*ECAI 2023*
*K. Gal et al. (Eds.)*

*doi:10.3233/FAIA230619*

# Revisiting the Efficiency-Accuracy Tradeoff in Adapting Transformer Models via Adversarial Fine-Tuning

**Minjia Zhang***, **Niranjan Uma Naresh and Yuxiong He**

Microsoft Corporation

**Abstract.** Adversarial fine-tuning (i.e., training on adversarial perturbed inputs) has demonstrated promising results in improving the accuracy of natural language understanding tasks. However, the improved accuracy does not come for free but is accompanied by a significantly prolonged training time, limiting their applicability to larger and more complex models. This work revisits the efficiency-accuracy trade-off in adversarial fine-tuning by systematically analyzing if adversarial fine-tuning methods, in conjunction with several efficiency optimizations, are suitable for adapting pre-trained Transformer models for natural language understanding tasks. Our results show that multiple design choices are crucial in determining the efficiency-accuracy trade-off, and we introduce a method, ScaLA, that achieves better accuracy-vs-speed trade-off than prior methods. We show in experiments that our proposed method attains up to 14.7× adaptation speedups on BERT, RoBERTa, and T5, while achieving comparable accuracy to existing methods.

## 1 Introduction

Pre-trained transformer-based language models (LM), such as BERT [6], RoBERTa [25], T5 [29], and GPT-3 [1] have achieved remarkable success. While these models provide a powerful general-purpose engine for processing language information, adapting them before use is necessary for many natural language understanding tasks[33]. This is because the pre-training objective used for many large LMs — predicting masked tokens in a sentence – is different from the downstream task objectives, e.g., question-answering tasks extract answers from a text based on questions posed by humans, which requires model adaptation. This adaptation is usually done by fine-tuning the parameters of a pre-trained model with domain-specific data.

The way adaptation is performed has a significant impact on the accuracy of downstream tasks. Among different approaches, adversarial fine-tuning (AF) adds adversarial perturbations to the training data throughout the fine-tuning procedure and has demonstrated promising improvements in fine-tuning accuracy [18, 42]. For instance, [18] (referred to as SMART) adopts the smoothness-inducing adversarial regularization technique [3, 34], which introduces norm-bounded perturbations to the lexical embeddings of token inputs. The model is then fine-tuned with these perturbed inputs and is encouraged to not let its output change drastically with those perturbations. Adversarial fine-tuning methods such as SMART have achieved state-of-the-art accuracy on several NLP tasks such as GLUE.

Although achieving great accuracy, existing adversarial fine-tuning methods have several limitations. (1) They solely focus on assessing the impact of generalization of adversarial fine-tuning, without considering the computational cost, which can be prohibitive when applying those methods to larger and more complex models. (2) Existing methods also mainly focus on small batch size, overlooking the scalability of their approach on large batches and scalability when more compute units are available. Increasing the batch size allows adversarial fine-tuning to achieve higher compute efficiency but may lead to sub-optimal accuracy. This results in a trade-off between accuracy and efficiency. (3) Existing work lacks theoretical analysis and its impact on the loss landscape. In this work, we address those challenges and introduce our lightweight and scalable method called ScaLA, while also providing theoretical and loss surface curvature analysis.

In summary, our contributions are as follows:

- We systematically assess how different training strategies affect the computational efficiency and generalization of fine-tuning Transformers. Our study reveals the computation-vs-accuracy dilemma in fine-tuning pre-trained transformers via adversarial fine-tuning and provides insights for optimizations (Sec. 4).
- We present ScaLA, a method that injects lightweight adversarial noise as well as groupwise adaptive learning rate to speed up the fine-tuning of pre-trained transformer models. Our design is based on careful analysis of factors that have major impacts on model accuracy and training speed of projected gradient descent-based adversarial training, especially under the multi-GPU fine-tuning regime (Sec. 5).
- We theoretically quantify the convergence rate of adversarial fine-tuning, which provides theoretical justification for our method (Appendix A).
- We conduct extensive evaluation, and our results show that ScaLA accelerates the fine-tuning of pre-trained Transformer-networks by up to 14.7 times over the baseline on BERT [6], RoBERTa [25], and T5 [29] on a wide range of natural language understanding (NLU) tasks.

## 2 Background and Related Work

The transformer network was originally proposed by [32] for neural machine translation, which shows its superiority in parallel computation and modeling long-range dependencies, compared to recurrent neural networks such as LSTM [13]. Subsequently, Devlin et al. propose a bidirectional transformer-based language model called BERT [6], which leverages transfer learning and has been demonstrated as an effective strategy for language representation learn-

* Corresponding Author. Email: minjiaz@microsoft.com.

*Please check ArXiv or contact the authors for any appendices or supplementary material mentioned in the paper.*

ing [25, 36, 22, 31]. The transfer learning paradigm consists of two major stages: a pre-training stage that involves training the network on a large corpus followed by a second fine-tuning stage to adapt the pre-trained model to different target tasks/domains, also known as domain adaptation.

There has been a lot of interest in improving domain adaptation. Numerous works suggest that a better pre-trained model helps improve the accuracy of downstream tasks [6, 25, 29, 12, 31, 1]. However, pre-training is expensive and time-consuming. Meanwhile, there has also been an enormous amount of interest in designing effective fine-tuning methods. Among different methods, adversarial training has shown promising results in improving the fine-tuning accuracy of pre-trained Transformers. Adversarial training has been studied extensively in the computer vision literature for improving the robustness against adversarial attacks [8, 26]. Subsequently, studies show that adversarial training can help improve the generalizability of language modeling [3, 34]. More recently, several studies, such as SMART [18] and FreeLb [42], show that adversarial training helps improve the generalization of fine-tuning of pre-trained Transformers. However, while many works focus on assessing the impact of generalization of adversarial training, few studies have examined the trade-off between speed and accuracy in adversarial fine-tuning. This work studies this trade-off and proposes methods that achieve better trade-offs in comparison to existing ones.

As the size of Transformer models has grown exponentially, there has been a lot of interest in developing efficient adaptation methods for pre-trained Transformers [15, 35, 16, 11]. Some methods insert so-called adapters to pre-trained model and only fine-tune the adapters while freezing the other weights [15]. [16] adds low-rank matrices to approximate parameter updates. [28] shows that it is possible to quickly adapt to new tasks by collectively learning knowledge from multiple tasks. These parameter-efficient methods focus on reducing memory consumption and trainable parameters in fine-tuning, which is complementary to the efficient adversarial fine-tuning introduced in this work.

