Defensive Dropout for Hardening Deep Neural Networks under Adversarial Attacks

Siyue Wang[1*], Xiao Wang[2*], Pu Zhao[1], Wujie Wen[3], David Kaeli[1], Peter Chin[2], Xue Lin[1]
[1] Northeastern University [2] Boston university
[3] Florida International University [*] Equal Contribution
wang.siy@husky.neu.edu kxw@bu.edu zhao.pu@@husky.neu.edu wwen@fiu.edu
kaeli@ece.neu.edu spchin@cs.bu.edu xue.lin@northeastern.edu

ABSTRACT

Deep neural networks (DNNs) are known vulnerable to adversarial attacks. That is, adversarial examples, obtained by adding delicately crafted distortions onto original legal inputs, can mislead a DNN to classify them as any target labels. This work provides a solution to hardening DNNs under adversarial attacks through defensive dropout. Besides using dropout during training for the best test accuracy, we propose to use dropout also at test time to achieve strong defense effects. We consider the problem of building robust DNNs as an attacker-defender two-player game, where the attacker and the defender know each others' strategies and try to optimize their own strategies towards an equilibrium. Based on the observations of the effect of test dropout rate on test accuracy and attack success rate, we propose a defensive dropout algorithm to determine an optimal test dropout rate given the neural network model and the attacker's strategy for generating adversarial examples. We also investigate the mechanism behind the outstanding defense effects achieved by the proposed defensive dropout. Comparing with stochastic activation pruning (SAP), another defense method through introducing randomness into the DNN model, we find that our defensive dropout achieves much larger variances of the gradients, which is the key for the improved defense effects (much lower attack success rate). For example, our defensive dropout can reduce the attack success rate from 100% to 13.89% under the currently strongest attack i.e., C&W attack on MNIST dataset.

1 INTRODUCTION

Deep neural networks (DNNs) are powerful models that achieve extraordinary performance in various speech and vision tasks, including speech recognition [15], natural language processing [7], scene understanding [17], and object recognition [20, 22, 23]. However, recent studies [13, 21, 31] show that DNNs are vulnerable to adversarial attacks implemented by generating adversarial examples, i.e., adding imperceptible but well-designed distortions to the original legal inputs. Delicately crafted adversarial examples can mislead a DNN to classify them as any target labels, while they appear recognizable and visually normal to human eyes.

ICCAD '18, November 5–8, 2018, San Diego, CA, USA

ACM ISBN 978-1-4503-5950-4/18/11…$15.00
https://doi.org/10.1145/3240765.3264699

Evidences have shown that audio/visual inputs sound/look like speeches/objects to DNNs but non-sense to humans [4, 26]. Recently Kurakin, Goodfellow, and Bengio have demonstrated the existence of adversarial attacks not only in theoretical models but also the physical world [21]. They mimicked the scenario of physical world application of DNNs by feeding the adversarial examples to a DNN through a cellphone camera to find that adversarial examples remain mis-classified by the DNN even when perceived through a camera.

Concerns have been aroused for applying DNNs in security-critical tasks. The security properties of DNNs have been widely investigated from two aspects: (i) crafting adversarial examples to test the vulnerability of DNNs and (ii) enhancing the robustness of DNNs under adversarial attacks. For the former aspect, adversarial examples have been generated by solving optimization problems [1, 5, 6, 13, 28, 31]. For the later aspect, research works have been conducted by either filtering out added distortions [2, 10, 14, 34] or revising DNN models [9, 11, 29] to defend against adversarial attacks. These two aspects mutually benefit each other towards hardening DNNs under adversarial attacks.

In this work, we provide a new solution to hardening DNNs under adversarial attacks through defensive dropout. Dropout is a commonly used regulation method to deal with the overfitting issue due to limited training data [30]. As a regulation method, dropout is applied during training that for each training case in a mini-batch, a sub-network is sampled by dropping some of the units (i.e., neural nodes). Based on some observations from preliminary experiments, we propose to use dropout also at test time as a defense method against adversarial attacks. By introducing dropout to test time, we achieve shorter and fatter distributions of the gradients, which is the key for the improved defense effects (lower attack success rate) compared with another model-randomness-based defense method i.e., stochastic activation pruning (SAP) [8]. For MNIST dataset, our defensive dropout reduces the attack success rate from 100% to 13.89% under the currently strongest attack i.e., C&W attack [5], while the distillation as a defense [29] and the adversarial training [32] are totally vulnerable under C&W attack. For CIFAR dataset, our defensive dropout reduces the attack success rate to 43.33%, while SAP can only reduce the attack success rate to 77.78% under C&W attack based on the same neural network model.

The contributions of this work are summarized as following:

(i) We consider the problem of building robust DNNs as an attacker-defender two-player game, where the attacker and the defender know each others' strategies and try to optimize their own strategies towards an equilibrium.

(ii) We propose a defensive dropout algorithm that determines an optimal test dropout rate given the neural network model and the

attacker's strategy for generating adversarial examples. Basically, we need to trade-off between the defense effects and test accuracy.

(iii) We explain the mechanism behind the outstanding defense effects by the proposed defensive dropout. The shorter and fatter gradient distributions make it difficult for the attacker to generate adversarial examples using the gradients from the sampled sub-networks.

