Adversarial Training as Stackelberg Game: An Unrolled Optimization Approach

Simiao Zuo[†], **Chen Liang**[†], **Haoming Jiang**[†], **Xiaodong Liu**[◇], **Pengcheng He**[*], **Jianfeng Gao**[◇], **Weizhu Chen**[*] **and Tuo Zhao**[†]
[†]Georgia Institute of Technology [◇]Microsoft Research [*]Microsoft Dynamics 365 AI

Abstract

Adversarial training has been shown to improve the generalization performance of deep learning models in various natural language processing tasks. Existing works usually formulate adversarial training as a zero-sum game, which is solved by alternating gradient descent/ascent algorithms. Such a formulation treats the adversarial and the defending players equally, which is undesirable because only the defending player contributes to the generalization performance. To address this issue, we propose Stackelberg Adversarial Training (SALT), which formulates adversarial training as a Stackelberg game. This formulation induces a competition between a leader and a follower, where the follower generates perturbations, and the leader trains the model subject to the perturbations. Different from conventional adversarial training, in SALT, the leader is in an advantageous position. When the leader moves, it recognizes the strategy of the follower and takes the anticipated follower's outcomes into consideration. Such a leader's advantage enables us to improve the model fitting to the unperturbed data. The leader's strategic information is captured by the Stackelberg gradient, which is obtained using an unrolling algorithm. Our experimental results on a set of machine translation and natural language understanding tasks show that SALT outperforms existing adversarial training baselines across all tasks.

1 Introduction

Adversarial training (Goodfellow et al., 2014b) has been shown to improve the generalization performance of deep learning models in various natural language processing (NLP) tasks, such as language modeling (Wang et al., 2019), machine translation (Sano et al., 2019), natural language understanding (Jiang et al., 2019), and reading comprehension (Jia and Liang, 2017). However, even though significant progress has been made, the power of adversarial training is not fully harnessed.

Conventional adversarial training is formulated as a zero-sum game (a min-max optimization problem), where the two players seek to minimize/maximize the same utility function. In this formulation, an adversarial player composes perturbations, and a defending player solves for the model parameters subject to the perturbed inputs. Existing algorithms find the equilibrium of this zero-sum game using alternating gradient descent/ascent (Madry et al., 2017). For example, in a classification problem, the adversarial player first generates the input perturbations by running gradient ascent to maximize the classification loss, and then the defending player updates the model using gradient descent, trying to decrease the classification error. Notice that in this case, neither of the players know the strategy of its competitor, i.e., the model does not know how the perturbations are generated, and vice versa. In other words, the two players are of the same priority, and either one of them can be advantageous in the game. It is possible that the adversarial player generates overstrong perturbations that hinder generalization of the model.

To resolve this issue, we grant the defending player (i.e., the model) a higher priority than the adversarial player by letting the defender recognize its competitor's strategy, such that it is advantageous in the game. Consequently, we propose **S**tackelberg **A**dversaria**l** **T**raining (SALT), where we formulate adversarial training as a Stackelberg game (Von Stackelberg, 2010). The concept arises from economics, where two firms are competing in a market, and one of the them is in the leading position by acknowledging the opponent's strategy. In Stackelberg adversarial training, a leader solves for the model parameters, and a follower generates input perturbations. The leader procures its advantage by considering what the best response of the follower is, i.e., how will the follower respond after observing the leader's decision. Then, the

leader minimizes its loss, anticipating the predicted response of the follower.

The SALT framework identifies the interaction between the leader and the follower by treating the follower's strategy (i.e., the input perturbations) as an operator of the leader's decision (i.e., the model parameters). Then we can solve for the model parameters using gradient descent. One caveat is that computing the gradient term, which we call the Stackelberg gradient, requires differentiating the interaction operator. To rigorously define this operator, recall that the follower can be approximately solved using gradient ascent. We can treat the perturbations in each iteration as an operator of the model parameters, and the interaction operator is then the composition of such update-induced operators. Correspondingly, the Stackelberg gradient is obtained by differentiating through these updates. This procedure is referred to as unrolling (Pearlmutter and Siskind, 2008), and the only computational overhead caused by it is computing Hessian vector products. As a result, when applying the finite difference method, computing the Stackelberg gradient requires two backpropagation and an extra $O(d)$ complexity operation, where d is the embedding dimension. Therefore, the unrolling algorithm computes the Stackelberg gradient without causing much computational burden.

We conduct experiments on neural machine translation (NMT) and natural language understanding (NLU) tasks. For the NMT task, we experiment on four low-resource and one rich-resource datasets. The SALT method improves upon existing adversarial training algorithms by notable margins, especially on low-resource datasets, where we achieve up to 2 BLEU score improvements. To test performance on NLU tasks, we evaluate SALT on the GLUE (Wang et al., 2018) benchmark. Our SALT method outperforms state-of-the-art models, such as BERT (Devlin et al., 2018), FreeAT (Shafahi et al., 2019), FreeLB (Zhu et al., 2019), and SMART (Jiang et al., 2019). We build SALT on the BERT-base architecture, and we achieve an average score of 84.5 on the GLUE development set, which is at least 0.7 higher than existing methods. Moreover, even though we adapt SALT to BERT-base, the performance is noticeably higher than the vanilla BERT-large model (84.5 vs. 84.0).

The unrolling procedure was first proposed for auto-differentiation (Pearlmutter and Siskind, 2008), and later applied in various context, such as hyper-parameter optimization (Maclaurin et al., 2015), meta-learning (Andrychowicz et al., 2016), and Generative Adversarial Networks (Metz et al., 2016). To the best of our knowledge, we are the first to apply the unrolling technique to adversarial training to improve generalization performance.

We summarize our contributions as the following: (1) We propose SALT, which employs a Stackelberg game formulation of adversarial training. (2) We use an unrolling algorithm to find the equilibrium of the Stackelberg game. (3) Extensive experiments on NMT and NLU tasks verify the efficacy of our method.

Notation We use $\mathrm{d}f(x)/\mathrm{d}x$ to denote the gradient of f with respect to x. We use $\partial f(x,y)/\partial x$ to denote the partial derivative of f with respect to x. For a d-dimensional vector v, its ℓ_2 norm is defined as $\|v\|_2 = (\sum_{i=1}^{d} v_i^2)^{1/2}$, and its ℓ_∞ norm is defined as $\|v\|_\infty = \max_{1\le i\le d} |v_i|$.