## 3 Problem Formulation and Setup

**Formulation.** Let $\mathbb{X}$ denote the parameter space and $\mathbb{Y}$ denote the data (mini-batch/sample) space and $Q$ denote a distribution supported on $\mathbb{Y}$. To improve the generalizability of transformer fine-tuning while retaining the scalability, we augment the usual stochastic optimization objective by constructing an adversarial [20, 26] regularization. In particular, we solve the following robust optimization problem, which is a stochastic minimax [24] optimization problem augmented with a regularization term involving a deterministic adversarial perturbation, instead of vanilla risk minimization:

$$\begin{aligned}\min_{x\in\mathbb{X}} \mathbb{E}_{\xi\sim Q}[g(x,\xi)] &= \min_{x\in\mathbb{X}} \mathbb{E}_{\xi\sim Q}[\underline{f}(x,\xi) + \lambda \underline{r}(x)] \\ &= \min_{x\in\mathbb{X}} \max_{y\in\mathbb{Y}} \mathbb{E}_{\xi\sim Q}[\underline{f}(x,\xi) + \lambda r(x,y)] \quad (1) \\ &:= \min_{x\in\mathbb{X}} \max_{y\in\mathbb{Y}} \mathbb{E}_{\xi\sim Q}[f(x,y,\xi)] \quad (2)\end{aligned}$$

where $g : \mathbb{X}\times\mathbb{Y}\to\mathbb{R}$ denotes the adversarial training objective, $\underline{f} : \mathbb{X}\times\mathbb{Y}\to\mathbb{R}$ denotes the standard training objective, $f : \mathbb{X}\times\mathbb{Y}\times\mathbb{Y}\to\mathbb{R}$ denotes the augmented objective, $\underline{r} : \mathbb{X}\to\mathbb{R}$ denotes a deterministic regularization term on the parameters controlled by a strength factor $\lambda \in (0,\infty)$, $r : \mathbb{X}\to\mathbb{R}$ denotes the augmented regularization and $\xi$ denotes samples drawn from $Q$ (for simplicity, we slightly abuse the notation in using $\xi$ to denote the random variable, e.g. $\mathbb{E}_\xi[g(x,\xi)]$, or its empirical realizations, e.g. $\frac{1}{K}\sum_{k=1}^{K} g(x,\xi_k)$ for any $K$; the meaning is clear from the context). The overall (outer) training objective involves a minimization problem in the parameter space while being stochastic with respect to the data space. The adversarial regularization (inner) term is a deterministic maximization problem operating in the data space conditioned on a fixed parameter configuration. We wish to emphasize that this formulation is a two-player sequential [19], not simultaneous, game wherein the goal is to optimize a transformer network that is robust to adversarial perturbation. In a given round, the first player (associated with the outer minimization) proposes a parameter configuration, and the second player (associated with the inner maximization) responds with a penalty to capture the effect of label errors due to perturbations in a large data batch size to undermine the performance of the transformer parameter configuration chosen by the first player. Specifically, for any given outer step $t$, let $x_t$ denote the parameter proposed by the first player. Since the exact inner maximization in Equation (2) is intractable for non-convex models such as transformers, we adopt truncated methods as in prior works. Specifically, we use Projected Gradient Ascent (PGA) [26, 18] to solve this problem, i.e., $y_{\tau+1} = \Pi_\omega(y_\tau + \rho_\tau \nabla_y r(x_t, y))$ where $\rho_\tau$ for $\tau \in [\mathcal{T}]$ is the step size sequence and $\Pi$ projects the result of the gradient ascent update into an $\ell_\infty$ ball of diameter $2\omega$ around the original input embeddings, $\xi$, considered by the first player.

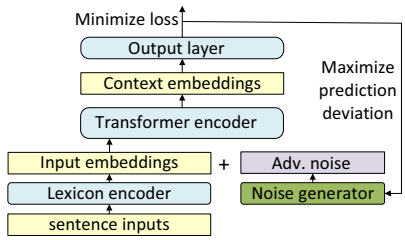



**Figure 1**: The overall setup for experiments.

**Setup.** Language expressions are quite sensitive to individual words or clauses, where perturbations against those would likely generate incorrect or biased training data with wrong labels [41]. Following prior success in applying adversarial training to NLP models [27, 42], we apply perturbations to the continuous word embeddings instead of directly to discrete words or tokens. The term $\underline{r}$ captures the prediction deviation from the perturbation. In a given round of the game, with respect to the first player's proposal, let $\Phi$ denote the transformer network under consideration (specifically, $\Phi$ is BERT in this paper) and $\xi$ be a large batch of data sampled from $Q$. We construct a label for the second player as $\gamma := \Phi(x,\xi)$. Next, for classification tasks, we choose $r$ to be the symmetric KL divergence [18], i.e., $r(x,y) := \mathrm{KL}_{\mathrm{sym}}(\gamma, \Phi(x,y))$. We follow [18] to use symmetric KL divergence to measure the distributional divergence to generate adversarial perturbation. For regression tasks, we choose $r$ to be the squared loss, i.e., $r(x,y) := (\gamma - \Phi(x,y))^2$. In practice, we add an $\ell_\infty$ constraint on $y$, which is achieved by simple clipping with a radius of $\omega$ (projection). Intuitively, a large $r$ corresponds to a situation wherein the transformer is highly sensitive to a given perturbation in the input, suggesting that the model parameters are close to a sharp minimum. Augmenting the original training objective with $r$ makes the first player incur an additional penalty if the outer minimization solution veers closer to sharp minima, thereby encouraging flatter solutions and better generalizability. Figure 1 shows the overall setup we used for experiments in this work.

