The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Adversarial Dropout for Supervised and Semi-Supervised Learning

Sungrae Park, JunKeon Park, Su-Jin Shin, Il-Chul Moon
Department of Industrial and System Engineering
KAIST
Deajeon, South Korea
{sungraepark, alex3012, sujin.shin, icmoon}@kaist.ac.kr

Abstract

Recently, training with adversarial examples, which are generated by adding a small but worst-case perturbation on input examples, has improved the generalization performance of neural networks. In contrast to the biased individual inputs to enhance the generality, this paper introduces *adversarial dropout*, which is a minimal set of dropouts that maximize the divergence between 1) the training supervision and 2) the outputs from the network with the dropouts. The identified adversarial dropouts are used to automatically reconfigure the neural network in the training process, and we demonstrated that the simultaneous training on the original and the reconfigured network improves the generalization performance of supervised and semi-supervised learning tasks on MNIST, SVHN, and CIFAR-10. We analyzed the trained model to find the performance improvement reasons. We found that adversarial dropout increases the sparsity of neural networks more than the standard dropout. Finally, we also proved that adversarial dropout is a regularization term with a rank-valued hyper parameter that is different from a continuous-valued parameter to specify the strength of the regularization.

Introduction

Deep neural networks (DNNs) have demonstrated the significant improvement on benchmark performances in a wide range of applications. As neural networks become deeper, the model complexity also increases quickly, and this complexity leads DNNs to potentially overfit a training data set. Several techniques (Hinton et al. 2012; Poole, Sohl-Dickstein, and Ganguli 2014; Bishop 1995b; Lasserre, Bishop, and Minka 2006) have emerged over the past years to address this challenge, and *dropout* has become one of dominant methods due to its simplicity and effectiveness (Hinton et al. 2012; Srivastava et al. 2014).

Dropout randomly disconnects neural units during training as a method to prevent the feature co-adaptation (Baldi and Sadowski 2013; Wager, Wang, and Liang 2013; Wang and Manning 2013; Li, Gong, and Yang 2016). The earlier work by Hinton et al. (2012) and Srivastava et al. (2014) interpreted dropout as an extreme form of model combinations, a.k.a. a model ensemble, by sharing extensive parameters on neural networks. They proposed learning the model combination through minimizing an expected loss of models perturbed by dropout. They also pointed out that the output of dropout is the geometric mean of the outputs from the model ensemble with the shared parameters. Extending the weight sharing perspective, several studies (Baldi and Sadowski 2013; Chen et al. 2014; Jain et al. 2015) analyzed the ensemble effects from the dropout.

The recent work by Laine & Aila (2016) enhanced the ensemble effect of dropout by adding self-ensembling terms. The self-ensembling term is constructed by a divergence between two sampled neural networks from the dropout. By minimizing the divergence, the sampled networks learn from each other, and this practice is similar to the working mechanism of the ladder network (Rasmus et al. 2015), which builds a connection between an unsupervised and a supervised neural network. Our method also follows the principles of self-ensembling, but we apply the adversarial training concept to the sampling of neural network structures through dropout.

At the same time that the community has developed the dropout, *adversarial training* has become another focus of the community. Szegedy et al. (2013) showed that a certain neural network is vulnerable to a very small perturbation in the training data set if the noise direction is sensitive to the models' label assignment y given x, even when the perturbation is so small that human eyes cannot discern the difference. They empirically proved that robustly training models against adversarial perturbation is effective in reducing test errors. However, their method of identifying adversarial perturbations contains a computationally expensive inner loop. To compensate it, Goodfellow et al. (2014) suggested an approximation method, through the linearization of the loss function, that is free from the loop. Adversarial training can be conducted on supervised learning because the adversarial direction can be defined when true y labels are known. Miyato et al. (2015) proposed a virtual adversarial direction to apply the adversarial training in the semi-supervised learning that may not assume the true y value. Until now, the adversarial perturbation can be defined as a unit vector of additive noise imposed on the input or the embedding spaces (Szegedy et al. 2013; Goodfellow, Shlens, and Szegedy 2014; Miyato et al. 2015).

Our proposed method, *adversarial dropout*, can be viewed from the *dropout* and from the *adversarial train-*

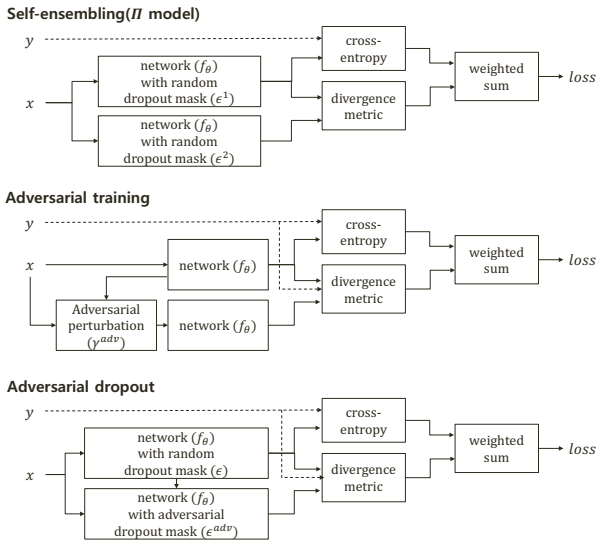

Figure 1: Diagram description of loss functions from Π model (Laine and Aila 2016), the adversarial training (Miyato et al. 2015), and our adversarial dropout.

ing perspectives. Adversarial dropout can be interpreted as dropout masks whose direction is optimized *adversarially* to the model's label assignment. However, it should be noted that adversarial dropout and traditional adversarial training with additive perturbation are different because adversarial dropout induces the sparse structure of neural network while the other does not make changes in the structure of the neural network, directly.

Figure 1 describes the proposed loss function construction of adversarial dropout compared to 1) the recent dropout model, which is Π model (Laine and Aila 2016) and 2) the adversarial training (Goodfellow, Shlens, and Szegedy 2014; Miyato et al. 2015). When we compare adversarial dropout to Π model, both divergence terms are similarly computed from two different dropped networks, but adversarial dropout uses an optimized dropped network to adapt the concept of adversarial training. When we compare adversarial dropout to the adversarial training, the divergence term of the adversarial training is computed from one network structure with two training examples: clean and adversarial examples. On the contrary, the divergence term of the adversarial dropout is defined with two network structures: a randomly dropped network and an adversarially dropped network.

