Interpretable Adversarial Perturbation in Input Embedding Space for Text

Motoki Sato1, Jun Suzuki2,4†, Hiroyuki Shindo3,4, and Yuji Matsumoto3,4

1Preferred Networks, Inc.,
2NTT Communication Science Laboratories,
3Nara Institute of Science and Technology,
4RIKEN Center for Advanced Intelligence Project

sato@preferred.jp, suzuki.jun@lab.ntt.co.jp, {shindo, matsu}@is.naist.jp

Abstract

Following great success in the image processing field, the idea of adversarial training has been applied to tasks in the natural language processing (NLP) field. One promising approach directly applies adversarial training developed in the image processing field to the input word embedding space instead of the discrete input space of texts. However, this approach abandons such interpretability as generating adversarial texts to significantly improve the performance of NLP tasks. This paper restores interpretability to such methods by restricting the directions of perturbations toward the existing words in the input embedding space. As a result, we can straightforwardly reconstruct each input with perturbations to an actual text by considering the perturbations to be the replacement of words in the sentence while maintaining or even improving the task performance.

1 Introduction

The existence of (small) perturbations, which induce prediction error in machine learning models, was first discovered and discussed in [Szegedy et al., 2014]. They called the perturbed inputs adversarial examples. Such perturbations can be easily found by optimizing the input to maximize the prediction error. After this discovery, a framework called adversarial training (AdvT) was proposed [Goodfellow et al., 2015] whose basic idea was to train models that can correctly classify both the original training data and adversarial examples generated based on the training data. Using AdvT, we can further improve the generalization performance of models. This improvement implies that the loss function of adversarial examples works as a good regularizer during model training. Currently, a technique for generating adversarial examples is crucial to neural image processing for both improving the task performance and analyzing the behaviors of black-box neural models.

Unlike its great success in the image processing field, AdvT cannot be straightforwardly applied to tasks in the natural language processing (NLP) field. This is because we cannot calculate the perturbed inputs for tasks in the NLP field since the inputs consist of discrete symbols, which are not a continuous space used in image processing. A novel strategy was recently proposed to improve AdvT for NLP tasks [Miyato et al., 2017] whose key strategy is simple and straightforward: applying AdvT to continuous word embedding space rather than the discrete input space of texts. Their method preserves a theoretical background developed in the image processing field and works well as a regularizer. In fact, this method significantly improved the task performance and achieved the state-of-the-art performance on several text classification tasks. Another notable merit of this method is succinct architecture. It only requires the gradient of the loss function to obtain adversarial perturbations (see Eq. 9). Note that the gradient calculation is a standard calculation procedure for updating the model parameters during training. We can obtain adversarial perturbations in the embedding space with a surprisingly small calculation cost without incorporating any additional sophisticated architecture.

In contrast, the main drawback of this method is that it abandons the generation of adversarial examples interpretable by people since how to appropriately reconstruct perturbed inputs in the input word embedding space to actual texts is not trivial. This implies that this approach lacks interpretability.
pretability. In fact, they declared that since their training strategy is no longer intended as a defense against adversaries, they exclusively proposed it as a regularizer to stabilize the model [Miyato et al., 2017]. It is often desirable for researchers and developers to generate adversarial examples (adversarial texts) to understand the behavior of \textit{black-box} neural models. Therefore, a trade-off exists between well-formed and low-cost (gradient-based) approaches and the interpretability of the AdvT methods used in the NLP field.

The main topic of this paper is the reduction of this trade-off gap. This paper restores interpretability while preserving the good ability of regularizer. Our main idea is to only restrict the directions of the perturbations toward the locations of existing words in the word embedding space. Fig. 1 illustrates an intuitive explanation of our idea. With our method, we can straightforwardly interpret each input with a perturbation as an actual sentence by considering the perturbations to be substitutions of the words in the sentence. To the best of our knowledge, our study is the first trial that discusses the interpretability of AdvT based on adversarial perturbation applied to tasks in the NLP field.

2 Related Work

Several studies have applied the ideas of AdvT to certain NLP tasks. A method was proposed that fooled reading comprehension systems by adding sentences to the ends of paragraphs using crowdsourcing [Jia and Liang, 2017]. Random character swaps can break the output of neural machine translation systems [Belinkov and Bisk, 2018; Hosseini et al., 2017], and thus they proposed AdvT methods that generate random character swaps and utilize the generated input sentences as additional training data for their models. Moreover, a method generated a large number of input sentences by replacing a word with its synonym [Samanta and Mehta, 2017].