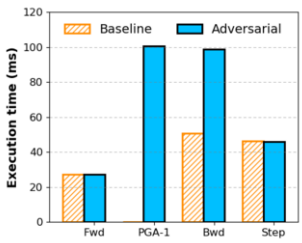



**Figure 2**: Time breakdown without and with PGA-1.

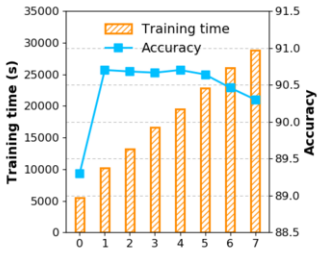



**Figure 3**: Impact of perturbation steps.

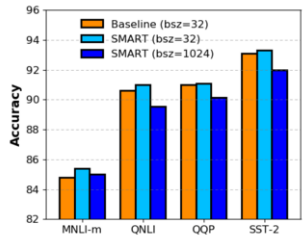



**Figure 4**: Accuracy improvements with adversarial fine-tuning varying batch sizes.

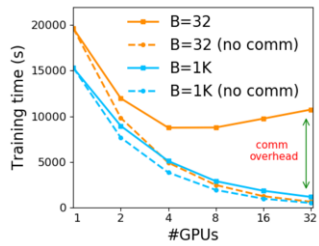



**Figure 5**: Scalability of adversarial fine-tuning varying the number of compute units.

## 4 Performance Characterization

Existing adversarial fine-tuning methods improve generalization but at what cost? To answer that question, we perform studies to analyze how the training computation efficiency and model accuracy are affected by adversarial training during the fine-tuning stage, using SMART [18] and pre-trained BERT$_{base}$ model on GLUE as an example. The hardware and software setup is described in detail in the evaluation section.

**Adding adversarial perturbation improves generalization but adds significant computational cost.** The generation of adversarial noise requires an extra PGA inner loop that standard training does not have. This slows down training by at least 2 times, even with a single PGA iteration (i.e.,$\mathcal{T}$=1). Figure 2 provides the time breakdown of optimization using PGA with $\mathcal{T} = 1$ (denoted as PGA-1). PGA-1 performs the perturbation and takes approximately the same time as making three forward passes (Fwd) through the network, i.e., one step of PGA requires making one forward and backward pass (Bwd) over the entire network. The backward pass of the optimization takes roughly twice the amount of time as the standard backward step because the back-propagation is triggered twice to calculate the noise and the gradients. This high overhead motivates the investigation of more lightweight approaches to generate adversaries while retaining their benefits of improving generalization.

**Inner maximization suffers from diminishing returns.** Inner maximization in PGA often requires multiple iterations (i.e., $K$) to find a good solution (i.e., adversary) [26, 42], adding significant computational overhead. To investigate the usefulness of inner maximization, we progressively increase the perturbation with various strengths (i.e., varying $\mathcal{T}$) Figure 3 shows that although using a large $\mathcal{T}$ helps to produce stronger noises, it has a diminishing return in final accuracy after one iteration. In fact, too many perturbations steps may even have a negative impact on the model accuracy, despite the fact that the training overhead still increases almost linearly. Given that a frequently encountered scenario in practical problems is the need of building accurate models under a limited computational budget, this raises the question of whether existing adversarial fine-tuning methods perform well under limited computational budgets.

**Existing methods for adversarial fine-tuning use small batch sizes, leading to sub-optimal efficiency and limited scalability.** Given the expensive computational cost of adversarial fine-tuning, one idea is to parallelize it with multiple compute units, where each unit processes a chunk of the adversarial inputs. However, existing methods mainly tried a few methods with small batch sizes. A small batch size has a profound implication on training efficiency and scalability. Modern GPUs have massive parallel compute units and are highly optimized for throughput. When the batch size is small, most GPU cores will stay idle or spend most of their time communicating with each other. To show this, we carry out a scalability test by varying the number of GPUs from 1 to $P$ (e.g., 32), with multi-node multi-GPU setups. Different from pre-training, the adversarial fine-tuning stage employs a much smaller batch size (e.g., 16, 32) than pre-training (e.g., 4096) [6, 25]. We choose a batch size $m$ (e.g., 32), by following the suggestion from existing literature [6, 25, 18], and we divide the samples in the mini-batch among $P$ GPUs. If the per-worker batch size (e.g., 16) is larger than the maximum admissible per-worker batch size (e.g., 8), we use local gradient accumulation [9] to avoid running out of memory. Figure 5 shows the scalability results. The training time decreases initially when P increases, but it quickly plateaus and even decreases with more GPUs. This is mainly caused by the small batch size, where the system spends more time communicating gradients than doing actual computation. As such, the communication overhead dominates the total execution time (e.g.,B=32 vs. B=32 (no comm)). The communication overhead is especially huge when there is cross-machine communication (e.g., from 16 to 32). Therefore, using adversarial fine-tuning on multi-node multi-GPU is inefficient.

Increasing the batch size can decrease training time, especially as the number of GPUs increases because an increased batch size increases the compute resource utilization and the computation-vs-communication ratio. However, this also leads to a large accuracy loss, even worse than the baseline accuracy without using adversarial fine-tuning, as shown in Fig.4. This problem is related to the "generalization gap" associated with large batch training [20, 14]. As such, the question remains whether adversarial fine-tuning can benefit from multi-node multi-GPU training.

## 5 Methods

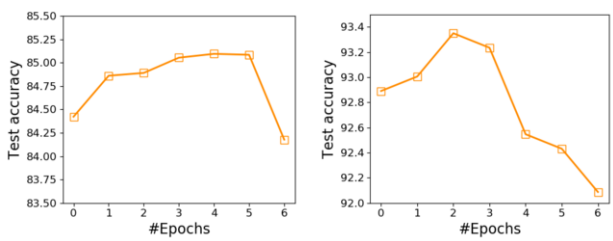
The design principle of ScaLA is to perform adversarial fine-tuning in a simple and computationally efficient way without sacrificing accuracy. This section describes three optimizations to make adversarial fine-tuning more efficient and scalable.

**Simplified inner maximization.** Existing works on adversarial fine-tuning [18] conjectured that $K > 1$ in PGA-K was necessary in order to find good adversaries for improving the fine-tuning accuracy, leading to huge computational costs. [42] tackles this issue by reusing the projected gradient results from PGA-K across multiple mini-batches. Contrary to these ideas, we instead use a simplified strategy where we only perform a one-shot best-effort perturbation. We show this simplification (referred to as PGA-1) preserves model quality, reducing adversarial fine-tuning costs significantly.

From an optimization perspective, the model has two components, the parameter space and data space. First, unlike the minimization in the parameter space, which is stochastic, the maximization in the data space is deterministic. Second, with respect to the testing phase,

the numerical convergence in the model's parameter space is of primary importance rather than the numerical convergence in the data space, i.e., the maximization is an auxiliary procedure that augments the training phase to make the parameter space "aware" of effects of the batch size across epochs. Due to these two points, we hypothesize that for a given batch, the marginal utility of an additional PGA step is low, and we are able to get away with inexact deterministic maximization.

