Drop-Activation: Implicit Parameter Reduction and Harmonious Regularization

Senwei Liang
Department of Mathematics
Purdue University, IN 47907, USA
liang339@purdue.edu

Yuehaw Khoo
Department of Statistics and the College
The University of Chicago, Chicago, IL 60637
ykhoo@galton.uchicago.edu

Haizhao Yang
Department of Mathematics
Purdue University, IN 47907, USA*
Department of Mathematics
National University of Singapore, Singapore†
haizhao@purdue.edu

March 27, 2020

Abstract

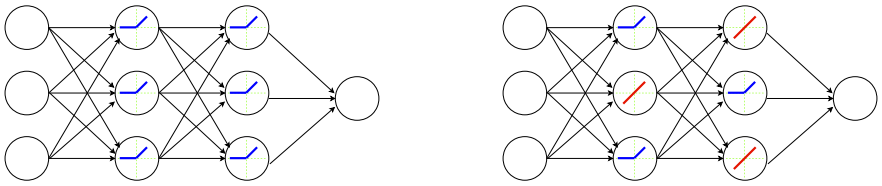
Overfitting frequently occurs in deep learning. In this paper, we propose a novel regularization method called Drop-Activation to reduce overfitting and improve generalization. The key idea is to drop nonlinear activation functions by setting them to be identity functions randomly during training time. During testing, we use a deterministic network with a new activation function to encode the average effect of dropping activations randomly. Our theoretical analyses support the regularization effect of Drop-Activation as implicit parameter reduction and verify its capability to be used together with Batch Normalization [11]. The experimental results on CIFAR-10, CIFAR-100, SVHN, EMNIST, and ImageNet show that Drop-Activation generally improves the performance of popular neural network architectures for the image classification task. Furthermore, as a regularizer Drop-Activation can be used in harmony with standard training and regularization techniques such as Batch Normalization and Auto Augment [3]. The code is available at `https://github.com/LeungSamWai/Drop-Activation`.

Keywords. Deep Learning, Image Classification, Overfitting, Regularization

1 Introduction

Convolution neural network (CNN) is a powerful tool for computer vision tasks. With the help of gradually increasing depth and width, CNNs [6, 7, 9, 29, 26] gain a significant improvement in image classification problems by capturing multiscale features [31]. However, when the number of trainable parameters is 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 [3], weight decay [12], Dropout [21] and Batch Normalization (BN) [11] to prevent overfitting and improve generalization.

*Current institute.
†Institute when the project was started.



(a) Standard neural network with non-linearity.

(b) After applying Drop-Activation during training.

Figure 1: Illustration 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).

Although regularization has been an essential part of 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 does not always give improved performance. For instance, the network trained with both Dropout and BN may not produce a better result [11, 16]. Dropout may change the statistical variance of layers output when we switch from training to testing, while BN requires the variance to be the same during both training and testing [16].

Our contributions: To deal with the aforementioned challenges, we propose a novel regularization method, Drop-Activation, inspired by the works in [21, 4, 10, 28, 24, 20, 14], 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 the most popular architectures and regularization methods, leading to improved performance in various datasets for image classification.

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 combination of identity mapping and the dropped nonlinearity, to represent the ensemble average of the random networks generated by Drop-Activation. Rectified linear unit (ReLU) has the advantage of reducing the saturation of gradient and accelerating the training compared with sigmoid or tanh activation function [13]. It is frequently adopted in modern deep neural networks [6, 7, 9, 29, 26]. In this paper, we focus on studying the random replacement of the ReLU activation function with the identity function.

The starting point of Drop-Activation is to randomly draw an ensemble of neural networks with either an identity mapping 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 ensemble work well when being assigned with these parameters. By "fitting" to many neural-networks instead of a fixed one, overfitting can potentially be prevented. Indeed, our theoretical analysis shows that Drop-Activation implicitly adds a penalty term to the loss function, aiming at network parameters such that the corresponding deep neural network can be approximated by a linear network, i.e, implicit parameter reduction.

Organizations: The remainder of this paper is structured as follows. In Section 2, we review some of the regularization methods and discuss their relations with our work. In Section 3, we formally introduce Drop-Activation. In Section 4, the theoretical analysis demonstrates the regularization of Drop-Activation and its synergy with BN. In Section 5, these advantages of Drop-Activation are further supported by our numerical experiments carried on different datasets and networks.

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 [13, 19, 4, 3] or output labels [32, 25]. Another class of methods regularize the network by adding randomness into various neural network structures such as nodes [21], connections [24], pooling layers [30], activations [27] and residual blocks [5, 10, 28]. In particular [21, 4, 10, 28, 24, 20, 14] add randomness by dropping some structures of neural networks at random in training.