The primary strategy for generating adversarial examples in the NLP field clearly differs from those developed in the image processing field, which are rather ad-hoc, e.g., using human knowledge [Jia and Liang, 2017], dictionaries [Samanta and Mehta, 2017], or require such costly procedures as exhaustive searches [Samanta and Mehta, 2017]. These methods are not essentially based on the previously discussed idea of perturbation that was first discussed [Szegedy et al., 2014] for generating adversarial examples.

In contrast, our baseline method [Miyato et al., 2017] preserves a theoretical background developed in the image processing field. Thus, note that the methods discussed in this paper borrow a distinct strategy from the current primal strategy taken in the NLP field as described above.

3 Target Tasks and Baseline Models

This section briefly explains the formal definitions of our target tasks, text classification and sequence labeling, and the baseline neural models for modeling these tasks. Fig. 2 shows the architecture of the baseline neural models.

3.1 Common Notation

Let $X$ represent an input sentence. $\mathcal{V}$ denotes the vocabulary of the input words. $x^{(i)} \in \mathcal{V}$ is the $i$-th word that appears in a given input sentence $X$, where $X = (x^{(1)}, \ldots, x^{(T)})$ if the number of words in $X$ is $T$. Here we introduce the following short notation of sequence $(x^{(1)}, \ldots, x^{(T)})$ as $(x^{(t)})_{t=1}^{T}$. $\mathcal{Y}$ denotes a set of output classes. To explain the text classification and the sequence labeling tasks in a single framework, this paper assumes that output $Y$ denotes sequence of class labels $Y = (y^{(t)})_{t=1}^{T}$, where $y^{(t)} \in \mathcal{Y}$ for all $t$ in the case of sequence labeling, and class label $Y = y$, where $y \in \mathcal{Y}$ for the text classification case.

Let $w^{(t)}$ be a word embedding vector that corresponds to $x^{(t)}$ whose dimension is $D$, where $w^{(t)} \in \mathbb{R}^D$. Thus, sequence of word embedding vectors $\hat{X}$ that corresponds to $X$ can be written as $\hat{X} = (w^{(t)})_{t=1}^{T}$. Then for text classification, $\hat{y}$ denotes a corresponding class ID of $y$ in $\mathcal{Y}$. $\hat{y}$ always takes one integer from $1$ to $|\mathcal{Y}|$, where $\hat{y} \in \{1, \ldots, |\mathcal{Y}|\}$. $\hat{y}^{(t)}$ also denotes a corresponding class ID of $y^{(t)}$ in $\mathcal{Y}$ for sequence labeling. Finally, $\hat{Y}$ represents $\hat{Y} = \hat{y}$ for text classification and $\hat{Y} = (\hat{y}^{(t)})_{t=1}^{T}$ for sequence labeling.

Here, without loss of generality, we formulate a text classification task or a sequence labeling task whose inputs and outputs are respectively $\hat{X}$ and $\hat{Y}$ instead of $X$ and $Y$. This is because we can uniquely convert from $X$ to $\hat{X}$ and from $Y$ to $\hat{Y}$. Thus, training data $\mathcal{D}$ can be represented as a set of $\hat{X}$ and $\hat{Y}$ pairs, namely, $\mathcal{D} = \{(\hat{X}^{(n)}, \hat{Y}^{(n)})\}_{n=1}^{N}$, where $N$ represents the amount of training data.

3.2 Baseline Model for Text Classification

We encode input $\hat{X}$ with a recurrent neural network (RNN)-based model, which consists of an LSTM unit [Hochreiter and Schmidhuber, 1997]. The (forward) LSTM unit calculates a hidden state in each step $t$ as $h^{(t)} = \text{LSTM}(w^{(t)}, h^{(t-1)})$, where $h^{(0)}$ is assumed to be a zero vector. Then we model the (conditional) probability of output $\hat{Y}$ given input $\hat{X}$ as follows:

$$
    p(\hat{Y} \mid \hat{X}, \mathcal{W}) = \frac{\exp(q_{\hat{y}})}{\sum_{m=1}^{N} \exp(q_{m})},
$$

where $q_{m}$ is the $m$-th factor of $q$ whose dimension is $|\mathcal{Y}|$. $\mathcal{W}$ is calculated through a standard feed-forward neural network.
from $T$-th final hidden state $h^{(T)}$, where $q = \text{FFNN}(h^{(T)})$. Here we omit an explanation of the detailed configurations of LSTM and FFNN, but they will be described in our experiment section, since the selection of their configurations affects none of this paper’s discussion.

