Drop-Activation: Implicit Parameter Reduction and Harmonious Regularization

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

Yuehaw Kwoo
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

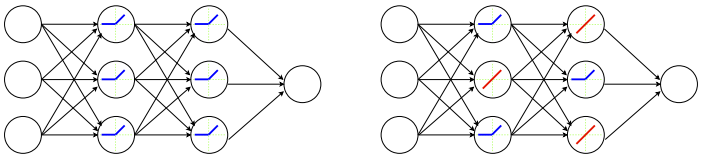
Overfitting frequently occurs in deep learning. In this paper, we propose a novel regularization method called Drop-Activation to reduce overfitting and improve generalization. The key idea is to drop nonlinear activation functions by setting them to be identity functions randomly during training time. 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, EMNIST, and ImageNet 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 verify 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 [1, 2, 3, 4, 5] gain a significant improvement in image classification problems by capturing multiscale features [6]. 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 [7], weight decay [8], Dropout [9] and Batch Normalization [10] 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 [10]. 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 [11].

Preprint. Under review.



((a)) Standard neural network with nonlinearity

((b)) After applying Drop-Activation during training

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).

Our contributions: To deal with the aforementioned challenges, we propose a novel regularization method, Drop-Activation, inspired by the works in [9, 12, 13, 14, 15], 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 draw an ensemble 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 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 shallower neural network, i.e.,, implicit parameter reduction.

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 different datasets. 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 [16, 17, 12, 7] or output labels [18, 19]. Another class of methods regularize the network by adding randomness into various neural network structures such as nodes [9], connections [15], pooling layers [20], activations [21] and residual blocks [22, 13, 14]. In particular [9, 12, 13, 14, 15] add randomness by dropping some structures of neural networks at random 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 [9] drops nodes along with its connection with some fixed probability during training. DropConnect [15] has a similar idea but masks out some weights randomly. [13] improves the performance of ResNet [1] 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 [14] when training ResNeXt [5]

type 2-residual-branch network. The idea of dropping also arises in data augmentation. Cutout [12] 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.,,

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

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 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.

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

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

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, if ReLU is used, then the new activation $(1-p)I + pf$ is the Leaky ReLU with slope $1-p$ [21].

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 on various datasets. This section is organized as followed. 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 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 [21] with the following training scheme for an input vector x,

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

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. The parameters a and b in RReLU are set at 1/8 and 1/3 respectively, as suggested in [21].

Improvement upon modern neural network architectures The residual-type neural network structures greatly facilitate the optimization for deep neural network [1] and are employed by ResNet [1], PreResNet [2], DenseNet [3], ResNeXt [5], WideResNet (WRN)[4] and SENet [23]. 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 (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 [12] and AutoAugment [7]. Cutout randomly masks a square region of training data and AutoAugment uses reinforcement learning to obtain an improved data augmentation scheme.

4.2 Datasets and implementation details

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 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 Appendix. Figure **??** 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.

Datasets and implementation: We train the models with Drop-Activation on CIFAR-10, CIFAR-100 [8], SVHN [24], EMNIST ("Balanced") [25] and ImageNet 2012 [26]. 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, for ImageNet, random cropping of the image to size 224×224 is used, and only ReLUs in the last two stages of networks are modified by Drop-Activation. All the models are optimized using SGD with the momentum of 0.9 [27]. The other implementation details are given in the Appendix.

4.3 Experiment Results

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

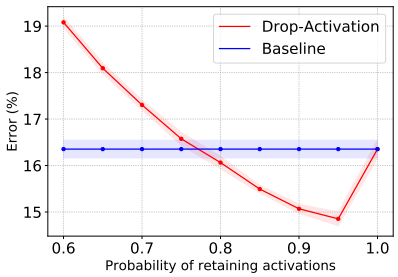


Figure 3: Testing error on CIFAR-10 with 95% confidence intervals with respect to the probability p of retaining activation (average of 20 runs). fig:parameters

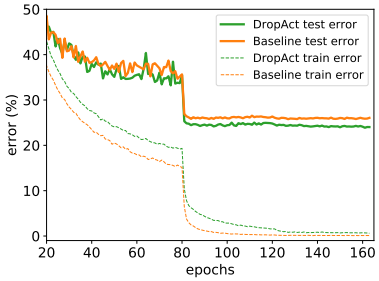


Figure 4: Training curves on CIFAR-100 with ResNet-164. fig:resnet-164

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 methods. Although Drop-Activation can not reduce the testing error of ResNeXt-8×64d, Drop-Activation with DenseNet-190-40 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*	6.91	**6.38**	28.67*	28.62	**28.55**
ResNet110	6.43	7.66	**6.17**	28.24*	29.64	**27.91**
ResNet164	5.94*	6.08	**5.62**	25.16	24.74	**23.88**
PreResNet164	5.46	5.33	**4.87**	24.33	23.22	**22.72**
WideResNet28-10	3.89	4.31	**3.74**	18.85	19.63	**18.14**
DenseNet100-12	4.51	5.02	**4.40**	22.27	22.82	**21.71**
DenseNet190-40	3.75*	3.78	**3.45**	17.63*	18.70	**16.92**
ResNeXt29-8×64	**3.65**	4.80	4.16	17.77	18.53	**17.68**