Dropout [21] drops nodes along with its connection with some fixed probability during training. DropConnect [24] has a similar idea but masks out some weights randomly. [10] improves the performance of ResNet [6] by dropping the entire residual block at random during training and passing through skip connections (identity mapping). This idea is also used in [28] when training ResNeXt [26] type 2-residual-branch network. The idea of dropping also arises in data augmentation. Cutout [4] randomly cuts out a square region of training images to prevent the neural network from putting too much emphasis on the specific region of features.

A related idea of dropping "functions" in neural networks were proposed in [20, 14], where subnetwork structures are discarded randomly, instead of replacing an activation function with the identify function. [20] proposes a framework of Swapout $\Theta_1 \otimes X + \Theta_2 \otimes F(X)$, where X is the input feature map, F is a sub-network, Θ_1 and Θ_2 are i.i.d Bernoulli distribution, and $\otimes$ is the element-wise product. [20] verified the effectiveness of the proposed Swapout framework on a large structure, i.e., F is a residual in ResNet [6] which consists of layers of BN, ReLU, Convolution. Similarly, Zoneout [14] discussed the case when F is a set of layers. Dropping a subnetwork structure can lead to instability of training and it requires more careful hyperparameter tuning. On the contrary, Drop-Activation focuses on the nonlinear activation functions, a smaller and more basic structure of networks.We will show the effectiveness of Drop-Activation on a wider range of networks and datasets than those in [20, 14].

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 many regularizers without conflicts.

3 Formulation of 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,

$$f(x) = \left[\sigma\left(x[1]\right), \cdots, \sigma\left(x[d]\right)\right]^T \in \mathbb{R}^d, \quad x = \left[x[1], \cdots, x[d]\right]^T \in \mathbb{R}^d, \tag{1}$$

where σ could be a ReLU, a sigmoid or a tanh function but we only consider that σ is a ReLU function in our paper. For a 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 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 nonlinearity.

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

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

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.

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

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

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:

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

and the new activation function $(1-p)I + pf$ is the convex combination of an identity operator I and an activation operator f. Eqn. (4) is the deteministic nonlinearity used to generate a deterministic neural network for testing. In particular, when ReLU is used, then the new activation $(1-p)I + pf$ is the leaky ReLU with slope $1-p$ in its negative part [27].

4 Theoretical Analysis

In Section 4.1, we show that in a ReLU neural-network with one-hidden-layer, Drop-Activation provides a regularization via penalizing the difference between nonlinear activation network and linear network, which can be understood as implicit parameter reduction, i.e, the intrinsic dimension of the parameter space is reduced. In Section 4.2, we further show that the use of Drop-Activation does not impact some other techniques such as BN, which ensures the practicality of using Drop-Activation in deep networks.

4.1 Drop-Activation as a regularizer

We use similar ideas in [21] and [23] to 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 output. The output of the one-hidden layer neural ReLU network 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

$$\hat{y} = W_2((I - P + Pr)W_1 x) \tag{6}$$

during training, and

$$\hat{y} = W_2((1-p)I + pr)W_1 x = W_2 r_p(W_1 x) \tag{7}$$

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:

$$\min_{W_1,W_2} \sum_{i=1}^{n} \mathbb{E}\|W_2[(I - P + Pr)W_1 x_i] - y_i\|_2^2, \tag{8}$$

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 1. *The optimization problem (8) is equivalent to*

$$\min_{W_1,W_2} \sum_{i=1}^{n} \|W_2 r_p(W_1 x_i) - y_i\|_2^2 + p^{-1}(1-p)\|W_2 W_1 x_i - W_2 r_p(W_1 x_i)\|_2^2. \tag{9}$$

Proof. Suppose that x is the input vector. Let $D_{W_1,x} = \mathrm{diag}\{(W_1 x > 0)\}$, where $(W_1 x > 0)$ is a 0-1 vector, and the j-th component of $(W_1 x > 0)$ is equal to 1 if the j-th component of $W_1 x$ is positive or is equal to 0 else. Then, the ReLU mapping of $W_1 x$ can be written as $r(W_1 x) = D_{W_1,x} W_1 x$. For simplification, we denote

$$S := I - P + PD_{W_1,x}, \;\; S_p := I - pI + pD_{W_1,x}, \;\; v := W_1 x.$$

On one hand, $\|W_2 r_p(W_1 x) - y\|_2^2 = \|W_2 S_p v - y\|_2^2$. We expand it and obtain

$$\|W_2 S_p W_1 x - y\|_2^2 = \mathrm{tr}(W_2 S_p v v^T S_p W_2^T) - 2\mathrm{tr}(W_2 S_p v y^T) + \mathrm{tr}(yy^T), \tag{10}$$