3.3 Baseline Model for Sequence Labeling

We employ a bi-directional LSTM to encode input $\hat{X}$. The hidden state of each step $t$, that is, $h^{(t)}$, can be obtained by the concatenation of two hidden states from forward and backward LSTMs: $h^{(t)} = \text{concat}(h_i^{(t)}, h_b^{(t)})$, where $h_i^{(t)} = \text{LSTM}(w_i^{(t)}, h_i^{(t-1)})$, and $h_b^{(t)} = \text{LSTM}(w_b^{(t)}, h_b^{(t+1)})$. We assume that $h_i^{(0)}$ and $h_b^{(T+1)}$ are always zero vectors. We also assume that probability $p(\tilde{Y} | \hat{X}, W)$ can be decomposed into each step $t$. This means that probability $p(\tilde{Y} | \hat{X}, W)$ can be calculated:

$$p(\tilde{Y} | \hat{X}, W) = \prod_{t=1}^{T} p(\tilde{y}^{(t)} | \hat{X}, W)$$

(2)

$$p(\tilde{y}^{(t)} | \hat{X}, W) = \frac{\exp(q_{\tilde{y}^{(t)}})}{\sum_{y \in \mathcal{Y}} \exp(q_y)}$$

(3)

where $q_m^{(t)}$ is the $m$-th factor of $q^{(t)}$ whose dimension is $|\mathcal{Y}|$. $q^{(t)}$ is calculated through a standard feed-forward neural network from $t$-th final hidden state $h^{(t)}$: $q^{(t)} = \text{FFNN}(h^{(t)})$.

3.4 Training

For training both the text categorization and the sequence labeling, we generally find the optimal parameters of an RNN-based model that can minimize the following optimization problem:

$$\text{argmin}_W \left\{ J(D, W) \right\},$$

(4)

where $W$ represents the overall parameters in the RNN-based model, $J(D, W)$ is the loss function over entire training data $D$, and $\ell(\tilde{X}, \tilde{Y}, W)$ is the loss function of individual training sample $(\tilde{X}, \tilde{Y})$ in $D$:

$$J(D, W) = \frac{1}{|D|} \sum_{(X,Y) \in D} \ell(\tilde{X}, \tilde{Y}, W)$$

(5)

$$\ell(\tilde{X}, \tilde{Y}, W) = -\log(p(\tilde{Y} | \hat{X}, W)).$$

(6)

4 Adversarial Training in Embedding Space

Adversarial training (AdvT) [Goodfellow et al., 2015] is a novel regularization method that improves the robustness of misclassifying small perturbed inputs. To distinguish the AdvT method in image processing, this paper specifically refers to AdvT that is applied to input word embedding space for NLP tasks as AdvT-Text, which was first introduced in [Miyato et al., 2017].

Let $r^{(t)}_{\text{AdvT}}$ be a (adversarial) perturbation vector for $t$-th word $x^{(t)}$ in input $\hat{X}$. We assume that $r^{(t)}_{\text{AdvT}}$ is a $D$-dimensional vector whose dimension always matches that of word embedding vector $w^{(t)}$. Fig. 3 shows the AdvT-Text architecture and our baseline neural models by applying AdvT. See also Fig. 2 for a comparison of the architecture with our baseline neural models. Let $r$ represent a concatenated vector of $r^{(t)}_{\text{AdvT}}$ for all $t$. We introduce $\hat{X}_{+r}$ that denotes $\hat{X}$ with additional small perturbations, where $\hat{X}_{+r} = (w^{(t)} + r^{(t)})_{t=1}^T$. To obtain (worst case) perturbations $r^{(t)}_{\text{AdvT}}$ for all $t$ for maximizing the negative log-likelihood (equivalent to minimizing the log-likelihood), we seek optimal solution $r_{\text{AdvT}}$ by maximizing the following equation:

$$r_{\text{AdvT}} = \arg\max_{\|r\| \leq \epsilon} \left\{ \ell(\tilde{X}_{+r}, \tilde{Y}, W) \right\},$$

(7)

where $\epsilon$ is a tunable hyper-parameter that controls the norm of the perturbation and $r_{\text{AdvT}}$ represents a concatenated vector of $r^{(t)}_{\text{AdvT}}$ for all $t$ that resemble $r$. Then based on adversarial perturbation $r_{\text{AdvT}}$, the loss function for AdvT-Text can be defined:

$$J_{\text{AdvT}}(D, W) = \frac{1}{|D|} \sum_{(X,Y) \in D} \ell(\tilde{X}_{+r_{\text{AdvT}}}, \tilde{Y}, W),$$

(8)

where $\tilde{X}_{+r_{\text{AdvT}}} = (w^{(t)} + r^{(t)}_{\text{AdvT}})_{t=1}^T$ similar to $\tilde{X}_{+r}$.