2 Background and Related Works

◇ **Neural machine translation** has achieved superior empirical performance (Bahdanau et al., 2014; Gehring et al., 2017; Vaswani et al., 2017). We focus on the Transformer architecture (Vaswani et al., 2017), which integrates the attention mechanism in an encoder-decoder structure. The encoder in a Transformer model first maps a source sentence into an embedding space, then the embeddings are fed into several encoding layers to generate hidden representations, where each of the encoding layers contains a self-attention mechanism and a feed-forward neural network (FFN). After which the Transformer decoder layers, each contains a self-attention, a encoder-decoder attention, and a FFN, decode the hidden representations.

◇ **Adversarial training** was originally proposed for training adversarial robust classifiers in image classification (Szegedy et al., 2013; Goodfellow et al., 2014b; Madry et al., 2017). The idea is to synthesize strong adversarial examples, and the classifier is trained to be robust to them. Different from computer vision, in NLP, the goal of adversarial training is to build models that generalize well on the unperturbed test data. Note that robustness and generalization are different concepts. Recent works (Raghunathan et al., 2020; Min et al., 2020) showed that adversarial training can hurt generalization performance, i.e., accuracy on clean data. As such, adversarial training needs to be treated with great caution. Therefore, in NLP, this tech-

nique requires refined tuning of, for example, the training algorithm and the perturbation strength.

◇ **Fine-tuning pre-trained language models** (Peters et al., 2018; Devlin et al., 2018; Radford et al., 2019; Liu et al., 2019b; He et al., 2020) is state-of-the-art for natural language understanding tasks such as the GLUE (Wang et al., 2018) benchmark. Recently, there are works that use adversarial pre-training (Liu et al., 2020a) and adversarial-regularized fine-tuning methods such as SMART (Jiang et al., 2019), FreeLB (Zhu et al., 2019), and FreeAT (Shafahi et al., 2019) to improve model generalization.

◇ **Generative adversarial training** is another line of works to conduct adversarial training. For example, Yang et al. (2017); Wu et al. (2018); Cheng et al. (2018, 2019); Huang et al. (2020) use Generative Adversarial Network (GAN, Goodfellow et al. 2014a) and reinforcement learning to synthesize adversaries. In our work, the adversaries are input perturbations, whereas in generative adversarial training, the adversary is a discriminator network, similar to that of GAN. Therefore, these works are fundamentally different from our focus.

3 Method

Natural language inputs are discrete symbols (e.g., words), instead of continuous ones. Therefore, a common approach to generate perturbations is to learn continuous embeddings of the inputs and operate on the embedding space (Miyato et al., 2016; Clark et al., 2018; Sato et al., 2018; Sano et al., 2019; Stutz et al., 2019). Let $f(x, \theta)$ be our model, where x is the input embedding, and θ is the model parameter. Further let y be the ground-truth output corresponding to x. For example, in NMT, f is a sequence-to-sequence model, x is the embedding of the source sentence, and y is the target sentence. In classification tasks, f is a classifier, x is the input sentence/document embedding, and y is the label. In both of these cases, the model is trained by minimizing the empirical risk over the training data, i.e.,

$$\min_{\theta} \mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i, \theta), y_i).$$

Here $\{(x_i, y_i)\}_{i=1}^{n}$ is our dataset, and ℓ is a task-specific loss function, e.g., cross-entropy loss.

3.1 Virtual Adversarial Training

Virtual adversarial training (VAT, Miyato et al., 2016) is a regularization technique that ensures smoothness of the model outputs around each input data point. Concretely, we define an adversarial regularizer for non-regression tasks as

$$\ell_v(x, \delta, \theta) = \mathrm{KL}\big(f(x, \theta) \,||\, f(x + \delta, \theta)\big),$$
$$\text{where } \mathrm{KL}(P \,||\, Q) = \sum_k p_k \log \frac{p_k}{q_k}.$$

Here $\mathrm{KL}(\cdot||\cdot)$ is the Kullback–Leibler (KL) divergence, δ is the perturbation corresponding to x, and $f(\cdot, \theta)$ is the prediction probability simplex given model parameters θ. In regression tasks, the model output $f(\cdot, \theta)$ is a scalar, and the adversarial regularizer is defined as

$$\ell_v\left(x, \delta, \theta\right) = \left(f(x, \theta) - f(x + \delta, \theta)\right)^2.$$

Then VAT solves

$$\min_{\theta} \mathcal{L}(\theta) + \frac{\alpha}{n} \sum_{i=1}^{n} \max_{\|\delta_i\| \le \epsilon} \ell_v(x_i, \delta_i, \theta), \quad (1)$$

where α is a tuning parameter, ϵ is a pre-defined perturbation strength, and $\|\cdot\|$ is either the ℓ_2 norm or the ℓ_∞ norm.

The min and max problems are solved using alternating gradient descent/ascent. We first generate the perturbations δ by solving the maximization problem using several steps of projected gradient ascent, and then we update the model parameters θ with gradient descent, subject to the perturbed inputs. More details are deferred to Appendix A.

One major drawback of the zero-sum game formulation (Eq. 1) is that it fails to consider the interaction between the perturbations δ and the model parameters θ. This is problematic because a small change in δ may lead to a significant change in θ, which renders the problem ill-conditioned. Thus, the model is susceptible to underfitting and generalize poorly on unperturbed test data.

3.2 Virtual Adversarial Training as Stackelberg Game

We formulate virtual adversarial training as a Stackelberg game (Von Stackelberg, 2010):

$$\min_{\theta} \mathcal{F}(\theta) = \mathcal{L}(\theta) + \frac{\alpha}{n} \sum_{i=1}^{n} \ell_v(x_i, \delta_i^K(\theta), \theta),$$
$$\text{s.t. } \delta_i^K(\theta) = U^K \circ U^{K-1} \circ \cdots \circ U^1(\delta_i^0). \quad (2)$$

Here "$\circ$" denotes operator composition, i.e., $f \circ g(\cdot) = f(g(\cdot))$. Following conventions, in this

Stackelberg game, we call the optimization problem in Eq. 2 the leader. Further, the follower in Eq. 2 is described using a equality constraint. Note that U^K is the follower's K-step composite strategy, which is the composition of K one-step strategies $\{U^k\}_{k=1}^K$. In practice, K is usually small. This is because in NLP, we target for generalization, instead of robustness, and choosing a small K prevents over-strong adversaries.