Table 1: Test error (%) on CIFAR-10 an 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 the results with "*" and ResNet-164 where the results are obtained from [2].

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 improvements when Drop-Activation is used. In particular, Drop-Activation improves ResNet, PreResNet and WRN by reducing the relative test error for CIFAR10, CIFAR100 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.

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 CIFAR-100 and CIFAR-10.

5 Theoretical Analysis

In Section 5.1, we show that in a neural-network with one-hidden-layer, Drop-Activation provides a regularization via penalizing the difference between deep and shallow networks, which can be understood as implicit parameter reduction, i.e.,, the intrinsic dimension of the parameter space is

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: Test error (%) on ImageNet.

	Dataset	with Cutout (CO)			with AutoAug (AA)		
		Baseline	CO	CO+DA	Baseline	AA	AA+DA
ResNet-18	CIFAR100	22.46	21.96	**20.99**	-	-	-
ResNet-164	CIFAR100	25.16	24.13	**22.29**	25.16	21.12	**20.39**
WideResNet28-10	CIFAR100	18.85	18.41	**17.86**	18.85	17.09	**16.20**
DenseNet190-40	CIFAR10	3.75	3.15	**2.79**	3.75	2.54	**2.36**

Table 4: Test error(%) for CIFAR-100 or CIFAR-10. Combination of Drop-Activation (DA) and Cutout (CO) or AutoAugement (AA). The results of Cutout are quoted from [12]. The WRN result of AutoAug is quoted from [7].

reduced. 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

We use similar ideas in [9] and [28] 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

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

during training, and

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

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:

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

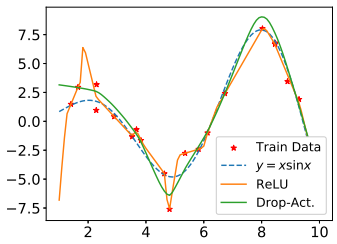
where the expectation is taken with respect to the feature noise $P_1, \cdots, P_d$. The use of Drop-Activation can be seen as applying a stochastic minimization to such an average loss. The result after averaging the loss function over P is summarized as follows.

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

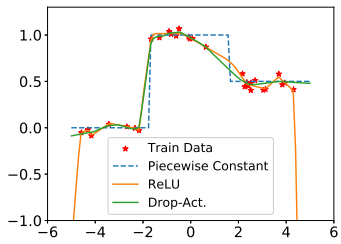
$$\min_{W_1, W_2} \sum_{i=1}^{n} \|W_2 r_p(W_1 x_i) - y_i\|_2^2 + p^{-1}(1-p)\|W_2 W_1 x_i - W_2 r_p(W_1 x_i)\|_2^2. \tag{10}$$

Proof of Property 5.1 can be found in Appendix. 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

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



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



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

Figure 4: 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.

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 close to 1 (typically 0.95).

The penalty (11) consists of the terms W_2W_1x and $W_2r_p(W_1x)$. Since W_2W_1x has no nonlinearity, it can be viewed as a shallow network. In contrast, since $W_2r_p(W_1x)$ 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 $W_2r_p(W_1x)$ should be somewhat close to that of a shallow network. In a classification or regression task, the shallow network has less representation power but 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. 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 4(a) and 4(b) 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. However, with the incurred penalty (11), the network with Drop-Activation yields a smooth curve. Furthermore, Drop-Activation reduces the influence of data noise.