It is generally infeasible to exactly estimate $r^{(t)}_{\text{AdvT}}$ in Eq. 7 for sophisticated deep neural models. As a solution, an approximation method was proposed by linearizing $\ell(\tilde{X}, \tilde{Y}, W)$ around $\tilde{X}$ [Goodfellow et al., 2015]. For our RNN-based models, the approximation method induces the following non-iterative solution for calculating $r^{(t)}_{\text{AdvT}}$ for all $t$:

$$r^{(t)}_{\text{AdvT}} = \frac{c g^{(t)}}{||g^{(t)}||_2}, \quad g^{(t)} = \nabla_{w^{(t)}} \ell(\tilde{X}, \tilde{Y}, W),$$

(9)

where $g$ is a concatenated vector of $g^{(t)}$ for all $t$.

Finally, we jointly minimize objective functions $J(D, W)$ and $J_{\text{AdvT}}(D, W)$:

$$\tilde{W} = \arg\min_W \left\{ J(D, W) + \lambda J_{\text{AdvT}}(D, W) \right\},$$

(10)

where $\lambda$ is a coefficient that controls the balance of two loss functions.
5 Interpretable Adversarial Perturbation

As described above, we extended Adv-Text to restore the ability to generate adversarial texts that are interpretable by people while maintaining the task performance. We only restrict the directions of the perturbations in the embedding space toward existing words in the input word embedding space. The intuition behind our method is that the directions to other words can be interpreted as the substitution of another word in the sentence, which may reconstruct adversarial texts. We refer to our Adv-Text extension as interpretable AdvT-Text or iAdvT-Text.

5.1 Definition of Interpretable AdvT-Text

Suppose \( w_k \) denotes a word embedding vector that corresponds to the \( k \)-th word in vocabulary \( \mathcal{V} \). We define direction vector \( d_k^{(t)} \) that indicates the direction from \( w(t) \) to \( w_k \) in the input word embedding space:

\[
d_k^{(t)} = \frac{d_k^{(t)}}{||d_k^{(t)}||_2}, \quad \text{where} \quad d_k^{(t)} = w_k - w^{(t)}. \tag{11}\]

Note that \( d_k^{(t)} \) for all \( t \) and \( k \) is always a unit vector, \( ||d_k^{(t)}||_2 = 1 \). If the \( t \)-th word in the given input sentence is the \( k \)-th word in the vocabulary, then \( w_k = w^{(t)} \), and thus, \( d_k^{(t)} \) becomes a zero vector.\(^2\)

Next let \( \alpha^{(t)} \) be a \( |\mathcal{V}| \)-dimensional vector, and let \( \alpha_k^{(t)} \) be the \( k \)-th factor of \( \alpha^{(t)} \), where \( \alpha^{(t)} = (\alpha_k^{(t)})_{k=1}^{|\mathcal{V}|} \). We define \( r(\alpha^{(t)}) \) that denotes the perturbation generated for the \( t \)-th word in \( X \), which is parameterized by \( \alpha^{(t)} \):

\[
r(\alpha^{(t)}) = \sum_{k=1}^{|\mathcal{V}|} \alpha_k^{(t)} d_k^{(t)}, \tag{12}\]

\( \alpha_k^{(t)} \) is a weight for the direction from the \( t \)-th word in the input to the \( k \)-th word in the vocabulary. Then, similar to the definition of \( \hat{X}_{+r} \), we also introduce \( \hat{X}_{+r(\alpha)} \) as follows:

\[
\hat{X}_{+r(\alpha)} = (w^{(t)} + r(\alpha^{(t)}))^T. \tag{13}\]

Similar to Eq. 7, we seek the worst case weights of the direction vectors that maximize the loss functions as follows:

\[
\alpha_{\text{AdvT}} = \arg \max_{\alpha, ||\alpha||_2 \leq 1} \left\{ \ell(\hat{X}_{+r(\alpha)}, \hat{Y}, \mathcal{W}) \right\}. \tag{14}\]

Then we define the loss functions of our method, iAdvT-Text, based on \( \alpha_{\text{AdvT}} \):

\[
J_{\text{iAdvT}}(D, \mathcal{W}) = \frac{1}{|D|} \sum_{(X,Y) \in D} \ell(\hat{X}_{+r(\alpha_{\text{AdvT}})}, \hat{Y}, \mathcal{W}). \tag{15}\]

We substitute \( J_{\text{AdvT}}(D, \mathcal{W}) \) in Eq. 10 with \( J_{\text{iAdvT}}(D, \mathcal{W}) \) for our method, where the form of the optimization problem can be simply written:

\[
\hat{W} = \arg \min_{\mathcal{W}} \left\{ J(D, \mathcal{W}) + \lambda J_{\text{iAdvT}}(D, \mathcal{W}) \right\}. \tag{16}\]