In Eq. 2, U^ks are the follower's one-step strategies, and we call them update operators, e.g., U^1 updates δ^0 to δ^1 using pre-selected algorithms. For example, projected gradient ascent can be applied as the update procedure, that is,

$$\begin{aligned}\delta^k(\theta) &= U^k(\delta^{k-1}(\theta)) \\ &= \Pi_{\|\cdot\|\le\epsilon}\left(\delta^{k-1}(\theta) + \eta\frac{\partial \ell_v(x, \delta^{k-1}(\theta), \theta)}{\partial \delta^{k-1}(\theta)}\right) \\ &\text{for } k = 1, \cdots, K, \end{aligned} \tag{3}$$

where $\delta^0 \sim \mathcal{N}(0, I)$ is a initial random perturbation, η is a pre-defined step size, and Π denotes projection to the ℓ_2-ball or the ℓ_∞-ball.

To model how the follower will react to a leader's decision θ, we consider the implicit function $\delta^K(\theta)$. Then, adversarial training can be viewed solely in terms of the leader decision θ.

We highlight that in our formulation, the leader knows the strategy, instead of only the outcome, of the follower. This information is captured by the Stackelberg gradient $\mathrm{d}\mathcal{F}(\theta)/\mathrm{d}\theta$, defined as

$$\begin{aligned}\frac{\mathrm{d}\mathcal{F}(\theta)}{\mathrm{d}\theta} &= \frac{\mathrm{d}\ell(f(x,\theta), y)}{\mathrm{d}\theta} + \alpha\frac{\mathrm{d}\ell_v(x, \delta^K(\theta), \theta)}{\mathrm{d}\theta} \\ &= \underbrace{\frac{\mathrm{d}\ell(f(x,\theta), y)}{\mathrm{d}\theta} + \alpha\frac{\partial \ell_v(x, \delta^K, \theta)}{\partial\theta}}_{\text{leader}} \\ &\quad + \underbrace{\alpha\frac{\partial \ell_v(x, \delta^K(\theta), \theta)}{\partial \delta^K(\theta)}\frac{\mathrm{d}\delta^K(\theta)}{\mathrm{d}\theta}}_{\text{leader-follower interaction}}. \end{aligned} \tag{4}$$

The underlying idea behind Eq. 4[1] is that given a leader decision θ, and a direction along which the leader's loss decreases the most, taking the follower's decision into account, we move θ in that direction. Note that the gradient used in standard adversarial training (Eq. 1) only contains the "leader" term, such that the "leader-follower interaction" is not taken into account.

[1]The second term in "leader" is written as $\partial \ell_v(x, \delta^K, \theta)/\partial\theta$, instead if $\partial \ell_v(x, \delta^K(\theta), \theta)/\partial\theta$. This is because the partial derivative of θ is only taken w.r.t. the third argument in $\ell_v(x, \delta^K, \theta)$. We drop the θ in $\delta^K(\theta)$ to avoid causing any confusion.

Algorithm 1: Stackelberg Adversarial Training with Unrolled Optimization.

Input: $\mathcal{D}$: dataset; T: total number of training epochs; K: number of unrolling steps; Optimizer: optimizer to update θ.
Initialize: model parameters θ;
for $t = 1, \cdots, T$ **do**
  for $(x, y) \in \mathcal{D}$ **do**
    Initialize $\delta^0 \sim \mathcal{N}(0, I)$;
    for $k = 1, \cdots, K$ **do**
      Compute δ^k using Eq. 3;
      Compute $\mathrm{d}\delta^k(\theta)/\mathrm{d}\theta$ using Eq. 6;
    end
    Compute $\mathrm{d}\mathcal{F}(\theta)/\mathrm{d}\theta$ based on $\mathrm{d}\delta^K(\theta)/\mathrm{d}\theta$ using Eq. 4;
    $\theta \leftarrow$ Optimizer$(\mathrm{d}\mathcal{F}(\theta)/\mathrm{d}\theta)$;
  end
end
Output: θ

3.3 SALT: Stackelberg Adversarial Training

We propose to use an unrolling method (Pearlmutter and Siskind, 2008) to compute the Stackelberg gradient (Eq. 4). The general idea is that since the interaction operator is defined as the composition of the $\{U^k\}$ operators, all of which are known, we can directly compute the derivative of $\delta^K(\theta)$ with respect to θ. Concretely, we first run a forward iteration to update δ, and then we differentiate through this procedure to acquire the Stackelberg gradient.

Note that the updates of δ can take any form, such as projected gradient ascent in Eq. 3, or more complicated alternatives like Adam (Kingma and Ba, 2014). For notation simplicity, we denote $\Delta(x, \delta^{k-1}(\theta), \theta) = \delta^k(\theta) - \delta^{k-1}(\theta)$. Accordingly, Eq. 3 can be rewritten as

$$\delta^k(\theta) = \delta^{k-1}(\theta) + \Delta(x, \delta^{k-1}(\theta), \theta). \tag{5}$$

The most expensive part in computing the Stackelberg gradient (Eq. 4) is to calculate $\mathrm{d}\delta^K(\theta)/\mathrm{d}\theta$, which involves differentiating through the composition form of the follower's strategy:

$$\begin{aligned}\frac{\mathrm{d}\delta^k(\theta)}{\mathrm{d}\theta} &= \frac{\mathrm{d}\delta^{k-1}(\theta)}{\mathrm{d}\theta} + \frac{\partial\Delta(x, \delta^{k-1}, \theta)}{\partial\theta} \\ &\quad + \frac{\partial\Delta(x, \delta^{k-1}(\theta), \theta)}{\partial\delta^{k-1}(\theta)}\frac{\mathrm{d}\delta^{k-1}(\theta)}{\mathrm{d}\theta} \\ &\quad \text{for } k = 1, \cdots, K. \end{aligned} \tag{6}$$

We can compute Eq. 6 efficiently using deep learning libraries, such as PyTorch (Paszke et al., 2019). Notice that $\Delta(x, \delta^{k-1}(\theta), \theta)$ already contains the first order derivative with respect to the perturbations. Therefore, the term $\partial\Delta(x, \delta^{k-1}(\theta), \theta)/\partial\delta^{k-1}(\theta)$ contains the Hessian of $\delta^{k-1}(\theta)$. As a result, in Eq. 4, the most expensive operation is the Hessian vector product (Hvp). Using the finite difference method, computing Hvp only requires two backpropagation and an extra $O(d)$ complexity operation. This indicates that in comparison with conventional adversarial training, SALT does not introduce significant computational overhead. The training algorithm is summarized in Algorithm 1.

4 Experiments

In all the experiments, we use *PyTorch*[2] (Paszke et al., 2019) as the backend. All the experiments are conducted on NVIDIA V100 32GB GPUs.