Our experiments demonstrated that 1) adversarial dropout improves the performance on MNIST supervised learning compared to the dropout suggested by Π model, and 2) adversarial dropout showed the state-of-the-art performance on the semi-supervised learning task on SVHN and CIFAR-10 when we compare the most recent techniques of dropout and adversarial training. Following the performance comparison, we visualize the neural network structure from adversarial dropout to illustrate that the adversarial dropout enables a sparse structure compared to the neural network of standard dropout. Finally, we theoretically show the original characteristics of adversarial dropout that specifies the strength of the regularization effect by the rank-valued parameter while the adversarial training specifies the strength with the conventional continuous-valued scale.

Preliminaries

Before introducing adversarial dropout, we briefly introduce stochastic noise layers for deep neural networks. Afterwards, we review adversarial training and temporal ensembling, or Π model, because two methods are closely related to adversarial dropout.

Noise Layers

Corrupting training data with noises has been well-known to be a method to stabilize prediction (Bishop 1995a; Maaten et al. 2013; Wager, Wang, and Liang 2013). This section describes two types of noise injection techniques, such as additive Gaussian noise and dropout noise.

Let $\mathbf{h}^{(l)}$ denote the l^{th} hidden variables in a neural network, and this layer can be replaced with a noisy version $\tilde{\mathbf{h}}^{(l)}$. We can vary the noise types as the followings.

- Additive Gaussian noise: $\tilde{\mathbf{h}}^{(l)} = \mathbf{h}^{(l)} + \boldsymbol{\gamma}$, where $\boldsymbol{\gamma} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_{d\times d})$ with the parameter σ^2 to restrict the degree of noises.
- Dropout noise: $\tilde{\mathbf{h}}^{(l)} = \mathbf{h}^{(l)} \odot \boldsymbol{\epsilon}$, where $\odot$ is the element-wise product of two vectors, and the elements of the noise vector are $\boldsymbol{\epsilon}_i \sim Bernoulli(1-p)$ with the parameter p. To simply put, this function specifies that $\boldsymbol{\epsilon}_i = 0$ with probability p and $\boldsymbol{\epsilon}_i = 1$ with probability $(1-p)$.

Both additive Gaussian noise and dropout noise are generalization techniques to increase the generality of the trained model, but they have different properties. The additive Gaussian noise increases the margin of decision boundaries while the dropout noise affects a model to be sparse (Srivastava et al. 2014). These noise layers can be easily included in a deep neural network. For example, there can be a dropout layer between two convolutional layers. Similarly, a layer of additive Gaussian noise can be placed on the input layer.

Self-Ensembling Model

The recently reported self-ensembling (SE) (Laine and Aila 2016), or Π model, construct a loss function that minimizes the divergence between two outputs from two sampled dropout neural networks with the same input stimulus. Their suggested regularization term can be interpreted as the following:

$$\mathcal{L}_{SE}(\mathbf{x};\boldsymbol{\theta}) := D[f_{\boldsymbol{\theta}}(\mathbf{x},\boldsymbol{\epsilon}^1), f_{\boldsymbol{\theta}}(\mathbf{x},\boldsymbol{\epsilon}^2)], \tag{1}$$

where $\boldsymbol{\epsilon}^1$ and $\boldsymbol{\epsilon}^2$ are randomly sampled dropout noises in a neural network $f_{\boldsymbol{\theta}}$, whose parameters are $\boldsymbol{\theta}$. Also, $D[\mathbf{y},\mathbf{y}']$ is a non-negative function that represents the distance between two output vectors: $\mathbf{y}$ and $\mathbf{y}'$. For example, D can be the cross entropy function, $D[\mathbf{y},\mathbf{y}'] = -\sum_i \mathbf{y}_i \log \mathbf{y}'_i$, where $\mathbf{y}$ and $\mathbf{y}'$ are the vectors whose i^{th} elements represent the probability of the i^{th} class. The divergence could be calculated between two outputs of two different structures, which

turn this regularization to be semi-supervised (Bachman, Alsharif, and Precup 2014). Π model is based on the principle of Γ model, which is the ladder network by Rasmus et al. (2015). Our proposed method, adversarial dropout, can be seen as a special case of Π model when one dropout neural network is adversarially sampled.

Adversarial Training

Adversarial dropout follows the training mechanism of adversarial training, so we briefly introduce a generalized formulation of the adversarial training. The basic concept of adversarial training (AT) is an incorporation of adversarial examples on the training process. Additional loss function by including adversarial examples (Szegedy et al. 2013; Goodfellow, Shlens, and Szegedy 2014; Miyato et al. 2015) can be defined as a generalized form:

$$\mathcal{L}_{AT}(\mathbf{x}, y; \boldsymbol{\theta}, \delta) := D[g(\mathbf{x}, y, \boldsymbol{\theta}), f_{\boldsymbol{\theta}}(\mathbf{x} + \boldsymbol{\gamma}^{adv})] \quad (2)$$

where $\boldsymbol{\gamma}^{adv} := argmax_{\boldsymbol{\gamma}; \|\boldsymbol{\gamma}\|_\infty \leq \delta} D[g(\mathbf{x}, y, \boldsymbol{\theta}), f_{\boldsymbol{\theta}}(\mathbf{x} + \boldsymbol{\gamma})]$.

Here, $\boldsymbol{\theta}$ is a set of model parameters, δ is a hyperparameter controlling the intensity of the adversarial perturbation $\boldsymbol{\gamma}^{adv}$. The function $f_{\boldsymbol{\theta}}(\mathbf{x})$ is an output distribution of a neural network to be learned. Adversarial training can be diversified by differentiating the definition of $g(\mathbf{x}, y, \boldsymbol{\theta})$, as the following.

