Revisiting Few-sample BERT Fine-tuning

Tianyi Zhang[*†] **Felix Wu**[*†] **Arzoo Katiyar**[†] **Kilian Q. Weinberger**[†‡] **Yoav Artzi**[†‡]
[†]ASAPP Inc. [‡]Cornell University
{tzhang, fwu, akatiyar, kweinberger, yoav}@asapp.com

Abstract

We study the problem of few-sample fine-tuning of BERT contextual representations, and identify three sub-optimal choices in current, broadly adopted practices. First, we observe that the omission of the gradient bias correction in the BERTADAM optimizer results in fine-tuning instability. We also find that parts of the BERT network provide a detrimental starting point for fine-tuning, and simply re-initializing these layers speeds up learning and improves performance. Finally, we study the effect of training time, and observe that commonly used recipes often do not allocate sufficient time for training. In light of these findings, we re-visit recently proposed methods to improve few-sample fine-tuning with BERT and re-evaluate their effectiveness. Generally, we observe a decrease in their relative impact when modifying the fine-tuning process based on our findings.

1 Introduction

Fine-tuning self-supervised pre-trained models has significantly boosted state-of-the-art performance on Natural Language Processing (NLP) tasks [13, 28, 43, 51, 56]. One of the most effective models for this process is BERT [7]. However, despite significant success this process remains unstable, especially when using the large variant of BERT ($\text{BERT}_{\text{Large}}$) on small datasets, where pre-training stands to provide the most significant benefit. Identical learning processes with different random seeds often result in significantly different and sometimes degenerate models following fine-tuning, even though only a few, seemingly insignificant aspects of the learning process are impacted by the random seed [8, 23, 33].[1] As a result, practitioners resort to multiple random trials for model selection. This increases model deployment costs and time, and makes scientific comparison challenging [8].

In this paper, we review the common optimization practices of BERT fine-tuning and show that the cause and solution of instability lie in the optimization process. We focus our discussion on three aspects of optimization: the optimization algorithm, model initialization, and the number of fine-tuning training iterations. We identify prevalent suboptimalities in community practices along all three aspects. We identify, rather surprisingly, that BERT fine-tuning is often optimized with a non-standard implementation of ADAM [20]. This widely used implementation introduces a bias in the gradient estimation leading to detrimental effects on few-sample fine-tuning performance. We also show that the top layers of the pre-trained BERT model provide a bad initialization point for fine-tuning optimization. Re-initializing these top layers with random weights speeds up fine-tuning convergence and improves performance. Finally, we stress the importance of allocating enough training iterations for fine-tuning, and show that BERT fine-tuning on many small datasets is stabilized by simply training with more iterations.

Once these suboptimal practices are addressed, we observe that degenerate runs are eliminated and performance becomes much more stable. It also becomes unnecessary to execute numerous

[*]Equal contribution.

[1]Fine-tuning instability is also receiving significant practitioner attention. For example: https://github.com/zihangdai/xlnet/issues/96 and https://github.com/huggingface/transformers/issues/265.

Preprint. Under review.

random restarts as proposed in Dodge et al. [8]. In the light of these findings, we re-evaluate several techniques [18, 23, 33] that are specialized for few-sample fine-tuning and show a significant decrease in their impact. Our work invites future studies to design new methods to further improve few-sample fine-tuning on top of the set of optimization practices we outline in this paper.

2 Background and Related Work

BERT The Bidirectional Encoder Representations from Transformers [BERT; 7] model is a Transformer encoder [42] trained on raw text using masked language modeling and next-sentence prediction objectives. It generates an embedding vector contextualized through a stack of Transformer blocks for each input token. BERT prepends a special `[CLS]` token to the input sentence or sentence pairs. The embedding of this token is used as a summary token for the input for classification tasks. This embedding is computed with an additional fully-connected layer with a $\tanh$ non-linearity, commonly referred to as the *pooler*, to aggregate the information for the `[CLS]` embedding.

Fine-tuning The common approach for using the pre-trained BERT model is to replace the original output layer with a new task-specific layer and fine-tune the complete model. This includes learning the new output layer parameters and modifying all the original weights, including the weights of word embeddings, Transformer blocks, and the pooler. For example, for sentence-level classification, an added linear classifier projects the `[CLS]` embedding to an unnormalized probability vector over the output classes. This process introduces two sources of randomness: the weight initialization of the new output layer and the data order in the stochastic fine-tuning optimization. Existing work [8, 23, 33] shows that these seemingly benign factors can influence the results significantly, especially on small datasets (i.e., $<$ 10K examples). Consequently, practitioners often conduct many random trials of fine-tuning and pick the best model based on validation performance [7].

Fine-tuning Instability The instability of the BERT fine-tuning process has been known since its introduction [7], and various methods have been proposed to address it. Phang et al. [33] show that fine-tuning the pre-trained model on a large intermediate task stabilizes later fine-tuning on small datasets. Lee et al. [23] introduce a new regularization method to constrain the fine-tuned model to stay close to the pre-trained weights and show that it stabilizes fine-tuning. Dodge et al. [8] propose an early stopping method to efficiently filter out random seeds likely to lead to bad performance.

BERT Representation Transferability BERT pre-trained representations have been widely studied using probing methods showing that the pre-trained features from intermediate layers are more transferable [15, 16, 25, 40, 41] or applicable [55] to new tasks than features from later layers, which change more after fine-tuning [31, 32]. Our work is inspired by these findings, but focuses on studying how the pre-trained weights influence the fine-tuning process. Concurrent to our work, Tamkin et al. [39] adopt a similar methodology of weight re-initialization (Section 5) to study the transferability of BERT. In contrast to our study, their work emphasizes pinpointing the layers that contribute the most in transfer learning, and the relation between probing performance and transferability.

3 Experimental Methodology

Data We follow the data setup of previous studies [8, 23, 33] to study few-sample fine-tuning using eight datasets from the GLUE benchmark [46]. The datasets cover four tasks: natural language inference (RTE, QNLI, MNLI), paraphrase detection (MRPC, QQP), sentiment classification (SST-2), and linguistic acceptability (CoLA). Appendix A provides dataset statistics and a description of each dataset. We primarily focus on four datasets (RTE, MRPC, STS-B, CoLA) that have fewer than 10k training samples, because BERT fine-tuning on these datasets is known to be unstable [7]. We also complement our study by downsampling all eight datasets to 1k training examples following Phang et al. [33]. While previous studies [8, 23, 33] focus on the validation performance, we split held-out test sets for our study.[2] For RTE, MRPC, STS-B, and CoLA, we divide the original validation set in half, using one half for validation and the other for test. For the other four larger datasets, we only study the downsampled versions, and split additional 1k samples from the training set as our validation data and test on the original validation set.

[2]The original test sets are not publicly available.

Algorithm 1: the ADAM pseudocode adapted from Kingma and Ba [20], and provided for reference. g_t^2 denotes the elementwise square $g_t \odot g_t$. β_1 and β_2 to the power t are denoted as β_1^t, β_2^t. All operations on vectors are element-wise. The suggested hyperparameter values according to Kingma and Ba [20] are: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. BERTADAM [7] omits the bias correction (lines 9–10), and treats m_t and v_t as $\widehat{m}_t$ and $\widehat{v}_t$ in line 11.

Require: α: learning rate; $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for the moment estimates; $f(\theta)$: stochastic objective function with parameters θ; θ_0: initial parameter vector; $\lambda \in [0, 1)$: decoupled weight decay.

1: $m_0 \leftarrow 0$ (Initialize first moment vector)
2: $v_0 \leftarrow 0$ (Initialize second moment vector)
3: $t \leftarrow 0$ (Initialize timestep)
4: **while** θ_t not converged **do**
5: $t \leftarrow t + 1$
6: $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
7: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
8: $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
9: $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
10: $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
11: $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
12: **end while**
13: **return** θ_t (Resulting parameters)

Experimental Setup Unless noted otherwise, we follow the hyperparameter setup of Lee et al. [23]. We fine-tune the uncased, 24-layer $\text{BERT}_{\text{Large}}$ model with batch size 32, dropout 0.1, and peak learning rate 2×10^{-5} for three epochs. We apply linear learning rate warm-up during the first 10% of the updates followed by a linear decay. We use mixed precision training using Apex[3] to speed up experiments. We show that mixed precision training does not affect fine-tuning performance in Appendix C. We evaluate ten times on the validation set during training and perform early stopping. We fine-tune with 20 random seeds to compare different settings.

4 Optimization Algorithm: Debiasing Omission in BERTADAM

The most commonly used optimizer for fine-tuning BERT is BERTADAM, a modified version of the ADAM first-order stochastic optimization method. It differs from the original ADAM algorithm [20] in omitting a bias correction step. This change was introduced by Devlin et al. [7], and subsequently made its way into common open source libraries, including the official implementation,[4] huggingface's Transformers [50],[5] AllenNLP [10], GluonNLP [12], jiant [47], MT-DNN [27], and FARM.[6] As a result, this non-standard implementation is widely used in both industry and research [4, 8, 17, 22, 23, 26, 33, 37, 38, 45]. We observe that the bias correction omission influences the learning rate, especially early in the fine-tuning process, and is one of the primary reasons for instability in fine-tuning BERT [7, 8, 23, 33].

Algorithm 1 shows the ADAM algorithm, and highlights the omitted line in the non-standard BERTADAM implementation. At each optimization step (lines 4–11), ADAM computes the exponential moving average of the gradients (m_t) and the squared gradients (v_t), where β_1, β_2 parameterize the averaging (lines 7–8). Because ADAM initializes m_t and v_t to 0 and sets exponential decay rates β_1 and β_2 close to 1, the estimates of m_t and v_t are heavily biased towards 0 early during learning when t is small. Kingma and Ba [20] computes the ratio between the biased and the unbiased estimates of m_t and v_t as $(1 - \beta_1^t)$ and $(1 - \beta_2^t)$. This ratio is independent of the training data. The model parameters θ are updated in the direction of the averaged gradient m_t divided by the square root of the second moment $\sqrt{v_t}$ (line 11). BERTADAM omits the debiasing (lines 9–10), and directly uses the biased estimates in the parameters update.