where function $\mathrm{tr}(\cdot)$ is the trace operator computing the sum of matrix diagonal. We denote $\mathrm{vec}(\cdot)$ as a function converting the diagonal matrix into a column vector. Rewrite the first term of Eqn. (10) and get

$$\begin{aligned} \mathrm{tr}(W_2 S_p v v^T S_p W_2^T) =& \mathrm{tr}(S_p v v^T S_p W_2^T W_2) \\ =& \mathrm{tr}(\mathrm{diag}(v)\mathrm{vec}(S_p)\mathrm{vec}(S_p)^T \mathrm{diag}(v) W_2^T W_2) \\ =& \mathrm{tr}(\mathrm{vec}(S_p)\mathrm{vec}(S_p)^T \mathrm{diag}(v) W_2^T W_2 \mathrm{diag}(v)). \end{aligned} \tag{11}$$

On the other hand, we have

$$\begin{aligned} \mathbb{E}\|W_2[(I - P + Pr)W_1 x] - y\|_2^2 =& \mathbb{E}[\|W_2 S v - y\|_2^2] \\ =& \mathbb{E}[\mathrm{tr}(W_2 S v v^T S W_2^T)] - 2\mathrm{tr}(W_2 S_p v y^T) + \mathrm{tr}(yy^T), \end{aligned} \tag{12}$$

where the expectation is taken with respect to the feature noise $P = \{P_1, \cdots, P_d\}$. Similar to Eqn. (11), we combine the matrices containing random variables and obtain

$$\mathrm{tr}(W_2 S v v^T S W_2^T) = \mathrm{tr}(\mathrm{vec}(S)\mathrm{vec}(S)^T \mathrm{diag}(v) W_2^T W_2 \mathrm{diag}(v)). \tag{13}$$

Since $\mathrm{tr}(\cdot)$ has property of linearity, taking the expectation of Eqn. (13) with respect to P obtains

$$\mathbb{E}\mathrm{tr}(W_2 S v v^T S W_2^T) = \mathrm{tr}(\mathbb{E}(\mathrm{vec}(S)\mathrm{vec}(S)^T)\mathrm{diag}(v) W_2^T W_2 \mathrm{diag}(v)). \tag{14}$$

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

$$\begin{aligned} \mathbb{E}[\mathrm{vec}(S)\mathrm{vec}(S)^T] - \mathrm{vec}(S_p)\mathrm{vec}(S_p)^T =& \mathrm{diag}(\{\mathbb{E}((1 - P_i + P_i d_i)^2) - (1 - p + p d_i)^2\}_{i=1}^k) \\ =& p(1-p)(I - D_{W_1,x})^2. \end{aligned} \tag{15}$$

Using Eqn. (15), Eqn. (11) and Eqn. (13), we can get the difference between Eqn. (10) and Eqn. (12),

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

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

$$p(1-p)\|W_2(I - D_{A,x}) W_1 x\|_2^2 = \frac{1-p}{p}\|W_2(I - S_p) W_1 x\|_2^2 = \frac{1-p}{p}\|W_2 W_1 x - W_2 r_p(W_1 x)\|_2^2.$$

Finally, we attain the difference between Eqn. (10) and Eqn. (12),

$$\frac{1-p}{p}\|W_2 W_1 x - W_2 r_p(W_1 x)\|_2^2.$$

□

We refer to the objective function of the optimization (9). 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 (7). Therefore, Property 1 shows that Drop-Activation incurs a penalty

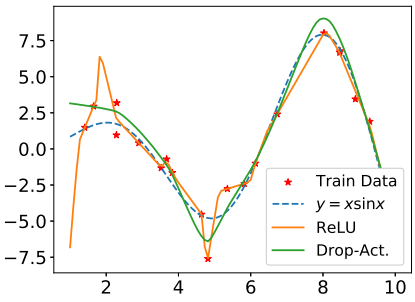
$$p^{-1}(1-p)\|W_2 W_1 x_i - W_2 r_p(W_1 x_i)\|_2^2 \tag{16}$$

on top of the prediction loss. In Eqn. (16), the coefficient $\frac{1-p}{p}$ influences the magnitude of the penalty. In our experiments, p is selected to be a large number close to 1 (typically 0.95). The magnitude of the penalty will not be large in our numerical experiments.