**Delayed adversary injection:** Even PGA-1 still adds an overhead factor of 2.2, so we investigate how useful adversarial perturbations are at different stages of fine-tuning. We conduct additional experiments to measure the final accuracy corresponding to starting from a regular fine-tuning and then enabling PGA-1 for $t \geq t_s$ where $t_s \in [T]$. Experiment results show that enabling PGA-1 from the beginning does not offer much improvement in accuracy, whereas adversaries become more potent as the model begins to stabilize toward the end of training. Intuitively, this makes sense because generally, the model is primarily doing exploration to the loss landscape at initialization and is less likely to benefit from perturbed inputs. When adversaries are added towards the end of the training, the model is more likely to remember those adversaries that help minimize the adversarial loss in Eqn. 2. This phenomenon has also been observed in computer vision tasks [2, 38, 10]. We show that it is possible to delay the injection of adversaries for adversarial fine-tuning of transformers for NLP tasks. Following prior studies [10], we set $t_s$ either as a hyperparameter or choose a $t_s$ when the training loss flattens.




(a) MNLI-m (b) SST-2

**Figure 6**: Accuracy results from delaying the injection of adversaries at different stages.

**Outer Minimization via layer-wise adaptive scaling:** The main obstacle to scaling up batch sizes for improved compute efficiency is the instability from training with larger learning rates. Prior work suggests that adaptive learning rates [39] are suitable for solving the instability issue from large-batch training and improving its convergence quality. However, few works have studied its interplay with adversarial training. Given that both could improve model accuracy, it poses the question: to what extent does adversarial fine-tuning benefit from layer-wise adaptive learning rate scaling? To investigate that, we solve the minimization problem in Equation (2) by $x_{t+1}^i = x_t^i - \eta_t \nu(\|x_t^i\|)\widehat{\nabla}_x^i g(x)/\|\widehat{\nabla}_x^i g(x)\|$, $\forall i \in [h]$ where $i$ denotes the $i^{\text{th}}$-layer of the transformer. The normalized gradient descent mitigates issues due to exploding gradients. The learning rate sequence $\eta_t$, $\forall t \in [T]$ is scaled by a clipping function $\nu(c) := \max(\mathcal{L}, \min(c, \mathcal{U}))$ where $\mathcal{L} < \mathcal{U}$, which ensures the norm of the update is of the same order as that of the weights. The normalization and scaling are done groupwise (every layer forms a group), hence the adaptive layer-wise learning rates. Note that we use gradient averaging on $\xi$, i.e., gradient accumulation and all-reduce, over a batch size $B$ distributed across $P$ workers in order to obtain a noisy gradient estimate $\widehat{\nabla}_x^i g(x)$ at epoch $t$.

**Putting it together.** Combining the formulation with the above investigations, we propose ScaLA, with its full procedure provided in Algorithm 1. Meanwhile, we also provide a theoretical analysis of its convergence rate in Theorem 5.1.

**Algorithm 1 ScaLA**

**Input:** Epochs $T$, delay $t_s$, perturbation (inner) step size $\rho$, clipping radius $\omega$, regularization strength $\lambda$, (outer) learning rate $\eta$
**Output**: $h$-layer transformer model $\Phi$ with converged robust parameters $\overline{x} := x_T$
**for** $t \in [T]$ **do** ▷ Loop through epochs
  **for** worker $p \in [P]$ **do** ▷ In parallel across homogeneous workers
    **for** mini-batch $\xi_p \sim Q$ **do** ▷ Subsample $\frac{B}{P}$ data instances on each worker
      $\underline{r}(x_t) \leftarrow 0$, $\gamma \leftarrow \Phi(x, \xi_p)$, select $y_0$ ▷ Initialize regularization and label
      **if** $t \geq t_s$ **then** ▷ Check delay condition
        $y_1 \leftarrow \Pi_\omega(y_0 + \rho \nabla_y r(x_t, y))$ ▷ Perform adversarial perturbation with PGA-1
        $\underline{r}(x_t) \leftarrow \text{KL}_{\text{sym}}(\gamma, \Phi(x_{t-1}, y_1))$ ▷ Calculate the adversarial regularization
      **end if**
      $g(x_t, \xi_p) \leftarrow \underline{f}(x_{t-1}, \xi_p) + \lambda \underline{r}(x_t)$ ▷ Calculate the augmented loss
      $\nabla_x g(x_t, \xi_p) \leftarrow$ Backward pass on $\Phi$ ▷ Compute local gradients using accumulation
    **end for**
  **end for**
  $\widehat{\nabla}_x g(x_t) \leftarrow \frac{1}{B} \sum_{p=1}^{P} \nabla_x g(x_t, \xi_p)$ ▷ Gradient averaging using all-reduce
  $x_t^i \leftarrow x_{t-1}^i - \eta_t \nu(\|x_t^i\|) \frac{\widehat{\nabla}_x^i g(x_t)}{\|\widehat{\nabla}_x^i g(x_t)\|}, \forall i \in [h]$ ▷ Update model parameters
**end for**

**Theorem 5.1** (Complexity of Algorithm 1; Informal – Details in Appendix A)**.** *Consider the problem in Equation 2. Let $t_s = 0$. Setting the outer learning rate as $\eta = O\left(1/\sqrt{T}\right)$ and scaling batch size as $b = O(T)$, for Algorithm 1, we have $\mathbb{E}\left[\|\nabla g_{1/2\alpha}(\overline{x})\|^2\right] \leq O\left(\epsilon + \kappa_\alpha/\sqrt{T}\right)$ where $\overline{x}$ is the estimator obtained from running $T$ steps of Algorithm 1 and picking $x_t$ uniformly at random for $t \in [T]$. Here, $\epsilon$ is the error due to the approximate inner maximization oracle, $\alpha$ characterizes the smoothness of $f(x, .)$, $g_{1/2\alpha}$ is the Moreau-envelope of $g$ and $\kappa_\alpha = \max_i \alpha_i / \min_i \alpha_i$.*

## 6 Evaluation

We evaluate the effectiveness of ScaLA in adapting pre-trained transformer networks over a set of NLP tasks.

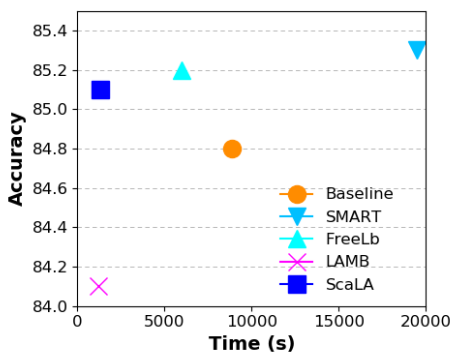
**Model/Dataset.** We study adaptation on pre-trained BERT$_{base}$, RoBERTa$_{large}$, and T5$_{base}$ hosted by HuggingFace, which are the top three most downloaded models by practitioners. We use SQuAD-v2 and the GLUE benchmark [33], which is a collection of sentence or sentence-pair natural language understanding tasks including question answering, sentiment analysis, and textual entailment. We exclude tasks in GLUE that have very small datasets (e.g.,CoLA, RTE).

