DROPATTACK: A MASKED WEIGHT ADVERSARIAL TRAINING METHOD TO IMPROVE GENERALIZATION OF NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Adversarial training has been proven to be a powerful regularization method to improve generalization of models. In this work, a novel masked weight adversarial training method, DropAttack, is proposed for improving generalization potential of neural network models. It enhances the coverage and diversity of adversarial attack by intentionally adding worst-case adversarial perturbations to both the input and hidden layers and randomly masking the attack perturbations on a certain proportion weight parameters. It then improves the generalization of neural networks by minimizing the internal adversarial risk generated by exponentially different attack combinations. Further, the method is a general technique that can be adopted to a wide variety of neural networks with different architectures. To validate the effectiveness of the proposed method, five public datasets were used in the fields of natural language processing (NLP) and computer vision (CV) for experimental evaluating. This study compared DropAttack with other adversarial training methods and regularization methods. It was found that the proposed method achieves state-of-the-art performance on all datasets. In addition, the experimental results of this study show that DropAttack method can achieve similar performance when it uses only a half training data required in standard training. Theoretical analysis revealed that DropAttack can perform gradient regularization at random on some of the input and weight parameters of the model. Further, visualization experiments of this study show that DropAttack can push the minimum risk of the neural network model to a lower and flatter loss landscapes. [1]

1 INTRODUCTION

Deep neural networks (DNNs) (LeCun et al., 2015) have achieved state-of-the-art performance in several artificial intelligence applications such as natural language processing and computer vision. However, one of the rebarbative characteristics of deep learning models is the ease to overfitting, which leads to low generalization potential of the models. Regularization methods, such as L1 (Tibshirani, 1996) and L2 (Tikhonov, 1943) as well as early stopping (Morgan & Bourlard, 1989) and Dropout method (Srivastava et al., 2014), plays an important role for the impressive performance of deep neural networks by controlling complexity of the models. The methods prevents overfitting and hence improve generalization potential of models. Adversarial training (Goodfellow et al., 2015) was originally proposed as a method of improving security of machine learning systems and to train a neural network that is robust to attack samples. Specifically, adversarial training process minimizes the maximal risk for label-preserving input perturbations of a model. Goodfellow et al. (2015) have previously demonstrated that adversarial training can result in even further regularization than dropout method. Therefore, adversarial training improves not only robustness to adversarial examples, but also generalization performance of original examples.

Most of the recent adversarial training (Goodfellow et al., 2015; Miyato et al., 2017; Athalye et al., 2018; Zhang et al., 2019; Zhu et al., 2020) only perturb the input layer, which to a large extent restricts attack range and weakens attack intensity. Moreover, the existing adversarial training methods add perturbation to every element in the input tensor during the attack and hence such an attack

[1] The code of DropAttack has been made public.

lacks diversity. It should be noted that the input in the NLP field is the embeddings of the text whereas in the CV field the input is the value of each pixel of the picture. However, in this paper, it is uniformly called the input for the convenience of expression.

In this work, focus was mainly on improving the generalization performance of models and preventing model from overfitting, rather than enhancing the robustness of models to attack samples. Therefore, a novel adversarial training method called DropAttack was proposed to enhance the diversity and scope of attacks. DropAttack expands the attack target from the input to the weight parameters of other hidden layers, thereby increasing the scope of attack. To make combinations of attacks diversification, DropAttack method randomly masks the attack on a certain proportion of weight parameters instead of attacking all the weight parameters in each attack iteration. Therefore, exponentially different attack combinations can be obtained while using random mask attack.

First, the impact of DropAttack was evaluated on the generalization performance of the model followed by other well-known regularization methods. Visual analysis of the training and verification accuracy of some models of different architectures were later conducted as the training progresses. Further, we also study the efficiency of DropAttack under training datasets of different sizes. Finally, a theoretical analysis was also conducted from another perspective to prove the effectiveness of DropAttack.

The main contributions of this paper are summarized as follows:

- In this work, a novel random masked weight adversarial training method, DropAttack, is proposed to improve the generalization of neural networks.
- To the best of our knowledge, DropAttack is the first to use random masking of some perturbations to increase the diversity of adversarial attack combinations.
- Experimental results on five public datasets show that DropAttack achieves extremely strong performances. And DropAttack can improve the generalization of neural networks with different structures.
- We theoretically show that our DropAttack can perform gradient regularization at random on some of the input and weight parameters of the model and push the minimum risk of the neural network model to a lower and flatter loss landscapes.

2 Related work

Adversarial training can be traced back to Goodfellow et al. (2015), in which robustness and generalization of the model was improved by generating adversarial examples and injecting them into training data. It is necessary to find the perturbation value that maximizes the adversarial loss because the effectiveness of adversarial training largely depends on the direction of the attack. Further, neural networks are easily attacked by linear perturbation due to their linear characteristics. Therefore, Goodfellow et al. (2015) proposed Fast Gradient Sign Method (FGSM) to calculate the perturbation of the input sample. They hence linearized the cost function around the current value of parameters, obtaining an optimal max-norm constrained pertubation of:

$$\boldsymbol{r_{adv}} = \epsilon \cdot \operatorname{sgn}(\nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)) \tag{1}$$

where $\boldsymbol{\theta}$ is the model parameter, $\boldsymbol{x}$ is the input of the model, y is the label corresponding to the input, L is the cost used to train the neural network, sgn is the symbolic function and ϵ is the perturbation coefficient. To find a better perturbation, Miyato et al. (2017) proposed the Fast Gradient Method (FGM), which made a simple modification to the calculation of perturbation in FGSM. In addition, FGM is the first time that adversarial training has been applied to text classification tasks. The formula is as follows:

$$\boldsymbol{r_{adv}} = \epsilon \cdot \boldsymbol{g}/\|\boldsymbol{g}\|_2 \text{ where } \boldsymbol{g} = \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y). \tag{2}$$

A adversarial training method called Projected Gradient Descent (PGD) was proposed by Athalye et al. (2018). The method obtains the final perturbation value through multiple forward and backward propagation iterations. The perturbation obtained in each iteration is limited to a set range and if it exceeds this range will be mapped to the "sphere" of the range. To put it simply, "walk in small steps, take a few more steps." The formula is as follows:

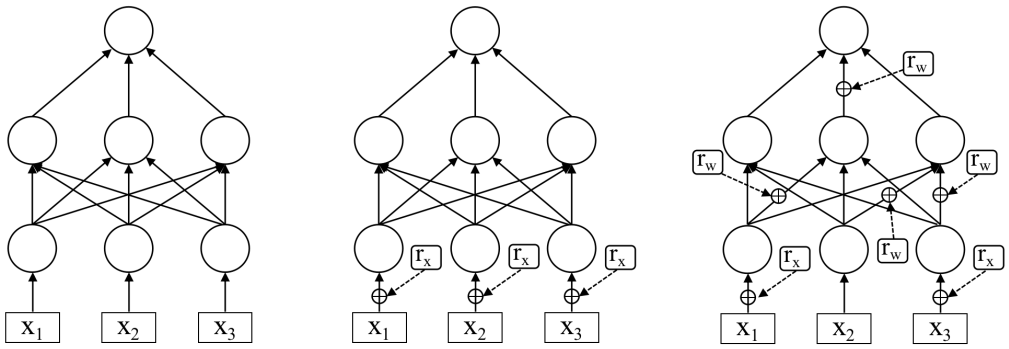
$$x_{t-1} = \prod_{x+S} (x_t + \alpha \cdot \boldsymbol{g}(\boldsymbol{x_t})/\|\boldsymbol{g}(\boldsymbol{x_t})\|_2) \text{ where } \boldsymbol{g}(\boldsymbol{x_t}) = \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x_t}, y). \tag{3}$$

where $S = r \in \mathbb{R}^d; \|\boldsymbol{r}\|_2 \leq \epsilon$ is the constraint space of the perturbation and a represent the step length of the "small step".