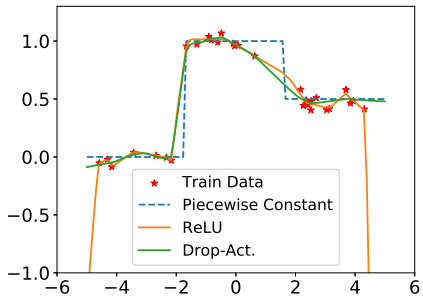
The penalty (16) consists of the terms $W_2 W_1 x$ and $W_2 r_p(W_1 x)$. $W_2 W_1 x$ has no nonlinearity, so it is a linear network. In contrast, since $W_2 r_p(W_1 x)$ has the nonlinearity r_p, it can be considered as a deep network. The two networks share the same parameters W_1 and W_2. Therefore the penalty (16) 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 linear network. 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. To generate the training dataset, we sample 20 (x_i, y_i) pairs from the ground truth function and add gaussian noise on the outputs. Then we train a fully connected network with three hidden layers of width 1000, 800, 200, respectively. Figure 2a and 2b show that the network with 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 lower prediction error. With the effect of incurred penalty (16), the network with Drop-Activation reduces the influence of data noise and yields a smooth curve.

Figure 6 shows the training of ResNet164 on CIFAR100, the training error with Drop-Activation is slightly larger than that without Drop-Activation. However, in terms of the generalization error, Drop-Activation gives improved performance. This verifies that the original network has been over-parametired and Drop-Activation can regularize the network by implicit parameter reduction.



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



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

Figure 2: 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 ReLU. Green: Regression results using Drop-Activation. "$*$": Training data perturbed by Gaussian noise.

4.2 Compatibility of Drop-Activation with BN

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 BN. [16] argues that BN assumes the output of each layer has the same variance during training and testing. However, Dropout [21] will shift the variance of the output during the testing time leading to disharmony when used in conjunction with BN. Using a similar analysis as [16], we show that unlike Dropout, Drop-Activation can be used together with BN since it maintains the output variance.

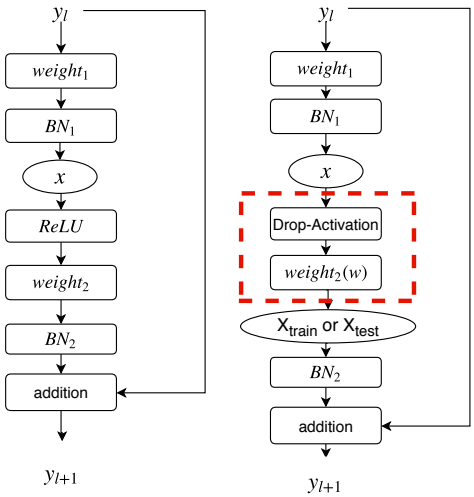


Figure 3: **Left**: A basic block in ResNet. **Right**: A basic block of a network with Drop-Activation.

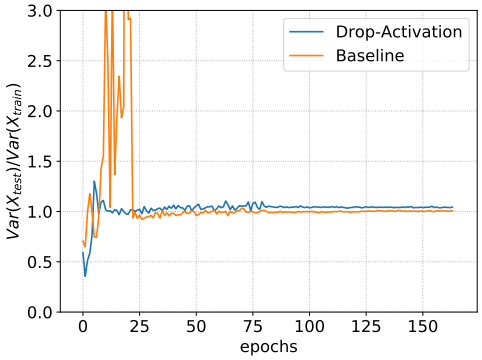


Figure 4: The shift ratio of the output of the second stage for ResNet-164. $\mathrm{Var}(X_{\mathrm{train}})$ and $\mathrm{Var}(X_{\mathrm{test}})$ denote the average of the variance for the output of the second stage during training and testing respectively.

To this end, we analyze the mappings in ResNet [6]. Figure 3 (Left) shows a basic block of ResNet while Figure 3 (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 3 is $x = (x[1], \cdots, x[d])$. [15] shows the hidden features converge in distribution to the Gaussian when d is large, so for simplification, we assume

that $x[i] \sim \mathcal{N}(0,1),\ i = 1, \ldots, d$ 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 $X_{\text{train}} := \sum_{i=1}^{d} w_i((1-P_i)x[i] + P_i r(x[i]))$, where $P = \text{diag}(P_1, \cdots, P_d)$ and $P_i \sim B(p)$. Similarly, during testing, taking the expectation over P_i's gives $X_{\text{test}} := \sum_{i=1}^{d} w_i((1-p)x[i] + pr(x[i]))$. The output of the rectangular box X_{train} (and X_{test} during testing) is then used as the input to BN_2 in Figure 3. Since for BN 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 3. 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 [16]: Shift Ratio $= \text{Var}(X_{\text{test}})/\text{Var}(X_{\text{train}})$ as a metric for evaluating the variance shift. The shift ratio is expected to be close to 1, since the BN layer BN_2 requires its input having similar variance in both training and testing time.