4.1 Baselines

We adopt seven baselines in the experiments.

◇ *Transformer* (Vaswani et al., 2017) achieves superior performance in neural machine translation.

◇ *Adversarial training* (Sano et al., 2019) in NMT can improve models' generalization power.

◇ *Virtual adversarial training* (Sano et al., 2019) achieves even better performance in NMT tasks than standard adversarial training.

◇ *BERT* (Devlin et al., 2018) is a pre-trained language model that exhibits outstanding performance after fine-tuned on downstream NLU tasks.

◇ *FreeLB* (Zhu et al., 2019) is a "free" large batch adversarial training method. This algorithm was originally proposed for NLU. We modify the algorithm so that it is also suitable for NMT tasks.

◇ *FreeAT* (Shafahi et al., 2019) enables "free" adversarial training by recycling the gradient information generated when updating the model parameters. This method was proposed for computer vision tasks, but was later modified for NLU. We further adjust the algorithm for NMT tasks.

◇ *SMART* (Jiang et al., 2019) is a state-of-the-art fine-tuning method that utilizes smoothness-inducing regularization and Bregman proximal point optimization.

[2] https://pytorch.org/

4.2 Low-Resource Neural Machine Translation

Datasets and pre-processing We adopt four low-resource datasets[3]: English-Vietnamese from IWSLT'15, German-English from IWSLT'14, French-English from IWSLT'16, and English-Romanian from WMT'17. The statistics are summarized in Table 1. We use byte-pair encoding (Sennrich et al., 2015) with 10,000 and 40,000 merge operations to build the vocabulary for the IWSLT ('14, '15, '16) and WMT'17 datasets, respectively. For other data pre-processing steps, we modify the scripts in Ott et al. (2019).

Data	Source	Train	Valid	Test
En-Vi	IWSLT'15	133k	768	1268
De-En	IWSLT'14	161k	7.2k	6.7k
Fr-En	IWSLT'16	224k	1080	1133
En-Ro	WMT'17	391k	1.9k	2.0k
En-De	WMT'16	4.5m	3.0k	3.0k

Table 1: Dataset source and statistics. Here "k" stands for thousand, and "m" stands for million.

Implementation Recall that to generate adversarial examples, we perturb the word embeddings. In NMT experiments, we perturb both the source-side and the target-side embeddings. This strategy is empirically demonstrated (Grefenstette et al., 2019) to be more effective than perturbing only one side of the inputs. We use *Fairseq*[4] (Ott et al., 2019) to implement our algorithms. We adopt the Transformer-base (Vaswani et al., 2017) architecture in all the experiments, except IWSLT'14 De-En. In this dataset, we use a model smaller than Transformer-base, which is defined in *Fairseq*[5].

Hyper-parameters We use Adam (Kingma and Ba, 2014) as the leader's (i.e., the upper level problem that solves for model parameters) optimizer, and we set $\beta = (0.9, 0.98)$. The follower's (i.e., the lower level problem that solves for perturbations) optimizer is chosen from Adam and SGD, where we observe only marginal empirical differences between these two choices. Learning rate of the leader $\text{lr}_{\text{leader}}$ is chosen from $\{5 \times 10^{-4}, 8 \times 10^{-4}, 1 \times 10^{-3}, 3 \times 10^{-3}\}$, and we set $\text{lr}_{\text{leader}} = 1 \times 10^{-3}$ in all the experiments. Learn-

[3] https://wit3.fbk.eu/

[4] https://github.com/pytorch/fairseq

[5] https://github.com/pytorch/fairseq/tree/master/examples/translation

Models	En-Vi	De-En	Fr-En	En-Ro
Transformer	30.64	34.40	37.89	22.25
Adv	31.05	34.83	38.81	23.01
FreeAT	30.99	35.19	38.55	23.18
SMART	31.96	35.66	39.33	23.63
SALT	**32.94**	**36.85**	**40.24**	**24.45**

Table 2: BLEU score on four low-resource datasets. All the baseline results are from our re-implementation.

Models	BLEU
ConvS2S (Gehring et al., 2017)	25.2
Transformer (Vaswani et al., 2017)	28.4
FreeAT (Shafahi et al., 2019)	29.0
FreeLB (Zhu et al., 2019)	29.0
SMART (Jiang et al., 2019)	29.1
SALT	**29.6**

Table 3: BLEU score on WMT'16 En-De dataset. All the baseline results are from our re-implementation.

ing rate of the follower $\text{lr}_{\text{follower}}$ is chosen from $\{5 \times 10^{-5}, 1 \times 10^{-4}, 5 \times 10^{-4}, \text{lr}_{\text{leader}}\}$, and we set $\text{lr}_{\text{follower}} = 1 \times 10^{-4}$ in all the experiments. We set the batch size to be equivalent to 64k tokens for all the datasets. The maximum number of training steps is set to be 15k, and we evaluate the models that yield the best validation performance. In all the experiments, we set the perturbation strength $\epsilon = 0.3$, and we constrain each perturbation according to its ℓ_2 norm, i.e., $\|\delta\|_2 \leq \epsilon$. We further set the unrolling steps $K = 2$. Discussions about these choices are presented in Section 4.5.

Experiment results Experiment results are summarized in Table 2. Notice that SMART, which utilizes virtual adversarial training, consistently outperforms standard adversarial training (Adv). Similar observations were also reported in Miyato et al. (2016); Sano et al. (2019). This is because Adv generates perturbations using the correct examples, thus, the label information are "leaked" (Kurakin et al., 2016). Additionally, we can see that SALT is particularly effective in this low-resource setting, where it outperforms all the baselines by large margins. In comparison with the vanilla Transformer model, SALT achieves more than 2 BLEU score improvements on all the four datasets.

4.3 Rich-Resource Neural Machine Translation

We further experiment on one rich-resource dataset: English-German from WMT'16 (Table 1), which contains about 4.5 million training samples.

Implementation We follow the same data pre-processing settings in Ott et al. (2018), and we use *Fairseq* to implement the algorithms. In this experiment, we adopt the Transformer-big architecture (Vaswani et al., 2017). Due to computational concerns, we first pre-train the model on clean samples for 24k steps. During this stage, we use 3.5k tokens per GPU with 8 GPUs, and we accumulate gradient for 16 steps. We use Adam as the optimizer with a learning rate of 10^{-3}. After 24k steps, we continue to train the model with adversarial training for 6k steps. During this stage, we change gradient accumulation steps to 4, and we reduce the leader's learning rate to 5×10^{-5}. We also add a gradient norm clipping of 0.1. For the follower, we set the perturbation strength $\epsilon = 0.05$, the learning rate to be 1×10^{-5}, and the unrolling steps $K = 1$.