To reduce the calculation cost, we also introduce an update formula derived by applying the same idea of the approximation method explained in Eq. 9:

\[
\alpha_{\text{iAdvT}}^{(t)} = \frac{\epsilon g^{(t)}}{||g||_2}, \quad g^{(t)} = \nabla_{\alpha^{(t)}} \ell(\hat{X}_{+r(\alpha)}, \hat{Y}, \mathcal{W}). \tag{17}\]

Similar to \( r_{\text{AdvT}}^{(t)} \), the intuitive interpretation of \( \alpha_{\text{iAdvT}}^{(t)} \) is the (normalized) strength of each direction \( d_k^{(t)} \) about how much to increase the loss function. Thus, we expect to evaluate which direction of words is a good adversarial perturbation.

5.2 Practical Computation

The most time-consuming part of our method is its calculation of the summation of all the words that appeared in Eq. 12, which includes the calculation of directions \( d_k^{(t)} \) for all the words from each word \( w_k \), as shown in Eqs. 11. At most, this creates a computational cost of \( |\mathcal{V}|^2 \), which might be unacceptable compared with the small computational cost of AdvT-Text (the previous method). Here we introduce \( \mathcal{V}^{(t)} \) as individual vocabularies of step \( t \), where \( \mathcal{V}^{(t)} \subseteq \mathcal{V} \) for all \( t \) and \( |\mathcal{V}^{(t)}| \ll |\mathcal{V}| \), i.e., \( |\mathcal{V}^{(t)}| = 10 \). In our method, we select the \( |\mathcal{V}^{(t)}| \) nearest neighbor word embeddings around each \( w_k \) for all \( t \) in each iteration during the training. This approximation is equivalent to treating \( \alpha_k^{(t)} = 0 \) for all \( k \) if \( w_k \notin \mathcal{V}^{(t)} \) for all \( t \). The intuition behind this approximation is that words with a large distance can be treated as nearly unrelated words.

5.3 Extension to Semi-Supervised Learning

Suppose \( \mathcal{D} \) denotes a set of labeled and unlabeled data. Virtual adversarial training (VAT) [Miyato et al., 2016] is a (regularization) method closely related to AdvT. VAT, a natural extension of AdvT to semi-supervised learning, can also be applied to tasks in NLP fields, which we refer to as VAT-Text. We borrow this idea and extend it to our iAdvT-Text for a semi-supervised setting, which we refer to as iVAT-Text. VAT-Text uses the following objective function for estimating the loss of adversarial perturbation \( r_{\text{VAT}} \):

\[
J_{\text{VAT}}(\mathcal{D}', \mathcal{W}) = \frac{1}{|\mathcal{D}'|} \sum_{\hat{X} \in \mathcal{D}'} \ell_{\text{KL}}(\hat{X}, \hat{X}_{+r_{\text{VAT}}}, \mathcal{W}) \tag{18}\]

\[
\ell_{\text{KL}}(\hat{X}, \hat{X}_{+r_{\text{VAT}}}, \mathcal{W}) = \text{KL}(p(\cdot | \hat{X}, \mathcal{W})||p(\cdot | \hat{X}_{+r_{\text{VAT}}}, \mathcal{W})), \tag{19}\]

where \( \text{KL}(\cdot | \cdot) \) denotes the KL divergence. To obtain \( r_{\text{VAT}} \), we solve the following optimization problem:

\[
r_{\text{VAT}} = \arg \max_{\mathcal{R}, ||\mathcal{R}||_2 \leq 1} \left\{ \text{KL}(p(\cdot | \hat{X}, \mathcal{W})||p(\cdot | \hat{X}_{+r}, \mathcal{W})) \right\}. \tag{19}\]

Then instead of solving the above optimization problem, an approximated method was proposed [Miyato et al., 2017]:

\[
r_{\text{VAT}}^{(t)} = \frac{\epsilon g^{(t)}}{||g||_2}, \quad g^{(t)} = \nabla_{w^{(t)}+r^{(t)}} \ell_{\text{KL}}(\hat{X}, \hat{X}_{+r}, \mathcal{W}). \tag{20}\]

By using the same derivation technique to obtain the above approximation, we introduce the following equation to calculate \( \alpha_{\text{iVAT}}^{(t)} \) for an extension to semi-supervised learning:

\[
\alpha_{\text{iVAT}}^{(t)} = \frac{\epsilon g^{(t)}}{||g||_2}, \quad g^{(t)} = \nabla_{\alpha^{(t)}} \ell_{\text{KL}}(\hat{X}, \hat{X}_{+r(\alpha)}, \mathcal{W}). \tag{21}\]
Table 1: Summary of datasets