Although many defenses were broken by Athalye et al. (2018), PGD-based adversarial training is one of the few methods that can withstand powerful attacks. According to Athalye et al. (2018) PGD can largely avoid the problem of gradient confusion and will still cause high convolution and non-linear loss surface. When K is very small, it will be lightly broken under powerful adversaries. To obtain a more efficient PGD-based adversarial training, it must iteratively calculate the gradient many times, but this will consume a lot of computing resources. A "free" adversarial training algorithm that eliminates the overhead cost of calculating adversarial perturbations by recycling the gradient information computed when updating model parameters was proposed byShafahi et al. (2019). Furthermore, the total number of full forward and backward propagations is effectively reduced in Zhang et al. (2019) through restriction of most forward and back propagation within the first layer of the network during adversary updates. Virtual adversarial training is proposed by Miyato et al. (2019) as a regularization method for semi-supervised learning in the text field. In addition, Zhu et al. (2020) propose FreeLB for improving the generalization of language models, which performs multiple PGD iterations to attack the embeddings and simultaneously accumulates the "free" parameter gradients in each iteration.

In this work, we are the first to propose an adversarial training method that simultaneously attacks the input of the model and the weight parameters of other layers to improve the generalization of the model. In addition, to the best our knowledge the proposed method, DropAttack, is the first to use random masking of some elements to increase the diversity of adversarial attack combinations.

3 The Proposed DropAttack Adversarial Training Method



(a) Standard Neural Net (b) Standard adversarial training (c) Model with DropAttack

Figure 1: (a) A standard neural network with 3 input and 12 weight parameters. (b) An new neural network produced by applying standard adversarial Training on the left, which attacks all inputs. (c) An new neural network produced by applying DropAttack to the network on the left. Assume that each input vector and weight parameter have a 2/3 and 1/3 probability of being attacked, respectively.

Standard adversarial training is aimed to explore the optimal parameters to minimize the maximum risk of adversarial attacks. The Min-Max formula is as follows:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(x,y)\sim D}[\max_{r_{adv}\in S} L(\boldsymbol{\theta}, \boldsymbol{x} + r_{adv}, y)] \tag{4}$$

where D is the data distribution, y is the label, and L is loss function. r_{adv} represent the perturbation under maximizing internal risk and S is the perturbation constraint space. In the current study a new adversarial training method, DropAttack is proposed. It simultaneously attacks the input x of the model and the weight parameters of other layers and randomly masks some attacks. The overall procedure is shown in Algorithm A and the Min-Max formula of DropAttack can be expressed as:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(x,y)\sim D}[\max_{r_x\in S} L(\boldsymbol{\theta}, \boldsymbol{x} + \boldsymbol{M_x} \cdot \boldsymbol{r_x}, y) + \max_{r_\theta \in S} L(\boldsymbol{\theta} + \boldsymbol{M_\theta} \cdot \boldsymbol{r_\theta}, \boldsymbol{x}, y)] \tag{5}$$

Algorithm 1: DropAttack Adversarial Training

Input: Training samples $\mathcal{X}$, model parameter $\boldsymbol{\theta}$, perturbation coefficients ϵ_x and ϵ_θ, Attack probabilities p_x and p_θ , Learning rate τ

for epoch $= 1 \ldots N_{ep}$ **do**
  for $(\boldsymbol{x}, y) \in \mathcal{X}$ **do**
    Compute gradient $\boldsymbol{g}$ of parameter $\boldsymbol{x}$ and $\boldsymbol{\theta}$:
      $\boldsymbol{g_x} \leftarrow \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)$; $\boldsymbol{g_\theta} \leftarrow \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)$
    Compute perturbation $\boldsymbol{r_x}$ and $\boldsymbol{r_\theta}$:
      $\boldsymbol{r_x} \leftarrow \epsilon_x \cdot \boldsymbol{g_x} / \|\boldsymbol{g_x}\|_2$; $\boldsymbol{r_\theta} \leftarrow \epsilon_\theta \cdot \boldsymbol{g_\theta} / \|\boldsymbol{g_\theta}\|_2$
    Random attack $\boldsymbol{M_x}$ and $\boldsymbol{M_\theta}$ mask:
      $\boldsymbol{M_{x_{ij}}} \sim \text{Bernoulli}(p_x)$; $\boldsymbol{M_{\theta_{ij}}} \sim \text{Bernoulli}(p_\theta)$
    Compute adversarial gradient $\boldsymbol{g}_{adv}$:
      $\boldsymbol{g}_{adv} \leftarrow \nabla_{\boldsymbol{\theta}} [L(\boldsymbol{\theta}, \boldsymbol{x} + \boldsymbol{M_x} \cdot \boldsymbol{r_x}, y) + L(\boldsymbol{\theta} + \boldsymbol{M_\theta} \cdot \boldsymbol{r_\theta}, \boldsymbol{x}, y)]$
    Update parameter $\boldsymbol{\theta}$:
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \tau(\boldsymbol{g} + \boldsymbol{g}_{adv})$
  end
end

Output: $\boldsymbol{\theta}$

where $\boldsymbol{r_x}$ and $\boldsymbol{r_\theta}$ are the perturbation of the input $\boldsymbol{x}$ and the parameter $\boldsymbol{\theta}$ under maximizing the internal risk. These values were approximated by linearizing $\nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x_t}, y)$ and $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x_t}, y)$ around $\boldsymbol{x}$ and $\boldsymbol{\theta}$ respectively. The resulting adversarial perturbation when using the linear approximation in equation (6) and the L2 norm constraint is:

$$\boldsymbol{r_x} \leftarrow \epsilon_x \cdot \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y) / \|\nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)\|_2; \ \boldsymbol{r_\theta} \leftarrow \epsilon_\theta \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y) / \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)\|_2 \quad (6)$$

These perturbations $\boldsymbol{r_x}$ and $\boldsymbol{r_\theta}$ can be easily calculated using backpropagation in a neural network.

The random attack masks of $\boldsymbol{r_x}$ and $\boldsymbol{r_\theta}$ are $\boldsymbol{M_x}$ and $\boldsymbol{M_\theta}$ respectively. For any attack mask, $\boldsymbol{M_{ij}}(\boldsymbol{M_{x_{ij}}} \sim \text{Bernoulli}(p_x); \boldsymbol{M_{\theta_{ij}}} \sim \text{Bernoulli}(p_\theta))$ is a matrix of independent Bernoulli random variables with the same dimension as the perturbation value and the probability of each Bernoulli random variable is 1. Multiplying the perturbation matrix and the attack mask matrix randomly masks a part of the element values in the perturbation matrix. The attack on the input in the text model is essentially an attack on the weight parameters of the embedding layer. It is evident that Figure 1 (a) is a standard neural network with 12 weight parameters. Figure 1 (b) is a new neural network produced by applying standard adversarial Training on the left, which attacks all inputs. Further, Figure 1 (c) is a new neural network after DropAttack is applied and some of the weights are added with the perturbation calculated by the corresponding gradient. It is noted that DropAttack maximizes internal risk through a wider range of attacks (not limited to the input layer) compared with standard adversarial training. In addition, it also randomly mask some dimensional perturbation, which will generate a more robust and diversified embedding space.