Figure 1 shows the ratio $\frac{\widehat{m}_t}{\sqrt{\widehat{v}_t}}$ between the update using the biased and the unbiased estimation as a function of training iterations. The bias is relatively high early during learning, indicating overestimation. It eventually converges to one, suggesting that when training for sufficient iterations, the estimation bias will have negligible effect. Therefore, the bias ratio term is most important early during learning to counteract the overestimation of m_t and v_t during early iterations. In practice,

[3] https://github.com/NVIDIA/apex
[4] https://github.com/google-research/bert/blob/f39e881/optimization.py#L108-L157
[5] The default was changed from BERTADAM to debiased ADAM in commit ec07cf5a on July 11, 2019.
[6] https://github.com/deepset-ai/FARM

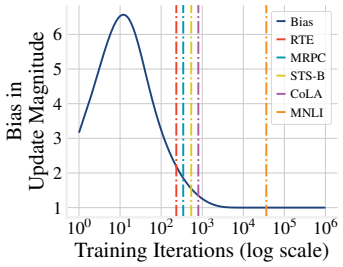

Figure 1: Bias in the ADAM update as a function of training iterations. Vertical lines indicate the typical number of iterations used to fine-tune BERT on four small datasets and one large dataset (MNLI). Small datasets use fewer iterations and are most affected.

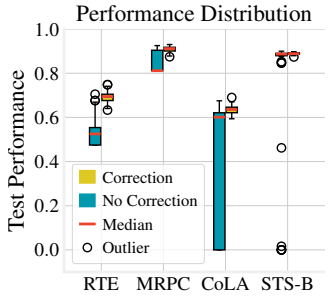

Figure 2: Performance distribution box plot across 50 random trials and the four datasets with and without ADAM bias correction. Bias correction reduces the variance of fine-tuning results by a large margin.

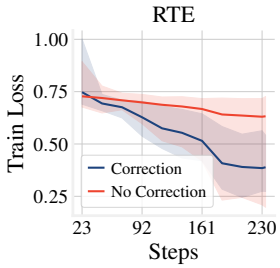

Figure 3: Mean (solid lines) and range (shaded region) of training loss during fine-tuning BERT, across 50 random trials. Bias correction speeds up convergence and shrinks the range of training loss.

ADAM adaptively re-scales the learning rate by $\frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$. This correction is crucial for BERT fine-tuning on small datasets with fewer than 10k training samples because they are typically fine-tuned with less than 1k iterations [7]. The figure shows the number of training iterations for RTE, MRPC, STS-B, CoLA, and MNLI. MNLI is the only one of this set with a large number of supervised training examples. For small datasets, the bias ratio is significantly higher than one for the entire fine-tuning process, implying that these datasets suffer heavily from overestimation in the update magnitude. In comparison, for MNLI, the majority of fine-tuning occurs in the region where the bias ratio has converged to one. This explains why fine-tuning on MNLI is known to be relatively stable [7].

We evaluate the importance of the debiasing step empirically by fine-tuning BERT with both BERTADAM and the debiased ADAM[7] for 50 random seeds on RTE, MRPC, STS-B, and CoLA. Figure 2 summarizes the performance distribution. The bias correction significantly reduces the performance variance across different random trials and the four datasets. Without the bias correction we observe many degenerate runs, where fine-tuned models fail to outperform the random baseline. For example, on RTE, 48% of fine-tuning runs have an accuracy less than 55%, which is close to random guessing. Figure 3 further illustrates this difference by plotting the mean and the range of training loss during fine-tuning across different random trials on RTE. Figure 12 in Appendix E shows similar plots for MRPC, STS-B, and CoLA. The biased BERTADAM consistently leads to worse averaged training loss, and on all datasets to higher maximum training loss. This indicates models trained with BERTAdam are underfitting and the root of instability lies in optimization.

We simulate a realistic setting of multiple random trials following Dodge et al. [8]. We use bootstrapping for the simulation: given the 50 fine-tuned models we trained, we sample models with replacement, perform model selection on the validation set, and record the test results; we repeat this process 1k times to estimate mean and variance. Figure 4 shows the simulated test results as a function of the number of random trials. Appendix D provides the same plots for validation performance. Using the debiased ADAM we can reliably achieve good results using fewer random trials; the difference in expected performance is especially pronounced when we perform less than 10 trials. Whereas the expected validation performance monotonically improves with more random trials [8], the expected test performance deteriorates when we perform too many random trials because the model selection process potentially overfits the validation set. Based on these observations, we recommend performing a moderate number of random trials (i.e., 5 or 10).

5 Initialization: Re-initializing BERT Pre-trained Layers

The initial values of network parameters have significant impact on the process of training deep neural networks, and various methods exist for careful initialization [6, 11, 14, 34, 54]. During fine-tuning, the BERT parameters take the role of the initialization point for the fine-tuning optimization process,

[7] We use the PyTorch ADAM implementation https://pytorch.org/docs/1.4.0/_modules/torch/optim/adamw.html.

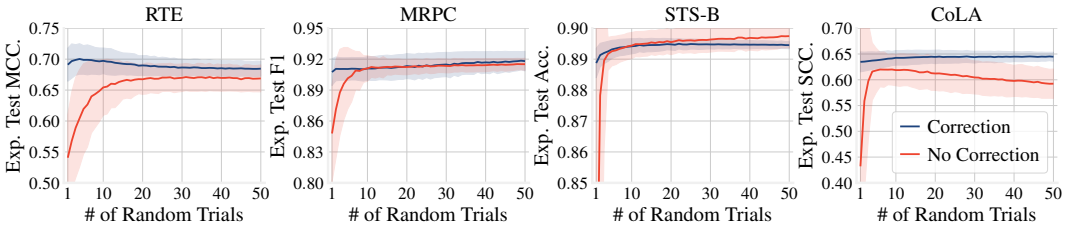

Figure 4: Expected test performance (solid lines) with standard deviation (shaded region) over the number of random trials allocated for fine-tuning BERT. With bias correction, we reliably achieve good results with few (i.e., 5 or 10) random trials.

Dataset	RTE		MRPC		STS-B		CoLA	
	3 Epochs	Longer	3 Epochs	Longer	3 Epochs	Longer	3 Epochs	Longer
Standard	69.5 ± 2.5	72.3 ± 1.9	90.8 ± 1.3	90.5 ± 1.5	89.0 ± 0.6	89.6 ± 0.3	63.0 ± 1.5	62.4 ± 1.7
Re-init	72.6 ± 1.6	73.1 ± 1.3	91.4 ± 0.8	91.0 ± 0.4	89.4 ± 0.2	89.9 ± 0.1	63.9 ± 1.9	61.9 ± 2.3
Dataset	RTE (1k)		MRPC (1k)		STS-B (1k)		CoLA (1k)	
	3 Epochs	Longer	3 Epochs	Longer	3 Epochs	Longer	3 Epochs	Longer
Standard	62.5 ± 2.8	65.2 ± 2.1	80.5 ± 3.3	83.8 ± 2.1	84.7 ± 1.4	88.0 ± 0.4	45.9 ± 1.6	48.8 ± 1.4
Re-init	65.6 ± 2.0	65.8 ± 1.7	84.6 ± 1.6	86.0 ± 1.2	87.2 ± 0.4	88.4 ± 0.2	47.6 ± 1.8	48.4 ± 2.1
Dataset	SST (1k)		QNLI (1k)		QQP (1k)		MNLI (1k)	
	3 Epochs	Longer	3 Epochs	Longer	3 Epochs	Longer	3 Epochs	Longer
Standard	89.7 ± 1.5	90.9 ± 0.5	78.6 ± 2.0	81.4 ± 0.9	74.0 ± 2.7	77.4 ± 0.8	52.2 ± 4.2	67.5 ± 1.1
Re-init	90.8 ± 0.4	91.2 ± 0.5	81.9 ± 0.5	82.1 ± 0.3	77.2 ± 0.7	77.6 ± 0.6	66.4 ± 0.6	68.8 ± 0.5

Table 1: Mean test performance and standard deviation. We compare fine-tuning with the complete BERT model (Standard) and fine-tuning with the partially re-initialized BERT (Re-init). We show results of fine-tuning for 3 epochs and for longer training (Sec 6). We underline and highlight in blue the best and number statistically equivalent to it among each group of 4 numbers. We use a one-tailed Student's t-test and reject the null hypothesis when $p < 0.05$.

while also capturing the information transferred from pre-training. The common approach for BERT fine-tuning is to initialize all layers except one specialized output layer with the pre-trained weights. We study the value of transferring all the layers in contrast to simply ignoring the information learned in some layers. This is motivated by object recognition transfer learning results showing that lower pre-trained layers learn more general features while higher layers closer to the output specialize more to the pre-training tasks [53]. Existing methods using BERT show that using the complete network is not always the most effective choice, as we discuss in Section 2. Our empirical results further confirm this: we observe that transferring the top pre-trained layers slows down learning and hurts performance.

We test the transferability of the top layers using a simple ablation study. Instead of using the pre-trained weights for all layers, we re-initialize the pooler layers and the top $L \in \mathbb{N}$ BERT Transformer blocks using the original BERT initialization, $\mathcal{N}(0, 0.02^2)$. Figure 5 illustrates the process. We compare two settings: (a) standard fine-tuning with BERT, and (b) Re-init fine-tuning of BERT. We evaluate Re-init by selecting $L \in \{1, \ldots, 6\}$ based on mean validation performance. All experiments use the debiased ADAM (Section 4) with 20 random seeds.