<table>
<thead>
<tr>
<th>Task</th>
<th>Dataset</th>
<th>Train</th>
<th>Dev</th>
<th>Test</th>
<th>Unlabeled</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMDB</td>
<td>21,246</td>
<td>3,754</td>
<td>25,000</td>
<td>50,000</td>
<td></td>
</tr>
<tr>
<td>SEC</td>
<td>22,500</td>
<td>2,500</td>
<td>25,000</td>
<td>200,000</td>
<td></td>
</tr>
<tr>
<td>Rotten Tomatoes</td>
<td>8,636</td>
<td>960</td>
<td>1,066</td>
<td>7,911,684</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Summary of hyper-parameters

<table>
<thead>
<tr>
<th>Word embed.</th>
<th>LSTM</th>
<th>FFNN</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimensions</td>
<td>state size</td>
<td>dimensions</td>
<td>algorithm</td>
</tr>
<tr>
<td>dropout</td>
<td>direction</td>
<td>activation</td>
<td>batch size</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>initial learning rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>decay rate</td>
</tr>
<tr>
<td>256</td>
<td>1024</td>
<td>30</td>
<td>Adam</td>
</tr>
<tr>
<td>300</td>
<td>200</td>
<td>128</td>
<td>0.001</td>
</tr>
<tr>
<td>0.3</td>
<td>Uni-LSTM</td>
<td>50</td>
<td>0.9998</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Then the objective function for iVAT-Text can be written:

\[
J_{iVAT}(D', \mathcal{W}) = \frac{1}{|D'|} \sum_{\tilde{x} \in D'} \ell_{KL}(\tilde{x}, \tilde{x}_t^{(\alpha_{iVAT})}) \mathcal{W}.
\]

6 Experiments

We conducted our experiments on a sentiment classification (SEC) task, a category classification (CAC) task, and a grammatical error detection (GED) task to evaluate the effectiveness of our methods, iAdvT-Text and iVAT-Text. SEC is a text classification task that classifies a given text into either a positive or a negative class. GED is a sequence labeling task that identifies ungrammatical words.

6.1 Datasets

For SEC, we used the following well-studied benchmark datasets, IMDB [Maas et al., 2011], Elec [Johnson and Zhang, 2015], and Rotten Tomatoes [Pang and Lee, 2005]. In our experiment with the Rotten Tomatoes dataset, we utilized unlabeled examples from the Amazon Reviews dataset\(^1\). For CAC, we utilized DBpedia [Lehmann et al., 2015] and RCV1 [Lewis et al., 2004]. Since the DBpedia dataset has no additional unlabeled examples, the DBpedia results are only for the supervised learning task. Following [Miyato et al., 2017], we split the original training data into training and development sentences. For GED, we utilized the First Certificate in the English dataset (FCE-public) [Yannakoudakis et al., 2011]. Table 1 summarizes the information about each dataset.

6.2 Model Settings

To fairly compare our methods with previous methods, we followed previously described model configurations [Miyato et al., 2017] for SEC and [Rei and Yannakoudakis, 2016; Kaneko et al., 2017] GED, shown in Fig. 3: left for SEC and right for GED. Moreover, following [Miyato et al., 2017], we initialized the word embeddings and the LSTM weights with a pre-trained RNN-based language model [Bengio et al., 2000] that was trained on labeled training and unlabeled data if they were available. To reduce the computational cost of softmax loss, we use the Adaptive Softmax [Grave et al., 2017] for training language model. We utilized an early stopping criterion [Caruana et al., 2000] based on the performance measured on development sets. The hyper-parameters are summarized in Table 2, with dropout [Srivastava et al., 2014] and Adam [Kingma and Ba, 2014]. In addition, we set \(\epsilon = 5.0\) for both AdvT-Text and VAT-Text and \(\epsilon = 15.0\) for our method. We also set \(\lambda = 1\) for all the methods. To find the best hyper-parameter, we picked models whose performances were best measured on development data.

In addition, we implemented our methods (iAdvT-Text and iVAT-Text) and re-implemented the previous methods (AdvT-Text and VAT-Text) using Chainer [Tokui et al., 2015] with GPU support. All four methods share sub-modules, such as RNN-based models, in our implementation. Therefore, our internal experiments are fairly compared under identical conditions.