- *Adversarial training (AT)* (Goodfellow, Shlens, and Szegedy 2014; Kurakin, Goodfellow, and Bengio 2016) defines $g(\mathbf{x}, y, \boldsymbol{\theta})$ as $g(y)$ ignoring $\mathbf{x}$ and $\boldsymbol{\theta}$, so $g(y)$ is an one-hot encoding vector of y.
- *Virtual adversarial training (VAT)* (Miyato et al. 2015; Miyato, Dai, and Goodfellow 2016) defines $g(\mathbf{x}, y, \boldsymbol{\theta})$ as $f_{\hat{\boldsymbol{\theta}}}(\mathbf{x})$ where $\hat{\boldsymbol{\theta}}$ is the current estimated parameter. This training method does not use any information from y in the adversarial part of the loss function. This enables the adversarial part to be used as a regularization term for the semi-supervised learning.

Method

This section presents our adversarial dropout that combines the ideas of adversarial training and dropout. First, we formally define the adversarial dropout. Second, we propose a training algorithm to find the instantiations of adversarial dropouts with a fast approximation method.

General Expression of Adversarial Dropout

Now, we propose the adversarial dropout (AdD), which could be an adversarial training method that determines the dropout condition to be sensitive on the model's label assignment. We use $f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon})$ as an output distribution of a neural network with a dropout mask. The below is the description of the additional loss function by incorporating adversarial dropout.

$$\mathcal{L}_{AdD}(\mathbf{x}, y, \boldsymbol{\epsilon}^s; \boldsymbol{\theta}, \delta) := D[g(\mathbf{x}, y, \boldsymbol{\theta}), f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon}^{adv})] \quad (3)$$

where $\boldsymbol{\epsilon}^{adv} := argmax_{\boldsymbol{\epsilon}; \|\boldsymbol{\epsilon}^s - \boldsymbol{\epsilon}\|_2 \leq \delta H} D[g(\mathbf{x}, y, \boldsymbol{\theta}), f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon})]$.

Here, $D[\cdot, \cdot]$ indicates a divergence function; $g(\mathbf{x}, y, \boldsymbol{\theta})$ represents an adversarial target function that can be diversified by its definition; $\boldsymbol{\epsilon}^{adv}$ is an adversarial dropout mask under the function $f_{\boldsymbol{\theta}}$ when $\boldsymbol{\theta}$ is a set of model parameters; $\boldsymbol{\epsilon}^s$ is a sampled random dropout mask instance; δ is a hyperparameter controlling the intensity of the noise; and H is the dropout layer dimension.

We introduce the boundary condition, $\|\boldsymbol{\epsilon}^s - \boldsymbol{\epsilon}\|_2 \leq \delta H$, which indicates a restriction of the number of the difference between two dropout conditions. An adversarial dropout mask should be infinitesimally different from the random dropout mask. Without this constraint, the network with adversarial dropout may become a neural network layer without connections. By restricting the adversarial dropout with the random dropout, we prevent finding such irrational layer, which does not support the back propagation. We found that the Euclidean distance between two $\boldsymbol{\epsilon}$ vectors can be calculated by using the graph edit distance or the Jaccard distance. In the supplementary material, we proved that the graph edit distance and the Jaccard distance can be abstracted as Euclidean distances between two $\boldsymbol{\epsilon}$ vectors.

In the general form of adversarial training, the key point is the existence of the linear perturbation $\boldsymbol{\gamma}^{adv}$. We can interpret the input with the adversarial perturbation as this adversarial noise input $\tilde{\mathbf{x}}^{adv} = \mathbf{x} + \boldsymbol{\gamma}^{adv}$. From this perspective, the authors of adversarial training limited the adversarial direction only on the space of the additive Gaussian noise $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\gamma}^0$, where $\boldsymbol{\gamma}^0$ is a sampled Gaussian noise on the input layer. In contrast, adversarial dropout can be considered as a noise space generated by masking hidden units, $\tilde{\mathbf{h}}^{adv} = \mathbf{h} \odot \boldsymbol{\epsilon}^{adv}$ where $\mathbf{h}$ is hidden units, and $\boldsymbol{\epsilon}^{adv}$ is an adversarially selected dropout condition. If we assume the adversarial training as the Gaussian additive perturbation on the input, the perturbation is linear in nature, but adversarial dropout could be non-linear perturbation if the adversarial dropout is imposed upon multiple layers.

Supervised Adversarial Dropout *Supervised Adversarial dropout (SAdD)* defines $g(\mathbf{x}, y, \boldsymbol{\theta})$ as y, so g is a one-hot vector of y as the typical neural network. The divergence term from Formula 3 can be converted as follows:

$$\mathcal{L}_{SAdD}(\mathbf{x}, y, \boldsymbol{\epsilon}^s; \boldsymbol{\theta}, \delta) := D[g(y), f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon}^{adv})] \quad (4)$$

where $\boldsymbol{\epsilon}^{adv} := argmax_{\boldsymbol{\epsilon}; \|\boldsymbol{\epsilon}^s - \boldsymbol{\epsilon}\|_2 \leq \delta H} D[g(y), f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon})]$.

In this case, the divergence term can be seen as the pure loss function for a supervised learning with a dropout regularization. However, $\boldsymbol{\epsilon}^{adv}$ is selected to maximize the divergence between the true information and the output from the dropout network, so $\boldsymbol{\epsilon}^{adv}$ eventually becomes the mask on the most contributing features. This adversarial mask provides the learning opportunity on neurons, so called *dead filter*, that was considered to be less informative.

Virtual Adversarial Dropout *Virtual adversarial dropout (VAdD)* defines $g(\mathbf{x}, y, \boldsymbol{\theta}) = f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon}^s)$. This uses the loss function as a regularization term for semi-supervised learning. The divergence term in Formula 3 can be represented as below:

$$\mathcal{L}_{VAdD}(\mathbf{x}, y, \boldsymbol{\epsilon}^s; \boldsymbol{\theta}, \delta) := D[f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon}^s), f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon}^{adv})] \quad (5)$$

where $\boldsymbol{\epsilon}^{adv} := argmax_{\boldsymbol{\epsilon}; \|\boldsymbol{\epsilon}^s - \boldsymbol{\epsilon}\|_2 \leq \delta H} D[f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon}^s), f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon})]$.