Property 2. *The shift ratio of X_{train} and X_{test} is*

$$Var(X_{test})/Var(X_{train}) = [(\pi-1)p^2 - 2\pi p + 2\pi]/[2\pi - \pi p - p^2]. \tag{17}$$

Proof. Since $x[i] \sim \mathcal{N}(0,1)$, it is easy to get $\mathbb{E}(x[i]) = 0$, $\mathbb{E}(r(x[i])) = \frac{1}{\sqrt{2\pi}}$, $\mathbb{E}(x[i]^2) = 1$, and $\mathbb{E}(r(x[i])^2) = \frac{1}{2}$, where the expectation is taken with respect to random variable $x[i]$. We have

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

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

Expand the square of X_{train} to get

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

Then we obtain its expectation,

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

Using the fact that $\text{Var}(X_{\text{train}}) = \mathbb{E}(X_{\text{train}}^2) - (\mathbb{E}X_{\text{train}})^2$, we get

$$\text{Var}(X_{\text{train}}) = \sum_{i=1}^{d} w_i^2(1 - p + \frac{1}{2}p) + \frac{p^2}{\pi}\sum_{i<j} w_i w_j - (\frac{1}{\sqrt{2\pi}}p\sum_{i=1}^{d} w_i)^2 = \sum_{i=1}^{d} w_i^2(1 - \frac{1}{2}p - \frac{1}{2\pi}p^2). \tag{18}$$

So far, we have finished $\mathrm{Var}(X_{\text{train}})$. Now we are going to compute $\mathrm{Var}(X_{\text{test}})$. Expand X_{test}^2 to get

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

We take expectation with respect to the input x,

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

Using the fact that $\mathrm{Var}(X_{\text{test}}) = \mathbb{E}(X_{\text{test}}^2) - (\mathbb{E}(X_{\text{test}}))^2$, we can obtain that

$$\mathrm{Var}(X_{\text{test}}) = \sum_{i=1}^{d} w_i^2((\frac{1}{2} - \frac{1}{2\pi})p^2 - p + 1). \tag{19}$$

With Eqn. 18 and Eqn. 19, we have

$$\frac{\mathrm{Var}(X_{\text{test}})}{\mathrm{Var}(X_{\text{train}})} = \frac{(\frac{1}{2} - \frac{1}{2\pi})p^2 - p + 1}{1 - \frac{1}{2}p - \frac{1}{2\pi}p^2}. \tag{20}$$

□

In Eqn. (17), the range of the shift ratio lies on the interval $[0.8, 1]$. In particular, when $p = 0.95$, $\mathrm{Var}(X_{\text{test}})/\mathrm{Var}(X_{\text{train}}) \approx 0.9377$, therefore $\mathrm{Var}(X_{\text{test}})$ is close to $\mathrm{Var}(X_{\text{train}})$. This shows that in Drop-Activation, the difference in the variance of inputs to a BN 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 ResNet164 with CIFAR100. ResNet164 consists of a stack of three stages. Each stage contains 54 convolution layers with the same spatial size. We observe the statistics of the output of the second stage 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 4, the shift ratio stabilizes close to 1 at the end of the training, which is consistent with our analysis.

In summary, by maintaining the statistical property of the internal output of hidden layers in testing time, Drop-Activation can be combined with BN to improve performance.

5 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 on various datasets. This section is organized as followed. Section 5.1 contains basic experiment settings. In Section 5.2, we introduce the datasets and implementation details. In section 5.3, we present the numerical results.

5.1 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 when having f as ReLU in Eqn. (5) and the randomized leaky rectified linear units (RReLU), 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 upon 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.

Comparison with RReLU: RReLU is proposed in [27] with the following training scheme for an input vector x,

$$\mathrm{RReLU}(x)[j] = \begin{cases} x[j], & x[j] \geq 0, \\ U_j x[j], & x[j] < 0, \end{cases} \tag{21}$$

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. (21) 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. (21) that RReLU passes the negative data with a random shrinking rate, while Drop-Activation randomly lets the complete information pass. The parameters a and b in RReLU are set at 1/8 and 1/3 respectively, as suggested in [27].

Improvement upon modern neural network architectures: The residual-type neural network structures greatly facilitate the optimization for deep neural network [6] and are employed by ResNet [6], Pre-ResNet [7], DenseNet [9], ResNeXt [26], WideResNet (WRN)[29] and SENet [8]. We demonstrate that Drop-Activation works well with these modern architectures. Moreover, since these networks use BN to accelerate training and may contain Dropout to improve generalization. e.g., WRN, these experiments also show the ability of Drop-Activation to work in synergy with the prevalent training techniques.

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 [4] and AutoAugment [3]. Cutout randomly masks a square region of training data and AutoAugment uses reinforcement learning to obtain an improved data augmentation scheme.