6.3 Evaluation by Task Performance

Table 3 shows the IMDB performance evaluated by the error rate. Random perturbation (Labeled) is the method with which we replaced \(r^{(t)}_\text{AdvT}\) with a random unit vector, and Rand-

---

\(^1\)http://snap.stanford.edu/data/web-Amazon.html
Table 5: Test performance (error rate) on DBpedia: lower is better

<table>
<thead>
<tr>
<th>Method</th>
<th>Test error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.94 (%)</td>
</tr>
<tr>
<td>AdvT-Text [Miyato et al., 2017]</td>
<td>0.92 (%)</td>
</tr>
<tr>
<td>iAdvT-Text (Ours)</td>
<td>0.99 (%)</td>
</tr>
<tr>
<td>VAT-Text [Miyato et al., 2017]</td>
<td>0.91 (%)</td>
</tr>
<tr>
<td>iVAT-Text (Ours)</td>
<td>0.93 (%)</td>
</tr>
</tbody>
</table>

Table 6: Test performance ($F_{0.5}$) on GED task: larger is better

<table>
<thead>
<tr>
<th>Method</th>
<th>$F_{0.5}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>39.21</td>
</tr>
<tr>
<td>Random Perturbation</td>
<td>39.90</td>
</tr>
<tr>
<td>AdvT-Text [Miyato et al., 2017]</td>
<td><strong>42.28</strong></td>
</tr>
<tr>
<td>iAdvT-Text (Ours)</td>
<td>42.26</td>
</tr>
<tr>
<td>VAT-Text [Miyato et al., 2017]</td>
<td>41.81</td>
</tr>
<tr>
<td>iVAT-Text (Ours)</td>
<td>41.88</td>
</tr>
<tr>
<td>BiLSTM w/ Skipgram [Rei and Yannakoudakis, 2016]</td>
<td>41.1</td>
</tr>
<tr>
<td>BiLSTM w/ GNE [Kaneko et al., 2017]</td>
<td>41.4</td>
</tr>
</tbody>
</table>

Figure 4: Visualization of perturbation at sentence-level: Texts at left of blue or green bars are sentences in datasets, and texts in blue or green bars are words reconstructed from perturbations.

Table 4 shows the performance on the other datasets.

The AdvT-Text and VAT-Text scores were obtained by our re-implemented code, which outperformed the original scores [Miyato et al., 2017] (Adv: 6.21 %, VAT: 5.91 %).

For RCV1 and DBpedia, our baselines were slightly weak due to the resource limitation of constructing the large-scale pre-trained language models.

A previous study [Nagata and Nakatani, 2010] suggested that since accurate prediction is more important than coverage in error detection applications, $F_{0.5}$ was selected rather than $F_1$. The AdvT-Text and VAT-Text scores were obtained by our re-implemented code, which outperformed the original scores [Miyato et al., 2017] (Adv: 6.21 %, VAT: 5.91 %). For RCV1 and DBpedia, our baselines were slightly weak due to the resource limitation of constructing the large-scale pre-trained language models.

A previous study [Nagata and Nakatani, 2010] suggested that since accurate prediction is more important than coverage in error detection applications, $F_{0.5}$ was selected rather than $F_1$. The AdvT-Text and VAT-Text scores were obtained by our re-implemented code, which outperformed the original scores [Miyato et al., 2017] (Adv: 6.21 %, VAT: 5.91 %). For RCV1 and DBpedia, our baselines were slightly weak due to the resource limitation of constructing the large-scale pre-trained language models.

A previous study [Nagata and Nakatani, 2010] suggested that since accurate prediction is more important than coverage in error detection applications, $F_{0.5}$ was selected rather than $F_1$. The AdvT-Text and VAT-Text scores were obtained by our re-implemented code, which outperformed the original scores [Miyato et al., 2017] (Adv: 6.21 %, VAT: 5.91 %). For RCV1 and DBpedia, our baselines were slightly weak due to the resource limitation of constructing the large-scale pre-trained language models.

A previous study [Nagata and Nakatani, 2010] suggested that since accurate prediction is more important than coverage in error detection applications, $F_{0.5}$ was selected rather than $F_1$. The AdvT-Text and VAT-Text scores were obtained by our re-implemented code, which outperformed the original scores [Miyato et al., 2017] (Adv: 6.21 %, VAT: 5.91 %). For RCV1 and DBpedia, our baselines were slightly weak due to the resource limitation of constructing the large-scale pre-trained language models.

A previous study [Nagata and Nakatani, 2010] suggested that since accurate prediction is more important than coverage in error detection applications, $F_{0.5}$ was selected rather than $F_1$. The AdvT-Text and VAT-Text scores were obtained by our re-implemented code, which outperformed the original scores [Miyato et al., 2017] (Adv: 6.21 %, VAT: 5.91 %). For RCV1 and DBpedia, our baselines were slightly weak due to the resource limitation of constructing the large-scale pre-trained language models.