VAdD is a special case of a self-ensembling model with two dropouts. They are 1) a dropout, $\boldsymbol{\epsilon}^s$, sampled from a random distribution with a hyperparameter and 2) a dropout, $\boldsymbol{\epsilon}^{adv}$, composed to maximize the divergence function of the learner, which is the concept of the noise injection from the virtual adversarial training. The two dropouts create a regularization as the virtual adversarial training, and the inference procedure optimizes the parameters to reduce the divergence between the random dropout and the adversarial dropout. This optimization triggers the self-ensemble learning in (Laine and Aila 2016). However, the adversarial dropout is different from the previous self-ensembling because one dropout is induced by the adversarial setting, not by a random sampling.

Learning with Adversarial Dropout The full objective function for the learning with the adversarial dropout is given by

$$l(y, f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon}^s)) + \lambda \mathcal{L}_{AdD}(\mathbf{x}, y, \boldsymbol{\epsilon}^s; \boldsymbol{\theta}, \delta) \tag{6}$$

where $l(y, f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon}^s))$ is the negative log-likelihood for y given x under the sampled dropout instance $\boldsymbol{\epsilon}^s$. There are two scalar-scale hyper-parameters: (1) a trade-off parameter, λ, for controlling the impact of the proposed regularization term and (2) the constraints, δ, specifying the intensity of adversarial dropout.

Combining Adversarial Dropout and Adversarial Training Additionally, it should be noted that the adversarial training and the adversarial dropout are not exclusive training methods. A neural network can be trained by imposing the input perturbation with the Gaussian additive noise, and by enabling the adversarially chosen dropouts, simultaneously. Formula 7 specifies the loss function of simultaneously utilizing the adversarial dropout and the adversarial training.

$$l(y, f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon}^s)) + \lambda_1 \mathcal{L}_{AdD}(\mathbf{x}, y, \boldsymbol{\epsilon}^s) + \lambda_2 \mathcal{L}_{AT}(\mathbf{x}, y) \tag{7}$$

where λ_1 and λ_2 are trade-off parameters controlling the impact of the regularization terms.

Fast Approximation Method for Finding Adversarial Dropout Condition

Once the adversarial dropout, $\boldsymbol{\epsilon}^{adv}$, is identified, the evaluation of $\mathcal{L}_{AdD}$ simply becomes the computation of the loss and the divergence functions. However, the inference on $\boldsymbol{\epsilon}^{adv}$ is difficult because of three reasons. First, we cannot obtain a closed-form solution on the exact adversarial noise value, $\boldsymbol{\epsilon}^{adv}$. Second, the feasible space for $\boldsymbol{\epsilon}^{adv}$ is restricted under $\|\boldsymbol{\epsilon}^s - \boldsymbol{\epsilon}^{adv}\|_2 \leq \delta H$, which becomes a constraint in the optimization. Third, $\boldsymbol{\epsilon}^{adv}$ is a binary-valued vector rather than a continuous-valued vector because $\boldsymbol{\epsilon}^{adv}$ indicates the activation of neurons. This discrete nature requires an optimization technique like *integer programming*.

To mitigate this difficulty, we approximated the objective function, $\mathcal{L}_{AdD}$, with the first order of the Taylor expansion by relaxing the domain space of $\boldsymbol{\epsilon}^{adv}$. This Taylor expansion of the objective function was used in the earlier works of adversarial training (Goodfellow, Shlens, and Szegedy 2014; Miyato et al. 2015). After the approximation, we found an adversarial dropout condition by solving an integer programming problem.

To define a neural network with a dropout layer, we separate the output function into two neural sub-networks, $f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon}) = f^{upper}_{\boldsymbol{\theta}_1}(\mathbf{h}(\mathbf{x}) \odot \boldsymbol{\epsilon})$, where $f^{upper}_{\boldsymbol{\theta}_1}$ is the upper part neural network of the dropout layer and $\mathbf{h}(\mathbf{x}) = f^{under}_{\boldsymbol{\theta}_2}(\mathbf{x})$ is the under part neural network. Our objective is optimizing an adversarial dropout noise $\boldsymbol{\epsilon}^{adv}$ by maximizing the following divergence function under the constraint $\|\boldsymbol{\epsilon}^s - \boldsymbol{\epsilon}^{adv}\|_2 \leq \delta H$:

$$D(\mathbf{x}, \boldsymbol{\epsilon}; \boldsymbol{\theta}, \boldsymbol{\epsilon}^s) = D[g(\mathbf{x}, y, \boldsymbol{\theta}, \boldsymbol{\epsilon}^s), f^{upper}_{\boldsymbol{\theta}_1}(\mathbf{h}(\mathbf{x}) \odot \boldsymbol{\epsilon}))] \tag{8}$$

where $\boldsymbol{\epsilon}^s$ is a sampled dropout mask, and $\boldsymbol{\theta}$ is a parameter of the neural network model. We approximate the above divergence function by deriving the first order of the Taylor expansion by relaxing the domain space of $\boldsymbol{\epsilon}$ from the multiple binary spaces, $\{0, 1\}^H$, to the real value spaces, $[0, 1]^H$. This conversion is a common step in the integer programming research as (Hemmecke et al. 2010):

$$D(\mathbf{x}, \boldsymbol{\epsilon}; \boldsymbol{\theta}, \boldsymbol{\epsilon}^s) \approx D(\mathbf{x}, \boldsymbol{\epsilon}^0; \boldsymbol{\theta}, \boldsymbol{\epsilon}^s) + (\boldsymbol{\epsilon} - \boldsymbol{\epsilon}^0)^T \mathbf{J}(\mathbf{x}, \boldsymbol{\epsilon}^0) \tag{9}$$

where $\mathbf{J}(\mathbf{x}, \boldsymbol{\epsilon}^0)$ is the Jacobian vector given by $\mathbf{J}(\mathbf{x}, \boldsymbol{\epsilon}^0) := \bigtriangledown_{\boldsymbol{\epsilon}} D(\mathbf{x}, \boldsymbol{\epsilon}; \boldsymbol{\theta}, \boldsymbol{\epsilon}^s)|_{\boldsymbol{\epsilon}=\boldsymbol{\epsilon}^0}$ when $\boldsymbol{\epsilon}^0 = 1$ indicates no noise injection. The above Taylor expansion provides a linearized optimization objective function by controlling ϵ. Therefore, we reorganized the Taylor expansion with respect to ϵ as the below:

$$D(\mathbf{x}, \boldsymbol{\epsilon}; \boldsymbol{\theta}, \boldsymbol{\epsilon}^s) \propto \sum_i \boldsymbol{\epsilon}_i \mathbf{J}_i(\mathbf{x}, \boldsymbol{\epsilon}^0) \tag{10}$$

where $\mathbf{J}_i(\mathbf{x}, \boldsymbol{\epsilon}^0)$ is the i^{th} element of $\mathbf{J}(\mathbf{x}, \boldsymbol{\epsilon}^0)$. Since we cannot proceed further with the given formula, we introduce an alternative Jaccobian formula that further specifies the dropout mechanism by $\odot$ and $\mathbf{h}(\mathbf{x})$ as the below.

$$J(\mathbf{x}, \boldsymbol{\epsilon}^0) \approx \mathbf{h}(\mathbf{x}) \odot \bigtriangledown_{\mathbf{h}(\mathbf{x})} D(\mathbf{x}, \boldsymbol{\epsilon}^0; \boldsymbol{\theta}, \boldsymbol{\epsilon}^s) \tag{11}$$

where $\mathbf{h}(\mathbf{x})$ is the output vector of the under part neural network of the adversarial dropout.

The control variable, $\boldsymbol{\epsilon}$, is a binary vector whose elements are either one or zero. Under this approximate divergence, finding a maximal point of $\boldsymbol{\epsilon}$ can be viewed as the 0/1 knapsack problem (Kellerer, Pferschy, and Pisinger 2004), which is one of the most popular integer programming problems.

To find $\boldsymbol{\epsilon}^{adv}$ with the constraint, we propose Algorithm 1 based on the dynamic programming for the 0/1 knapsack problem. In the algorithm, $\boldsymbol{\epsilon}^{adv}$ is initialized with $\boldsymbol{\epsilon}^s$, and $\boldsymbol{\epsilon}^{adv}$ changes its value by the order of the degree increasing the objective divergence until $\|\boldsymbol{\epsilon}^s - \boldsymbol{\epsilon}^{adv}\|_2 \leq \delta H$; or there is no increment in the divergence. After using the algorithm, we obtain $\boldsymbol{\epsilon}^{adv}$ that maximizes the divergence with the constraint, and we evaluate the loss function $\mathcal{L}_{AdD}$.

We should notice that the complex vector of the Taylor expansion is not $\boldsymbol{\epsilon}^s$, but $\boldsymbol{\epsilon}^0$. In the case of virtual adversarial dropout, whose divergence is formed as $D[f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon}^s), f_{\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\epsilon})]$, $\boldsymbol{\epsilon}^s$ is the minimal point leading the gradient to be zero because of the identical distribution between the random and the optimized dropouts. This zero gradient affects the approximation of the divergence term as zero. To avoid the zero gradients, we set the complex vector of the Taylor expansion as $\boldsymbol{\epsilon}_0$.

Algorithm 1: Finding Adversarial Dropout Condition

Input : $\boldsymbol{\epsilon}^s$ is current sampled dropout mask
Input : δ is a hyper-parameter for the boundary
Input : $\mathbf{J}$ is the Jacobian vector
Input : H is the layer dimension.
Output: $\boldsymbol{\epsilon}_{adv}$

begin
  $\boldsymbol{z} \longleftarrow |\mathbf{J}|$ // absolute values of the Jacobian
  $\boldsymbol{i} \longleftarrow$ Arg Sort $\boldsymbol{z}$ as $z_{i_1} \geq ... \geq z_{i_H}$
  $\boldsymbol{\epsilon}^{adv} \longleftarrow \boldsymbol{\epsilon}^s$
  $d \longleftarrow 1$
  while $\|\boldsymbol{\epsilon}^s - \boldsymbol{\epsilon}^{adv}\|_2 \leq \delta H$ *and* $d \leq H$ **do**
    if $\epsilon^{adv}_{i_d} = 0$ *and* $\mathbf{J}_{i_d} > 0$ **then**
      $\epsilon^{adv}_{i_d} \longleftarrow 1$
    else if $\epsilon^{adv}_{i_d} = 1$ *and* $\mathbf{J}_{i_d} < 0$ **then**
      $\epsilon^{adv}_{i_d} \longleftarrow 0$
    end
    $d \longleftarrow d + 1$
  end
end

Table 1: Test performance with 1,000 labeled (semi-supervised) and 60,000 labeled (supervised) examples on MNIST. Each setting is repeated for eight times.

	Error rate (%) with # labels	
Method	1,000	All (60,000)
Plain (only dropout)	2.99 ± 0.23	0.53 ± 0.03
AT	-	0.51 ± 0.03
VAT	1.35 ± 0.14	0.50 ± 0.01
Π model	1.00 ± 0.08	0.50 ± 0.02
SAdD	-	$\mathbf{0.46 \pm 0.01}$
VAdD (KL)	$\mathbf{0.99 \pm 0.07}$	0.47 ± 0.01
VAdD (QE)	$\mathbf{0.99 \pm 0.09}$	$\mathbf{0.46 \pm 0.02}$

Experiments

This section evaluates the empirical performance of adversarial dropout for supervised and semi-supervised classification tasks on three benchmark datasets, MNIST, SVHN, and CIFAR-10. In every presented task, we compared adversarial dropout, Π model, and adversarial training. We also performed additional experiments to analyze the sparsity of adversarial dropout.

Supervised and Semi-supervised Learning on MNIST task

In the first set of experiments, we benchmark our method on the MNIST dataset (LeCun et al. 1998), which consists of 70,000 handwritten digit images of size 28×28 where 60,000 images are used for training and the rest for testing.

Our basic structure is a convolutional neural network (CNN) containing three convolutional layers, which filters are 32, 64, and 128, respectively, and three max-pooling layers sized by 2×2. The adversarial dropout applied only on the final hidden layer. The structure detail and the hyper-parameters are described in Appendix B.1.