5.2 Datasets and implementation details

Choosing the 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 CIFAR10 without data augmentation and perform a grid search for p on the interval $[0.6, 1.0]$, with a step size equal to 0.05. The simple network consists of three convolution layers and two fully connected layers, and details are in the Appendix. We split the train set of CIFAR10 into two parts, 10% for validation and 90% for training. The Figure 5 shows the validation error on CIFAR10 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.

Datasets and implementation: We train the models with Drop-Activation on CIFAR10, CIFAR100 [12], SVHN [17], EMNIST ("Balanced") [2] and ImageNet 2012 [18] (random cropping size 224×224). When applying Drop-Activation to these models, we directly substitute all the original ReLU function with Drop-Activation except for the case of ImageNet. In particular, due to the relatively underfitting of training on ImageNet, only ReLUs in the last two stages of networks are modified by Drop-Activation. All the models are optimized using SGD with a momentum of 0.9 [22]. The other implementation details are given in the Appendix.

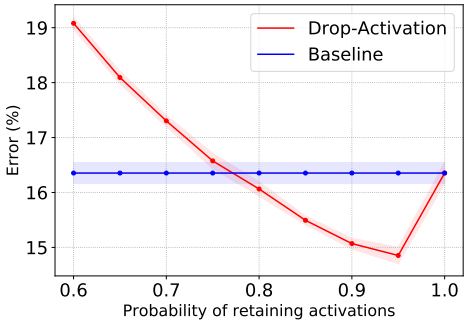


Figure 5: Validation error on CIFAR10 with 95% confidence intervals with respect to the probability p of retaining activation (average of 20 runs).

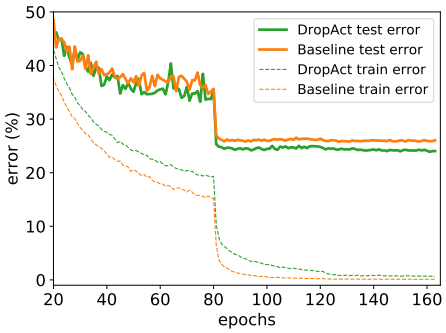


Figure 6: Training curves on CIFAR100 with ResNet164.

5.3 Experiment results

Table 1, 2 and 3 show the testing error on different datasets. The baseline results are from original networks without Drop-Activation. Table 5 shows the training time of different models. In what follows, we discuss how our results support the points raised in Section 5.1 and analyse the training time of applying Drop-Activation.

Comparison with RReLU: As shown in Table 1, RReLU may have worse performance than the baseline method. However, Drop-Activation consistently results in superior performance over RReLU and almost all baseline networks. Although Drop-Activation can not reduce the testing error of ResNeXt-8×64d on CIFAR10, Drop-Activation with DenseNet190-40 has the best testing error smaller than that of the original ResNeXt29-8×64d.

	CIFAR10			CIFAR100		
	Baseline	RReLU	Drop-Act	Baseline	RReLU	Drop-Act
VGG19(BN)	6.56±0.13	6.60±0.22	**6.38±0.07**	28.67±0.30	28.62±0.15	**28.55±0.28**
ResNet110	6.77±0.27	7.37±0.22	**6.25±0.06**	28.24±0.13	29.64±0.06	**27.91±0.18**
ResNet164	5.94±0.27	6.08±0.03	**5.62±0.08**	25.86±0.42	24.78±0.43	**24.18±0.22**
PreResNet164	5.01±0.03	5.17±0.12	**4.87±0.16**	23.49±0.17	23.21±0.05	**22.79±0.16**
WideResNet28-10	3.85±0.13	4.32±0.05	**3.74±0.05**	18.84±0.26	19.53±0.13	**18.14±0.22**
DenseNet100-12	4.73±0.10	5.06±0.03	**4.38±0.09**	22.66±0.25	22.59±0.20	**21.80±0.21**
DenseNet190-40	3.91±0.15	3.84±0.08	**3.51±0.06**	17.28±0.45	18.58±0.12	**16.80±0.12**
ResNeXt29-8×64	3.95±0.05	4.56±0.16	3.95±0.18	18.56±0.38	18.65±0.08	**17.65±0.16**

Table 1: Test error (%) on CIFAR10 an CIFAR100. The test accuracy is averaged over three repeated experiments. We use Baseline to indicate the usage of the original architecture without modifications.

Application to modern models: As shown in Table 1, Drop-Activation in almost all cases improves the testing accuracy consistently comparing to Baseline for CIFAR10 and CIFAR100. To further demonstrate this, we apply Drop-Activation to various neural-network architectures and demonstrate the successes on the datasets SVHN, EMNIST, and ImageNet. Again, in Table 2 and 3 we see a consistent improvement when Drop-Activation is used.