3.1 DropAttack with Multiple Internal Ascent Steps

Most of the latest adversarial training are PGD-based methods. The PGD-based methods are a series of adversarial training algorithms (Kurakin et al., 2017) for solving the maximum-minimum problem of cross-entropy loss. This can be reliably achieved by using multiple projection gradient ascent steps and then performing an SGD (Stochastic Gradient Descent) (Bottou, 2012) step. Multiple PGD iterations can get a more optimized perturbation. Generally, DropAttack can also use multiple forward and backward propagation methods to update the perturbation. In practice, when DropAttack is used, each ascent step of updating perturbation optimizes for a different weight network. This will affect the inability to obtain the optimal perturbation because $\boldsymbol{r_t}$ and $\boldsymbol{r_{t-1}}$ are an iterative relationship. Therefore, the same mask matrix is used for each step of perturbation update. The overall procedure is shown in Algorithm B. Specifically, the initial gradients $\boldsymbol{g_x^{(0)}} = \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)$ and $\boldsymbol{g_\theta^{(0)}} = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)$ of $\boldsymbol{x}$ and $\boldsymbol{\theta}$ and initial perturbation $\boldsymbol{r_x^{(0)}} = \epsilon_x \cdot \boldsymbol{g_x^{(0)}} / \|\boldsymbol{g_x^{(0)}}\|_2$ and $\boldsymbol{r_\theta^{(0)}} = \epsilon_\theta \cdot \boldsymbol{g_\theta^{(0)}} / \|\boldsymbol{g_\theta^{(0)}}\|_2$ are first calculated, then random attack masks $\boldsymbol{M_x}$ and $\boldsymbol{M_\theta}$ are generated and used in the forward and backward propagation for each step of perturbation update. Therefore, the mask matrix is fixed after the first step of perturbation update.

Algorithm 2: PGD-based DropAttack-K Adversarial Training

Input: Training samples $\mathcal{X}$, model parameter $\boldsymbol{\theta}$, perturbation coefficient ϵ_x and ϵ_θ, Attack probability p_x and p_θ, number of forward-backward propagation K, Learning rate τ

for epoch $= 1 \dots N_{ep}$ **do**
  for $(\boldsymbol{x}, y) \in \mathcal{X}$ **do**
    for $\boldsymbol{t} = 1, 2 \dots K$ **do**
      Compute initial gradient $\boldsymbol{g}$ of parameter $\boldsymbol{x}$ and $\boldsymbol{\theta}$:
        $\boldsymbol{g_x^{(0)}} \leftarrow \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)$; $\boldsymbol{g_\theta^{(0)}} \leftarrow \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)$
      Compute initial perturbation $\boldsymbol{r_x^{(0)}}$ and $\boldsymbol{r_\theta^{(0)}}$:
        $\boldsymbol{r_x^{(0)}} \leftarrow \epsilon_x \cdot \boldsymbol{g_x^{(0)}} / \|\boldsymbol{g_x^{(0)}}\|_2$; $\boldsymbol{r_\theta^{(0)}} \leftarrow \epsilon_\theta \cdot \boldsymbol{g_\theta^{(0)}} / \|\boldsymbol{g_\theta^{(0)}}\|_2$
      if $\boldsymbol{t} = 1$ **then**
        Generate random attack masks $\boldsymbol{M_x}$ and $\boldsymbol{M_\theta}$:
          $\boldsymbol{M_{x_{ij}}} \sim \text{Bernoulli}(p_x)$; $\boldsymbol{M_{\theta_{ij}}} \sim \text{Bernoulli}(p_\theta)$
      end
      Update the perturbation $\boldsymbol{r}$ via gradient ascend :
        $\boldsymbol{g_x^{(t)}} \leftarrow \boldsymbol{g_x^{(t-1)}} + \frac{1}{K} \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x} + \boldsymbol{M_x} \cdot \boldsymbol{r_x^{(t-1)}}, y)$
        $\boldsymbol{g_\theta^{(t)}} \leftarrow \boldsymbol{g_\theta^{(t-1)}} + \frac{1}{K} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x} + \boldsymbol{M_\theta} \cdot \boldsymbol{r_\theta^{(t-1)}}, y)$
        $\boldsymbol{r_x^{(t)}} \leftarrow \epsilon_x \cdot \boldsymbol{g_x^{(t)}} / \|\boldsymbol{g_x^{(t)}}\|_2$; $\boldsymbol{r_\theta^{(t)}} \leftarrow \epsilon_\theta \cdot \boldsymbol{g_\theta^{(t)}} / \|\boldsymbol{g_\theta^{(t)}}\|_2$
    end
    Update parameter $\boldsymbol{\theta}$:
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \tau(\boldsymbol{g_x^{(K)}} + \boldsymbol{g_\theta^{(K)}})$
  end
end

Output: $\boldsymbol{\theta}$

The perturbation values $\boldsymbol{r_x^{(t)}} = \epsilon_x \cdot \boldsymbol{g_x^{(t)}} / \|\boldsymbol{g_x^{(t)}}\|_2$ and $\boldsymbol{r_\theta^{(t)}} = \epsilon_\theta \cdot \boldsymbol{g_\theta^{(t)}} / \|\boldsymbol{g_\theta^{(t)}}\|_2$ are updated through the gradient ascend in each iteration, where $\boldsymbol{g_x^{(t)}} = \boldsymbol{g_x^{(t-1)}} + \frac{1}{K} \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x} + \boldsymbol{M_x} \cdot \boldsymbol{r_x^{(t-1)}}, y)$ and $\boldsymbol{g_\theta^{(t)}} = \boldsymbol{g_\theta^{(t-1)}} + \frac{1}{K} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x} + \boldsymbol{M_\theta} \cdot \boldsymbol{r_\theta^{(t-1)}}, y)$. Finally, the model parameter $\boldsymbol{\theta}$ is updated once with the accumulated gradient of each adversarial iteration.The DropAttack-K of K iterations can be expressed as:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(x,y)\sim D} \{ \frac{1}{K} \sum_{t=0}^{K-1} [\max_{r_x^{(t)} \in S} L(\boldsymbol{\theta}, \boldsymbol{x} + \boldsymbol{M_x} \cdot \boldsymbol{r_x^{(t)}}, y) + \max_{r_\theta^{(t)} \in S} L(\boldsymbol{\theta} + \boldsymbol{M_\theta} \cdot \boldsymbol{r_\theta^{(t)}}, \boldsymbol{x}, y)] \}. \quad (7)$$

The training process is equivalent to replacing the original batch with a K-times virtual batch which consist of samples whose embeddings are $\boldsymbol{x} + \boldsymbol{M_x} \cdot \boldsymbol{r_x^{(0)}}, \boldsymbol{x} + \boldsymbol{M_x} \cdot \boldsymbol{r_x^{(1)}}, \dots, \boldsymbol{x} + \boldsymbol{M_x} \cdot \boldsymbol{r_x^{(k-1)}}$. Similarly, multiple virtual neural networks with different weight parameters will be trained and their weight parameters are $\boldsymbol{\theta} + \boldsymbol{M_\theta} \cdot \boldsymbol{r_\theta^{(0)}}, \boldsymbol{\theta} + \boldsymbol{M_\theta} \cdot \boldsymbol{r_\theta^{(1)}}, \dots, \boldsymbol{\theta} + \boldsymbol{M_\theta} \cdot \boldsymbol{r_\theta^{(k-1)}}$ respectively.

It is worth noting that our perturbation constraint S does not take an additional fixed value and only uses the L2 norm to constrain the gradient value. This is because the diversity of perturbations is needed at each step, rather than forcibly constraining it in a fixed spherical space. Furthermore, the proposed method, DropAttack-K also inherits the "free" ability, using the gradient average calculated by each backpropagation for external minimization.

Intuitively, DropAttack can generate a richer adversarial sample in the spherical space of the original sample compared to the previous adversarial training method, which can prevent the model from overfitting on the adversarial sample to a certain extent. Empirically, there is a gap between the features of the training dataset and the features of the test dataset in the high-dimensional feature space. Improving generalization is essentially to narrow the feature distribution gap between the training dataset and the test dataset. However, more diverse adversarial samples are needed to fill this gap because it is uncertain. In theory, DropAttack has a more significant improvement in the generalization of the model.