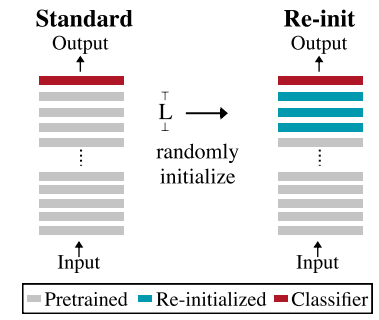

Figure 5: Re-init tests the effect of the last L layers in initializing fine-tuning.

Re-init Impact on Performance Table 1 shows our results on all the datasets from Section 3. We show results for the common setting of using 3 epochs, and also for longer training, which we discuss and study in Section 6. Re-init consistently improves mean performance on all the datasets, showing that not all layers are beneficial for transferring. It usually also decreases the variance across all datasets. Appendix E shows similar benefits for pre-trained models other than BERT.

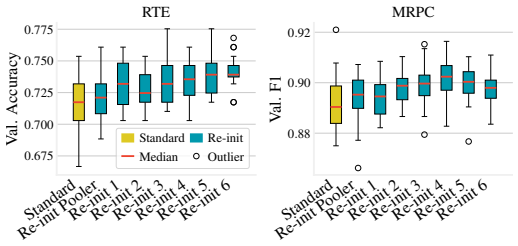

Figure 6: Validation performance distribution of re-initializing different number of layers of the BERT model.

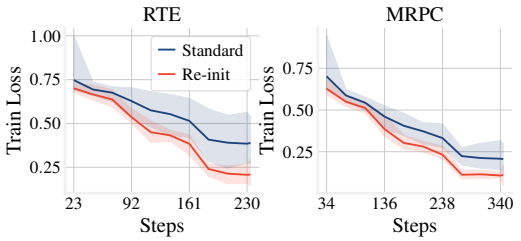

Figure 7: Mean (solid lines) and Range (shaded region) of training loss during fine-tuning BERT, across 20 random trials. Re-init leads to faster convergence and shrinks the range.

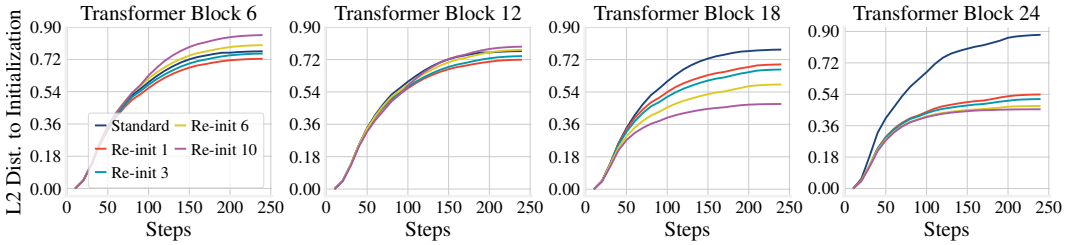

Figure 8: $L2$ distance to the initial parameters during fine-tuning BERT on RTE. Re-init reduces the amount of change in the weights of top Transformer blocks. However, re-initializing too many layers causes a larger change in the bottom Transformer blocks.

Sensitivity to Number of Layers Re-initialized Figure 6 shows the effect of the choice of L, the number of blocks we re-initialize, on RTE and MRPC. Figure 14 in Appendix E shows similar plots for the rest of the datasets. We observe more significant improvement in the worst-case performance than the best performance, suggesting that Re-init is more robust to unfavorable random seed. We already see improvements when only the pooler layer is re-initialized. Re-initializing further layers helps more. For larger L though, the performance plateaus and even decreases as re-initialize pre-trained layers with general important features. The best L varies across datasets.

Effect on Convergence and Parameter Change Figure 7 shows the training loss for both the standard fine-tuning and Re-init on RTE and MRPC. Figure 14, Appendix E shows the training loss for all other datasets. Re-init leads to faster convergence. We quantify the change in the weights of different Transformer blocks. For each block, we concatenate all parameters and record the $L2$ distance between these parameters and their initialized values during fine-tuning. In general, Re-init decreases the $L2$ distance to initialization for top Transformer blocks (i.e., 18 and 24). Re-initializing more layers leads to a larger reduction, indicating that Re-init decreases the fine-tuning workload. The effect of Re-init is not local; even re-initializing only the topmost Transformer block can affect the whole network. However, while Re-init 1 and Re-init 3 continue to benefit the bottom Transformer blocks, re-initializing too many layers (e.g., Re-init 10) can increase the $L2$ distance in the bottom Transformer blocks, suggesting a tradeoff between the bottom and the top Transformer blocks. We provide similar plots for all 24 Transformer blocks and for other datasets in Appendix E, Figures 16–19. Collectively, these results suggest that Re-init finds a better initialization for fine-tuning and the top L layers of BERT are potentially overspecialized to the pre-training objective.

6 Training Iterations: Fine-tuning BERT for Longer

BERT is typically fine-tuned with a slanted triangular learning rate, which applies linear warm-up to the learning rate followed by a linear decay. This learning schedule warrants deciding the number of training iterations upfront. Devlin et al. [7] recommend fine-tuning GLUE datasets for three epochs. This recommendation has been adopted broadly for fine-tuning [8, 23, 33]. We study the impact of this choice, and observe that this one-size-fits-all three-epochs practice for BERT fine-tuning is sub-optimal. Fine-tuning BERT longer can improve both training stability and model performance.

Experimental setup We study the effect of increasing the number of fine-tuning iterations for the datasets in Section 3. For the 1k downsampled datasets, where three epochs correspond to 96 steps,

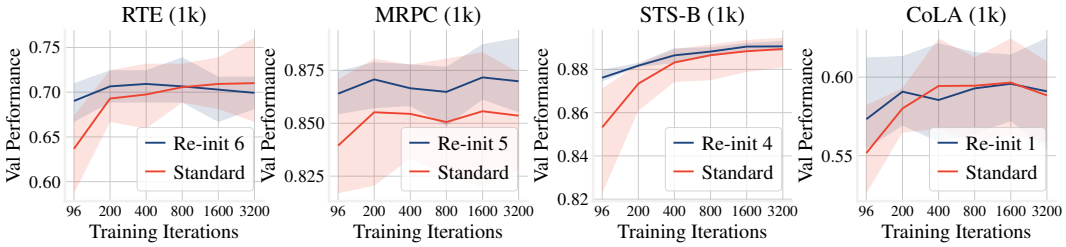

Figure 9: Mean (solid lines) and range (shaded region) of validation performance trained with different number of iterations, across eight random trials.

we tune the number of iterations in $\{200, 400, 800, 1600, 3200\}$. For the four small datasets, we tune the number of iterations in the same range but skip values smaller than the number of iterations used in three epochs. We evaluate our models ten times on the validation set during fine-tuning. This number is identical to the experiments in Sections 4–5, and controls for the set of models to choose from. We tune with eight different random seeds and select the best set of hyperparameters based on the mean validation performance to save experimental costs. After the hyperparameter search, we fine-tune with the best hyperparameters for 20 seeds and report the test performance.

Results Table 1 shows the result under the *Longer* column. Training longer can improve over the three-epochs setup most of the time, in terms of both performance and stability. This is more pronounced on the 1k downsampled datasets. We also find that training longer reduces the gap between standard fine-tuning and Re-init, indicating that training for more iterations can help these models recover from bad initializations. However, on datasets such as MRPC and MNLI, Re-init still improves the final performance even with training longer. We show the validation results on the four downsampled datasets with different number of training iterations in Figure 9. We provide a similar plot in Figure 15, Appendix F for the other downsampled datasets. We observe that different tasks generally require different number of training iterations and it is difficult to identify a one-size-fits-all solution. Therefore, we recommend practitioners to tune the number of training iterations on their datasets when they discover instability in fine-tuning. We also observe that on most of the datasets, Re-init requires fewer iterations to achieve the best performance, corroborating that Re-init provides a better initialization for fine-tuning.

7 Revisiting Existing Methods for Few-sample BERT Fine-tuning

Instability in BERT fine-tuning, especially in few-sample settings, is receiving increasing attention recently [7, 8, 23, 33]. We revisit these methods given our analysis of the fine-tuning process, focusing on the impact of using the debiased ADAM instead of BERTADAM (Section 4).

We revisit several recently proposed methods that were evaluated with sub-optimal optimization strategies when published. We find that when these methods are re-evaluated with the unbiased ADAM they are now less effective with respect to the improvement in fine-tuning stability and performance.

7.1 Overview

Pre-trained Weight Decay Weight decay (WD) is a common regularization technique [21]. At each optimization iteration, $\lambda \mathbf{w}$ is subtracted from the model parameters, where λ is a hyperparameter for the regularization strength and $\mathbf{w}$ is the model parameters. Pre-trained weight decay adapts this method for fine-tuning pre-trained models [3, 5] by subtracting $\lambda(\mathbf{w} - \hat{\mathbf{w}})$ from the objective, where $\hat{\mathbf{w}}$ is the pre-trained parameters. Lee et al. [23] empirically show that pre-trained weight decay works better than conventional weight decay in BERT fine-tuning and can stabilize fine-tuning.

Mixout Mixout [23] is a stochastic regularization technique motivated by Dropout [36] and DropConnect [44]. At each training iteration, each model parameter is replaced with its pre-trained value with probability p. The goal is to prevent catastrophic forgetting, and [23] proves it constrains the fine-tuned model from deviating too much from the pre-trained initialization.