**Hyperparameters.** For all configurations, we perform a grid search of learning rates in the range of {1e-5, 3e-5, 5e-5, 7e-5, 9e-5, 1e-4, 3e-4} for small batch sizes and {5.6e-5, 8e-5, 1e-4, 1.7e-4, 2.4e-4, 2.8e-4, 4e-4, 5.6e-4, 1e-3} for large batch sizes. We report the best-performing results on the validation datasets for all baselines and our method. For a fair comparison, we set the maximum number of epochs to 6 for all configs. We use a linear learning rate decay schedule with a warm-up ratio of 0.1. For ScaLA, we set $\lambda = 1$, perturbation clipping radius $\omega = 10^{-5}$, step size $\rho = 10^{-4}$, and $t_s$={3,5}.

**Hardware.** We conduct the evaluation using two nodes, where each node consists of 16 NVIDIA V100 GPUs.

## 6.1 Main Results

We compare the following schemes: (1) **Baseline:** This is the existing PyTorch implementation of Transformer fine-tuning from HuggingFace (HF). The multi-GPU training is implemented using DistributedDataParallel [23], (2) **SMART**: This is the adversarial fine-tuning method described in [18], (3) **FreeLB:**, this is the work described in [42] that allows multiple mini-batches to reuse gradients (with an accumulated batch size of 1K), (4) **LAMB:**, this configuration uses LAMB optimizer [39] for large-batch fine-tuning, and (4) **ScaLA:** This is our approach as described in Algorithm 1.




(a) BERT

**Figure 7**: Performance ($y$ axis) and speed ($x$ axis) of different methods. Top-left represents a better model quality with faster training speed.

Fig. 7 shows the trade-off between model accuracy and speed on MNLI fine-tuned on BERT. Among different methods, LAMB, together with large-batch optimization, achieves the fastest training time, but its accuracy is much worse than the best accuracy (e.g., > 1 point lower). SMART achieves the best accuracy (85.3), but it also incurs 16.5x longer training time than LAMB. While ScaLA does not achieve the highest accuracy, it achieves a significantly faster training speed than other adversarial fine-tuning methods, such as SMART and FreeLb, at the cost of slightly lower accuracy (e.g., <0.2 points).

**BERT.** Table 1 presents more results on MNLI, QNLI, QQP, and SST2. Overall, we observe that ScaLA often achieves a better trade-off in terms of speed and accuracy in comparison to other baseline methods. ScaLA achieves up to 14.7× speedups over SMART with 0.3 lower average accuracy on BERT (89.4 vs. 89.7). The speedups come from two aspects: (1) the lightweight adversarial fine-tuning incurs lower computation cost per training step while retaining the accuracy improvements; (2) the groupwise adaptive learning rate allows ScaLA to use a drastically much larger batch size for fine-tuning, which improves the hardware utilization. ScaLA is up to 4.5 times faster than FreeLb while achieving similar accuracy on BERT (89.4 vs. 89.5). ScaLA is faster than FreeLb because the PGA-based maximization performs multiple ascent steps to calculate adversaries across the full training process, which adds significant computation cost. In contrast, ScaLA reduces the computational complexity of adversarial large-batch optimization, allowing it to be more efficiently executed across multiple GPUs and bring end-to-end speedups. In our experiments, LAMB leads to only marginal accuracy improvements than the baseline and is 1.1 points lower than SMART. This is because LAMB is primarily designed for improving the training stability of pre-training with large-batch sizes. It allows large-batch optimization to achieve similar accuracy as its small-batch counterpart, but it does not seem it can help further improve fine-tuning accuracy.

**RoBERTa.** We also studied the accuracy vs. speed trade-off of adversarial fine-tuning on RoBERTa$_{large}$, which also uses an encoder transformer as its backbone architecture. Table 2 shows the accuracy vs. speed of different methods. The fine-tuning time increases significantly because RoBERTa$_{large}$ is 3.2× larger than BERT$_{base}$ (354M vs. 1110M). Despite the increased model scale, we observed similar accuracy vs. speed patterns. ScaLA achieves up to 11.8× speedup over SMART with 0.1 points lower average accuracy, and 4.5× speedup over FreeLb with 0.4 points lower average accuracy. These results indicate that the accuracy-speed trade-off in adversarial fine-tuning exists as we further scale the model size, and it is possible to achieve significant speedups while retaining model accuracy with ScaLA even as the model size increases.

**T5.** We evaluated our approach on T5, a representative encoder-decoder Transformer based model. We use T5$_{base}$ from the HuggingFace model zoo's checkpoint. We apply ScaLA to fine-tune pre-trained T5 on SQuAD-v2. Table 3 shows the comparison results. Overall, ScaLA achieves comparable EM/F1 scores score to FreeLb and SMART but with 1.5x and 3.6x faster speeds. These results confirm that our approach can bring speedups to encoder-decoder architecture.

## 6.2 Ablation study

We study the importance of components in ScaLA. We set $t_s$ to 0, which denotes as *w/o Delaying PGA-1*. We replace the outer minimization to use ADAM [21], which is noted as *w/o Groupwise LR*. We set $\lambda$ to 0, which denotes as *w/o PGA-1*. The results in Table 4 show that the removal of either design element would result in a performance drop. For example, removing PGA-1 leads to 0.8 points accuracy drop (88.6 vs. 89.4), indicating that adversarial noise is crucial for improving the generalizability of large-batch adaptation. Moreover, if we perform PGA-1 without delayed injection, the average accuracy increases by 0.1 points (89.5 vs. 89.4), but the execution time is increased by 1.5–1.9x, indicating the importance of having lightweight adversarial noise for speeding up the adaptation. Finally, removing group-wise learning rates leads to a small 0.2 points accuracy drop (89.2 vs. 89.4).