A previous study [Nagata and Nakatani, 2010] suggested that since accurate prediction is more important than coverage in error detection applications, $F_{0.5}$ was selected rather than $F_1$. The AdvT-Text and VAT-Text scores were obtained by our re-implemented code, which outperformed the original scores [Miyato et al., 2017] (Adv: 6.21 %, VAT: 5.91 %). For RCV1 and DBpedia, our baselines were slightly weak due to the resource limitation of constructing the large-scale pre-trained language models. Moreover, our methods successfully reached performances that almost matched AdvT-Text and VAT-Text. Again, we emphasize that these results are substantially positive for our methods since they did not degrade the performance even when we added a strong restriction for calculating the perturbations.

In addition, in contrast to SEC, VAT-Text did not outperform AdvT-Text. Since the GED dataset does not contain a large amount of unlabeled data, we confirmed that it is hard for VAT-Text to improve the performance.

6.4 Visualization of Sentence-Level Perturbations

We visualized the perturbations computed by our method (iAdvT-Text) in Fig. 4 for understanding its behavior. We also visualized the perturbations by the previous method (AdvT-Text) for comparison. The words at the left of each (blue or green) bar indicate the words in the (true) sentences in the dataset. We selected the highest direction toward a word from each word in the sentence. In our method, it can be easily obtained by selecting the maximum values of $\alpha_{k_t}$ for all $t$. The performance of two simple methods (iAdvT-Rand and iAdvT-Best) is poor. Tables 4 and 5 show the performance on the other datasets.
For AdvT-Text, we calculated the cosine similarities between the perturbation and direction to every word $w_k$ and selected a word with the highest cosine similarity. Each word written in the (blue or green) bar represents the selected word by the above operations, and shades of color are the relative strengths of the obtained perturbations toward the selected words.

For the SEC task, the correct label of the sentence in the blue bar is positive. The iAdvT-Text successfully found the directions for replacing better with worse to increase the loss. In other words, the direction might change the class label from positive to negative. For the GED task, the sentence in the green bar contains a grammatical error word (practise), which should be replaced with play. The iAdvT-Text also found directions for replacing practise with play.

In contrast, the perturbations of AdvT-Text (Previous) were uninterpretable (replacing <eos> with Analyze, and replacing practise with UNFORTUNATELY). This is mainly because the perturbations of AdvT-Text barely matched the direction toward any existence points of word embeddings, and we just visualized the most cosmoe similar words with perturbation.

These results revealed that the directions of the perturbations in iAdvT-Text are understandable by humans, and thus, offer a chance for researchers to interpret black-box neural models, regardless whether the model properly learned certain phenomena that the researchers are interested in. We believe that such interpretability is critical, especially for sophisticated neural models. The usefulness of this visualization is the main claim of our proposed methods.

### 6.5 Adversarial Texts

We reconstructed adversarial examples, which misclassified the trained models, from the adversarial perturbations in the input word embedding space given by iAdvT-Text. To obtain adversarial texts, we first identified the largest perturbation and replaced the original word with one that matches the largest perturbation.

Table 7 shows typical examples, where the top two rows show an example for SEC, and the bottom two rows show an example for GED. For example, the second example in Table 7 was generated by changing this to that. Even though this example does not alter the meaning, the prediction was changed from Negative $\rightarrow$ Positive. The generated adversarial texts for GED still contain grammatical error; however the model predicts that they are grammatically correct. Thus, these two examples are adversarial texts.

Note that the previous methods, AdvT-Text and VAT-Text, hardly reconstruct such effective adversarial texts. Thus, this is a clear advantage of our methods compared with the previous ones.

### 7 Conclusion

This paper discussed the interpretability of adversarial training based on adversarial perturbation that was applied to tasks in the NLP field. Our proposal restricted the directions of perturbations toward the locations of existing words in the word embedding space. We demonstrated that our methods can successfully generate reasonable adversarial texts and interpretable visualizations of perturbations in the input embedding space, which we believe will greatly help researchers analyze a model’s behavior. In addition, we confirmed that our methods, iAdvT-Text and iVAT-Text, maintained or improved the state-of-the-art performance obtained by our baseline methods, AdvT-Text and VAT-Text, in well-studied sentiment classification (SEC), category classification (CAC), and grammatical error detection (GED) benchmark datasets.

### Acknowledgments

We thank four anonymous reviewers for their helpful comments. We also thank Takeru Miyato who suggested that we reproduce the result of a previous work [Miyato et al., 2017] as well as Masahiro Kaneko and Tomoya Mizumoto who provided helpful comments.

### References


[Maas et al., 2011] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *ACL*, pages 142–150, 2011.