We conducted both supervised and semi-supervised learnings to compare the performances from the standard dropout, Π model, and adversarial training models utilizing linear perturbations on the input space. The supervised learning used 60,000 instances for training with full labels. The semi-supervised learning used 1,000 randomly selected instances with their labels and 59,000 instances with only their input images. Table 1 shows the test error rates including the baseline models. Over all experiment settings, SAdD and VAdD further reduce the error rate from Π model, which had the best performance among the baseline models. In the table, KL and QE indicate Kullback-Leibler divergence and quadratic error, respectively, to specify the divergence function, $D[\mathbf{y}, \hat{\mathbf{y}}]$.

Supervised and Semi-supervised Learning on SVHN and CIFAR-10

We experimented the performances of the supervised and the semi-supervised tasks on the SVHN (Netzer et al. 2011) and the CIFAR-10 (Krizhevsky and Hinton 2009) datasets consisting of 32×32 color images in ten classes. For these experiments, we used the large-CNN (Laine and Aila 2016; Miyato et al. 2017). The details of the structure and the settings are described in Appendix B.2.

Table 2 shows the reported performances of the close family of CNN-based classifiers for the supervised and semi-supervised learning. We did not consider the recently advanced architectures, such as ResNet (He et al. 2016) and DenseNet (Huang et al. 2016), because we intend to compare the performance increment by the dropout and other training techniques.

In supervised learning tasks using all labeled train data, adversarial dropout models achieved the top performance compared to the results from the baseline models, such as Π model and VAT, on both datasets. When applying adversarial dropout and adversarial training together, there were further improvements in the performances.

Additionally, we conducted experiments on the semi-supervised learning with randomly selected labeled data and unlabeled images. In SVHN, 1,000 labeled and 72,257 unlabeled data were used for training. In CIFAR-10, 4,000 labeled and 46,000 unlabeled data were used. Table 2 lists the performance of the semi-supervised learning models, and our implementations with both VAdD and VAT achieved the top performance compared to the results from (Sajjadi, Javanmardi, and Tasdizen 2016).

Our experiments demonstrate that VAT and VAdD are complementary. When applying VAT and VAdD together by simply adding their divergence terms on the loss function, see Formula 7, we achieved the state-of-the-art performances on the semi-supervised learning on both datasets; 3.55% of test error rates on SVHN, and 10.04% and 9.22% of test error rates on CIFAR-10. Additionally, VAdD alone achieved a better performance than the self-ensemble model (Π model). This indicates that considering an adversarial perturbation on dropout layers enhances the self-ensemble effect.

Table 2: Test performances of semi-supervised and supervised learning on SVHN and CIFAR-10. Each setting is repeated for five times. KL and QE indicate Kullback-Leibler divergence and quadratic error, respectively, to specify the divergence function, $D[\mathbf{y}, \hat{\mathbf{y}}]$

	SVHN with # labels		CIFAR-10 with # labels	
Method	1,000	73,257 (All)	4,000	50,000 (All)
Π model (Laine and Aila 2016)	4.82	2.54	12.36	5.56
Tem. ensembling (Laine and Aila 2016)	4.42	2.74	12.16	5.60
Sajjadi et al. (Sajjadi, Javanmardi, and Tasdizen 2016)	-	-	11.29	-
VAT (Miyato et al. 2017)	3.86	-	10.55	5.81
Π model (our implementation)	4.35 ± 0.04	2.53 ± 0.05	12.62 ± 0.29	5.77 ± 0.11
VAT (our implementation)	$\mathbf{3.74 \pm 0.09}$	2.69 ± 0.04	11.96 ± 0.10	5.65 ± 0.17
SAdD	-	2.46 ± 0.05	-	5.46 ± 0.16
VAdD (KL)	4.16 ± 0.08	$\mathbf{2.31 \pm 0.01}$	11.68 ± 0.19	5.27 ± 0.10
VAdD (QE)	4.26 ± 0.14	2.37 ± 0.03	$\mathbf{11.32 \pm 0.11}$	$\mathbf{5.24 \pm 0.12}$
VAdD (KL) + VAT	$\mathbf{3.55 \pm 0.05}$	$\mathbf{2.23 \pm 0.03}$	10.07 ± 0.11	$\mathbf{4.40 \pm 0.12}$
VAdD (QE) + VAT	$\mathbf{3.55 \pm 0.07}$	2.34 ± 0.05	$\mathbf{9.22 \pm 0.10}$	4.73 ± 0.04

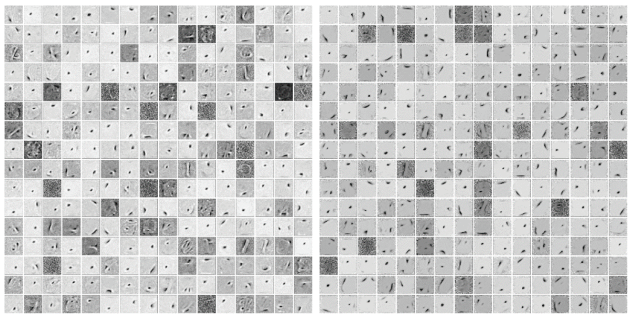

Figure 2: Features of one hidden layer autoencoders trained on MNIST; a standard dropout (left) and an adversarial dropout (right).

Effect on Features and Sparsity from Adversarial Dropout

Dropout prevents the co-adaptation between the units in a neural network, and the dropout decreases the dependency between hidden units (Srivastava et al. 2014). To compare the adversarial dropout and the standard dropout, we analyzed the co-adaptations by visualizing features of autoencoders on the MNIST dataset. The autoencoder consists with one hidden layer, whose dimension is 256, with the ReLU activation. When we trained the autoencoder, we set the dropout with $p = 0.5$, and we calculated the reconstruction error between the input data and the output layer as a loss function to update the weight values of the autoencoder with the standard dropout. On the other hand, the adversarial dropout error is also considered when we update the weight values of the autoencoder with the parameters, $\lambda = 0.2$, and $\delta = 0.3$. The trained autoencoders showed similar reconstruction errors on the test dataset.