4 EXPERIMENT

4.1 DATASETS

The DropAttack algorithm is evaluated using five public datasets, IMDB (Maas et al., 2011), PHEME (Zubiaga et al., 2016), AGnews (Zhang et al., 2015), MNIST (LeCun, 1998) and GIFAR-10 (Krizhevsky, 2009). The brief description of the datasets are as shown in Table 1. The IMDB is a standard benchmark movie review dataset for sentiment analysis. The PHEME dataset contains a collection of Twitter rumours and non-rumours posted during breaking news. The AGnews topic classification dataset is constructed by Zhang et al. (2015) from the original AG news sources. The MNIST is a standard and the commonly used toy dataset of handwritten digits. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. Each dataset is divided into training set, validation set and test set.

Table 1: Overview of the datasets used in this paper.

Dataset	Task	Classes	Training	Validation	Test
IMDB	Sentiment analysis	2	40,000	5,000	5,000
PHEME	Rumor detection	2	5,145	643	637
AGNEWS	News classification	4	110,000	10,000	7,600
MNIST	Image classification	10	50,000	10,000	10,000
CIFAR-10	Image classification	10	40,000	10,000	10,000

4.2 EXPERIMENTAL SETUP

In the present experiment, the rnn-based and cnn-based models were chosen to handle nlp tasks and cv tasks, respectively. For nlp tasks, IMDB uses an LSTM (Hochreiter & Schmidhuber, 1997) (300-300 dim) layer and a fully connected layer (300-2 dim); PHEME uses a BiLSTM (Schuster & Paliwal, 1997) (300-300 dim) layer and a fully connected layer (600-2 dim) whereas AGnews uses two BiGRU (Chung et al., 2014) (300-300 dim) layers and a fully connected layer (600-4 dim). In addition, for cv tasks, MNIST uses the LeNet-5 (LeCun et al., 1998) model, which contains two layers of CNN (1-6-16 channels, kernel size = 5) and three fully connected layers (400-120-84-10 dim); CIFAR-10 uses the VGGNet-16 (Simonyan & Zisserman, 2014) model, which contains 13 layers of CNN (3-64-64-128-128-256-256 -256-512-512-512-512 channels, kernel size = 3) and three fully connected layers (512-4096-4096-10 dim). All the models are implemented based on Pytorch, the Batch sizes value is 128, the optimizer is Adam and the learning rate is 0.001.

4.3 EXPERIMENTAL RESULTS AND DISCUSSION

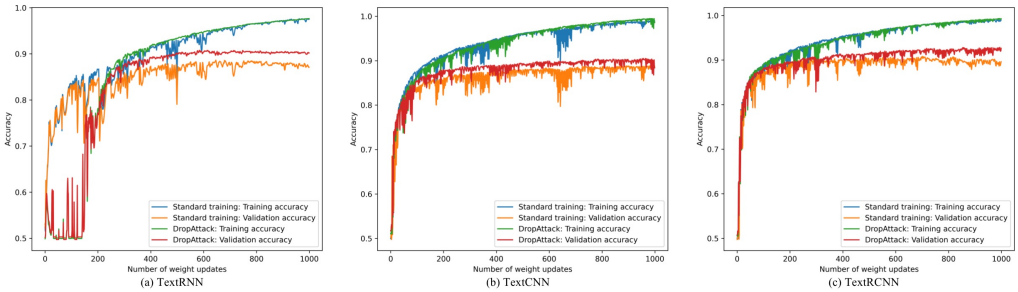
Table 2 show that the performance of the model after using DropAttack has improved on the five datasets compared to the original model without DropAttack. The improvements observed on the three nlp datasets are 2.24%, 3.07% and 1.5% respectively. It is seen that the overall effect of adversarial training is better compared with other regularization methods. Among them, the efficiency of FGSM is relatively unstable, whereas the performance on IMDB and CIFAR-10 is only 88.04% and 71.86% respectively. Because the naive perturbation value may destroy the distribution of the original data. It should be noted that PGD, FreeAT and FreeLB are all PGD-based adversarial training methods, which require multiple forward and back propagation iterations to calculate the optimal perturbation value. In this experiment, the number of forward-backward passes k is 3.

Our method achieves state-of-the-art performance on five datasets by calculating the perturbation value through only one backpropagation. In this study the influence of the number of forward-backward propagation on DropAttack was evaluated later. It is seen that the performance of DropAttack-(I-H) is better than DropAttack-(I) on all five datasets, which proves that it is effective to attack the parameters of the hidden layer of the model. In addition, it is evident that based on the results in Table 2, the current method has more improvements on the NLP datasets and less on the CV datasets. This finding is consistent with the experimental results in other previous adversarial training research studies (Cheng et al., 2019; Zhao et al., 2018; Zhu et al., 2020). It is believed that

Table 2: Comparing the effects of DropAttack with various well-known regularization and other state-of-the-art adversarial training methods. The reported results are calculated from 5 runs with the same hyper-parameters except for the random seeds. The "Baseline" used in the five datasets are LSTM, BiLSTM, BiGRU, LeNet and VGGNet, which are common neural-based deep learning models. The best method and the best competitor are highlighted by **bold** and underline, respectively. Additional experimental details and results are provided in the Appendix A.

Methods	NLP Datasets			CV Datasets	
	IMDB	PHEME$^{\circledast}$	AGnews	MNIST	CIFAR10
Baseline	88.12	84.08/78.99	91.87	98.95	84.67
Baseline + L1 (Tibshirani, 1996)	88.02	85.34/79.55	92.29	99.07	84.74
Baseline + L2 (Tikhonov, 1943)	88.27	85.67/81.29	92.43	99.14	84.63
Baseline + Dropout (Srivastava et al., 2014)	88.64	85.85/81.08	92.22	99.07	85.39
Baseline + FGSM (Goodfellow et al., 2015)	88.04	85.61/80.40	92.54	99.16	71.86
Baseline + FGM Miyato et al. (2017)	89.26	84.97/78.52	92.53	99.15	85.64
Baseline + PGD Athalye et al. (2018)	89.38	85.28/79.30	92.76	99.10	85.57
Baseline + FreeAT Shafahi et al. (2019)	89.17	85.29/79.32	92.45	99.09	85.45
Baseline + FreeLB Zhu et al. (2020)	89.25	85.69/81.18	92.58	99.11	85.47
Baseline + DropAttack-(I)†	89.76	85.75/81.33	93.35	99.16	86.05
Baseline + DropAttack-(I-H)‡	**90.36**	**87.15/81.31**	**93.37**	**99.27**	**86.09**

$^{\circledast}$ Note that although PHEME is a two-class classification, the labels are not balanced, so we use accuracy and f1-score (accuracy/f1) as the evaluation criteria. † Only attack the input layer; ‡ Attack the input layer and hidden layer simultaneously.



(a) TextRNN (b) TextCNN (c) TextRCNN

Figure 2: Training and validation accuracy of different models (TextRNN, TextCNN and TextRCNN) with and without DropAttack on IMDB dataset.

the reason for this phenomenon is that the perturbation is directly added to the original pixel value for the picture and the text is not directly modified to the word rather, the perturbation value is added to the embedding vector. Further, the pixel value of the image is fixed and the perturbation may change the distribution of the original sample. However, the word vector of the text is not unique and certain, hence it is more likely to learn a better word vector after adding perturbation.