Experiment results Table 3 summarizes experiment results. We can see that SALT outperforms all the baseline methods by notable margins, and it improves upon the vanilla Transformer model by 1.2 BLEU score.

4.4 Natural Language Understanding

Datasets and pre-processing We demonstrate the effectiveness of the Stackelberg game formulation on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018), which is a collection of nine NLU tasks. The benchmark includes question answering (Rajpurkar et al., 2016), linguistic acceptability (CoLA, Warstadt et al. 2019), sentiment analysis (SST, Socher et al. 2013), text similarity (STS-B, Cer et al. 2017), paraphrase detection (MRPC, Dolan and Brockett 2005), and natural language inference (RTE & MNLI, Dagan et al. 2006; Bar-Haim et al. 2006; Giampiccolo et al. 2007; Bentivogli et al. 2009; Williams et al. 2018) tasks. Dataset details can be found in Appendix C.

Implementation We evaluate our algorithm by fine-tuning a pre-trained BERT-base (Devlin et al., 2018) model. Our implementation is based on the *SMART* (Jiang et al., 2019) framework in the *MT-DNN* code-base (Liu et al., 2019a, 2020b)[6].

[6] https://github.com/microsoft/MT-DNN

	RTE Acc	MRPC Acc/F1	CoLA Mcc	SST-2 Acc	STS-B P/S Corr	QNLI Acc	QQP Acc/F1	MNLI-m/mm Acc	Average Score
$\text{BERT}_{\text{LARGE}}$	71.1	86.0/89.6	61.8	93.5	89.6/89.3	92.4	91.3/88.4	86.3/86.2	84.0
$\text{BERT}_{\text{BASE}}$	63.5	84.1/89.0	54.7	92.9	89.2/88.8	91.1	90.9/88.3	84.5/84.4	81.5
FreeAT	68.0	85.0/89.2	57.5	93.2	89.5/89.0	91.3	91.2/88.5	84.9/85.0	82.6
FreeLB	70.0	86.0/90.0	58.9	93.4	89.7/89.2	91.5	91.4/88.4	85.4/85.5	83.3
SMART	71.2	87.7/91.3	59.1	93.0	90.0/89.4	91.7	91.5/88.5	85.6/86.0	83.8
SALT	**72.9**	**88.4/91.8**	**61.0**	**93.6**	**90.4/90.0**	**92.0**	**91.7/88.6**	**86.1/85.8**	**84.5**

Table 4: Evaluation results on the GLUE development set. All the rows use $\textit{BERT}_{\textit{BASE}}$, except the top one, which is included to demonstrate the effectiveness of our model. Best results on each dataset, excluding $\textit{BERT}_{\textit{LARGE}}$, are shown in **bold**. Results of $\textit{BERT}_{\textit{BASE}}$ (Devlin et al., 2018), $\textit{BERT}_{\textit{LARGE}}$ (Devlin et al., 2018), *FreeAT* (Shafahi et al., 2019), and *FreeLB* (Zhu et al., 2019) are from our re-implementation. *SMART* results are from Jiang et al. (2019).

	RTE Acc	MRPC Acc/F1	CoLA Mcc	SST-2 Acc	STS-B P/S Corr	QNLI Acc	QQP Acc/F1	MNLI-m/mm Acc	Average Score
$\text{BERT}_{\text{BASE}}$	66.4	84.8/88.9	52.1	93.5	87.1/85.8	90.5	71.2/89.2	84.6/83.4	80.0
FreeLB	70.1	83.5/88.1	54.5	93.6	87.7/86.7	91.8	72.7/89.6	85.7/84.6	81.2
SALT	**72.2**	**85.8/89.7**	**55.6**	**94.2**	**88.0/87.1**	**92.1**	**72.8/89.8**	**85.8/84.8**	**82.0**

Table 5: GLUE test set results on the GLUE evaluation server. All the methods fine-tune a pre-trained $\text{BERT}_{\text{BASE}}$ model. Model references: $\textit{BERT}_{\textit{BASE}}$ (Devlin et al., 2018), *FreeLB* (Zhu et al., 2019).

Hyper-parameters We use Adam as both the leader's and the follower's optimizer, and we set $\beta = (0.9, 0.98)$. The learning rate of the leader $\text{lr}_{\text{leader}}$ is chosen from $\{5 \times 10^{-5}, 1 \times 10^{-4}, 5 \times 10^{-4}\}$, and the follower's learning rate is chosen from $\{1 \times 10^{-5}, \text{lr}_{\text{leader}}\}$. We choose the batch size from $\{4, 8, 16, 32\}$, and we train for a maximum 6 epochs with early-stopping based on the results on dev. We apply a gradient norm clipping of 1. We set the dropout rate in task specific layers to 0.1. We choose the perturbation strength ϵ from $\{1 \times 10^{-5}, 1 \times 10^{-4}\}$, and the ℓ_2 constraint is applied. We set the unrolling steps $K = 2$. We report the best validation performance on each dataset.

Experiment results Table 4 summarizes experiment results on the GLUE development set. We can see that SALT outperforms $\text{BERT}_{\text{BASE}}$ in all the tasks. Further, our method is particularly effective for small datasets, such as RTE, MRPC, and CoLA, where we achieve 9.4, 4.3, and 6.3 absolute improvements, respectively. Comparing with other adversarial training baselines, i.e., FreeAT, FreeLB, and SMART, our method achieves notable improvements in all the tasks.

We highlight that SALT achieves a 84.5 average score, which is significantly higher than that of the vanilla $\text{BERT}_{\text{BASE}}$ (+3.0) fine-tuning approach. Also, our average score is higher than the scores of baseline adversarial training methods (+1.9, +1.2, +0.7 for FreeAT, FreeLB, SMART, respectively). Moreover, the 84.5 average score is even higher than fine-tuning $\text{BERT}_{\text{LARGE}}$ (+0.5), which contains three times more parameters than the backbone of SALT.

Table 5 summarizes results on the GLUE test set. We can see that SALT consistently outperforms $\text{BERT}_{\text{BASE}}$ and FreeLB across all the tasks.