Layer-wise Learning Rate Decay (LLRD) LLRD [18] is a method that applies higher learning rates for top layers and lower learning rates for bottom layers. This is accomplished by setting the learning rate of the top layer and using a multiplicative decay rate to decrease the learning rate

	Standard	Int. Task	LLRD	Mixout	Pre-trained WD	WD	Re-init	Longer
RTE	69.5 ± 2.5	$\underline{81.8} \pm 1.7$	69.7 ± 3.2	$\underline{71.3} \pm 1.4$	69.6 ± 2.1	69.5 ± 2.5	$\underline{72.6} \pm 1.6$	$\underline{72.3} \pm 1.9$
MRPC	90.8 ± 1.3	$\underline{91.8} \pm 1.0$	91.3 ± 1.1	90.4 ± 1.4	90.8 ± 1.3	90.8 ± 1.3	$\underline{91.4} \pm 0.8$	91.0 ± 1.3
STS-B	89.0 ± 0.6	89.2 ± 0.3	$\underline{89.2} \pm 0.4$	$\underline{89.2} \pm 0.4$	89.0 ± 0.5	89.0 ± 0.6	$\underline{89.4} \pm 0.2$	$\underline{89.6} \pm 0.3$
CoLA	63.0 ± 1.5	$\underline{63.9} \pm 1.8$	63.0 ± 2.5	61.6 ± 1.7	63.4 ± 1.5	63.0 ± 1.5	$\underline{64.2} \pm 1.6$	62.4 ± 1.7

Table 2: Mean test performance and standard deviation on four datasets. Numbers that are statistically significantly better than the standard setting (left column) are in blue and underlined. The results of Re-init and Longer are copied from Table 1. All experiments use ADAM with debiasing (Section 4). Except Longer, all methods are trained with three epochs.

layer-by-layer from top to bottom. The goal is to modify the lower layers that encode more general information less than the top layers that are more specific to the pre-training task. This method is adopted in fine-tuning several recent pre-trained models, including XLNet [52] and ELECTRA [4].

Transferring via an Intermediate Task Phang et al. [33] propose to conduct supplementary fine-tuning on a larger, intermediate task before fine-tuning on few-sample datasets. They show that this approach can reduce variance across different random trials and improve model performance. Their results show that transferring models fine-tuned on MNLI [49] can lead to significant improvement on several downstream tasks including RTE, MRPC, and STS-B. In contrast to the other methods, this approach requires large amount of additional annotated data.

7.2 Experiments

We evaluate all methods on RTE, MRPC, STS-B, and CoLA. We fine-tune a $\text{BERT}_{\text{Large}}$ model using the ADAM optimizer with debiasing for three epochs, the default number of epochs used with each of the methods. For intermediate task fine-tuning, we fine-tune a $\text{BERT}_{\text{Large}}$ model on MNLI and then fine-tune for our evaluation. For other methods, we perform hyperparameter search with a similar size search space for each method, as described in Appendix G. We do model selection using the average validation performance across 20 random seeds. We additionally report results for standard fine-tuning with longer training time (Section 6), weight decay, and Re-init (Section 5).

Table 2 provides our results. Compared to published results [23, 33], our test performance for Int. Task, Mixout, Pre-trained WD, and WD are generally higher when using the ADAM with debiasing.[8] However, we observe less pronounced benefits for all surveyed methods compared to results originally reported. At times, these methods do not outperform the standard baselines or simply training longer. Using additional annotated data for intermediate task training continues to be effective, leading to consistent improvement over the average performance across all datasets. LLRD and Mixout show less consistent performance impact. We observe no noticeable improvement using pre-trained weight decay and conventional weight decay in improving or stabilizing BERT fine-tuning in our experiments, contrary to existing work [23]. This indicates that these methods potentially ease the optimization difficulty brought by the debiasing omission in BERTADAM, and when we add the debiasing, the positive effects are reduced.

8 Conclusion

We have demonstrated that optimization plays a vital role in the few-sample BERT fine-tuning. First, we show that the debiasing omission in BERTADAM is the main cause of degenerate models on small datasets commonly observed in previous work [8, 23, 33]. Second, we observe the top layers of the pre-trained BERT provide a detrimental initialization for fine-tuning and delay learning. Simply re-initializing these layers not only speeds up learning but also leads to better model performance. Third, we demonstrate that the common one-size-fits-all three-epochs practice for BERT fine-tuning is sub-optimal and allocating more training time can stabilize fine-tuning. Finally, we revisit several methods proposed for stabilizing BERT fine-tuning and observe that their positive effects are reduced with the debiased ADAM. In the future, we plan to extend our study to different pre-training objectives and model architectures, and study how model parameters evolve during fine-tuning.

[8] The numbers in Table 2 are not directly comparable with previously published validation results [23, 33] because we are reporting test performance. However, the relatively large margin between our results and previously published results indicates an improvement. More important, our focus is the relative improvement, or lack of improvement compared to simply training longer.

Broader Impact

Our work critically reviews and carefully investigates the current, broadly adopted optimization practices in BERT fine-tuning. Our findings significantly stabilize BERT fine-tuning on small datasets. Stable training has multiple benefits. It reduces deployment costs and time, potentially making natural language processing applications more feasible and affordable for companies and individuals with limited computational resources. Our findings are focused on few-sample training scenarios, which opens, or at least eases the way for new applications at reduced data costs. The reduction in costs broadens the accessibility and reduces the energy footprint of BERT-based models. For example, applications that require frequent re-training are now easier to deploy given the reduced training costs. Our work also simplifies the scientific comparison between future fine-tuning methods by making training more stable, and therefore easier to reproduce. Our findings shed new light on several recently proposed techniques for BERT fine-tuning, potentially affecting methods relying on these techniques, which are not discussed in our paper. More importantly, we recommend a set of solid optimization practices on which future research can be built upon.

Our work is applicable to many different text processing applications. Societal impact largely depends on the application using our methods. We hope that broadening the set of organizations and individuals able to experiment with fine-tuning methods for a diverse set of tasks will allow for developing better informed societal norms about NLP applications. Similar to existing work using the same set of datasets as ours, our work may also be influenced by biases in these datasets. We use multiple broadly-used datasets to limit our exposure for specific dataset bias. We focus on fine-tuning dynamics, and therefore the risk of damage through application failure is less relevant. At worst, failure of our proposed practices will lead to lower performance in end applications. We generally recommend developers to consider the implications of low performance in their systems. These are often scenario specific.

Acknowledgments and Disclosure of Funding

We thank Cheolhyoung Lee for his help in reproducing previous work. We thank Lili Yu, Ethan R. Elenberg, Varsha Kishore, and Rishi Bommasani for their insightful comments, and Hugging Face for the Transformers project, which enabled our work.

References

[1] Luisa Bentivogli, Ido Kalman Dagan, Dang Hoa, Danilo Giampiccolo, and Bernardo Magnini. The fifth pascal recognizing textual entailment challenge. In *TAC 2009 Workshop*, 2009.

[2] Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *SemEval-2017*, 2017.

[3] Ciprian Chelba and Alex Acero. Adaptation of maximum entropy capitalizer: Little data can help a lot. In *EMNLP*, 2004.

[4] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*, 2020.

[5] Hal Daumé III. Frustratingly easy domain adaptation. In *ACL*, 2007.

[6] Yann N Dauphin and Samuel Schoenholz. Metainit: Initializing learning by learning to initialize. In *NeurIPS*, 2019.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[8] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*, 2020.

[9] William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *IWP*, 2005.

[10] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. Allennlp: A deep semantic natural language processing platform. In *NLP-OSS*, 2018.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[12] Jian Guo, He He, Tong He, Leonard Lausen, Mu Li, Haibin Lin, Xingjian Shi, Chenguang Wang, Junyuan Xie, Sheng Zha, Aston Zhang, Hang Zhang, Zhi Zhang, Zhongyue Zhang, and Shuai Zheng. Gluoncv and gluonnlp: Deep learning in computer vision and natural language processing. *arXiv preprint arXiv:1907.04433*, 2019.

[13] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*, 2020.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[15] J. Hewitt and P. Liang. Designing and interpreting probes with control tasks. In *EMNLP*, 2019.

[16] John Hewitt and Christopher D. Manning. A structural probe for finding syntax in word representations. In *NAACL*, 2019.

[17] 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 *ICML*, 2019.

[18] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *ACL*, 2018.

[19] Shankar Iyer, Nikhil Dandekar, and Kornel Csernai. First quora dataset release: Question pairs. `https://tinyurl.com/y2y8u5ed`, 2017.

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

[21] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *NeurIPS*, 1992.

[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 *ICLR*, 2020.

[23] Cheolhyoung Lee, Kyunghyun Cho, and Wanmo Kang. Mixout: Effective regularization to finetune large-scale pretrained language models. In *ICLR*, 2020.

[24] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[25] Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. Linguistic knowledge and transferability of contextual representations. *arXiv preprint arXiv:1903.08855*, 2019.

[26] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. In *ACL*, 2019.

[27] Xiaodong Liu, Yu Wang, Jianshu Ji, Hao Cheng, Xueyun Zhu, Emmanuel Awa, Pengcheng He, Weizhu Chen, Hoifung Poon, Guihong Cao, and Jianfeng Gao. The microsoft toolkit of multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:2002.07972*, 2020.

[28] Yang Liu. Fine-tune BERT for extractive summarization. *arXiv preprint arXiv:1903.10318*, 2019.

[29] 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. *arXiv*, abs/1907.11692, 2019.

[30] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 1975.

[31] Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, and Ian Tenney. What happens to bert embeddings during fine-tuning? *arXiv preprint arXiv:2004.14448*, 2020.

[32] Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. To tune or not to tune? adapting pretrained representations to diverse tasks. *arXiv preprint arXiv:1903.05987*, 2019.

[33] Jason Phang, Thibault Févry, and Samuel R Bowman. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*, 2018.

[34] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[35] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014.

[37] Asa Cooper Stickland and Iain Murray. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *ICML*, 2019.

[38] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *CCL*, 2019.