2 RELATED WORK

2.1 Preliminaries

In this paper we focus on neural networks used as image classifiers. In this case, the input images can be denoted as 3-dimensional tensors $x \in \mathbb{R}^{h \times w \times c}$, where h, w and c denote the height, width and number of channels. For a gray scale image (e.g. MNIST), $c = 1$; and for a colored RGB image (e.g. CIFAR-10), $c = 3$. For both attacks and defends, all pixel values in the images are scaled to $[0, 1]$ for easy calculation, and therefore a valid input image should be inside a unit cube in the high dimensional space. We use model $F(x) = y$ to denote a neural network, where F accepts an input x and generates an output y.

Suppose the neural network is an m-class classifier and the output layer performs softmax operation. Let $Z(x)$ denote the output of all layers except for the softmax layer, and we have $F(x) = \text{softmax}(Z(x)) = y$. The input to the softmax layer, $Z(x)$, is called logits. The element y_i of the output vector y represents the probability that input x belongs to the i-th class. The output vector y is treated as a probability distribution and its elements satisfy $0 \leq y_i \leq 1$ and $y_1 + y_2 + \cdots + y_m = 1$. The neural network classifies input x according to the maximum probability i.e., $C(x) = \arg\max_i y_i$.

The adversarial attack can be either targeted or untargeted. Given an original legal input x with its correct label t^* and a target label $t \neq t^*$, the targeted adversarial attack is to find an input x' such that $C(x') = t$ and x and x' are close according to some measure of the distortion between x and x'. The input x' is then called as an adversarial example. The untargeted adversarial attack is to find an input x' satisfying $C(x') \neq t^*$ and x and x' are close according to some measure of the distortion. The untargeted adversarial attack does not specify any target label t to mislead the classifier. In this work, we consider targeted adversarial attacks since they are believed stronger than untargeted attacks.

The general problem of constructing adversarial examples can be formulated as: **Given** an original legal input x,

$$\begin{array}{ll} \text{minimize} & D(\delta) \\ \text{subject to} & C(x+\delta) = t \\ & x + \delta \in [0,1]^n \end{array} \tag{1}$$

where δ is the distortion added onto input x, $D(\delta)$ is a measure of the added distortion.

We need to measure the distortion between the original legal input x and the adversarial example $x' = x + \delta$. L_p norms are the most commonly used measures in the literature. The L_p norm of the distortion is defined as:

$$\|x - x'\|_p = \left(\sum_{i=1}^{n} |x_i - x'_i|^p \right)^{\frac{1}{p}} \tag{2}$$

We see the use of L_0, L_1, L_2, and L_∞ norms in different attacks.

- L_0 *norm*: measures the number of mismatched elements between x and x'.
- L_1 *norm*: measures the sum of the absolute values of the differences between x and x'.
- L_2 *norm*: measures the standard Euclidean distance between x and x'.
- L_∞ *norm*: measures the maximum difference between x_i and x'_i for all i's.

2.2 Attacks

2.2.1 Fault Injection [24]: published in ICCAD 2017, proposes two kinds of fault injection attacks that only require slight changes to the DNN's parameters to achieve misclassification: single bias attack (SBA) and gradient descent attack (GDA). SAB is able to achieve misclassification by modifying only one bias value in the network. And GDA achieves higher stealthiness and efficiency by using layer-wise searching and modification compression techniques. It implements very efficient attacks on MNIST and CIFAR-10 datasets. This work perceives the DNN attack problem from a different angle, i.e., modifying the DNN models, while all the other attacks and defends mentioned in this paper assume the modifications are performed onto the inputs.

2.2.2 Fast Gradient Sign Method (FGSM) [13]: is an L_∞ attack and utilizes the gradient of the loss function to determine the direction to modify the pixels. They are designed to be fast, rather than optimal. They can be used for adversarial training by directly changing the loss function instead of explicitly injecting adversarial examples into the training data. FGSM generates adversarial examples following:

$$x' = x - \epsilon \cdot \text{sign}(\nabla(loss_{F,t}(x))) \tag{3}$$

where ϵ is the magnitude of the added distortion, t is the target label. Using backpropagation, FGSM calculates the gradient of the loss function with respect to the label t to determine the direction to change the pixel values.

2.2.3 Jacobian-based Saliency Map Attack (JSMA) [28]: is an L_0 attack and uses a greedy algorithm that picks the most influential pixels by calculating Jacobian-based Saliency Map and modifies the pixels iteratively. The computational complexity of this attack method is very high.

2.2.4 C&W [5]: is a series of L_0, L_2, and L_∞ attacks that achieve 100% attack success rate with much lower distortions comparing with the above-mentioned attacks. In particular, the C&W L_2 attack is superior to other L_2 attacks because it uses a better objective function. C&W formulates the problem of generating adversarial examples in an alternative way that can be better optimized:

$$\begin{array}{ll} \text{minimize} & D(\delta) + c \cdot f(x+\delta) \\ \text{subject to} & x + \delta \in [0,1]^n \end{array} \tag{4}$$

where $c > 0$ is a constant to be chosen and objective function f has the following form:

$$f(x+\delta) = \max(\max\{Z(x+\delta)_i : i \neq t\} - Z(x+\delta)_t, -\kappa) \tag{5}$$

Here, κ is a parameter that controls the confidence in attacks. Stochastic gradient decent methods can be used to solve this problem. For example, the Adam optimizer [18] is used due to its fast and robust convergence behavior.