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 BN or Dropout used in training these networks.

Models	SVHN		EMNIST	
	Base	Drop-Act	Base	Drop-Act
ResNet164	-	-	8.85	**8.82**
PreResNet164	-	-	8.88	**8.72**
WRN16-8	1.54	**1.46**	-	-
WRN28-10	-	-	8.97	**8.72**
DenseNet100-12	1.76	**1.71**	**8.81**	8.90
ResNeXt29,8*64	1.79	**1.69**	9.07	**8.91**

Table 2: Test error (%) on SVHN, EMNIST (Balanced). The Baseline results of WRN and DenseNet for SVHN are obtained from the original papers.

	ImageNet 2012	
Models	Baseline	Drop-Act
ResNet34	26.07	**25.85**
SENet50	23.39	**23.18**

Table 3: Validation error (%) on ImageNet.

Compatibility with other regularization approaches: We apply Drop-Activation to network models that use Cutout or AutoAugment. As shown in Table 4, Drop-Activation can further improve with Cutout or AutoAugment by decreasing the test error on CIFAR100 and CIFAR10.

Model	Dataset	Baseline	DA	with Cutout (CO)		with AutoAug (AA)	
				CO	CO+DA	AA	AA+DA
ResNet18	CIFAR100	22.46	21.61	21.96	20.99	-	-
ResNet164	CIFAR100	25.86	24.18	-	-	21.12	20.39
WideResNet28-10	CIFAR100	18.84	18.14	18.41	17.86	17.09	16.20
DenseNet190-40	CIFAR10	3.91	3.51	3.15	2.79	2.54	2.36

Table 4: Test error (%) for CIFAR100 or CIFAR10 with combination of Drop-Activation (DA) and Cutout (CO) or AutoAugement (AA). The results of Cutout are quoted from [4]. The WideResNet result of AutoAug is quoted from [3].

Training time: The increment of the computational cost of the Drop-Activation network compared with the ReLU network comes from the different realizations of Bernoulli random variables for each activation function. This results in an unavoidable increment of training time. Table 5 shows the training time of each batch for different models. In particular, RReLU that we use is Pytorch official function. We train ResNet164 and WideResNet28 with batch size 128 on CIFAR10 using the workstation with CPU AMD Ryzen Threadripper 1920X and 2 GPUs 2080Ti. From Table 5, we can see that both Drop-Activation and officially implemented RReLU suffer from the training time increment.

Model	Baseline	RReLU	Drop-Activation
ResNet164	0.151	0.175	0.223
WideResNet28-10	0.128	0.212	0.179

Table 5: The training time (sec) for each batch of different models.

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 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 time, 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.

7 Conflict of Interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

Acknowledgments. S. Liang and H. Yang gratefully acknowledge the support of National Supercomputing Center (NSCC) Singapore [1] and High-Performance Computing (HPC) of the National University of Singapore for providing computational resources, and the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research. H. Yang was partially supported by National Science Foundation under the grant award 1945029.

References

[1] The computational work for this article was partially performed on resources of the national supercomputing centre, singapore (https://www.nscc.sg).

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

[3] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation strategies from data. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 113–123, 2019.

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

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

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In European conference on computer vision, pages 630–645. Springer, 2016.

[8] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 7132–7141, 2018.

[9] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700–4708, 2017.

[10] 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.

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

[12] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[13] 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.

[14] D. Krueger, T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N. R. Ke, A. Goyal, Y. Bengio, A. Courville, and C. Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. arXiv preprint arXiv:1606.01305, 2016.

[15] J. Lee, L. Xiao, S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In Advances in neural information processing systems, pages 8570–8581, 2019.

[16] X. Li, S. Chen, X. Hu, and J. Yang. Understanding the disharmony between dropout and batch normalization by variance shift. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2682–2690, 2019.

[17] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. International journal of computer vision, 115(3):211–252, 2015.

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

[20] S. Singh, D. Hoiem, and D. Forsyth. Swapout: Learning an ensemble of deep architectures. In Advances in neural information processing systems, pages 28–36, 2016.

[21] 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.

[22] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In International conference on machine learning, pages 1139–1147, 2013.

[23] S. Wager, S. Wang, and P. S. Liang. Dropout training as adaptive regularization. In Advances in neural information processing systems, pages 351–359, 2013.

[24] 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.

[25] 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.

[26] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1492–1500, 2017.

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

[28] Y. Yamada, M. Iwamura, T. Akiba, and K. Kise. Shakedrop regularization for deep residual learning. arXiv preprint arXiv:1802.02375, 2018.

[29] S. Zagoruyko and N. Komodakis. Wide residual networks. arXiv preprint arXiv:1605.07146, 2016.