## 6.3 Loss surface curvature analysis

Prior work correlates the low generalization with positive curvature of large magnitude in the parameter space [20]. The indication is that a sharp local minimum also reflects a higher sensitivity of the loss even within the neighborhood of training data points and can attribute to the difficulty in generalization. To verify if the generalization gap

**Table 1**: The fine-tuning time vs. accuracy results on GLUE benchmark and BERT$_{base}$.

| BERT$_{\text{base}}$ | bsz | MNLI-m | | | QNLI | | | QQP | | | SST-2 | | | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Steps | Time | Acc. | Steps | Time | Acc. | Steps | Time | Acc/F1 | Steps | Time | Acc. | |
| Baseline | 32 | 73632 | 8848 | 84.8 | 19644 | 2408 | 90.6 | 68226 | 11311 | 91/88.0 | 12630 | 1494 | 93.1 | 89.4 |
| SMART [18] | 32 | 73632 | 19466 | 85.3 | 19644 | 5538 | 91.0 | 68226 | 24885 | 91.1/88.1 | 12630 | 3436 | 93.3 | 89.7 |
| FreeLb [42] | 1K | 2301 | 5953 | 85.2 | 615 | 1944 | 90.3 | 2133 | 19030 | 91.2/88.2 | 396 | 680 | 92.8 | 89.5 |
| LAMB [39] | 1K | 2301 | 1180 | 84.1 | 615 | 359 | 89.6 | 2133 | 2978 | 90.5/87.0 | 396 | 139 | 92.4 | 88.6 |
| ScaLA | 1K | 2301 | 1323 | 85.1 | 615 | 432 | 90.0 | 2133 | 4229 | 90.9/87.7 | 396 | 151 | 93.5 | 89.4 |

**Table 2**: The fine-tuning time vs. accuracy results on GLUE benchmark and RoBERTa$_{large}$.

| BERT$_{\text{base}}$ | bsz | MNLI-m | | | QNLI | | | QQP | | | SST-2 | | | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Steps | Time | Acc. | Steps | Time | Acc. | Steps | Time | Acc/F1 | Steps | Time | Acc. | |
| Baseline | 32 | 73632 | 18114 | 90.5 | 19644 | 4842 | 94.7 | 68226 | 16614 | 92.0/89.4 | 12630 | 3072 | 96.4 | 92.5 |
| SMART [18] | 32 | 73632 | 39850 | 90.9 | 19644 | 11136 | 95.1 | 68226 | 38212 | 92.5/90.0 | 12630 | 6758 | 96.6 | 93.0 |
| FreeLb [42] | 1K | 2301 | 15133 | 91.2 | 615 | 5256 | 95.2 | 2133 | 10818 | 92.5/90.0 | 396 | 1804 | 96.9 | 93.3 |
| LAMB [39] | 1K | 2301 | 2646 | 90.5 | 615 | 973 | 94.5 | 2133 | 1998 | 91.3/88.5 | 396 | 324 | 96.2 | 92.1 |
| ScaLA | 1K | 2301 | 3363 | 90.9 | 615 | 1168 | 95.1 | 2133 | 2404 | 92.3/89.8 | 396 | 401 | 96.7 | 92.9 |

**Table 4**: Ablation study of ScaLA using BERT$_{base}$ on GLUE tasks.

| | MNLI-m | | QNLI | | QQP | | SST-2 | | Avg. | Speedup |
|---|---|---|---|---|---|---|---|---|---|---|
| | Time | Acc. | Time | Acc. | Time | Acc/F1 | Time | Acc. | | |
| Baseline | 19635 | 84.8 | 5535 | 90.6 | 16494 | 91/88.0 | 2736 | 93.1 | 89.4 | 1 |
| ScaLA | 1323 | 85.1 | 432 | 90 | 4229 | 90.9/87.7 | 151 | 93.5 | 89.4 | 12.4 |
| w/o Delaying PGA-1 | 2503 | 85.2 | 726 | 90.2 | 6407 | 91.3/88.3 | 272 | 93.1 | 89.5 | 7.0 |
| w/o Groupwise LR | 1290 | 85.0 | 422 | 89.9 | 4212 | 90.7/87.6 | 146 | 93.0 | 89.2 | 12.7 |
| w/o PGA-1 | 1180 | 84.1 | 359 | 89.6 | 2978 | 90.5/87.0 | 139 | 92.4 | 88.6 | 14.3 |

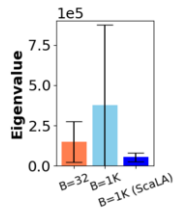



**Figure 8**: Eigenvalue results.

**Table 3**: Evaluation results of T5$_{base}$ on SQuAD-v2.

| T5/SQuAD-v2 | EM | F1 | Time |
|---|---|---|---|
| SMART | 78.9 | 82.4 | 28m |
| FreeLb | 79.0 | 82.6 | 66m |
| ScaLA | 78.9 | 82.5 | 18m |

during fine-tuning is actually caused by using large-batch sizes, we perform a Hessian-based curvature analysis.

We quantitatively measure the steepness of loss landscape by loading the checkpoint of a fine-tuned model and computing the curvature, i.e., the second derivative of the model, with respect to its parameters, for a fixed batch of samples. Inspired by [37], for a model $\Phi(x)$, we compute the largest eigenvalue of the model's Hessian, $L_{\max}[\nabla_x^2\Phi(x)]$, using the Hessian-vector product primitive and the power method. We use the largest eigenvalue as a measure of sharpness since the corresponding (top) eigenvector characterizes the direction of the largest change in gradient at a given point in the parameter space. From Figure 8, the largest eigenvalue of the model trained with a large batch (e.g., 1K) is much larger (e.g., 2.6x) than the small-batch baseline and with higher deviations (e.g., 3.9x). This result confirms that large-batch adaptation makes the loss landscape of the model more prone to ill-conditioning and less robust to perturbation, which helps explain the loss in generalization.

To verify if ScaLA improves the large-batch fine-tuning generalizability, we measure again the steepness of the loss landscape again after applying ScaLA, similar as what we do in Section 4. As shown in Fig. 8, the largest eigenvalue of the model becomes much smaller (6.9×) with lower deviations with ScaLA and is slightly better than the small batch baseline, which is a strong indication that our approach enforces the smoothness of the model that leads to the accuracy improvement.

## 6.4 *Training speed scalability*

Figure 9 shows the scalability results of ScaLA, varying the number of GPUs from 1 to 32. Overall, as the number of GPUs increases, configurations with large batch sizes (e.g., 1K) see a continual decrease in training time and achieve much higher throughput than configs with smaller batch sizes (e.g., 32). As expected, ScaLA scales better than ScaLA without delaying PGA-1, and achieves a much faster training speed, due to reduced computation complexity in each training iteration.