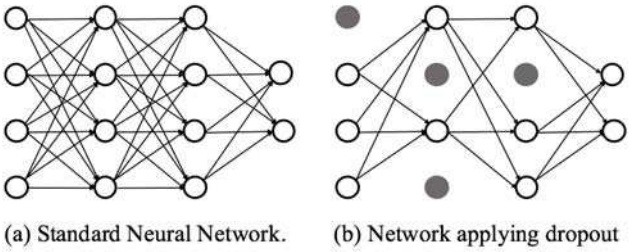


Figure 1: (a) A standard neural network with 2 hidden layers. (b) A sub-network produced by applying dropout. Units in grey color are dropped from the whole network.

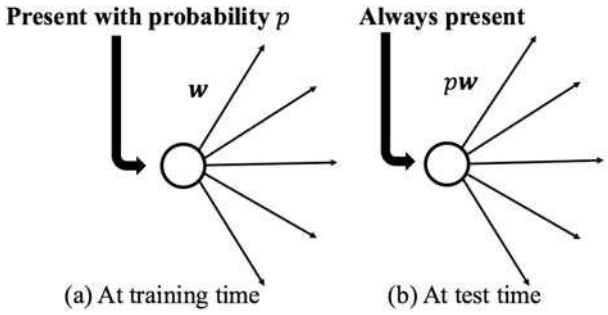


Figure 2: (a) At training time, a unit presents with probability p. (b) At test time, the unit is always present and the outgoing weights are multiplied by p.

2.3 Defense Methods

2.3.1 Adversarial Training [33]: injects adversarial examples with correct labels into the training dataset and then retrains the neural network, thus increasing robustness of DNNs under adversarial attacks.

2.3.2 Distillation as a Defense [29]: introduces *temperature T* into the softmax layer and uses a higher temperature for training and a lower temperature for testing. The training phase first trains a teacher model that can produce soft labels for the training dataset and then trains a distilled model using the training dataset with soft labels. The distilled model with reduced temperature will be preserved for testing. The modified softmax function utilized in the distilled model is given by:

$$\text{softmax}(x, T)_i = \frac{e^{Z(x)_i/T}}{\sum_j e^{Z(x)_j/T}} \tag{6}$$

where $Z(x)_i$ is the i-th logit corresponding to the last hidden layer output before softmax.

2.3.3 Stochastic Activation Pruning (SAP) [8]: SAP uses L_1 normalization to calculate the multinomial distribution regarding every activation layer. For every hidden activation vector $h^i \in \mathbb{R}^{a^i}$ at the i-th layer, SAP defines the probability p_j^i of whether or not to prune the j-th activation output h_j^i with the following,

$$p_j^i = \frac{|h_j^i|}{\sum_{k=1}^{a^i} |h_k^i|} \tag{7}$$

The remaining activation outputs are scaled up according to the pruning probability and the number of outputs kept at this layer by using the reweighting factor,

$$q_j^i = 1 - (1 - p_j^i)^{r_p^i} \tag{8}$$

where r_p^i is the number of outputs sampled.

3 PROPOSED DEFENSIVE DROPOUT

3.1 Dropout Preliminaries

Motivated by the mixability theory in the evolutionary biology [25], dropout was proposed as a regularization method in machine learning to prevent the overfitting issue with limited training data [30]. The term "dropout" refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, the unit along with all its incoming and outgoing connections are temporarily removed from the network. Fig. 1 shows applying dropout to a neural network amounts to sampling a sub-network from it.

The feedforward operation of a standard neural network can be described as:

$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)} \mathbf{y}^{(l)} + b_i^{(l+1)} \tag{9}$$

$$y_i^{(l+1)} = f(z_i^{(l+1)}) \tag{10}$$

where $\mathbf{y}^{(l)}$ denotes the vector of outputs from layer l, $\mathbf{z}^{(l)}$ denotes the vector of inputs to layer l, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are weight matrix and bias vector of layer l, and f is any activation function. With dropout, the feedforward operation becomes [16]:

$$r_j^{(l)} \sim \text{Bernoulli}(p) \tag{11}$$

$$\widetilde{\mathbf{y}}^{(l)} = \mathbf{r}^{(l)} \odot \mathbf{y}^{(l)} \tag{12}$$

$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)} \widetilde{\mathbf{y}}^{(l)} + b_i^{(l+1)} \tag{13}$$

$$y_i^{(l+1)} = f(z_i^{(l+1)}) \tag{14}$$

where $\mathbf{r}^{(l)}$ is a vector of independent Bernoulli random variables, each with probability p of being 1, and $\odot$ denotes an element-wise product.

A neural network with n units can been seen as a collection of 2^n sampled sub-networks, which all share weights so that the total number of parameters is still $O(n^2)$. During training a neural network with dropout, stochastic gradient descent is used as in standard training, except that for each training case in a mini-batch, a sub-network is sampled by dropping out units, and forward and backpropagation for that training case are done only on this sub-network. The gradients for each parameter are averaged over the training cases in each mini-batch. Therefore, training a neural network with n units using dropout can be seen as training a collection of 2^n sub-networks with extensive weight sharing.

The purpose of applying dropout is to prevent units from co-adapting too much by combining the predictions of many sub-networks with shared weights. However, at test time, it is not feasible to explicitly average the predictions from exponentially many sub-networks. A very simple approximate averaging method is to use a single neural network at test time without dropout, the weights of which are scaled-down versions of the trained weights. If a unit presents with probability p during training with dropout,

the outgoing weights of that unit are multiplied by p at test time, as shown in Fig. 2. This ensures that the output at test time is the same as the expected output at training time.