[39] Alex Tamkin, Trisha Singh, Davide Giovanardi, and Noah Goodman. Investigating transferability in pretrained language models, 2020.

[40] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. In *ACL*, 2019.

[41] Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, and Ellie Pavlick. What do you learn from context? probing for sentence structure in contextualized word representations. In *ICLR*, 2019.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[43] David Wadden, Ulme Wennberg, Yi Luan, and Hannaneh Hajishirzi. Entity, relation, and event extraction with contextualized span representations. In *EMNLP-IJCNLP*, 2019.

[44] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *ICML*, 2013.

[45] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In *NeurIPS*, 2019.

[46] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*, 2019.

[47] Alex Wang, Ian F. Tenney, Yada Pruksachatkun, Phil Yeres, Jason Phang, Haokun Liu, Phu Mon Htut, Katherin Yu, Jan Hula, Patrick Xia, Raghu Pappagari, Shuning Jin, R. Thomas McCoy, Roma Patel, Yinghui Huang, Edouard Grave, Najoung Kim, Thibault Févry, Berlin Chen, Nikita Nangia, Anhad Mohananey, Katharina Kann, Shikha Bordia, Nicolas Patry, David Benton, Ellie Pavlick, and Samuel R. Bowman. `jiant` 1.3: A software toolkit for research on general-purpose text understanding models. `http://jiant.info/`, 2019.

[48] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *TACL*, 2019.

[49] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *ACL*, 2018.

[50] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[51] Wei Yang, Haotian Zhang, and Jimmy Lin. Simple applications of BERT for ad hoc document retrieval. *arXiv preprint arXiv:1903.10972*, 2019.

[52] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, 2019.

[53] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NeurIPS*, 2014.

[54] Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Residual learning without normalization via better initialization. In *ICLR*, 2019.

[55] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating Text Generation with BERT. In *ICLR*, 2020.

[56] Jinhua Zhu, Yingce Xia, Lijun Wu, Di He, Tao Qin, Wengang Zhou, Houqiang Li, and Tieyan Liu. Incorporating bert into neural machine translation. In *ICLR*, 2020.

Task	RTE NLI	MRPC Paraphrase	STS-B Similarity	CoLA Acceptibility	SST-2 Sentiment	QNLI NLI	QQP Paraphrase	MNLI NLI
# of training samples	2.5k	3.7k	5.8k	8.6k	61.3k	104k	363k	392k
# of validation samples	139	204	690	521	1k	1k	1k	1k
# of test samples	139	205	690	521	1.8k	5.5k	40k	9.8k
Evaluation metric	Acc.	F1	SCC	MCC	Acc.	Acc.	Acc.	Acc.
Majority baseline (val)	52.9	81.3	0	0	50.0	50.0	50.0	33.3
Majority baseline (test)	52.5	81.2	0	0	49.1	50.5	63.2	31.8

Table 3: The datasets used in this work. We apply non-standard data splits to create test sets. SCC stands for Spearman Correlation Coefficient and MCC stands for Matthews Correlation Coefficient.

A Datasets

Table 3 summarizes dataset statistics and describes our validation/test splits. We also provide a brief introduction for each datasets:

RTE Recognizing Textual Entailment [1] is a binary entailment classification task. We use the GLUE version.

MRPC Microsoft Research Paraphrase Corpus [9] is binary classification task. Given a pair of sentences, a model has to predict whether they are paraphrases of each other. We use the GLUE version.

STS-B Semantic Textual Similarity Benchmark [2] is a regression tasks for estimating sentence similarity between a pair of sentences. We use the GLUE version.

CoLA Corpus of Linguistic Acceptability [48] is a binary classification task for verifying whether a sequence of words is a grammatically correct English sentence. Matthews correlation coefficient [30] is used to evaluate the performance. We use the GLUE version.

MNLI Multi-Genre Natural Language Inference Corpus [49] is a textual entailment dataset, where a model is asked to predict whether the premise entails the hypothesis, predicts the hypothesis, or neither. We use the GLUE version.

QQP Quora Question Pairs [19] is a binary classification task to determine whether two questions are semantically equivalent (i.e., paraphrase each other). We use the GLUE version.

SST-2 The binary version of the Stanford Sentiment Treebank [35] is a binary classification task for whether a sentence has positive or negative sentiment. We use the GLUE version.

B Isolating the Impact of Different Sources of Randomness

The randomness in BERT fine-tuning comes from three sources: (a) weight initialization, (b) data order, and (c) Dropout regularization [36]. We control the randomness using two separate random number generators: one for weight initialization and the other for both data order and Dropout (both of them affect the stochastic loss at each iteration). We fine-tune BERT on RTE for three epochs using ADAM with 10 seeds for both random number generators. We compare the standard setup with Re-init 5, where $L = 5$. This experiment is similar to Dodge et al. [8], but we use ADAM with debiasing instead of BERTADAM and control for the randomness in Dropout as well. When fixing a random seed for weight initialization, Re-init 5 shares the same initialized classifier weights with the standard baseline. Figure 10 shows the validation accuracy of each individual run as well as the minimum, average, and maximum scores when fixing one of the random seeds. Figure 11 summarizes the standard deviations when one of the random seeds is controlled. We observe several trends. Re-init 5 usually improves the performance regardless of the weight initialization or data order and Dropout. Second, Re-init 5 still reduces the instability when one of the sources of randomness is controlled. Third, the standard deviation of fixing the weight initialization roughly matches the one of controlled data order and Dropout, which aligns with the observation of Dodge et al. [8].

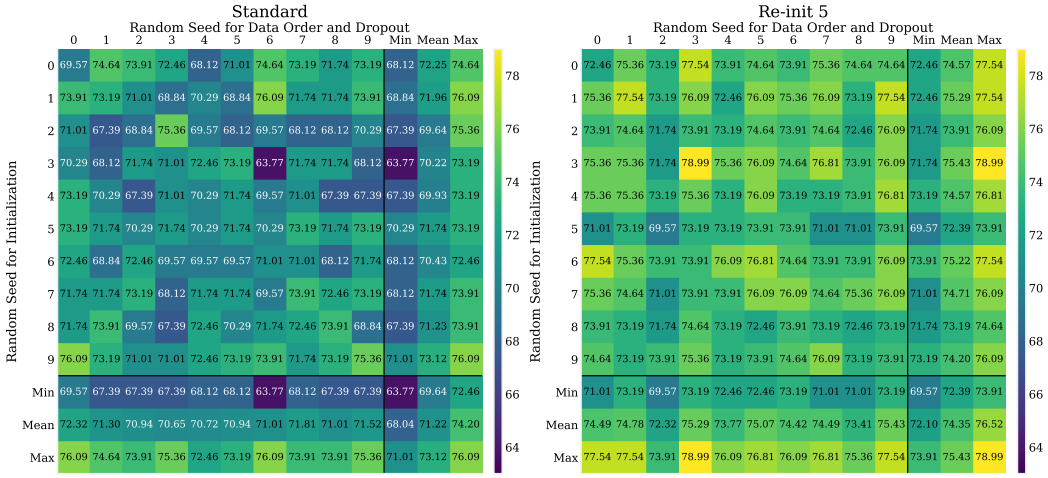

Figure 10: Validation accuracy on RTE with controlled random seeds. The min, mean, and max values of controlling one of the random seeds are also included. Re-init 5 usually improves the validation accuracy.

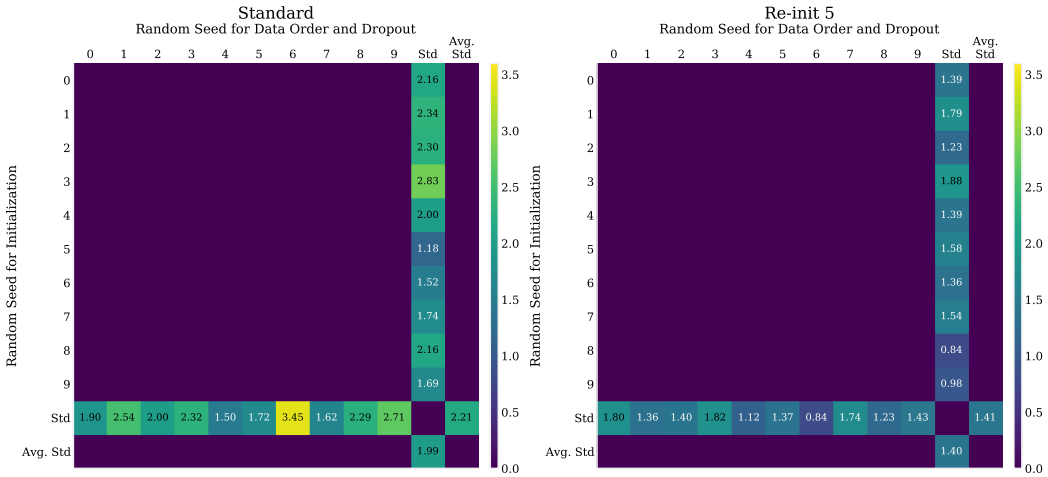

Figure 11: The standard deviation of the validation accuracy on RTE with controlled random seeds. We show the standard deviation of fixing either the initialization or data order and Dropout. Re-init 5 consistently reduces the instability regardless of the sources of the randomness.

C Mixed Precision Training

Mixed precision training can accelerate model training while preserving performance by replacing some 32-bit floating-point computation with 16-bit floating-point computation. We use mixed precision training in all our experiments using huggingface's Transformers [50]. Transformers uses O1-level optimized mixed precision training implemented with the Apex library.[9] We evaluate if this mixed precision implementation influences our results. We fine-tune BERT with 20 random trials on RTE, MRPC, STS-B, and CoLA. We use two-tailed t-test to test if the distributions of the two methods are statistically different. Table 4 shows the mean and standard deviation of the test performance. The performance of mixed precision matches the single precision counterpart, and there is no statistically significant difference.