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

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

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

8 Appendix

8.1 The simple model for finding the best parameter p

To find the best parameter for Drop-Activation, we perform a grid search on a simple model. The simple network consists of the following layers: We first stack three blocks, and each block contains convolution with 3×3 filter, BN, ReLU, and average pooling, as shown in Figure 7. The number of 3×3 filters for Block_1, Block_2, Block_3 is 32, 64, 128 respectively. The widths for fully connected layers are 1000 and 10 respectively.

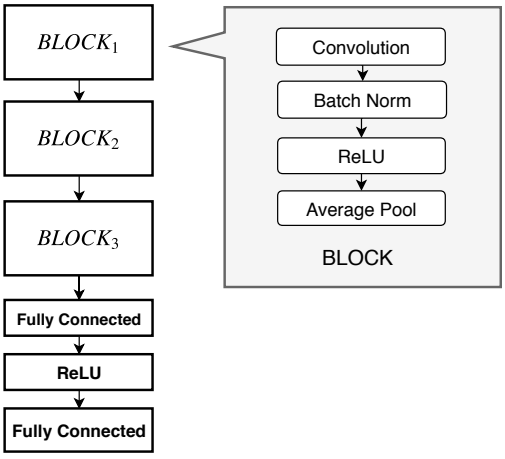


Figure 7: The model for finding the best parameter for Drop-Activation.

8.2 Introdcution of datasets

We use the following datasets in our numerical experiments,

CIFAR: Both CIFAR10 and CIFAR100 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. CIFAR100 is similar to CIFAR10, 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 [6].

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 [29]. The pixel values of the images are rescaled to $[0, 1]$, and no data augmentation is applied.

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 the "Balanced" split, there are 131,600 images in total, including 112,800 for training and 18,800 for testing.

ImageNet 2012: The ImageNet 2012 dataset consists of 1.28 million training images and 50K validation images from 1,000 classes. The models are evaluated on the validation set. We train the models for 120 epochs with an initial learning rate 0.1.

8.3 Implementation detail

The hyper-parameters for different networks are shown in Table 6, 7 and 8, and we offer the explanation of hyper-parameter names in Table 9.

	ResNet	PreResNet	WRN-28	ResNext29-8*64	VGG19(BN)	DenseNet190	DenseNet100
Batch size	128	128	128	128	128	32	64
Epoch	164	164	200	300	200	300	300
Optimizer	SGD(0.9)	SGD(0.9)	SGD(0.9)	SGD(0.9)	SGD(0.9)	SGD(0.9)	SGD(0.9)
Depth	-	-	28	29	19	190	100
Schedule	81/122	81/122	80/120/160	150/225	80/140	150/225	150/225
Weight-decay	1.00E-04	1.00E-04	5.00E-04	5.00E-04	1.00E-04	1.00E-04	1.00E-04
Gamma	0.1	0.1	0.2	0.1	0.1	0.1	0.1
Grow-rate	-	-	-	-	-	40	12
Widen-factor	-	-	10	4	-	-	-
Cardinality	-	-	-	8	-	-	-
LR	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Dropout	-	-	0.3	-	-	-	-

Table 6: Hyper-parameter setting for training models on CIFAR10/100 and EMNIST.

	ResNet34	SENet50
Batch size	256	256
Epoch	120	120
Optimizer	SGD(0.9)	SGD(0.9)
Depth	34	50
Schedule	30/60/90	30/60/90
Weight-decay	1.00E-04	1.00E-04
Gamma	0.1	0.1
LR	0.1	0.1

Table 7: Hyper-parameter setting for training models on ImageNet.

	WRN-16	ResNext29-8*64	DenseNet100
Batch size	128	128	64
Epoch	160	100	40
Optimizer	SGD(0.9)	SGD(0.9)	SGD(0.9)
Depth	16	29	100
Schedule	80/120	40/70	20/30
Weight-decay	5.00E-04	5.00E-04	1.00E-04
Gamma	0.2	0.1	0.1
Grow-rate	-	-	12
Widen-factor	8	4	-
Cardinality	-	8	-
LR	0.01	0.1	0.1
Dropout	0.4	-	-

Table 8: Hyper-parameter setting for training models on SVHN.

Batch size	Number of samples for training at each iteration
Epoch	Number of total epochs to train
Depth	The depth of network
Schedule	Decrease learning rate at these epochs
Weight-decay	The coefficient of l2 loss
Gamma	Learning rate is multiplied by Gamma on schedule
Widen-factor	Widen factor
Cardinality	Model cardinality (group)
LR	initial learning rate
Dropout	Dropout ratio

Table 9: The explanation of hyper-parameter names.