To show the effect of DropAttack in preventing neural network overfitting more clearly, classification experiments were carried out with many different models of keeping all hyperparameters, including ϵ and p fixed. As shown in Figure 2, the training and validation accuracy obtained for these models of different architectures (TextRNN, TextCNN and TextRCNN) as training progresses. The training accuracy under DropAttack training is basically the same as that under standard training. However, the validation accuracy is higher, which proves that using the DropAttack adversarial training method can alleviate model overfitting. Furthermore, it can be seen that DropAttack adversarial training may be more difficult to converge in the early stage, because it is more difficult to optimize the target under attack. Nevertheless, the validation accuracy of the model is relatively more stable after enough weight updates and learning. Therefore, DropAttack gives an obvious improvement across all neural networks of different architectures, without using hyperparameters that were tuned specifically for each architecture.

Table 3: The ability of our method DropAttack to prevent overfitting under different sizes of training data. The hyperparameters are set to $k = 3, \epsilon_x = \epsilon_\theta = 5, p_x = p_\theta = 0.7$.

Methods	Size of the training set							
	100	500	1,000	2,500	5,000	10,000	20,000	40,000
Standard Training	63.26	74.26	78.30	81.14	82.62	84.92	85.42	88.12
DropAttack-3 Training	65.46	76.34	80.70	83.88	85.22	87.02	88.86	90.42
Improvement ↑	2.20	2.08	2.40	2.66	2.60	2.10	3.42	2.30

Table 4: Comparing the performance of DropAttack under different number of forward-backward propagation K. The reported results are calculated from 5 runs with the same hyper-parameters.

Methods	IMDB	PHEME	AGNEWS	MNIST	CIFAR-10
DropAttack-1	90.36	87.15/82.31	93.37	99.27	86.09
DropAttack-2	90.38	87.27/82.43	93.38	99.24	86.09
DropAttack-3	90.42	87.36/82.78	93.34	99.27	86.07
DropAttack-4	90.43	87.25/82.63	93.41	99.26	86.10
DropAttack-5	90.42	87.26/82.67	93.39	99.25	86.09

In addition, the efficiency of DropAttack under training datasets of different sizes is also studied. The IMDB training set is divided into different sizes and use the LSTM model with the same structure as the above and the experimental results are shown in Table 3. Compared with standard training, the performance of the model using the DropAttack training method on all seven training datasets of different sizes has been improved by more than 2%. Furthermore, it is found that DropAttack can achieve and even exceed the accuracy of standard training using only half of the training data. For example, DropAttack can achieve 83.88% using 2500 training data and the accuracy of using 5000 data based on standard training is 82.62%.

For the PGD-based DropAttack-k the influence of the number of forward-backward propagation on DropAttack is studied and the experimental results are as shown in Table 4. It is found that multiple iterative calculations can indeed further improve the generalization of the neural network, because multiple iterations are more likely to get the optimal disturbance value. However, it is evident that more forward-backward propagation will greatly increase the training time. Therefore, a reasonable number of iterations K can be selected based on time and computing resources. Additional hyperparameter sensitivity analysis are provided in the Appendix B.

5 Theoretical Analysis

We provide another theoretical perspectives to explain why the adversarial training method DropAttack can be used as regularization to improve the generalization of the model and prevent overfitting. According to Section 3, the task of DropAttack can be to minimize the maximum internal adversarial risk, that is, to approximately optimize the following goals:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{(x,y)\sim D}[\max_{r_x \in S} L(\boldsymbol{\theta}, \boldsymbol{x} + \boldsymbol{M_x} \cdot \boldsymbol{r_x}, y) + \max_{r_\theta \in S} L(\boldsymbol{\theta} + \boldsymbol{M_\theta} \cdot \boldsymbol{r_\theta}, \boldsymbol{x}, y)] \tag{8}$$

For formula 8, we second-order Taylor expand functions $f(x) = L(\boldsymbol{\theta}, \boldsymbol{x} + \boldsymbol{M_x} \cdot \boldsymbol{r_x}, y)$ and $f(\theta) = L(\boldsymbol{\theta} + \boldsymbol{M_\theta} \cdot \boldsymbol{r_\theta}, \boldsymbol{x}, y)$ at points $(\boldsymbol{x} + \boldsymbol{M_x} \cdot \boldsymbol{r_x})$ and $(\boldsymbol{\theta} + \boldsymbol{M_\theta} \cdot \boldsymbol{r_\theta})$ respectively:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \mathbb{E}_{(x,y)\sim D}\{ &\max_{r_x \in S}[L(\boldsymbol{\theta}, \boldsymbol{x}, y) + < \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y), \boldsymbol{M_x} \cdot \boldsymbol{r_x} >] \\ &+ \max_{r_\theta \in S}[L(\boldsymbol{\theta}, \boldsymbol{x}, y) + < \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y), \boldsymbol{M_\theta} \cdot \boldsymbol{r_\theta} >]\} \end{aligned} \tag{9}$$

Then, after substituting the values $\boldsymbol{r_x} = \epsilon_x \cdot \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y) / \|\nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)\|_2$, $\boldsymbol{r_\theta} = \epsilon_\theta \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y) / \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)\|_2$ of the perturbation that maximize the antagonistic Loss, the following formula 10 is obtained:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \mathbb{E}_{(x,y)\sim D}[&L(\boldsymbol{\theta}, \boldsymbol{x}, y) + < \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y), \boldsymbol{M_x} \cdot \epsilon_x \cdot \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y) / \|\nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)\|_2 > \\ &+ L(\boldsymbol{\theta}, \boldsymbol{x}, y) + < \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y), \boldsymbol{M_\theta} \cdot \epsilon_\theta \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y) / \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)\|_2 >] \end{aligned} \tag{10}$$

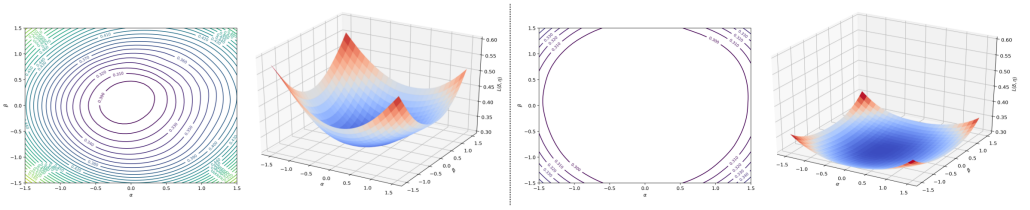


Figure 3: 2D and 3D visualization of the minima of the empirical risk generated by standard training (left) and DropAttack (right) on IMDB dataset.

$$\Rightarrow \min_{\boldsymbol{\theta}} \mathbb{E}_{(x,y)\sim D}[L(\boldsymbol{\theta}, \boldsymbol{x}, y) + \epsilon_x \cdot \boldsymbol{M_x} < \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y), \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)/\|\nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)\|_2 > + L(\boldsymbol{\theta}, \boldsymbol{x}, y) + \epsilon_\theta \cdot \boldsymbol{M_\theta} < \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y), \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)/\|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)\|_2 >] \tag{11}$$

$$\Rightarrow \min_{\boldsymbol{\theta}} \mathbb{E}_{(x,y)\sim D}[2L(\boldsymbol{\theta}, \boldsymbol{x}, y) + \epsilon_x \cdot \|\boldsymbol{M_x} \cdot \nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)\|_2 + \epsilon_\theta \cdot \|\boldsymbol{M_\theta} \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{x}, y)\|_2] \tag{12}$$

$$\Rightarrow \min_{\boldsymbol{\theta}} \mathbb{E}_{(x,y)\sim D}[2\text{Loss} + \epsilon_x \cdot \|\boldsymbol{M_x} \cdot \boldsymbol{g_x}\|_2 + \epsilon_\theta \cdot \|\boldsymbol{M_\theta} \cdot \boldsymbol{g_\theta}\|_2] \tag{13}$$

We can see the final optimization function formula 13, which actually adds the implicit gradient regularization $\epsilon_x \cdot \|\boldsymbol{M_x} \cdot \boldsymbol{g_x}\|_2$ and $\epsilon_\theta \cdot \|\boldsymbol{M_\theta} \cdot \boldsymbol{g_\theta}\|_2$ to a certain proportion of input $\boldsymbol{x}$ and parameters $\boldsymbol{\theta}$ after loss every time the parameter $\boldsymbol{\theta}$ is updated. Gradient penalty pushes the gradient of some parameters and inputs to approach zero, so that the model is likely to be optimized to a flatter minimum.