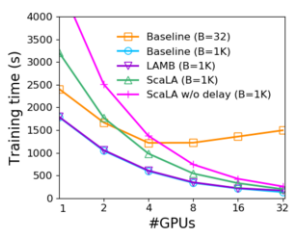



**Figure 9**: Comparison of scalability using different large-batch optimization methods on SST-2.

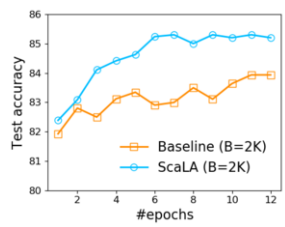



**Figure 10**: Comparison of test accuracy by training the baseline longer.

## 6.5 *Fine-tuning accuracy with different batch sizes*

We also evaluate how different batch sizes affect the fine-tuning accuracy of adapting transformers. Figure 11 shows the results on MNLI-m varying the batch size from 32 to 8K. We make two major observations: (1) The accuracy tends to drop as the batch size increases. (2) While both the baseline and LAMB suffer from significant accuracy drop by drastically increasing the batch size (e.g., from 32 to 8K), both SMART and ScaLA are able to mitigate the

generalization gap and consistently achieve higher accuracy than the baseline and LAMB. Although ScaLA achieves similar accuracy as SMART at larger batch sizes, it offers much faster training speed due to its lightweight adversary injection mechanism. These results also indicate the benefit of ScaLA is maintained by further increasing the batch size, which could bring even greater speedups when increasing the data parallelism degree.

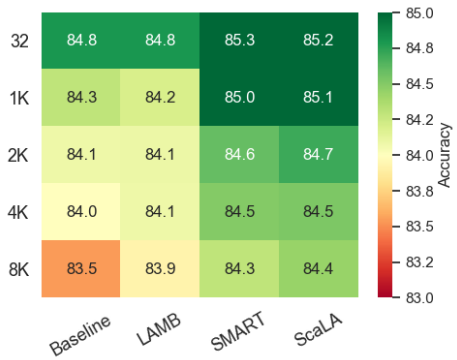



**Figure 11**: Comparison of accuracy under different batch sizes.

## 6.6 *Would train longer and tune hyperparameters lead to better fine-tuning accuracy?*

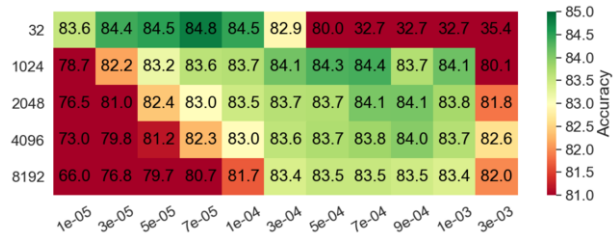
Despite improved adaptation speed, one may still wonder whether simply performing large-batch fine-tuning longer would also close the generalization gap. Figure 10 shows the comparison between ScaLA and the baseline on a batch size of 2K. ScaLA obtains an accuracy of 85.2 after 6 epochs of training, whereas the baseline has difficulty to reach 84 after training twice longer (e.g., 12 epochs). ScaLA achieves better accuracy because it explicitly penalizes model weights from getting stuck at sharp minima, leading to better generalizability.

If increasing the batch size improves the training speed, one may also wonder whether one can simply fine-tune pre-trained models with large batch sizes while tuning other hyperparameters, such as learning rates, to obtain high accuracy. We show that this is non-trivial, and one may not get the desired results even with significant tuning costs. In particular, we conduct an analysis of large-batch fine-tuning on pre-trained Transformers by performing a hyperparameter sweep on batch sizes {1K, 2K, 4K, 8K}, learning rates {1e-4, 3e-4,5e-4, 7e-4, 9e-4, 1e-3, 3e-3}, and two learning rate schedules: linear scaling [9] and sqrt scaling [39]. Figure 12 reports the validation accuracy. The results in Figure 12 show that the model can get poor accuracy when the learning rates are not properly chosen (e.g., accuracy in red boxes), (2) the best learning rates do not always follow the *sqrt* rule, making choosing the right learning rates challenging for large batch sizes; (3) even with a wide learning rate sweep, there is still a gap between the accuracy achieved via large batches and the best accuracy.

## 6.7 *Simplified hyperparameter tuning for ScaLA*

We looked into simple methods for adjusting hyperparameters when batch sizes increase in ScaLA for better training efficiency. The square root scaling rule (sqrt) was introduced in [39] to determine learning rates for large batch sizes. Table 5 compares the accuracy

**Figure 12**: Fine-tuning BERT$_{base}$ on MNLI with large batch sizes.




results of using sqrt scaling with tuned learning rates. Tuning learning rates improved results on some datasets like MNLI-m and SST-2, while sqrt scaling gave the same accuracy on QNLI and QQP. Since sqrt scaling provides comparable accuracy, we recommend trying it first as an easy and low-cost way to adjust ScaLA's learning rates.

**Table 5**: Evaluation results on hyperparameter tuning vs. using heuristic learning rate scaling.

| | MNLI-m | QNLI | QQP | SST-2 | Avg | Trials |
|---|---|---|---|---|---|---|
| Tuning lr | 85.1 | 90.8 | 91.4/88.4 | 93.5 | 89.9 | 10 |
| Sqrt lr | 84.9 | 90.8 | 91.4/88.4 | 92.9 | 89.7 | 1 |

## 7 Conclusions and Future Directions

Adversarial fine-tuning (i.e., fine-tuning using adversarially perturbed input data) is a promising method for improving the accuracy of pre-trained Transformer-based language models. However, the improved accuracy does not come for free but rather is accompanied by a significantly increased training time. This raises questions on whether adversarial fine-tuning can be applied to larger or more complicated models or even during pre-training. This work revisited the trade-off between accuracy and efficiency in adversarial fine-tuning by analyzing whether several optimizations can speed up the process while retaining its accuracy improvements.

We tested our proposed method, ScaLA, on five datasets and three Transformer models for natural language understanding including both encoder and encoder-decoder architectures. Our results show that ScaLA achieves a better accuracy-vs-speed tradeoff compared to existing methods, such as SMART, FreeLb, and LAMB. Nevertheless, our results also suggest that a significant improvement in the accuracy-vs-speed gap can be made, and future research direction may include applying ScaLA to pre-training language models using lightweight adversarial training.

## Acknowledgement

We are grateful for the insightful feedback provided by our anonymous reviewers, which has helped to enhance the quality of our work. Our thanks also go to Xiaodong Liu for engaging in productive discussions about SMART, which were instrumental in conducting this study. We appreciate the support of DeepSpeed, Microsoft WebXT, and the Azure ITP platform for providing the resources necessary for this research.