[9] https://github.com/NVIDIA/apex

	CoLA	MRPC	RTE	STS-B
Mixed precision	60.3 ± 1.5	89.2 ± 1.2	71.8 ± 2.1	90.1 ± 0.7
Full precision	59.9 ± 1.5	88.7 ± 1.4	71.4 ± 2.2	90.1 ± 0.7

Table 4: Comparing BERT fine-tuning with mixed precision and full precision. The difference between the two numbers on any dataset is not statistically significant.

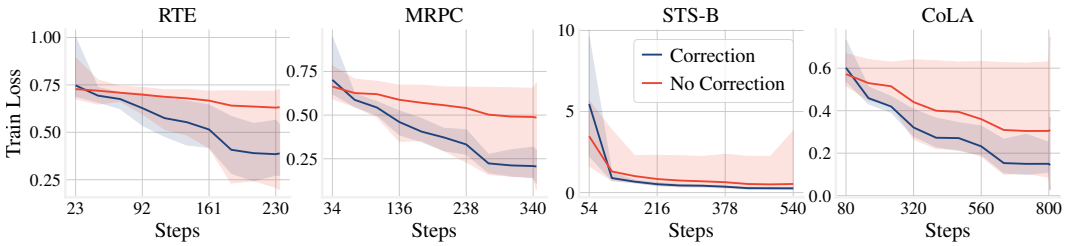

Figure 12: Mean (solid lines) and range (shaded region) of training loss during fine-tuning BERT, across 50 random trials. Bias correction speeds up convergence and reduces the range of the training loss.

D Supplementary Material for Section 4

Effect of ADAM with Debiasing on Convergence. Figure 12 shows the training loss as a function of the number of training iterations. Using bias correction effectively speeds up convergence and reduces the range of the training loss, which is consistent with our observation in Figure 3.

Effect of ADAM with Debiasing on the Expected Validation Performance. Figure 13 shows the expected validation performance as a function of the number of random trials. Comparing to Figure 4, we observe several trends. First, using ADAM with debiasing consistently leads to faster convergence and improved validation performance, which is similar to our observation about the test performance. Second, we observe that the expected validation performance monotonically increases with the number of random trials, contrary to our observation about the test performance. This suggests that using too many random trials may overfit to the validation set and hurt generalization performance.

E Supplementary Material for Section 5

Effect of L on Re-init Figure 14 shows the effect of Re-init in fine-tuning on the eight downsampled datasets. We observe similar trends in Figure 14 and Figure 6. Re-init's improvement is more pronounced in the wort-case performance across different random trials. Second, the best value of L is different for each dataset.

Effect of Re-init on Model Parameters We use the same setup as in Figure 8 to plot the change in the weights of different Transformer blocks during fine-tuning on RTE, MRPC, STS-B, and CoLA in Figures 16–19.

Effect of Re-init on Other Models We study more recent pre-trained contexual embedding models beyond $\text{BERT}_{\text{Large}}$. We investigate whether Re-init provides better fine-tuning initialization in $\text{XLNet}_{\text{Large}}$ [52], $\text{RoBERTa}_{\text{Large}}$ [29], $\text{BART}_{\text{Large}}$ [24], and $\text{ELECTRA}_{\text{Large}}$ [4]. XLNet is an autoregressive language model trained by learning all permutations of natural language sentences. RoBERTa is similar to BERT in terms of model architecture but is only pre-trained on the mask language modeling task only, but for longer and on more data. BART is a sequence-to-sequence model trained as a denoising autoencoder. ELECTRA is a BERT-like model trained to distinguish tokens generated by masked language model from tokens drawn from the natural distribution. Together, they represent a diverse range of modeling choices in pre-training, including different model architectures, objectives, data, and training strategies. We use ADAM with debiasing to fine-tune these models on RTE, MRPC, STS-B, and COLA, using the hyperparameters that are either described in the paper or in the official repository of each model. Table 5 summarizes the hyper-parameters of each model for each dataset. We use the huggingface's Transformers library [50]. The experimental setup is kept the same as our

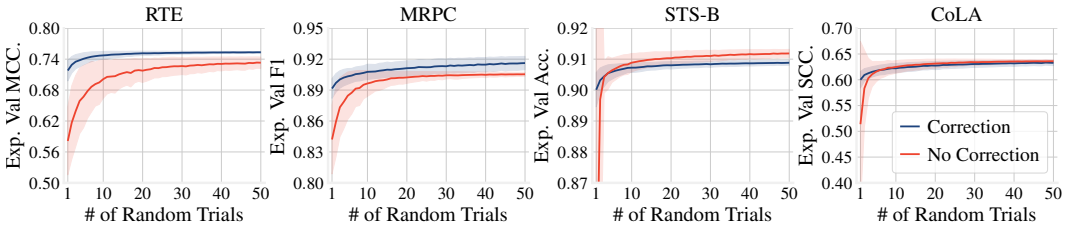

Figure 13: Expected validation performance (solid lines) with standard deviation (shaded region) over the number of random trials allocated for fine-tuning BERT. With bias correction, we can reliably achieve good results with few (i.e., 5 or 10) random trials.

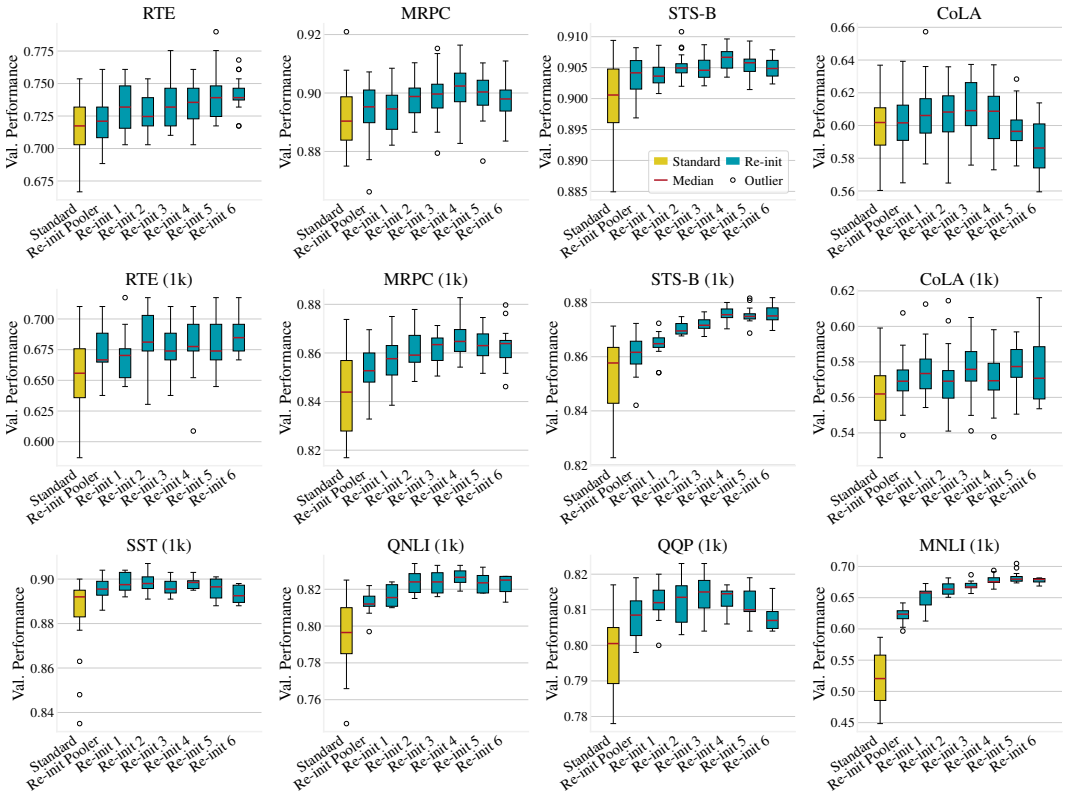

Figure 14: Validation performance distribution of re-initializing different number of layers of BERT on the downsampled datasets.

other experiments. Table 6 displays the average test performance on these datasets. We observe that several models suffer from high instability on these datasets and in most cases, Re-init can reduce the performance variance. We observe that for some models, like $\text{XLNet}_{\text{Large}}$ or $\text{RoBERTa}_{\text{Large}}$, Re-init can improve the average performance and reduce variance. However, the behavior of Re-init varies significantly across different models and Re-init have less significant improvement for $\text{ELECTRA}_{\text{Large}}$ and $\text{BART}_{\text{Large}}$. Further study of the entire model family requires significant computational resources, and we leave it as an important direction for future work.

F Supplementary Material for Section 6

Figure 15 plots the validation performance as a function of the number of training iteration, using the same setting as in Figure 9. Similar to our observations in Figure 9, we find that training longer generally improves fine-tuning performance and reduces the gap between standard fine-tuning and Re-init. On MNLI, Re-init still outperforms standard fine-tuning.

Model	Dataset	Learning Rate	Training Epochs / Steps	Batch Size	Warmup Ratio / Steps	LLRD
BERT	all	2×10^{-5}	3 epochs	32	10%	-
XLNet	RTE	3×10^{-5}	800 steps	32	200 steps	-
	MRPC	5×10^{-5}	800 steps	32	200 steps	-
	STS-B	5×10^{-5}	3000 steps	32	500 steps	-
	CoLA	3×10^{-5}	1200 steps	128	120 steps	-
RoBERTa	RTE	2×10^{-5}	2036 steps	16	122 steps	-
	MRPC	1×10^{-5}	2296 steps	16	137 steps	-
	STS-B	2×10^{-5}	3598 steps	16	214 steps	-
	CoLA	1×10^{-5}	5336 steps	16	320 steps	-
ELECTRA	RTE	5×10^{-5}	10 epochs	32	10%	0.9
	MRPC	5×10^{-5}	3 epochs	32	10%	0.9
	STS-B	5×10^{-5}	10 epochs	32	10%	0.9
	CoLA	5×10^{-5}	3 epochs	32	10%	0.9
BART	RTE	1×10^{-5}	1018 steps	32	61 steps	-
	MRPC	2×10^{-5}	1148 steps	64	68 steps	-
	STS-B	2×10^{-5}	1799 steps	32	107 steps	-
	CoLA	2×10^{-5}	1334 steps	64	80 steps	-

Table 5: Fine-tuning hyper-parameters of BERT and its variants as reported in the official repository of each model.