In order to further visually analyze the effectiveness of the proposed method, we draw the high-dimensional non-convex loss function with a visualization method proposed by (Li et al., 2018). We visualize the loss landscapes around the minima of the empirical risk generated by standard training or DropAttack, the 2D and 3D visualization are plotted in Figure 3. Additional loss visualization are provided in the Appendix C. Define two direction vectors, α and β with the same dimensions as θ, drawn from a Gaussian distribution with zero mean and a scale of the same order of magnitude as the variance of layer weights. Then we choose a center point θ^* and add a linear combination of α and β to obtain a loss that is a function of the contribution of the two random direction vectors. And we define a grid of points to evaluate the loss on i.e. range of values for δ and η for which $L(\delta, \eta)$ is evaluated and stored.

$$L(\delta, \eta) = \mathcal{L}(\theta^* + \delta\alpha + \delta\beta) \tag{14}$$

The results show that the test loss $L(\delta, \eta)$becomes lower and flatter during the training with DropAttack. And DropAttack indeed selects flatter loss landscapes via masked adversarial perturbations. Many studies have shown that a flatter loss landscape usually means better generalization (Hochreiter & Schmidhuber, 1997; Keskar et al., 2019; Ishida et al., 2020).

6 Conclusion

In this work, a masked weight adversarial training method, DropAttack is proposed to improve the generalization ability of neural network models and prevent overfitting. The proposed algorithm uses the gradient-based method to attack the weight parameters of input layer and hidden layer according to a certain probability and enhances the generalization of the model by minimizing the resultant adversarial risk. Experimental results prove that DropAttack can effectively improve the generalization of models and prevent overfitting, especially in the field of NLP. In addition, it was theoretically proven that the algorithm of the current work can regularize the gradient of model weight parameters. Finally, the visual experiments indicate that the model with DropAttack drift into an area with a flat loss landscape. Therefore, DropAttack can improve the robustness and generalization of neural network and prevent overfitting. Further, the current adversarial training still consumes more computing resources and time than the standard stochastic gradient descent. This is thus a valuable research direction to accelerate the adversarial training while improving generalization in the future.

REFERENCES

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pp. 274–283. PMLR, 2018.

Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pp. 421–436. Springer, 2012.

Yong Cheng, Lu Jiang, and Wolfgang Macherey. Robust neural machine translation with doubly adversarial inputs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4324–4333, 2019.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *ICLR*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Takashi Ishida, Ikko Yamane, Tomoya Sakai, Gang Niu, and Masashi Sugiyama. Do we need zero training loss after achieving zero training error? In *International Conference on Machine Learning*, pp. 4604–4614. PMLR, 2020.

Nitish Shirish Keskar, Jorge Nocedal, Ping Tak Peter Tang, Dheevatsa Mudigere, and Mikhail Smelyanskiy. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017*, 2019.

A Krizhevsky. Learning multiple layers of features from tiny images. *Master's thesis, University of Tront*, 2009.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *ICLR*, 2017.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 6391–6401, 2018.

Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150, 2011.

Takeru Miyato, Andrew M Dai, and Ian Goodfellow. Adversarial training methods for semi-supervised text classification. *ICLR*, 2017.

Takeru Miyato, Shin-Ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2019.

Nelson Morgan and Hervé Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. *Advances in neural information processing systems*, 2:630–637, 1989.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 3358–3369, 2019.

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

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.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Andrey Nikolayevich Tikhonov. On the stability of inverse problems. In *Dokl. Akad. Nauk SSSR*, volume 39, pp. 195–198, 1943.

Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. *Advances in Neural Information Processing Systems*, 32:227–238, 2019.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28:649–657, 2015.

Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. In *International Conference on Learning Representations*, 2018.

Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. Freelb: Enhanced adversarial training for natural language understanding. In *ICLR*, 2020.

Arkaitz Zubiaga, Maria Liakata, and Rob Procter. Learning reporting dynamics during breaking news for rumour detection in social media. *arXiv preprint arXiv:1610.07363*, 2016.

A Additional Experimental Details and Results

To prove the effectiveness of DropAttack, several experiments we conducted on five datasets: IMDB, PHEME, AGnews, MNIST and CIFAR-10. The detailed experimental settings and results are shown in Table 5, Table 6, Table 7, Table 8 and Table 9. The square brackets after DropAttack indicate the object of perturbation. For example, DropAttack[Embedding, Lstm.ih.w] means adding perturbation to Embedding and Lstm.ih.w parameters.

For an important discussion and research question: In addition to the perturbation of the input layer, which layer's weight parameters are better to perturb? According to our experimental results and experience, for the perturbation of hidden layer parameters, the effect of being close to the input layer will be better than that of being close to the output layer. Essentially, the perturbation of the hidden layer is the perturbation of the higher-dimensional embedding of the input. Due to the highly linearization of neural networks, small changes in the input vector may result in changes in the output of multiple layers. Therefore, the deeper weight parameters need to be robust enough to resist overfitting and prevent fitting to overly sensitive input features.

Table 5: Experimental details on the IMDB dataset.

Method	Accuracy (%)
LSTM	88.12
LSTM + DropAttack[Embedding] (e = 5, p = 0.5)	89.76
LSTM + DropAttack[Embedding, Fc.w] (e = 5, p = 0.5)	88.60
LSTM + DropAttack[Lstm.hh.w] (e = 5, p = 0.5)	86.64
LSTM + DropAttack[Lstm.ih.w] (e = 5, p = 0.5)	89.80
LSTM + DropAttack[Embedding, Lstm.hh.w] (e = 5, p = 0.5)	87.90
LSTM + DropAttack[Embedding, Lstm.ih.w] (e = 5, p = 0.5)	90.21
LSTM + DropAttack[Embedding, Lstm.ih.w] (e = 3, p = 0.7)	90.22
LSTM + DropAttack[Embedding, Lstm.ih.w] (e = 5, p = 0.7)	90.34
LSTM + DropAttack[Embedding, Lstm.ih.w] (e = 7, p = 0.7)	**90.36**
LSTM + DropAttack[Embedding, Lstm.ih.w] (e = 9, p = 0.7)	90.23
LSTM + DropAttack[Embedding, Lstm.ih.w] (e = 7, p = 0.9)	90.12
LSTM + DropAttack[Embedding, Lstm.hh.w, Lstm.ih.w] (e = 5, p = 0.5)	89.56
LSTM + DropAttack[Embedding, Lstm.hh.w, Lstm.ih.w] (e = 5, p = 0.6)	89.57
LSTM + DropAttack[Embedding, Lstm.hh.w, Lstm.ih.w] (e = 5, p = 07)	89.66

Table 6: Experimental details on the PHEME dataset.