## References

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh,

Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei, 'Language models are few-shot learners', *CoRR*, (2020).

[2] Qi-Zhi Cai, Chang Liu, and Dawn Song, 'Curriculum adversarial training', in *IJCAI 2018*, (2018).

[3] Yong Cheng, Lu Jiang, and Wolfgang Macherey, 'Robust neural machine translation with doubly adversarial inputs', in *ACL 2019*, (2019).

[4] Damek Davis and Dmitriy Drusvyatskiy, 'Stochastic subgradient method converges at the rate $o(k\{-1/4\})$ on weakly convex functions', *arXiv preprint arXiv:1802.02988*, (2018).

[5] Damek Davis and Dmitriy Drusvyatskiy, 'Stochastic model-based minimization of weakly convex functions', *SIAM Journal on Optimization*, **29**(1), 207–239, (2019).

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, 'BERT: pre-training of deep bidirectional transformers for language understanding', in *NAACL-HLT 2019)*, (2019).

[7] Saeed Ghadimi and Guanghui Lan, 'Stochastic first-and zeroth-order methods for nonconvex stochastic programming', *SIAM Journal on Optimization*, **23**(4), 2341–2368, (2013).

[8] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy, 'Explaining and harnessing adversarial examples', in *3rd International Conference on Learning Representations, ICLR 2015*, (2015).

[9] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He, 'Accurate, large minibatch SGD: training imagenet in 1 hour', *CoRR*, **abs/1706.02677**, (2017).

[10] Sidharth Gupta, Parijat Dube, and Ashish Verma, 'Improving the affordability of robustness training for dnns', in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020*, (2020).

[11] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig, 'Towards a unified view of parameter-efficient transfer learning', *CoRR*, **abs/2110.04366**, (2021).

[12] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen, 'Deberta: decoding-enhanced bert with disentangled attention', in *9th International Conference on Learning Representations, ICLR*, (2021).

[13] Sepp Hochreiter and Jürgen Schmidhuber, 'Long Short-Term Memory', *Neural Computation*, **9**(8), 1735–1780, (1997).

[14] Elad Hoffer, Itay Hubara, and Daniel Soudry, 'Train longer, generalize better: closing the generalization gap in large batch training of neural networks', in *NeurIPS 2017*, (2017).

[15] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly, 'Parameter-efficient transfer learning for NLP', in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, (2019).

[16] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen, 'Lora: Low-rank adaptation of large language models', *CoRR*, **abs/2106.09685**, (2021).

[17] Prateek Jain and Purushottam Kar, 'Non-convex optimization for machine learning', *arXiv preprint arXiv:1712.07897*, (2017).

[18] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao, 'SMART: robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization', in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, (2020).

[19] Chi Jin, Praneeth Netrapalli, and Michael Jordan, 'What is local optimality in nonconvex-nonconcave minimax optimization?', in *ICML*, (2020).

[20] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang, 'On large-batch training for deep learning: Generalization gap and sharp minima', in *5th International Conference on Learning Representations*, (2017).

[21] Diederik P. Kingma and Jimmy Ba, 'Adam: A Method for Stochastic Optimization', in *3rd International Conference on Learning Representations, ICLR 2015*, (2015).

[22] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut, 'ALBERT: A lite BERT for self-supervised learning of language representations', in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, (2020).

[23] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala, 'Pytorch distributed: Experiences on accelerating data parallel training', *Proc. VLDB Endow.*, (2020).

[24] Tianyi Lin, Chi Jin, and Michael Jordan, 'On gradient descent ascent for nonconvex-concave minimax problems', in *ICML*, (2020).

[25] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov, 'Roberta: A robustly optimized BERT pretraining approach', *CoRR*, **abs/1907.11692**, (2019).

[26] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu, 'Towards deep learning models resistant to adversarial attacks', in *6th International Conference on Learning Representations, ICLR 2018*, (2018).

[27] Takeru Miyato, Andrew M. Dai, and Ian J. Goodfellow, 'Adversarial training methods for semi-supervised text classification', in *5th International Conference on Learning Representations, ICLR 2017*, (2017).

[28] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych, 'Adapterfusion: Non-destructive task composition for transfer learning', in *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2021*, (2021).

[29] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu, 'Exploring the limits of transfer learning with a unified text-to-text transformer', *CoRR*, **abs/1910.10683**, (2019).

[30] Ralph Tyrell Rockafellar, *Convex analysis*, Princeton university press, 2015.

[31] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro, 'Megatron-lm: Training multi-billion parameter language models using model parallelism', *CoRR*, **abs/1909.08053**, (2019).

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, 'Attention is all you need', in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, (2017).

[33] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman, 'GLUE: A multi-task benchmark and analysis platform for natural language understanding', in *7th International Conference on Learning Representations*, (2019).

[34] Dilin Wang, ChengYue Gong, and Qiang Liu, 'Improving neural language modeling via adversarial training', in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, (2019).

[35] Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Jianshu Ji, Guihong Cao, Daxin Jiang, and Ming Zhou, 'K-adapter: Infusing knowledge into pre-trained models with adapters', in *Findings of the Association for Computational Linguistics: ACL/IJCNLP*, (2021).

[36] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le, 'Xlnet: Generalized autoregressive pretraining for language understanding', in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*, (2019).

[37] Zhewei Yao, Amir Gholami, Qi Lei, Kurt Keutzer, and Michael W. Mahoney, 'Hessian-based analysis of large batch training and robustness to adversaries', in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*, (2018).

[38] Nanyang Ye, Qianxiao Li, Xiao-Yun Zhou, and Zhanxing Zhu, 'Amata: An annealing mechanism for adversarial training acceleration', **abs/2012.08112**, (2020).

[39] Yang You, Jing Li, Jonathan Hseu, Xiaodan Song, James Demmel, and Cho-Jui Hsieh, 'Reducing BERT pre-training time from 3 days to 76 minutes', *CoRR*, **abs/1904.00962**, (2019).

[40] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh, 'Large batch optimization for deep learning: Training bert in 76 minutes', *arXiv preprint arXiv:1904.00962*, (2019).

[41] Dongxu Zhang and Zhichao Yang, 'Word embedding perturbation for sentence classification', *CoRR*, **abs/1804.08166**, (2018).

[42] Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu, 'Freelb: Enhanced adversarial training for natural language understanding', in *8th International Conference on Learning Representations*, (2020).