3.2 Defensive Dropout Implementations in Training and Test

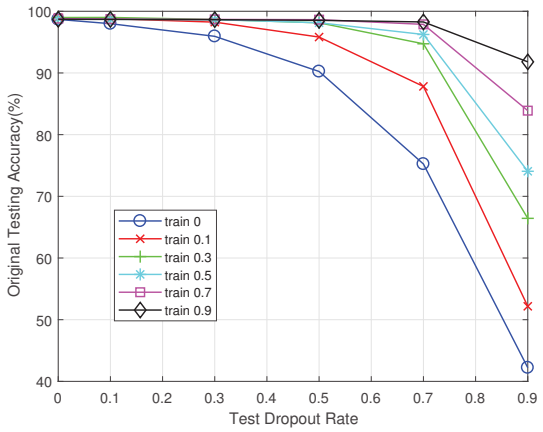
Dropout is a commonly used regularization method. To achieve very good test accuracy, in practice dropout is usually applied to units in the fully-connected layer close to the output layer of the neural network [12]. Also, the dropout rate $r = 1 - p$, instead of the probability p of presence for a unit is used during training with dropout [3]. For each training case in a mini-batch, the units are dropped with rate r and a sub-network is sampled for the training case. The gradient for each parameter (weight) is then calculated based on the sampled sub-network. Please note that, for a unit with rate r of being dropped, if it presents in the sub-network, we need to divide the output of its activation function by $1-r$ when evaluating the loss function for gradient calculation. This is for making the output at test time roughly the same as the expected output at training time. If a parameter is not used in the sub-network, a zero gradient is set for that. Gradients for each parameter are averaged over all training cases in the mini-batch. When dropout is applied as a regulation method to deal with overfitting, at test time the whole neural network without dropout is used, but with the output of the activation function divided by $1 - r$ if the unit is dropped with rate r during training.

Intuitively, introducing randomness into the test time can also help to harden deep neural networks against adversarial attacks. Therefore, we propose to apply dropout also **at the test time** as a defense method. If dropout was applied to units in a specific layer during training with dropout rate r, we are going to apply dropout to the same layer at test time with dropout rate r'. For each test case, units are dropped with rate r' and a sub-network is sampled for it. To have roughly the same expected output as the whole neural network at test time, we also need to scale-up the activation functions of the retained units in the dropout layer of the sub-network by $\frac{1}{1-r'}$. Please note that r is optimized during deep neural network training, and r' is the optimization variable of our defense method to be derived by the Algorithm in Section 3.4.

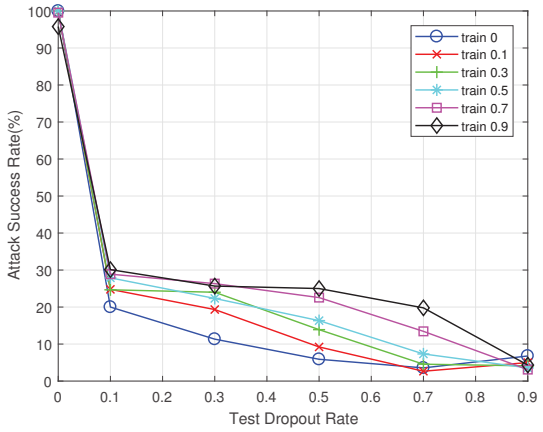
Our work aims at defending against the strongest attacks, i.e., the *white-box* attacks, that is, the attacker has perfect information about the neural network architecture and parameters. Therefore, when we defend against adversarial attacks, we assume the attacker knows not only the complete neural network model but also the stochastics in the model (i.e., which layer applied dropout and the dropout rates r and r'). Specifically, when the attacker generating adversarial examples by solving an optimization problem based on stochastic gradient descent, the gradients are calculated in the similar manner as training with dropout i.e., using sampled sub-networks and the activation functions scaled-up by $\frac{1}{1-r'}$. By doing this, we give the attacker full access to the neural network model and we are able to evaluate our defense method against the strongest white-box attacks.

3.3 Observations and Motivations

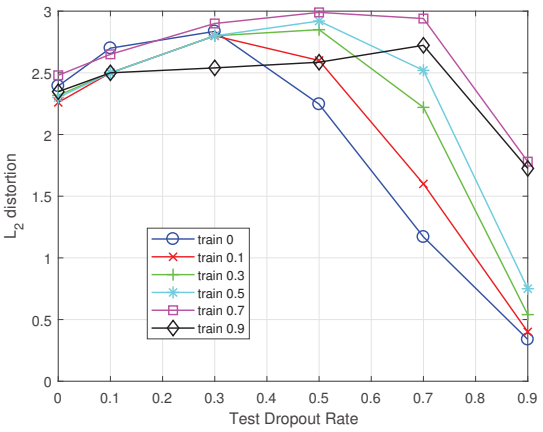
We perform some preliminary experiments that motivate and support our defensive dropout method. We pick the currently strongest



(a) Test Accuracy



(b) C&W Attack Success Rate



(c) C&W Attack L_2 Norm

Figure 3: (a) Test accuracy, (b) Attack success rate, and (c) L_2 norm under C&W attack on MNIST dataset using different training dropout rates and test dropout rates.

attack, i.e., C&W attack [5] and experiment on different training dropout rates and different test dropout rates to analyze test accuracy and defense effects. Fig. 3 presents the results, where the x-axes denote test dropout rate and each curve represents one training dropout rate. From Fig. 3 (a), we can observe that test accuracy decreases with increasing test dropout rate. Also, a training dropout rate of 0.3 achieves the highest test accuracy for MNIST dataset. The increase in test accuracy is more prominent in other datasets. For example, a training dropout rate of 0.7 increases the test accuracy by 7.5% in our neural network model on CIFAR-10 dataset. In summary, the decrease in test accuracy due to the test dropout rate can be compensated in some extent by the training dropout rate.