Method	Accuracy/F1 (%)
BiLSTM	84.08/78.99
BiLSTM + DropAttack[Embedding] (e = 5, p = 0.5)	85.69/79.97
BiLSTM + DropAttack[Lstm.hh.w] (e = 5, p = 0.5)	86.00/80.03
BiLSTM + DropAttack[Lstm.ih.w] (e = 5, p = 0.5)	83.40/78.64
BiLSTM + DropAttack[Fc.w] (e = 5, p = 0.5)	84.60/79.32
BiLSTM + DropAttack[Embedding, Lstm.hh.w] (e = 5, p = 0.5)	87.14/81.02
BiLSTM + DropAttack[Embedding, Lstm.hh.w] (e = 5, p = 0.6)	87.14/81.04
BiLSTM + DropAttack[Embedding, Lstm.ih.w] (e = 5, p = 0.5)	86.74/80.13
BiLSTM + DropAttack[Embedding, Lstm.hh.w] (e = 5, p = 0.7)	**87.15/81.31**
BiLSTM + DropAttack[Embedding, Lstm.hh.w] (e = 7, p = 0.5)	86.85/80.19
BiLSTM + DropAttack[Embedding, Lstm.hh.w] (e = 7, p = 0.7)	86.95/80.27
BiLSTM + DropAttack[Embedding, Lstm.hh.w] (e = 5, p = 0.8)	87.11/81.24
BiLSTM + DropAttack[Embedding, Lstm.hh.w, Lstm.ih.w] (e = 5, p = 0.5)	85.54/79.57
BiLSTM + DropAttack[Embedding, Lstm.hh.w, Lstm.ih.w] (e = 5, p = 0.7)	85.35/79.36

Table 7: Experimental details on the AGnews dataset.

Method	Accuracy (%)
BiGRU	91.87
BiGRU+ DropAttack[Embedding] (e = 5, p = 0.5)	93.35
BiGRU+ DropAttack[Embedding] (e = 5, p = 0.7)	93.34
BiGRU + DropAttack[Gru.hh.w] (e = 5, p = 0.5)	92.25
BiGRU + DropAttack[Gru.ih.w] (e = 5, p = 0.5)	92.46
BiGRU + DropAttack[Gru.hh.w, Gru.ih.w] (e = 5, p = 0.5)	92.70
BiGRU + DropAttack[Fc.w] (e = 5, p = 0.5)	92.24
BiGRU + DropAttack[Embedding, Gru.ih.w] (e = 5, p = 0.5)	93.12
BiGRU + DropAttack[Embedding, Gru.ih.w] (e = 5, p = 0.6)	93.21
BiGRU + DropAttack[Embedding, Gru.ih.w] (e = 5, p = 0.7)	**93.37**
BiGRU + DropAttack[Embedding, Gru.hh.w] (e = 5, p = 0.9)	92.70
BiGRU + DropAttack[Embedding, Gru.hh.w, Gru.ih.w] (e = 5, p = 0.5)	93.12
BiGRU + DropAttack[Embedding, Gru.ih.w, Gru.ih.w.reverse] (e = 5, p = 0.5)	92.88
BiGRU + DropAttack[Embedding, Gru.ih.w, Gru.ih.w.reverse] (e = 5, p = 0.7)	92.94

Table 8: Experimental details on the MNIST dataset.

Method	Accuracy (%)
LeNet-5	98.95
LeNet-5 + DropAttack[Input] (e = 5, p = 0.5)	99.16
LeNet-5 + DropAttack[Conv.1.w] (e = 5, p = 0.5)	99.08
LeNet-5 + DropAttack[Input, Conv.1.w] (e = 5, p = 0.5)	**99.27**
LeNet-5 + DropAttack[Input, Conv.1.w] (e = 5, p = 0.7)	99.25
LeNet-5 + DropAttack[Input, Conv.2.w] (e = 5, p = 0.5)	99.12
LeNet-5 + DropAttack[Input, Conv.1.b] (e = 5, p = 0.5)	99.10
LeNet-5 + DropAttack[Conv.2.w] (e = 5, p = 0.5)	99.11
LeNet-5 + DropAttack[Conv.1.b, Conv.2.w] (e = 5, p = 0.5)	99.09
LeNet-5 + DropAttack[Conv.1.w, Conv.2.w] (e = 5, p = 0.5)	98.78
LeNet-5 + DropAttack[Conv.1.w, Fc.1] (e = 5, p = 0.5)	98.93
LeNet-5 + DropAttack[Conv.2.w, Fc.1] (e = 5, p = 0.5)	99.10
LeNet-5 + DropAttack[Conv.2.w, Fc.2] (e = 5, p = 0.5)	99.05
LeNet-5 + DropAttack[Conv1.w, Conv2.w, fc1.w2] (e = 5, p = 0.5)	98.48

Table 9: Experimental details on the CIFAR-10 dataset.

Method	Accuracy (%)
VGGNet-16	84.67
VGGNet-16 + DropAttack[Input] (e = 5, p = 0.5)	86.02
VGGNet-16 + DropAttack[Conv.1.w] (e = 5, p = 0.5)	83.61
VGGNet-16 + DropAttack[Input, Conv.1.w] (e = 1, p = 0.5)	86.01
VGGNet-16 + DropAttack[Input, Conv.1.w] (e = 5, p = 0.5)	**86.09**
VGGNet-16 + DropAttack[Input, Conv.1.w] (e = 5, p = 0.7)	86.02
VGGNet-16 + DropAttack[Input, Conv.3.w] (e = 5, p = 0.7)	85.13
VGGNet-16 + DropAttack[Input, Conv.5.w] (e = 5, p = 0.7)	85.13
VGGNet-16 + DropAttack[BatchNorm.1.w] (e = 5, p = 0.5)	85.51
VGGNet-16 + DropAttack[Conv.2.w] (e = 5, p = 0.5)	83.16
VGGNet-16 + DropAttack[Conv.6.w] (e = 5, p = 0.5)	85.27
VGGNet-16 + DropAttack[BatchNorm.8.w] (e = 5, p = 0.5)	85.32
VGGNet-16 + DropAttack[Conv.1.w, BatchNorm.1.w] (e = 5, p = 0.5)	84.41
VGGNet-16 + DropAttack[Input, Conv.1.w, BatchNorm.1.w] (e = 5, p = 0.5)	85.01

B HYPERPARAMETER SENSITIVITY ANALYSIS

DropAttack has three tunable hyperparameters; perturbation coefficient ϵ, attack probability p (the probability of attacking a weight param in the network) and number of forward-backward propagation K. The effect of varying these hyperparameters is explored. First, the value of K was fixed at 1 and let ϵ to take up the values of 0.001, 0.1, 1, 3, 5, 7 and 9 in turn whereas p take the values of 0, 0.1, 0.3, 0.5, 0.7, 0.9, 1 in turn, so that there is a total of 7 x 7 = 49 kinds of hyperparameter combinations and the experimental results are as shown in Table 10. The best performance is 90.36% when = ϵ = 7 and p = 0.7. When p = 0, it means that the model is the standard training without any attack and its accuracy is the lowest. Further it was found that the effect is significantly lowest when p is less than 0.3 or greater than 0.9. When p =1, random masking is not used, although the performance is improved but not much, because the attack combination is relatively single. From these experimental results, it is thought that the hyperparameter p is best between 0.5 and 0.7.

Table 10: The influence of hyperparameters perturbation coefficient ϵ and attack probability p on model performance. The tested dataset is IMDB, and the model structure is the same as that in Table 2. The top ten performance is emphasized in **bold**.

		Perturbation coefficient †						
		$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 1$	$\epsilon = 3$	$\epsilon = 5$	$\epsilon = 7$	$\epsilon = 9$
Attack probability‡	P = 0.0	88.12	88.12	88.12	88.12	88.12	88.12	88.12
	P = 0.1	89.60	89.78	89.40	88.74	89.30	88.88	89.26
	P = 0.3	89.98	90.12	90.02	90.10	90.04	**90.28**	89.74
	P = 0.5	90.13	90.16	90.01	**90.25**	**90.21**	**90.30**	89.94
	P = 0.7	90.17	**90.32**	**90.18**	**90.20**	**90.18**	**90.36**	90.14
	P = 0.9	89.76	**90.22**	90.16	89.22	90.12	90.04	90.18
	P = 1.0	89.86	89.54	89.74	89.40	90.02	89.90	90.10

† Note that ϵ_x and ϵ_θ are uniformly denoted by ϵ; ‡ p_x and p_θ are uniformly denoted by p.