Figure **??** shows the training of ResNet164 on CIFAR100, 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. [11] 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 [11], 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 [1]. Figure **??** (Left) shows a basic block of ResNet while Figure **??** (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 **??** 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 $X_{\text{train}} := \sum_{i=1}^{d} w_i((1-P_i)x[i] + P_i r(x[i]))$, where $P =$

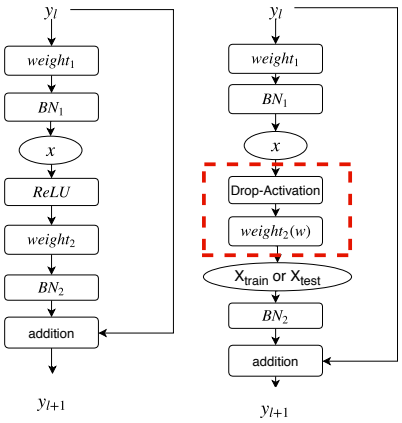


Figure 6: **Left**: A basic block in ResNet. **Right**: A basic block of a network with Drop-Activation.
fig:basicblock

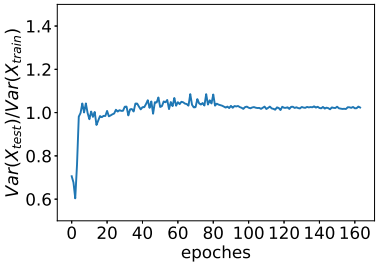


Figure 7: The shift ratio of the output of the second module for ResNet-164. $\text{Var}(X_{\text{train}})$ and $\text{Var}(X_{\text{test}})$ denote the average of the variance for the output of the second stage during training and testing respectively.
fig:var$_s hift$

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 **??**. 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 **??**. 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 [11]: 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 Batch Normalization layer BN_2 requires its input having similar variance in both training and testing time.

Property 5.2 *The the shift ratio of X_{train} and X_{test} is*

$$Var(X_{test})/Var(X_{train}) = [(\pi - 1)p^2 - 2\pi p + 2\pi]/[2\pi - \pi p - p^2]. \tag{12}$$

The proof of Property 5.2 is provided in Appendix. In Eqn. (12), the range of the shift ratio lies on the interval $[0.8, 1]$. In particular, when $p = 0.95$, $\text{Var}(X_{\text{test}})/\text{Var}(X_{\text{train}}) \approx 0.9377$, 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. ResNet-164 consists of a stack of three stages. Each stage contains 54 convolution layers. We observe the statics 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 **??**, the shift ratio stabilizes at 1 in the end of training.

In summary, by maintaining the statistical property of the internal output of hidden layers in testing time, 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 code of this paper will be available in the authors' personal webpages.

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.

7 Appendix

7.1 The simple model for finding the best parameters p

To find the best parameter for Drop-Activation, we perform grid search on the simple models. 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, as shown in Figure 7. The number of 3×3 filters for Block_1, Block_2, Block_3 is 32, 64, 128 respectively. The number of output nodes for fully connected layers is 1000 and 10 respectively.

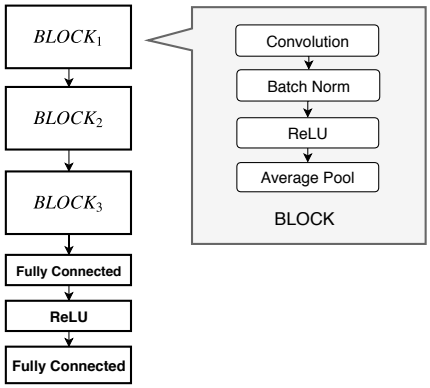


Figure 7: The model for finding the best parameter for Drop-Activation.

8 Introdcution of datasets and Implementation detail

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 [1].

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 [4]. 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 Balanced, 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. Due to the relatively underfitting of training on ImageNet, we only apply Drop-Activation to the last two stages of networks. We train the models for 120 epoches with initial learning rate 0.1.

	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 5: Hyper-parameter setting for training models on CIFAR-10/100 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
wd	1.00E-04	1.00E-04
gamma	0.1	0.1
lr	0.1	0.1

Table 6: 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 7: Hyper-parameter setting for training models on SVHN.

8.1 Proof of Property (5.1)

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, the ReLU map of W_1x can be written as $r(W_1x) = D_{W_1,x}W_1x$. For simplification, we 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

$$\begin{aligned} &\|W_2S_pW_1x - y\|_2^2 \\ =&\text{tr}(W_2S_pvv^TS_pW_2^T) - 2\text{tr}(W_2S_pvy^T) + \text{tr}(yy^T), \end{aligned} \tag{13}$$

where function 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 rewrite the first term of

Eqn. (13), and get

$$\begin{aligned}&\text{tr}(W_2 S_p v v^T S_p W_2^T)\\ =&\text{tr}(S_p v v^T S_p W_2^T W_2)\\ =&\text{tr}(\text{diag}(v)\text{vec}(S_p)\text{vec}(S_p)^T\text{diag}(v) W_2^T W_2)\\ =&\text{tr}(\text{vec}(S_p)\text{vec}(S_p)^T\text{diag}(v) W_2^T W_2 \text{diag}(v)).\end{aligned} \tag{14}$$