Fig. 3 (b) and (c) demonstrate the defense effects of training and test dropout rates. In general, increasing test dropout rate can reduce the attack success rate, see Fig. 3 (b). And the L_2 norm of the added distortions in the adversarial examples reaches a peak at certain test dropout rate, see Fig. 3 (c). The solution for defending against C&W attack on MNIST dataset is to use a test dropout rate of 0.5 when the training dropout rate is 0.3, which loses 0.8% test accuracy but decreases the attack success rate from 100% to 13.89% with the largest L_2 norm of the distortion (the peak point in Fig. 3 (c)), indicating that the added distortion in the adversarial examples might be large enough to be recognized by humans. Please note that under C&W attack, the distillation as a defense [29] and the adversarial training [32] could not decrease the attack success rate at all [5], i.e., still 100% attack success rate under these defense methods.

We also investigate the mechanism behind the outstanding defense effects achieved by the proposed defensive dropout. It is intuitive to explain the defense effects as model randomness through adding dropout at test time. However, there are also other defenses through introducing model randomness, such as stochastic activation pruning (SAP) [8] and migration through randomization (MTR) [34], that only achieve limited defense effects against C&W. We are trying to explain the mechanism in a different way. Fig. 4 is plotted for the probability density of uniformly sampled gradients when generating adversarial example using C&W attack on CIFAR-10 dataset. Please note that the gradients have $32 \times 32 \times 3$ dimensions, and therefore we select 5 dimensions for visualization, each presented by a color and each containing 50 data points throughout Fig. 4 (a)~(f). Also for fair comparison, we use the same neural network model with training dropout rate of 0.7 for the best test accuracy, and the same original input image for Fig. 4 (a)~(f), including our defensive dropout and SAP.

From Fig. 4 (a)~(e), with increasing test dropout rate, the probability densities become shorter and fatter, demonstrating increasing variances of the gradients, which is the key for the improved defense effects (decreasing attack success rate) with increasing test dropout rate. The larger variances of the gradients, the more difficult for the attacker to generate effective adversarial examples by using stochastic gradient descent when solving the optimization problem. It cross-validates the conclusion from Fig. 3 (a) and (b). Of course, we could not use the largest test dropout rate for the strongest defense effects, because the test accuracy might be very low. We need to trade-off defense effects for the test accuracy. Fig. 4 (f) is the probability densities of the gradients from stochastic activation pruning (SAP) [8], which shows very small variances of the gradients comparing with our defense method. That is the reason our defensive dropout outperforms SAP.

3.4 Defensive Dropout Algorithm

Towards hardening deep neural networks under adversarial attacks, the attacker and the defender improve their own strategies like in a two-player game. In such a game, the attacker and the defender know each others' strategies and try to optimize their own strategies towards an equilibrium. The defender can benefit from the improvement of the attacker's strategy. Therefore, we need to take into consideration the attacker's strategy of generating adversarial examples when designing our defense.

Based on the observations on the test dropout rate's effect on test accuracy and attack success rate, we design the defensive dropout algorithm that helps us to determine an optimal test dropout rate given the neural network model and the attacker's strategy for generating adversarial examples. In the algorithm, we also optimize the training dropout rate along with the test dropout rate, but that is only for the purpose of training neural network for the best test accuracy. Basically, we first train a neural network model F by finding a proper training dropout rate r. Then we fix the model F and search for the largest test dropout rate r' for the strongest defense effects while satisfying the test accuracy requirement. Pseudo code of the defensive dropout algorithm is given in Algorithm 1.

Algorithm 1 Defensive Dropout Algorithm

Require: $\mathbf{X}$: Dataset
F: Neural Network Model
π: Attacker Strategy (e.g., C&W, FGSM, JSMA)
ε: Maximum decrease of test accuracy
Ensure: r': Test Dropout Rate
r: Training Dropout Rate

```
Retrain neural network model F using a proper training dropout rate r ∈ [0.3, 0.7] and obtain the best test accuracy a_test^max;
a ← a_test^max;
r' ← 0;
while a > a_test^max − ε do
    r' ← r' + step_size;
    Generate adversarial examples using attacker strategy π, neural network model F, and test dropout rate r';
    Evaluate test accuracy a using neural network model F and test dropout rate r';
    Evaluate attack success rate using neural network model F and test dropout rate r';
end while
```

4 EXPERIMENTAL RESULTS

4.1 Setup

As in other attack and defense work, we are also using the two datasets: *MNIST* and *CIFAR-10*. The *MNIST* dataset (Modified National Institute of Standards and Technology database) [35] is a collection of handwritten digits that is commonly used for training and test various machine learning tasks. It consists of $70,000$ 28×28 ($60,000$ training images and $10,000$ test images) grey-scale