As shown in Figure 4, the impact of different attack probabilities is studied on model performance under different perturbation coefficients. It is observed that the attack probability ranges from 0 to 0.7 and the performance of the model increases with increase in the attack probability, because the intensity of the model attack is getting stronger. However, it was found that the attack probability increased from 0.7 to 1, and the performance decreased instead. Therefore, because an excessively high attack probability will reduce the diversity of attack combinations when p = 0, it is approximately a standard adversarial attack.

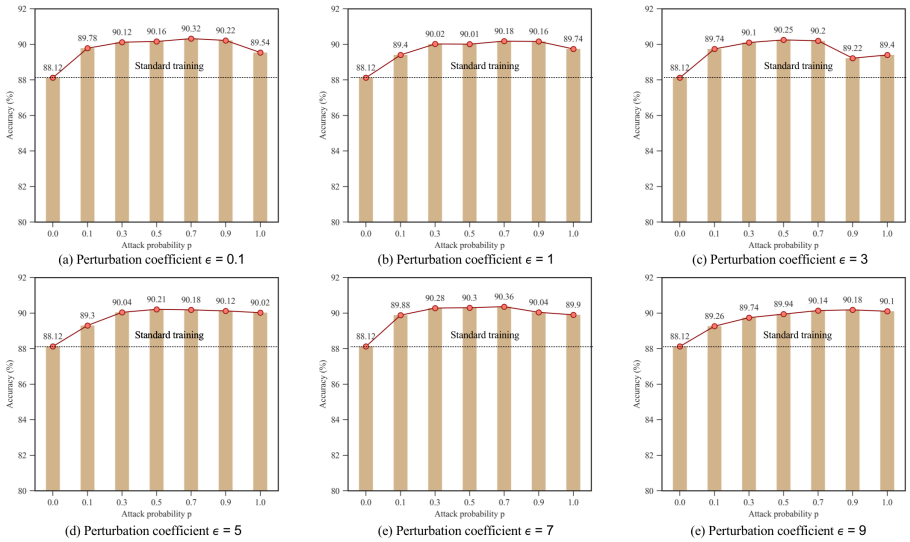
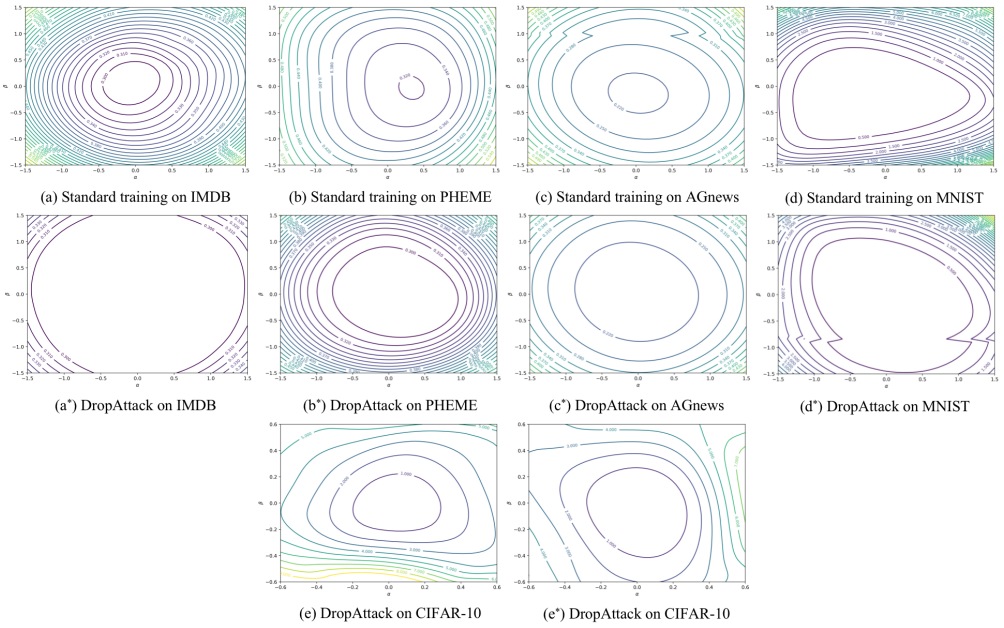


Figure 4: The impact of attack probability and perturbation coefficient on model performance.

C ADDITIONAL LOSS VISUALIZATION

We visualize the test loss function landscapes of the standard training and DropAttack adversarial training models separately. The 2D and 3D visualization results are shown in Figure 5 and Figure 6, respectively. The structure and parameters of the models are derived from Section 4.2.

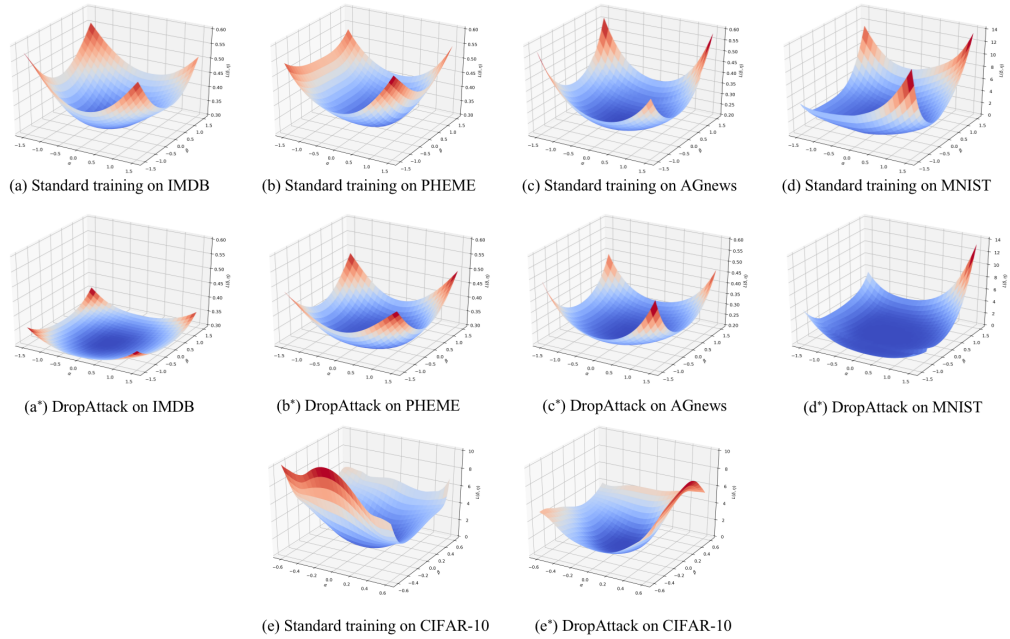


(a) Standard training on IMDB (b) Standard training on PHEME (c) Standard training on AGnews (d) Standard training on MNIST

(a*) DropAttack on IMDB (b*) DropAttack on PHEME (c*) DropAttack on AGnews (d*) DropAttack on MNIST

(e) DropAttack on CIFAR-10 (e*) DropAttack on CIFAR-10

Figure 5: 2D visualization of the minima of the risk selected by standard training and DropAttack on IMDB, PHEME, AGnews, MNIST, CIFAR-10 datasets, respectively. * With DropAttack.



(a) Standard training on IMDB (b) Standard training on PHEME (c) Standard training on AGnews (d) Standard training on MNIST

(a*) DropAttack on IMDB (b*) DropAttack on PHEME (c*) DropAttack on AGnews (d*) DropAttack on MNIST

(e) Standard training on CIFAR-10 (e*) DropAttack on CIFAR-10

Figure 6: 3D visualization of the minima of the risk selected by standard training and DropAttack on IMDB, PHEME, AGnews, MNIST, CIFAR-10 datasets, respectively. * With DropAttack.