Figure 2 shows the visualized features from the autoencoders. There are two differences identified from the visualization; 1) adversarial dropout prevents that the learned weight matrix contains black boxes, or *dead filters*, which may be all zero for many different inputs and 2) adversarial dropout tends to standardize other features, except for localized features viewed as black dots, while the standard dropout tends to ignore the neighborhoods of the localized features. These show that adversarial dropout standardizes the other features while preserving the characteristics of localized features from the standard dropout . These could be the main reason for the better generalization performance.

The important side-effect of the standard dropout is the sparse activations of the hidden units (Hinton et al. 2012). To analyze the sparse activations by adversarial dropout, we compared the activation values of the auto-encoder models with no-dropout, dropout, and adversarial dropout on the MNIST test dataset. A sparse model should only have a few highly activated units, and the average activation of any unit across data instances should be low (Hinton et al. 2012). Figure 3 plot the distribution of the activation values and their means across the test dataset. We found that the adversarial dropout has fewer highly activated units compared to others. Moreover, the mean activation values of the adversarial dropout were the lowest. These indicate that adversarial dropout improves the sparsity of the model than the standard dropout does.

Disucssion

The previous studies proved that the adversarial noise injections were an effective regularizer (Goodfellow, Shlens, and Szegedy 2014). In order to investigate the different properties of adversarial dropout, we explore a very simple case of applying adversarial training and adversarial dropout to the linear regression.

Linear Regression with Adversarial Training

Let $\mathbf{x}_i \in \mathbb{R}^D$ be a data point and $y_i \in \mathbb{R}$ be a target where $i = \{1, ..., N\}$. The objective of the linear regression is finding $\mathbf{w} \in \mathbb{R}^D$ that minimizes $l(\mathbf{w}) = \sum_i \|y_i - \mathbf{x}_i^T \mathbf{w}\|^2$.

To express adversarial examples, we denote $\tilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{r}_i^{adv}$ as the adversarial example of $\mathbf{x}_i$ where $\mathbf{r}_i^{adv} = \delta sign(\bigtriangledown_{\mathbf{x}_i} l(\mathbf{w}))$ utilizing the fast gradient sign method (FGSM) (Goodfellow, Shlens, and Szegedy 2014), δ is a

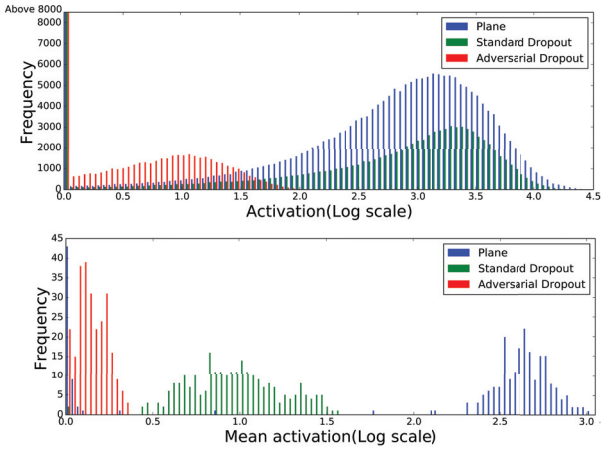

Figure 3: Histograms of the activation values and the mean activation values from a hidden layer of autoencoders in 1,000 MNIST test images. All values are converted by the log scale for the comparison.

control parameter representing the degree of adversarial noises. With the adversarial examples, the objective function of the adversarial training can be viewed as follows:

$$l_{AT}(\mathbf{w}) = \sum_i \|y_i - (\mathbf{x}_i + \mathbf{r}_i^{adv})^T \mathbf{w}\|^2 \qquad (12)$$

The above equation is translated into the below formula by isolating the terms with $\mathbf{r}_i^{adv}$ as the additive noise.

$$l(\mathbf{w}) + \sum_{ij} |\delta \bigtriangledown_{x_{ij}} l(\mathbf{w})| + \delta^2 \mathbf{w}^T \Gamma_{AT} \mathbf{w} \qquad (13)$$

where $\Gamma_{AT} = \sum_i sign(\bigtriangledown_{\mathbf{x}_i} l(\mathbf{w}))^T sign(\bigtriangledown_{\mathbf{x}_i} l(\mathbf{w}))$. The second term shows the L_1 regularization by multiplying the degree of the adversarial noise, δ, at each data point. Additionally, the third term indicates the L_2 regularization with Γ_{AT}, which form the scales of $\mathbf{w}$ by the gradient direction differences over all data points. The penalty terms are closely related with the hyper-parameter δ. When δ approaches to zero, the regularization term disappears because the inputs become adversarial examples, not anymore. For a large δ, the regularization constant grows larger than the original loss function, and the learning becomes infeasible. The previous studies proved that the adversarial objective function based on the FGSM is an effective regularizer. This paper investigated that training a linear regression with adversarial examples provides two regularization terms of the above equation.

Linear Regression with Adversarial Dropout

Now, we turn to the case of applying adversarial dropout to a linear regression. To represent the adversarial dropout, we denote $\tilde{\mathbf{x}}_i = \boldsymbol{\epsilon}_i^{adv} \odot \mathbf{x}_i$ as the adversarially dropped input of $\mathbf{x}_i$ where $\boldsymbol{\epsilon}_i^{adv} = argmax_{\boldsymbol{\epsilon}; \|\boldsymbol{\epsilon}_i - 1\|_2 \leq k} \|y_i - (\boldsymbol{\epsilon}_i \odot \mathbf{x}_i)^T \mathbf{w}\|^2$ with the hyper-parameter, k, controlling the degree of the adversarial dropout. For simplification, we used one vector as the sampled dropout, $\boldsymbol{\epsilon}^s$, of the adversarial dropout. If we apply Algorithm 1, the adversarial dropout can be defined as follows:

$$\epsilon_{ij}^{adv} = \begin{cases} 0 & \text{if } \mathbf{x}_{ij} \bigtriangledown_{\mathbf{x}_{ij}} l(\mathbf{w}) \leq min\{s_{ik}, 0\} \\ 1 & \text{otherwise} \end{cases} \qquad (14)$$

where s_{ik} is the k^{th} lowest element of $\mathbf{x}_i \odot \bigtriangledown_{\mathbf{x}_i} l(\mathbf{w})$. This solution satisfies the constraint, $\|\boldsymbol{\epsilon}_i - \boldsymbol{\epsilon}^s\|_2 \leq k$. With this adversarial dropout condition, the objective function of the adversarial dropout can be defined as the belows:

$$l_{AdD}(\mathbf{w}) = \sum_i \|y_i - (\boldsymbol{\epsilon}_i^{adv} \odot \mathbf{x}_i)^T \mathbf{w}\|^2 \qquad (15)$$