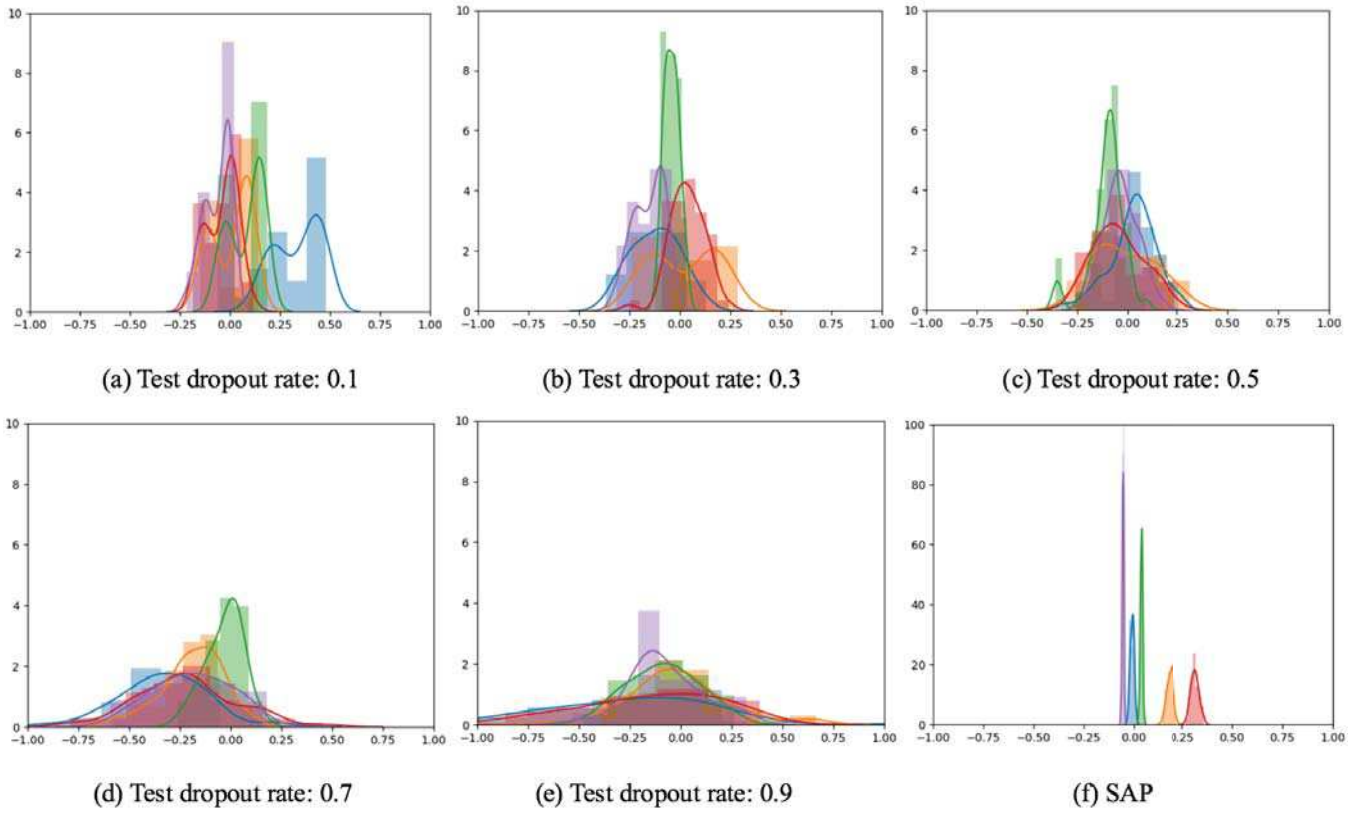


Figure 4: Probability density of sampled gradients when generating adversarial example using C&W attack on CIFAR-10. The same neural network architecture, the same original input image, and the same training dropout rate of 0.7 for the best test accuracy is used throughout (a)~(f). Histogram and the corresponding fitted probability density curve in each color denote one out of $32 \times 32 \times 3$ dimensions in the sampled gradients and include 50 data points. (a)~(e) are from our proposed defensive dropout for different test dropout rate, and (f) is from stochastic activation pruning (SAP).

images for digits 0 to 9. The *CIFAR-10* dataset (Canadian Institute For Advanced Research) [19] is a collection of color images. It contains $60,000$ 32×32 images in 10 different classes (e.g., cars, birds, airplanes, etc.).

For DNN models in our experiment, we use standard convolutional neural networks with 4 convolutional layers and 2 fully-connected layers. This architecture has been used as standard model in many previous work [5, 29]. While the overall neural network architecture for MNIST or CIFAR-10 dataset is the same, the size of the neural network model for CIFAR-10 is slightly larger than that for MNIST, since CIFAR-10 images have higher resolution. The activation function is rectified linear unit (ReLU) for all convolutional and fully connected layers. The architectures of the neural network models for MNIST and CIFAR-10 are summarized in Table 1. In both models we apply defensive dropout to *Fully connected layer 1*. After training, the models achieve the state-of-the-art 99.4% and 80% test accuracies for *MNIST* dataset and *CIFAR-10* dataset, respectively.

We implement FGSM, JSMA, and C&W attacks based on the CleverHans package [27]. For FGSM, we use a fixed $\epsilon = 0.25$ as suggested in the original paper [13]. For JSMA, we use the codes from CleverHans directly. For C&W, we perform 10 iterations of binary search for constant c. With a selected c, we then run 100 iterations of gradient decent with the Adam optimizer. We compare with other defenses such as adversarial training [33], distillation as a defense [29], and stochastic activation pruning (SAP) [8], among which we implement SAP ourselves, and defense effects of the adversarial training and distillation as a defense are cited from [5].