	RTE		MRPC		STS-B		CoLA	
	Standard	Re-init	Standard	Re-init	Standard	Re-init	Standard	Re-init
XLNet	71.7 ± 12.6	$\mathbf{80.1 \pm 1.6}$	92.3 ± 4.3	$\mathbf{94.5 \pm 0.8}$	86.8 ± 20.4	$\mathbf{91.7 \pm 0.3}$	51.8 ± 22.5	$\mathbf{62.0 \pm 2.1}$
RoBERTa	78.2 ± 12.1	$\mathbf{83.5 \pm 1.4}$	94.4 ± 0.9	$\mathbf{94.8 \pm 0.9}$	$\mathbf{91.8 \pm 0.3}$	$\mathbf{91.8 \pm 0.2}$	$\mathbf{68.4 \pm 2.2}$	67.6 ± 1.5
ELECTRA	$\mathbf{87.1 \pm 1.2}$	86.1 ± 1.9	$\mathbf{95.7 \pm 0.8}$	95.3 ± 0.8	91.8 ± 1.9	$\mathbf{92.1 \pm 0.5}$	$\mathbf{62.1 \pm 20.4}$	61.3 ± 20.1
BART	$\mathbf{84.1 \pm 2.0}$	83.5 ± 1.5	93.5 ± 0.9	$\mathbf{93.7 \pm 1.2}$	91.7 ± 0.3	$\mathbf{91.8 \pm 0.3}$	$\mathbf{65.4 \pm 1.9}$	64.9 ± 2.3

Table 6: Average Test performance with standard deviation on four small datasets with four different pre-trained models. For each setting, the better numbers are bolded and are in blue if the improvement is statistically significant.

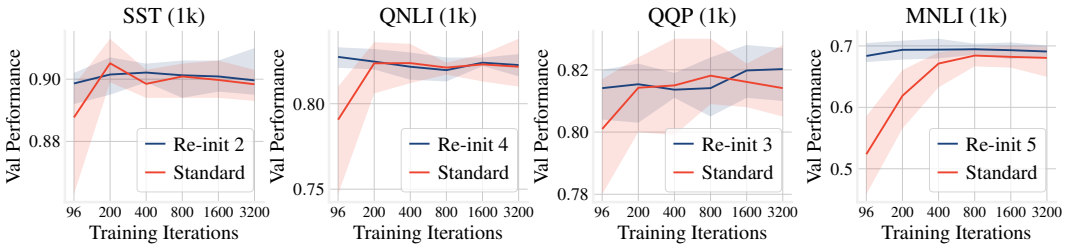

Figure 15: Mean (solid lines) and range (shaded region) of validation performance trained with different number of iterations, across eight random trials.

G Experimental Details in Section 7

The hyperparameter search space allocated for each method in our experiments is:

Layerwise Learning Rate Decay (LLRD) We grid search the initial learning rate in $\{2\times10^{-5}, 5\times10^{-5}, 1\times10^{-4}\}$ and the layerwise decay rate in $\{0.9, 0.95\}$.

Mixout We tune the mixout probability $p \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$.

Weight decay toward the pre-trained weight We tune the regularization strength $\lambda \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$.

Weight decay We tune the regularization strength $\lambda \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$.

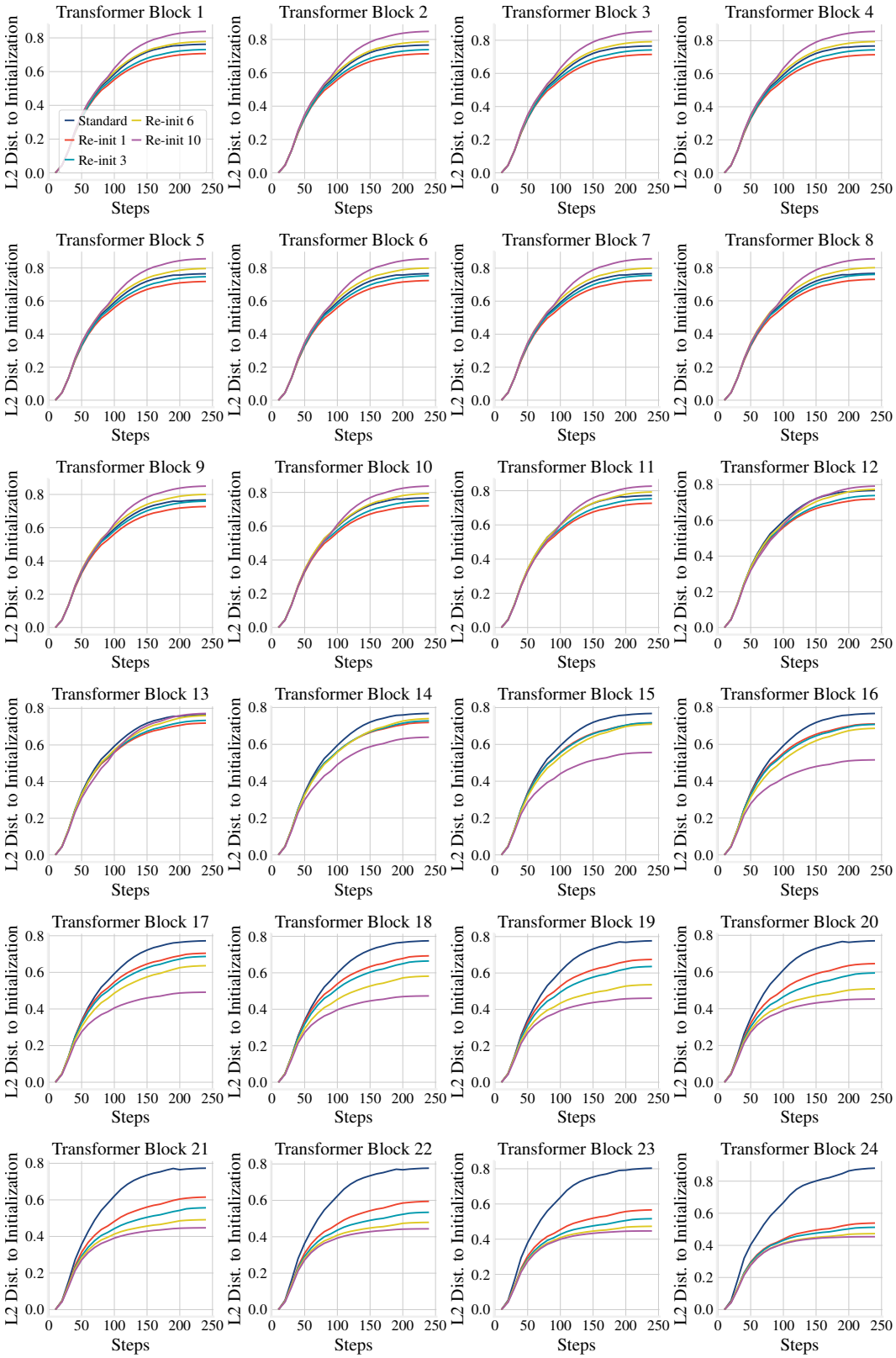

Figure 16: $L2$ distance to the initialization during find-tuning BERT on RTE.