4.5 Analysis

$\diamond$ **Unrolling reduces bias** In Figure 1, we visualize the training and the validation error on the STS-B and the SST datasets from the GLUE benchmark. As mentioned, conventional adversarial training suffers from over-strong perturbations, such that the model cannot fit the unperturbed data well. This is supported by the fact that the training loss of SALT is smaller than that of SMART, which means SALT fits the data better. SALT also yields a smaller loss than SMART on the validation data, indicating that the Stackelberg game-formulated model exhibits better generalization performance.

$\diamond$ **Comparison with Unrolled-GAN** The unrolling technique has been applied to train GANs (Unrolled-GAN, Metz et al. 2016). However, sub-

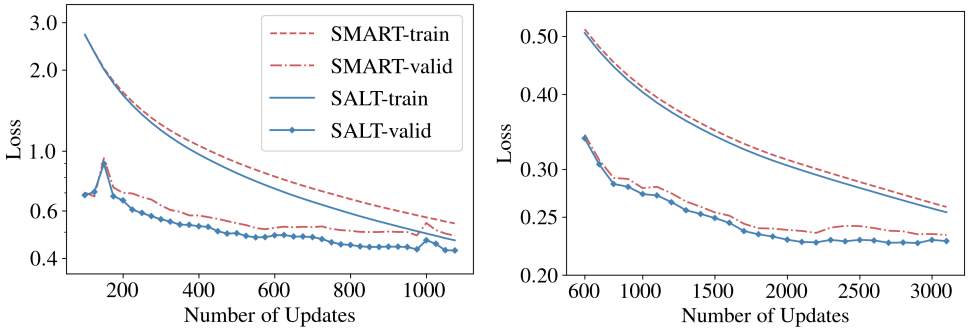
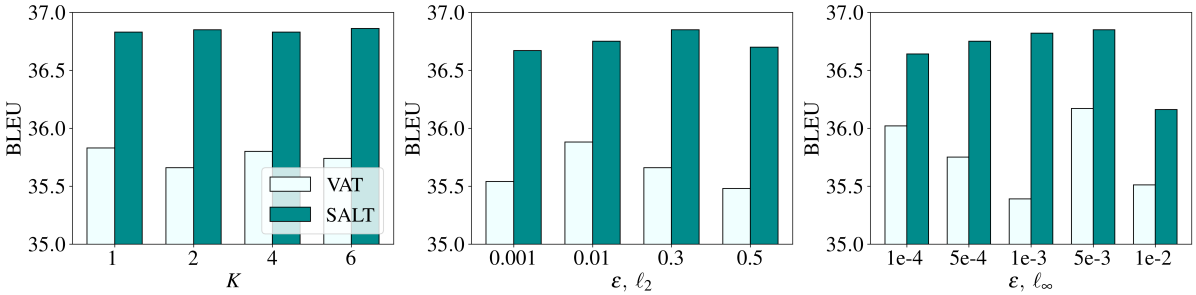
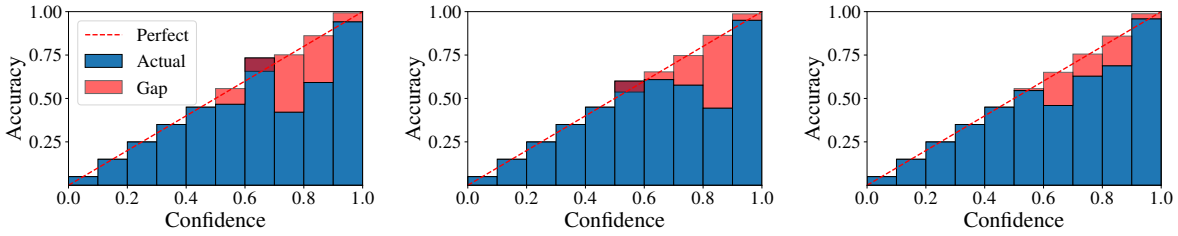


Figure 1: Training and validation loss of *SMART* and *SALT* on STS-B (left) and SST (right) datasets.



(a) Number of unrolling steps. (b) Perturbation strength ϵ, ℓ_2 case. (c) Perturbation strength ϵ, ℓ_∞ case.

Figure 2: Relation between BLEU score and different factors on the IWSLT'14 De-En dataset.



(a) $\text{BERT}_{\text{BASE}}$ (ECE: 6.09%). (b) SMART (ECE: 5.08%). (c) SALT (ECE: 4.06%).

Figure 3: Reliability diagrams on SST. Perfect Calibration: confidence = accuracy; ECE: the lower the better.

sequent works find that this approach not necessarily improves training (Grnarova et al., 2017; Tran et al., 2019; Doan et al., 2019). This is because Unrolled-GAN unrolls its discriminator, which has a significant amount of parameters. Consequently, the unrolling algorithm operates on a very large space, rendering the stochastic gradients that are used for updating the discriminator considerably noisy. In SALT, the unrolling space is the sample embedding space, the dimension of which is much smaller than the unrolling space of GANs. Therefore, unrolling is more effective for NLP tasks.

◇ **Robustness to the number of unrolling steps** From Figure 2a, we can see that SALT is robust to the number of unrolling steps. As such, setting the unrolling steps $K = 1$ or 2 suffices to build models that generalize well.

◇ **Robustness to the perturbation strength** Unrolling is robust to the perturbation strength within a wide range, as indicated in Figure 2b. Meanwhile, the performance of VAT consistently drops when we increase ϵ from 0.01 to 0.5. This indicates that the unrolling algorithm can withstand stronger perturbations than conventional approaches.

◇ ℓ_2 **constraints vs.** ℓ_∞ **constraints** Figure 2c illustrates model performance with respect to different perturbation strength in the ℓ_∞ case. Notice that in comparison with the ℓ_2 case (Figure 2b), SALT achieves the same level of performance, but the behavior of VAT is unstable. Additionally, SALT is stable within a wider range of perturbation strength in the ℓ_2 than in the ℓ_∞ case, which is the reason that we adopt ℓ_2 constraints in the experiments.

◇ **Classification Model Calibration** Recent studies show that adversarial training also helps model calibration (Stutz et al., 2020). A well-calibrated

model should produce reliable confidence estimation (i.e., confidence $\simeq$ actual accuracy), where the confidence is defined as the maximum output probability calculated by the model. We evaluate the calibration performance of $\text{BERT}_{\text{BASE}}$, SMART, and SALT by the Expected Calibration Error (ECE, Niculescu-Mizil and Caruana 2005). We plot the reliability diagram (confidence vs. accuracy) on the SST task in Figure 3 (See Appendix B for details about metrics). As we can see, $\text{BERT}_{\text{BASE}}$ and SMART are more likely to make overconfident predictions. SALT reduces the ECE error, and the corresponding reliability diagram aligns better with the perfect calibration curve.