Table 1: Architectures of neural network models for MNIST and CIFAR-10

	Model for MNIST	Model for CIFAR-10
Conv layer	32 filters with size (3,3)	64 filters with size (3,3)
Conv layer	32 filters with size (3,3)	64 filters with size (3,3)
Pooling layer	pool size (2,2)	pool size (2,2)
Conv layer	64 filters with size (3,3)	128 filters with size (3,3)
Conv layer	64 filters with size (3,3)	128 filters with size (3,3)
Pooling layer	pool size (2,2)	pool size (2,2)
Fully connected 1	200 units	256 units
Fully connected 2	200 units	256 units
Output layer	10 units	10 units

Table 2: Test accuracy, attack success rate, and L_2 norm using SAP against C&W attack on CIFAR-10.

	train 0	train 0.1	train 0.3	train 0.5	train 0.7	train 0.9
Test acc.	72.07%	75.69%	76.39%	78.12%	78.15%	68.35%
C&W ASR	54.44%	64.44%	77.78%	78.89%	85.56%	70%
L_2 norm	0.504	0.522	0.618	0.498	0.679	0.784

Table 3: Attack success rate using our defensive dropout against FGSM attack on CIFAR-10.

Dropout rate	test 0	test 0.1	test 0.3	test 0.5	test 0.7	test 0.9
train 0	32.48%	–	–	–	–	–
train 0.5	15.87%	14.46%	13.89%	–	–	–

Table 4: Attack success rate using our defensive dropout against FGSM attack on MNIST.

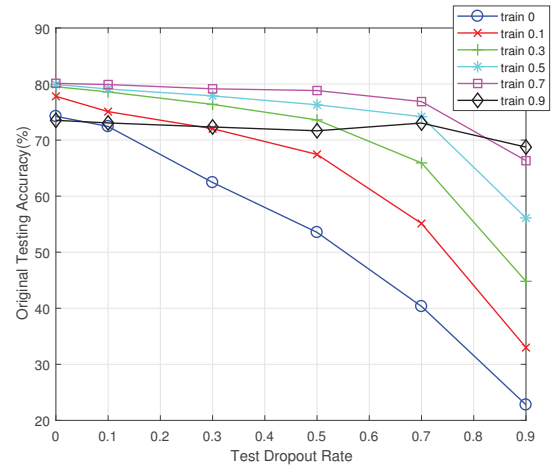
Dropout rate	test 0	test 0.1	test 0.3	test 0.5	test 0.7	test 0.9
train 0.7	22.74%	21.89%	20.67%	19.56%	16.44%	–

4.2 Results

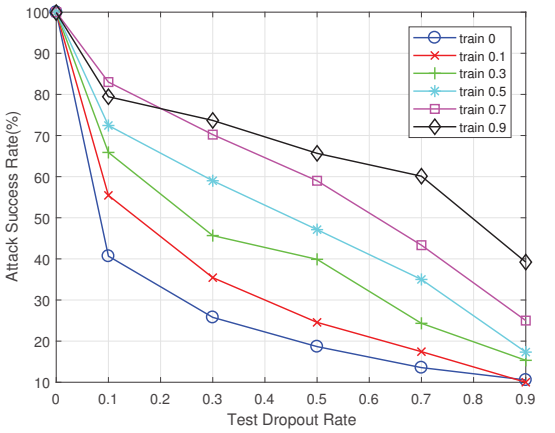
We use two metrics to evaluate the defense effects against adversarial attacks, i.e., attack success rate (ASR) and L_2 norm of the distortion. The lower attack sucssess rate and the higher L_2 norm imply the stronger defense effects.

We first compare with SAP on the defense effects against C&W attack using CIFAR-10 dataset. The results of SAP are summarized in Table 2. The results of our defensive dropout are summarized through Fig. 5. In Table 2, we perform SAP on neural network models using different training dropout rates (in columns). The second to forth rows report test accuracy, attack success rate (ASR), and L_2 norm. If we allow test accuracy decrease within 4%, the SAP can reduce the attack success rate from 100% to 77.78% with a test accuracy of 76.39%. From Fig. 5, we can observe that at training dropout rate of 0.7 and test dropout rate of 0.7, our defensive dropout can reduce the attack success rate to 43.33% with a test accuracy of 77%, demonstrating superior defense effects to SAP. Also the L_2 norm of our defensive dropout is around 1.1 which is much higher than that of SAP (i.e., 0.618 from Table 2). Table 3 shows the attack success rate using our defensive dropout against FGSM attack on CIFAR-10, observing that the defensive dropout reduces attack success rate from 32.48% to 13.89% at test accuracy of 77.89%. In general, attack success rate of FGSM is much lower than C&W, because it is a faster, but not optimal attack.