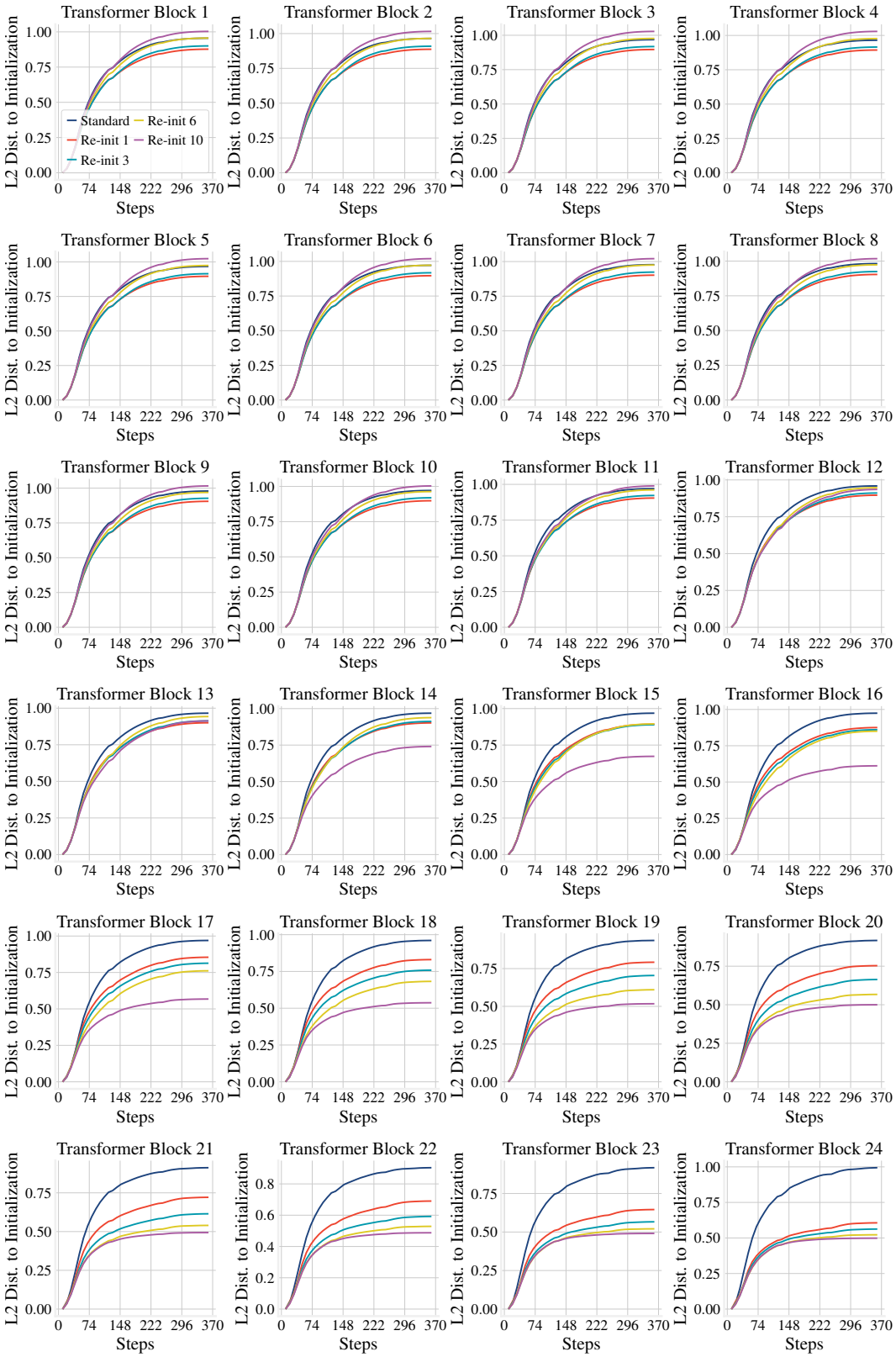

Figure 17: $L2$ distance to the initialization during find-tuning BERT on MRPC.

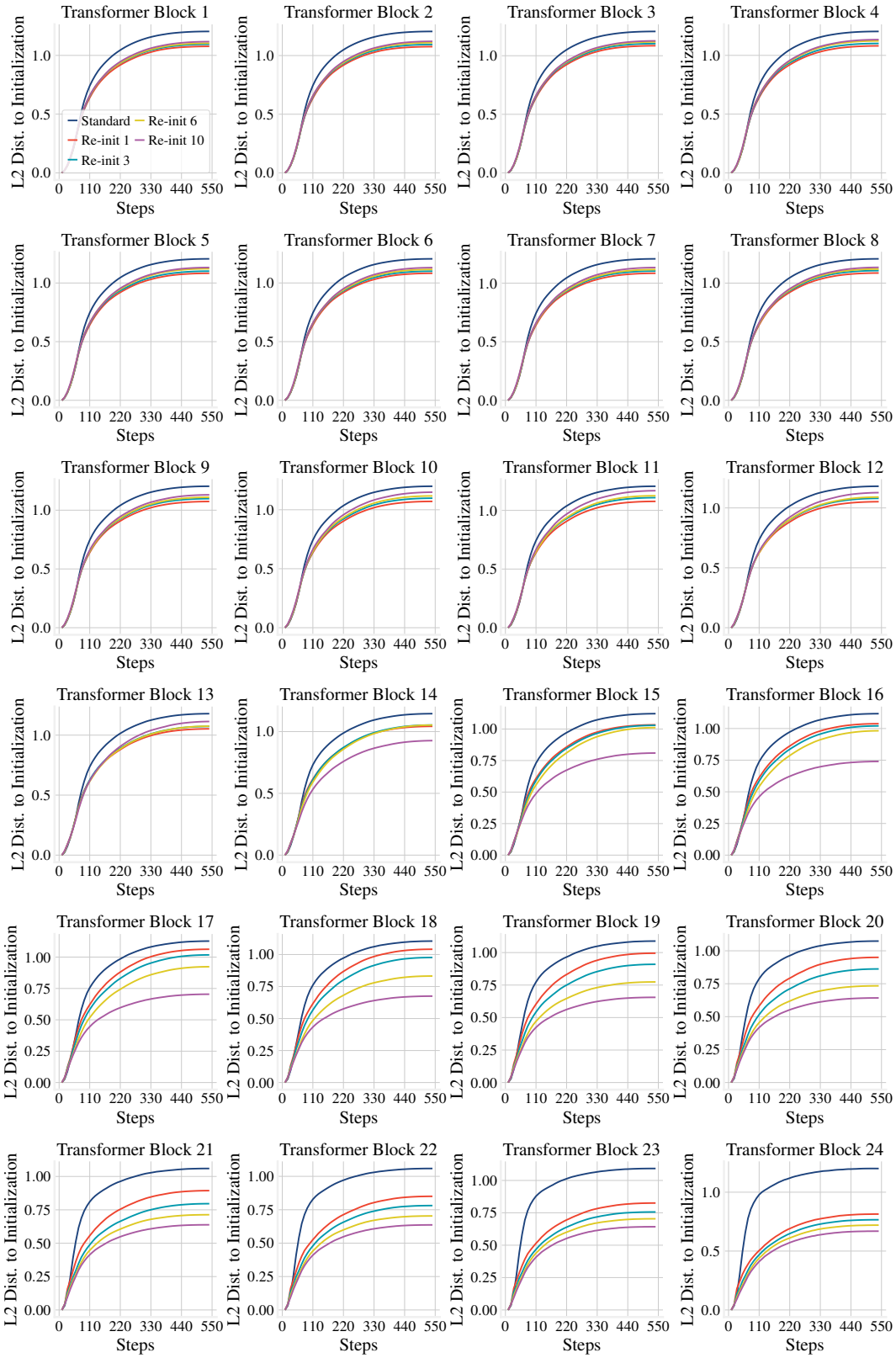

Figure 18: $L2$ distance to the initialization during find-tuning BERT on STS-B.

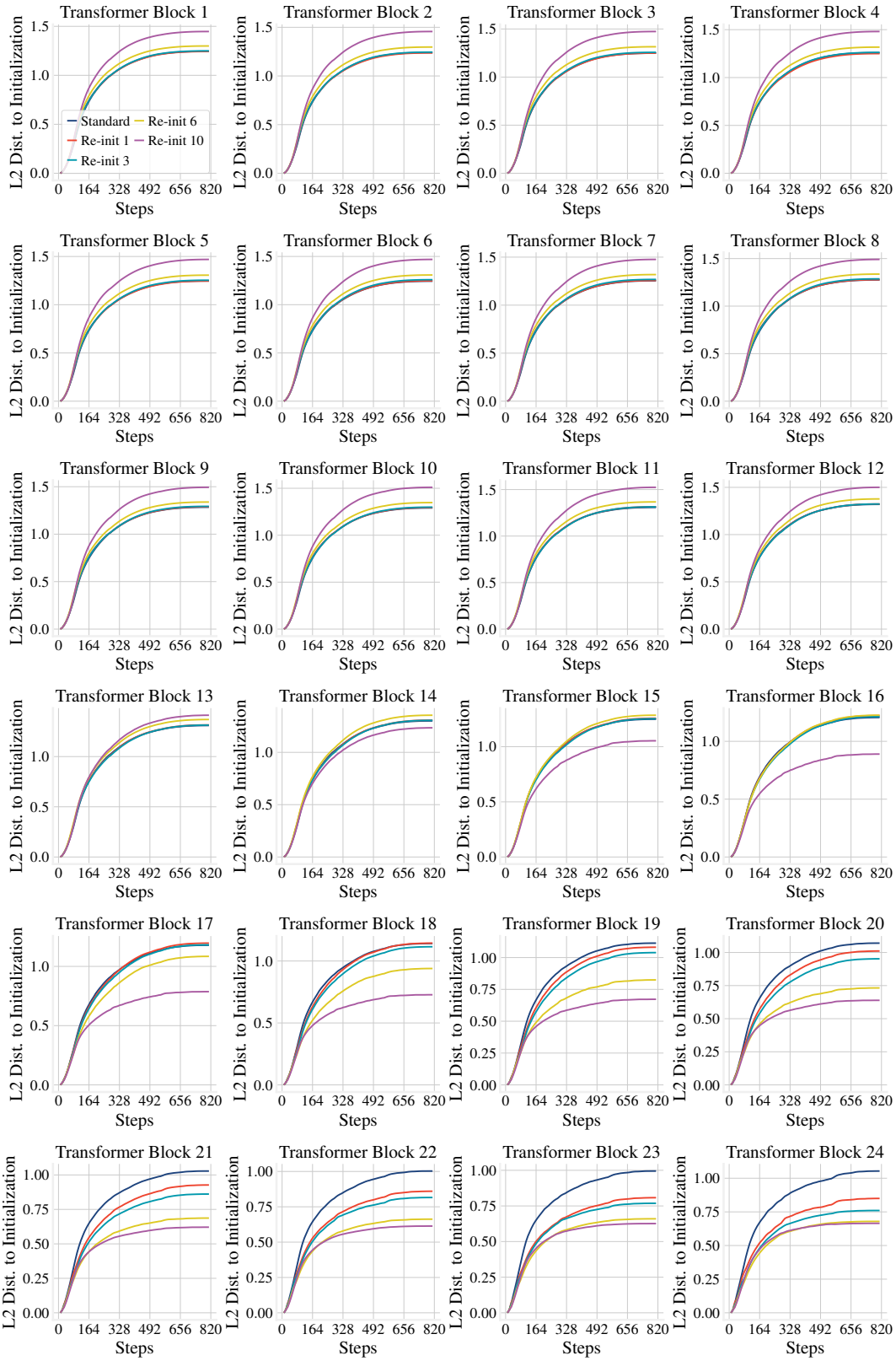

Figure 19: $L2$ distance to the initialization during find-tuning BERT on CoLA.