When we isolate the terms with $\boldsymbol{\epsilon}^{adv}$, the above equation is translated into the below formula.

$$l(\mathbf{w}) + \sum_i \sum_{j \in S_i} |x_{ij} \bigtriangledown_{x_{ij}} l(\mathbf{w})| + \mathbf{w}^T \Gamma_{AdD} \mathbf{w} \qquad (16)$$

where $S_i = \{j | \epsilon_{ij}^{adv} = 0\}$ and $\Gamma_{AdD} = \sum_i ((1 - \boldsymbol{\epsilon}_i^{adv}) \odot \mathbf{x}_i)^T ((1 - \boldsymbol{\epsilon}_i^{adv}) \odot \mathbf{x}_i)$. The second term is the L_1 regularization of the k largest loss changes from the features of each data point. The third term is the L_2 regularization with Γ_{AdD}. These two penalty terms are related with the hyper-parameter k controlling the degree of the adversarial dropout, because the k indicates the number of elements of the set $S_i, \forall i$. When k becomes zero, the two penalty terms disappears because there will be no dropout by the constraint on $\boldsymbol{\epsilon}$.

There are two differences between the adversarial dropout and the adversarial training. First, the regularization terms of the adversarial dropout are dependent on the scale of the features of each data point. In L_1 regularization, the gradients of the loss function are re-scaled with the data points. In L_2 regularization, the data points affect the scales of the weight costs. In contrast, the penalty terms of adversarial training are dependent on the degree of adversarial noise, δ, which is a static term across the instances because δ is a single-valued hyper parameter given in the training process. Second, the penalty terms of the adversarial dropout are selectively activated by the degree of the loss changes while the penalty terms of the adversarial training are always activated.

Conclusion

The key point of our paper is combining the ideas from the adversarial training and the dropout. The existing methods of the adversarial training control a linear perturbation with additive properties only on the input layer. In contrast, we combined the concept of the perturbation with the dropout properties on hidden layers. Adversarially dropped structure becomes a poor ensemble model for the label assignment even when very few nodes are changed. However, by learning the model with the poor structure, the model prevents over-fitting using a few effective features. The experiments showed that the generalization performances are improved by applying our adversarial dropout. Additionally, our approach achieved the-state-of-the-art performances of 3.55% on SVHN and 9.22% on CIFAR-10 by applying VAdD and VAT together for the semi-supervised learning.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(2017R1D1A1A01058209)

References

Bachman, P.; Alsharif, O.; and Precup, D. 2014. Learning with pseudo-ensembles. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc. 3365–3373.

Baldi, P., and Sadowski, P. J. 2013. Understanding dropout. In *Advances in Neural Information Processing Systems*. 2814–2822.

Bishop, C. M. 1995a. Training with noise is equivalent to tikhonov regularization. *Neural computation* 7(1):108–116.

Bishop, C. M. 1995b. Regularization and complexity control in feed-forward networks.

Chen, N.; Zhu, J.; Chen, J.; and Zhang, B. 2014. Dropout training for support vector machines. *arXiv preprint arXiv:1404.4171*.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, 630–645. Springer.

Hemmecke, R.; Köppe, M.; Lee, J.; and Weismantel, R. 2010. Nonlinear integer programming. In *50 Years of Integer Programming 1958-2008*. Springer. 561–618.

Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. R. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Huang, G.; Liu, Z.; Weinberger, K. Q.; and van der Maaten, L. 2016. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*.

Jain, P.; Kulkarni, V.; Thakurta, A.; and Williams, O. 2015. To drop or not to drop: Robustness, consistency and differential privacy properties of dropout. *arXiv preprint arXiv:1503.02031*.

Kellerer, H.; Pferschy, U.; and Pisinger, D. 2004. Introduction to np-completeness of knapsack problems. In *Knapsack problems*. Springer. 483–493.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images.

Kurakin, A.; Goodfellow, I.; and Bengio, S. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.

Laine, S., and Aila, T. 2016. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*.

Lasserre, J. A.; Bishop, C. M.; and Minka, T. P. 2006. Principled hybrids of generative and discriminative models. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, 87–94. IEEE.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Li, Z.; Gong, B.; and Yang, T. 2016. Improved dropout for shallow and deep learning. In *Advances in Neural Information Processing Systems*, 2523–2531.

Maaten, L.; Chen, M.; Tyree, S.; and Weinberger, K. Q. 2013. Learning with marginalized corrupted features. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 410–418.

Miyato, T.; Maeda, S.-i.; Koyama, M.; Nakae, K.; and Ishii, S. 2015. Distributional smoothing with virtual adversarial training. *arXiv preprint arXiv:1507.00677*.

Miyato, T.; Maeda, S.-i.; Koyama, M.; and Ishii, S. 2017. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *arXiv preprint arXiv:1704.03976*.

Miyato, T.; Dai, A. M.; and Goodfellow, I. 2016. Virtual adversarial training for semi-supervised text classification. *stat* 1050:25.

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

Poole, B.; Sohl-Dickstein, J.; and Ganguli, S. 2014. Analyzing noise in autoencoders and deep networks. *arXiv preprint arXiv:1406.1831*.

Rasmus, A.; Berglund, M.; Honkala, M.; Valpola, H.; and Raiko, T. 2015. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, 3546–3554.

Sajjadi, M.; Javanmardi, M.; and Tasdizen, T. 2016. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Advances in Neural Information Processing Systems*, 1163–1171.

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

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Wager, S.; Wang, S.; and Liang, P. S. 2013. Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems*, 351–359.

Wang, S., and Manning, C. 2013. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 118–126.