5 Conclusion

We propose SALT, an adversarial training framework that employs a Stackelberg game formulation. Such a formulation induces a competition between a leader (the model) and a follower (the adversary). In SALT, the leader is in an advantageous position by recognizing the follower's strategy, and this strategic information is captured by the Stackelberg gradient. We compute the Stackelberg gradient, and hence find the equilibrium of the Stackelberg game, using an unrolled optimization approach. Empirical results on the NMT datasets and the GLUE benchmark suggest the superiority of SALT to existing adversarial training methods.

References

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, and Danilo Giampiccolo. 2006. The second PASCAL recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*.

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. 2009. The fifth pascal recognizing textual entailment challenge. In *In Proc Text Analysis Conference (TAC'09*.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14.

Yong Cheng, Lu Jiang, and Wolfgang Macherey. 2019. Robust neural machine translation with doubly adversarial inputs. *arXiv preprint arXiv:1906.02443*.

Yong Cheng, Zhaopeng Tu, Fandong Meng, Junjie Zhai, and Yang Liu. 2018. Towards robust neural machine translation. *arXiv preprint arXiv:1805.06130*.

Kevin Clark, Minh-Thang Luong, Christopher D Manning, and Quoc V Le. 2018. Semi-supervised sequence modeling with cross-view training. *arXiv preprint arXiv:1809.08370*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The pascal recognising textual entailment challenge. In *Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*, MLCW'05, pages 177–190, Berlin, Heidelberg. Springer-Verlag.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Thang Doan, Joao Monteiro, Isabela Albuquerque, Bogdan Mazoure, Audrey Durand, Joelle Pineau, and R Devon Hjelm. 2019. On-line adaptative curriculum learning for gans. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3470–3477.

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 1–9, Prague. Association for Computational Linguistics.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014a. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014b. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. 2019. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*.

Paulina Grnarova, Kfir Y Levy, Aurelien Lucchi, Thomas Hofmann, and Andreas Krause. 2017. An online learning approach to generative adversarial networks. *arXiv preprint arXiv:1706.03269*.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*.

Shujian Huang, Jun Xie, Xinyu Dai, CHEN Jiajun, et al. 2020. A reinforced generation of adversarial examples for neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3486–3497.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2019. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv:1911.03437*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lingkai Kong, Haoming Jiang, Yuchen Zhuang, Jie Lyu, Tuo Zhao, and Chao Zhang. 2020. Calibrated language model fine-tuning for in-and out-of-distribution data. *arXiv preprint arXiv:2010.11506*.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*.

Xiaodong Liu, Hao Cheng, Pengcheng He, Weizhu Chen, Yu Wang, Hoifung Poon, and Jianfeng Gao. 2020a. Adversarial training for large neural language models. *arXiv preprint arXiv:2004.08994*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019a. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*.

Xiaodong Liu, Yu Wang, Jianshu Ji, Hao Cheng, Xueyun Zhu, Emmanuel Awa, Pengcheng He, Weizhu Chen, Hoifung Poon, Guihong Cao, and Jianfeng Gao. 2020b. The Microsoft toolkit of multi-task deep neural networks for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. 2015. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. 2016. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*.

Yifei Min, Lin Chen, and Amin Karbasi. 2020. The curious case of adversarially robust models: More data can help, double descend, or hurt generalization. *arXiv preprint arXiv:2002.11080*.

Takeru Miyato, Andrew M Dai, and Ian Goodfellow. 2016. Adversarial training methods for semi-supervised text classification. *arXiv preprint arXiv:1605.07725*.

Mahdi Pakdaman Naeini, Gregory F Cooper, and Milos Hauskrecht. 2015. Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence*, volume 2015, page 2901. NIH Public Access.

Alexandru Niculescu-Mizil and Rich Caruana. 2005. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. *arXiv preprint arXiv:1806.00187*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037.

Barak A Pearlmutter and Jeffrey Mark Siskind. 2008. Reverse-mode ad in a functional framework:

Lambda the ultimate backpropagator. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 30(2):1–36.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John Duchi, and Percy Liang. 2020. Understanding and mitigating the tradeoff between robustness and accuracy. *arXiv preprint arXiv:2002.10716*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Motoki Sano, Jun Suzuki, and Shun Kiyono. 2019. Effective adversarial regularization for neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 204–210.

Motoki Sato, Jun Suzuki, Hiroyuki Shindo, and Yuji Matsumoto. 2018. Interpretable adversarial perturbation in input embedding space for text. *arXiv preprint arXiv:1805.02917*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. 2019. Adversarial training for free! In *Advances in Neural Information Processing Systems*, pages 3358–3369.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

David Stutz, Matthias Hein, and Bernt Schiele. 2019. Disentangling adversarial robustness and generalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6976–6987.

David Stutz, Matthias Hein, and Bernt Schiele. 2020. Confidence-calibrated adversarial training: Generalizing to unseen attacks. In *International Conference on Machine Learning*.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Ngoc-Trung Tran, Viet-Hung Tran, Bao-Ngoc Nguyen, Linxiao Yang, and Ngai-Man Man Cheung. 2019. Self-supervised gan: Analysis and improvement with multi-class minimax game. *Advances in Neural Information Processing Systems*, 32:13253–13264.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Heinrich Von Stackelberg. 2010. *Market structure and equilibrium*. Springer Science & Business Media.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Dilin Wang, Chengyue Gong, and Qiang Liu. 2019. Improving neural language modeling via adversarial training. *arXiv preprint arXiv:1906.03805*.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Lijun Wu, Yingce Xia, Fei Tian, Li Zhao, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2018. Adversarial neural machine translation. In *Asian Conference on Machine Learning*, pages 534–549. PMLR.

Zhen Yang, Wei Chen, Feng Wang, and Bo Xu. 2017. Improving neural machine translation with conditional sequence generative adversarial nets. *arXiv preprint arXiv:1703.04887*.

Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Thomas Goldstein, and Jingjing Liu. 2019. Freelb: Enhanced adversarial training for language understanding. *arXiv preprint arXiv:1909.11764*.