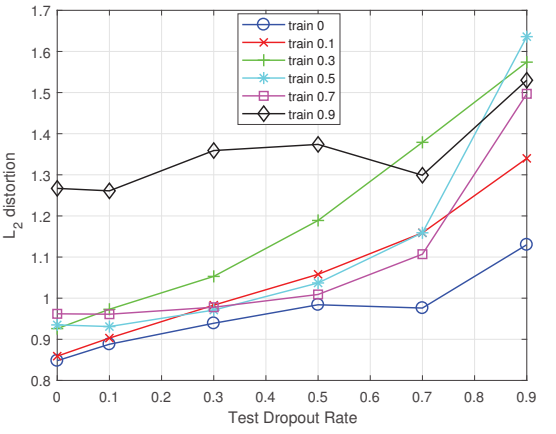
We also summarize the results on MNIST dataset using our defensive dropout against FGSM, JSMA, and C&W attacks, respectively, in Tables 4, 5, and 6, with <1% test accurary drop. For FGSM, defensive dropout reduces attack success rate from 40.67% to 16.44%. For JSMA, defensive dropout reduces attack success rate from 91.89% to 26.78%. For C&W, defensive dropout reduces attack success rate from 100% to 13.89%. Please note that adversarial training and distillation as a defense are totally vulnerable to C&W attack [5].



(a) Test Accuracy



(b) C&W Attack Success Rate



(c) C&W Attack L_2 Norm

Figure 5: (a) Test accuracy, (b) Attack success rate, and (c) L_2 norm under C&W attack on CIFAR-10 dataset using different training dropout rates and test dropout rates.

Table 5: Attack success rate using our defensive dropout against JSMA attack on MNIST.

Dropout rate	test 0	test 0.1	test 0.3	test 0.5	test 0.7	test 0.9
train 0.7	90.67%	60.56%	43.78%	35.67%	26.78%	–

Table 6: Attack success rate using our defensive dropout against C&W attack on MNIST.

Dropout rate	test 0	test 0.1	test 0.3	test 0.5	test 0.7	test 0.9
train 0.3	100%	24.66%	24.00%	13.89%	–	–

5 CONCLUSION

In this paper, we propose defensive dropout for hardening deep neural networks under adversarial attacks. Considering the problem of building robust DNNs as an attacker-defender two-player game, we provide a defensive dropout algorithm that determines an optimal test dropout rate given the neural network model and the attacker's strategy for generating adversarial examples. We also explain the mechanism behind the outstanding defense effects achieved by the proposed defensive dropout.

ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation (CCF-1733701, CNS-1618379, DMS-1737897, and CNS-1840813), Air Force Research Laboratory FA8750-18-2-0058, Naval Research Laboratory. We thank researchers at the US Naval Research Laboratory for their comments on previous drafts of this paper.

REFERENCES

[1] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420* (2018).
[2] Arjun Nitin Bhagoji, Daniel Cullina, and Prateek Mittal. 2017. Dimensionality reduction as a defense against evasion attacks on machine learning classifiers. *arXiv preprint arXiv:1704.02654* (2017).
[3] Xavier Bouthillier, Kishore Konda, Pascal Vincent, and Roland Memisevic. 2015. Dropout as data augmentation. *arXiv preprint arXiv:1506.08700* (2015).
[4] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. 2016. Hidden Voice Commands.. In *USENIX Security Symposium*. 513–530.
[5] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 39–57.
[6] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. 2017. EAD: elastic-net attacks to deep neural networks via adversarial examples. *arXiv preprint arXiv:1709.04114* (2017).
[7] Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*. ACM, 160–167.
[8] Guneet S. Dhillon, Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, and Animashree Anandkumar. 2018. Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations*. https://openreview.net/forum?id=H1uR4GZRZ
[9] Guneet S Dhillon, Kamyar Azizzadenesheli, Zachary C Lipton, Jeremy Bernstein, Jean Kossaifi, Aran Khanna, and Anima Anandkumar. 2018. Stochastic Activation Pruning for Robust Adversarial Defense. *arXiv preprint arXiv:1803.01442* (2018).
[10] Gintare Karolina Dziugaite, Zoubin Ghahramani, and Daniel M Roy. 2016. A study of the effect of jpg compression on adversarial images. *arXiv preprint arXiv:1608.00853* (2016).
[11] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. 2017. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410* (2017).
[12] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.
[13] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).
[14] Chuan Guo, Mayank Rana, Moustapha Cissé, and Laurens van der Maaten. 2017. Countering Adversarial Images using Input Transformations. *arXiv preprint arXiv:1711.00117* (2017).
[15] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.
[16] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).
[17] Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3128–3137.
[18] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *2015 ICLR* arXiv preprint arXiv:1412.6980 (2015). arXiv:1412.6980 http://arxiv.org/abs/1412.6980
[19] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).
[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
[21] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).
[22] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551.
[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[24] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. 2017. Fault injection attack on deep neural network. In *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 131–138.
[25] Adi Livnat, Christos Papadimitriou, Nicholas Pippenger, and Marcus W Feldman. 2010. Sex, mixability, and modularity. *Proceedings of the National Academy of Sciences* 107, 4 (2010), 1452–1457.
[26] Anh Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 427–436.
[27] Nicolas Papernot, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Fartash Faghri, Alexander Matyasko, Karen Hambardzumyan, Yi-Lin Juang, Alexey Kurakin, Ryan Sheatsley, et al. 2016. cleverhans v2. 0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768* (2016).
[28] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 372–387.
[29] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 582–597.
[30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.
[31] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).
[32] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Dan Boneh, and Patrick McDaniel. 2017. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204* (2017).
[33] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. 2018. Ensemble Adversarial Training: Attacks and Defenses. *2018 ICLR* arXiv preprint arXiv:1705.07204 (2018).
[34] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. 2017. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991* (2017).
[35] LeCun Yann, Cortes Corinna, and JB Christopher. 1998. The MNIST database of handwritten digits. *URL http://yhann. lecun. com/exdb/mnist* (1998).