On the other hand, we have

$$\begin{aligned}&\mathbb{E}\|W_2[(I-P+Pr)W_1 x]-y\|_2^2\\ =&\mathbb{E}[\|W_2 S v - y\|_2^2]\\ =&\mathbb{E}[\text{tr}(W_2 S v v^T S W_2^T)] - 2\text{tr}(W_2 S_p v y^T) + \text{tr}(yy^T).\end{aligned} \tag{15}$$

where the expectation is taken with respect to the feature noise $P = \{P_1, \cdots, P_d\}$. Similar to Eqn. (14), we combine the matrices containing random variables and obtain

$$\begin{aligned}&\text{tr}(W_2 S v v^T S W_2^T)\\ =&\text{tr}(\text{vec}(S)\text{vec}(S)^T\text{diag}(v) W_2^T W_2 \text{diag}(v)).\end{aligned} \tag{16}$$

Since tr() has property of linearity, taking the expectation of Eqn. (16) with respect to P to obtain

$$\begin{aligned}&\mathbb{E}\text{tr}(W_2 S v v^T S W_2^T)\\ =&\text{tr}(\mathbb{E}(\text{vec}(S)\text{vec}(S)^T)\text{diag}(v) W_2^T W_2 \text{diag}(v)).\end{aligned} \tag{17}$$

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

$$\begin{aligned}&\mathbb{E}[\text{vec}(S)\text{vec}(S)^T] - \text{vec}(S_p)\text{vec}(S_p)^T\\ =&\text{diag}(\{\mathbb{E}((1-P_i+P_i d_i)^2) - (1-p+pd_i)^2\}_{i=1}^k)\\ =&p(1-p)(I-D_{W_1,x})^2.\end{aligned} \tag{18}$$

Then, using Eqn. (18), Eqn. (14), Eqn. (16), we can get the difference between Eqn. (13) and Eqn. (15), this is,

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

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

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

Finally, we attain the difference between Eqn. (13) and Eqn. (15),

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

8.2 Proof of Property (5.2)

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

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

where we take expectation with respect to features 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 take expectation and obtain,

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

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

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

So far, we have finished $\text{Var}(X_{\text{train}})$. Then we are going to compute $\text{Var}(X_{\text{test}})$. 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,

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

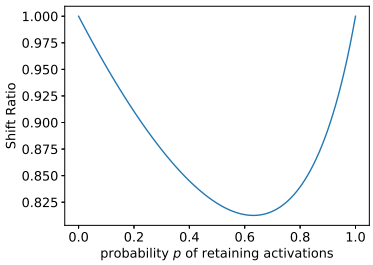


Figure 8: Shift ratio with respect to the probability p of retaining activations from 0 to 1.

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

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

Finally we have

$$\frac{\text{Var}(X_{\text{test}})}{\text{Var}(X_{\text{train}})} = \frac{(\frac{1}{2} - \frac{1}{2\pi})p^2 - p + 1}{1 - \frac{1}{2}p - \frac{1}{2\pi}p^2}. \tag{19}$$

To find the range of shift ratio, we plot the figure of shift ratio with respect to the p. As shown in Figure (8), the range of the shift ratio (19) lies on the interval $[0.8, 1]$.

Acknowledgments

S. Liang and H. Yang gratefully acknowledge the support of NATIONAL SUPERCOMPUTING CENTRE (NSCC) SINGAPORE [29] and High Performance Computing (HPC) of 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 thanks the support of the start-up grant by the Department of Mathematics at the National University of Singaporet, the Ministry of Education in Singapore for the grant MOE2018-T2-2-147.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[3] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[4] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.

[5] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

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

[7] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018.

[8] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

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

[11] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift. *arXiv preprint arXiv:1801.05134*, 2018.

[12] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[13] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pages 646–661. Springer, 2016.

[14] Yoshihiro Yamada, Masakazu Iwamura, and Koichi Kise. Shakedrop regularization. *arXiv preprint arXiv:1802.02375*, 2018.

[15] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[17]

[18] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[19] Lingxi Xie, Jingdong Wang, Zhen Wei, Meng Wang, and Qi Tian. Disturblabel: Regularizing cnn on the loss layer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4753–4762, 2016.

[20] Matthew D Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

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

[22] Xavier Gastaldi. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.

[23] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[24] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew 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.

[25] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[27] Ilya Sutskever, James Martens, George Dahl, and Geoffrey 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.

[28] Stefan Wager, Sida Wang, and Percy Liang. Dropout training as adaptive regularization. In *Proceedings of the 26th International Conference on NeurIPS - Volume 1*, pages 351–359, USA, 2013. Curran Associates Inc.

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