A Virtual Adversarial Training

Virtual adversarial training (VAT, Miyato et al. 2016) solves the following min-max optimization problem:

$$\min_{\theta} \mathcal{F}(\theta, \delta^*) = \mathcal{L}(\theta) + \frac{\alpha}{n}\sum_{i=1}^{n} \ell_v(x_i, \delta_i^*, \theta),$$
$$\delta_i^* = \underset{\|\delta_i\| \le \epsilon}{\operatorname{argmax}}\, \ell_v(x_i, \delta_i, \theta),$$

where

$$\ell_v(x_i, \delta_i, \theta) = \mathrm{KL}\big(f(x_i, \theta) \,||\, f(x_i + \delta_i, \theta)\big).$$

Note that the objective of the minimization problem is a function of both the model parameters and the perturbations.

Because the min problem and the max problem are operating on the same loss function, i.e., the min problem seeks to minimize ℓ_v, while the max problem tries to maximize ℓ_v, this min-max optimization is essentially a zero-sum game. And we can find the game's equilibrium using gradient descent/ascent algorithms.

Specifically, the adversarial player first generate an initial perturbation δ^0, and then refines it using K steps of projected gradient ascent, i.e.,

$$\delta^k = \Pi_{\|\cdot\| \le \epsilon}\left(\delta^{k-1} + \eta \frac{\partial \ell_v(x, \delta^{k-1}, \theta)}{\partial \delta^{k-1}}\right),$$
$$\text{for } k = 1, \cdots, K.$$

Here Π denotes projection onto the ℓ_2-ball or the ℓ_∞-ball. Empirically, we find that these two choices yield very similar performance, although adversarial training models is robust to ϵ within a wider range when applying the ℓ_2 constraint.

After obtaining the K-step refined perturbation δ^K, we use gradient descent to update the model parameters θ. Concretely, the gradient of the model parameters is computed as

$$\frac{\partial \mathcal{F}(\theta, \delta^K)}{\partial \theta} = \frac{\mathrm{d}\ell(f(x_i, \theta), y_i)}{\mathrm{d}\theta} + \alpha \frac{\partial \ell_v(x, \delta^K, \theta)}{\partial \theta}. \quad (7)$$

The training algorithm is demonstrated in Algorithm 2.

Note that in this paper, we target for models' generalization performance on the unperturbed test data, therefore we do not want a strong adversary that "traps" the model parameters to a bad local optima. Most of the existing algorithms achieve this goal by carefully tuning the hyper-parameters ϵ and K, i.e., a small ϵ usually generates weaker adversaries, so does a small K. However, these heuristics do not work well, and at times δ^K is too strong. Consequently, conventional adversarial training results in undesirable underfitting on the clean data.

Algorithm 2: Virtual Adversarial Training.

Input: $\mathcal{D}$: dataset; T: total number of training iterations; K: number of inner training iterations; η: step size to update δ; Optimizer: optimizer to update θ.

Initialize: model parameters θ;

for $t = 1, \cdots, T$ **do**
  for $(x, y) \in \mathcal{D}$ **do**
    Initialize $\delta^0 \sim \mathcal{N}(0, I)$;
    for $k = 1, \cdots, K$ **do**
      $g^k \leftarrow \partial \ell_v(x_i, \delta_i, \theta)/\partial \delta_i$;
      $\delta^k \leftarrow \Pi(\delta^{k-1} + \eta g^k)$;
    Compute the gradient g_θ using Eq. 7;
    $\theta \leftarrow \text{Optimizer}(g_\theta)$;

Output: θ

B Model Calibration

Many applications require trustworthy predictions that need to be not only accurate but also well calibrated (Kong et al., 2020). A well-calibrated model is expected to output prediction confidence comparable to its classification accuracy. For example, given 100 data points with their prediction confidence 0.6, we expect 60 of them to be correctly classified. More precisely, for a data point X, we denote by $Y(X)$ the ground truth label, $\widehat{Y}(X)$ the label predicted by the model, and $\widehat{P}(X)$ the output probability associated with the predicted label. The calibration error of the predictive model for a given confidence $p \in (0, 1)$ is defined as:

$$\mathcal{E}_p = \left|\mathbb{P}\left[\widehat{Y}(X) = Y(X) | \widehat{P}(X) = p\right] - p\right|. \quad (8)$$

Since Eq. 8 involves population quantities, we usually adopt empirical approximations (Guo et al., 2017) to estimate the calibration error. Specifically, we partition all data points into 10 bins of equal size according to their prediction confidence. Let $\mathcal{B}_m$ denote the bin with prediction confidence bounded between ℓ_m and u_m. Then, for any $p \in [\ell_m, u_m)$, we define the empirical calibration error as:

$$\widehat{\mathcal{E}}_p = \widehat{\mathcal{E}}_m = \frac{1}{|\mathcal{B}_m|}\Big|\sum_{i \in \mathcal{B}_m}\left[\mathbf{1}(\widehat{y}_i = y_i) - \widehat{p}_i\right]\Big|, \quad (9)$$

Corpus	Task	#Train	#Dev	#Test	#Label	Metrics
Single-Sentence Classification (GLUE)						
CoLA	Acceptability	8.5k	1k	1k	2	Matthews corr
SST	Sentiment	67k	872	1.8k	2	Accuracy
Pairwise Text Classification (GLUE)						
MNLI	NLI	393k	20k	20k	3	Accuracy
RTE	NLI	2.5k	276	3k	2	Accuracy
QQP	Paraphrase	364k	40k	391k	2	Accuracy/F1
MRPC	Paraphrase	3.7k	408	1.7k	2	Accuracy/F1
QNLI	QA/NLI	108k	5.7k	5.7k	2	Accuracy
Text Similarity (GLUE)						
STS-B	Similarity	7k	1.5k	1.4k	1	Pearson/Spearman corr

Table 6: Summary of the GLUE benchmark.

where y_i, $\widehat{y}_i$ and $\widehat{p}_i$ are the true label, predicted label and confidence for sample i.

Reliability Diagram is a bar plot that compares $\widehat{\mathcal{E}}_p$ against each bin, i.e., p. A perfectly calibrated would have $\widehat{\mathcal{E}}_p = (\ell_m + u_m)/2$ for each bin.

Expected Calibration Error (ECE) is the weighted average of the calibration errors of all bins (Naeini et al., 2015) defined as:

$$\mathrm{ECE} = \sum_{m=1}^{M} \frac{|\mathcal{B}_m|}{n} \widehat{\mathcal{E}}_m, \tag{10}$$

where n is the sample size.

We remark that the goal of calibration is to minimize the calibration error without significantly sacrificing prediction accuracy. Otherwise, a random guess classifier can achieve zero calibration error.

C GLUE Benchmark

Details of the GLUE benchmark, including tasks, statistics, and evaluation metrics, are summarized in Table 6.