Jointly Reparametrized Multi-Layer Adaptation for Efficient and Private Tuning

Umang Gupta
USC Information Sciences Institute
umanggup@usc.edu

Aram Galstyan
USC Information Sciences Institute

Greg Ver Steeg
University of California Riverside

Abstract

Efficient finetuning of pretrained language transformers is becoming increasingly prevalent for solving natural language processing tasks. While effective, it can still require a large number of tunable parameters. This can be a drawback for low-resource applications and training with differential-privacy constraints, where excessive noise may be introduced during finetuning. To this end, we propose a novel language transformer finetuning strategy that introduces task-specific parameters in multiple transformer layers. These parameters are derived from fixed random projections of a single trainable vector, enabling finetuning with significantly fewer parameters while maintaining performance. We achieve within 5% of full finetuning performance on GLUE tasks with as few as 4,100 parameters per task, outperforming other parameter-efficient finetuning approaches that use a similar number of per-task parameters. Besides, the random projections can be precomputed at inference, avoiding additional computational latency. All these make our method particularly appealing for low-resource applications. Finally, our method achieves the best or comparable utility compared to several recent finetuning methods when training with the same privacy constraints, underscoring its effectiveness and potential real-world impact.

1 Introduction

Transformer-based bidirectional language models (LMs), pretrained on a sizeable text corpus and finetuned on task-specific objectives, outperform models trained from scratch by large margins (Devlin et al., 2019; Liu et al., 2019). The straightforward approach to finetune a language model is to initialize with pretrained parameters and train the model on the downstream task. However, it is inefficient to finetune language models for each task as it requires training and storing a massive number of parameters per task (roughly the same as the size of language models) (Radford et al., 2019; Devlin et al., 2019). These inefficiencies are exacerbated in resource-constrained settings, such as personal devices with limited or federated learning scenarios where the costs of communicating parameter updates may limit the scope of applications (Xu et al., 2022; Ro et al., 2022).

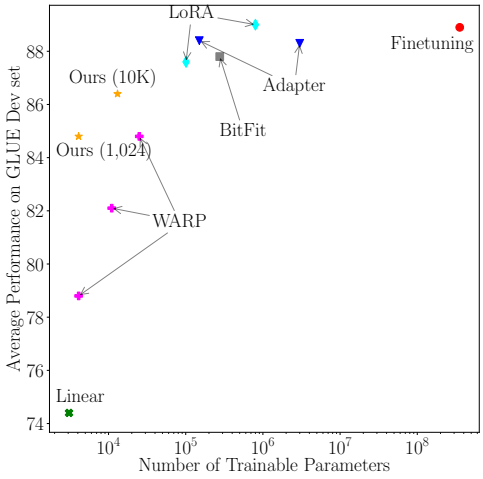

Figure 1: Performance vs. Parameters trade-off on GLUE benchmark with pretrained `RoBERTa-large`.

The shortcomings of naive finetuning methods have motivated research into approaches that identify and train fewer task-specific parameters (Treviso et al., 2022). Those parameter-efficient finetuning methods work by introducing task-specific trainable layers while freezing most of the pretrained language model parameters (*e.g.*, Adapter (Houlsby et al., 2019; Pfeiffer et al., 2021), LoRA (Hu et al., 2022)) or by introducing task-specific trainable prompts or inputs (*e.g.*, prompt-tuning based WARP (Hambardzumyan et al., 2021), prefix-tuning (Li and Liang, 2021)). We summarize the key properties of prominent efficient finetuning methods in Table 1. Among these methods, WARP is particularly interesting. It demonstrated comparable performance to full-finetuning with as few as 25K trainable parameters on natural language understanding (NLU) tasks.

Findings of the Association for Computational Linguistics: ACL 2023, pages 12612–12629
July 9-14, 2023

Method	Parameter Sharing	Efficient Inference	Multi-layer
Adapter	✗	✗	✓
LoRA	✗	✓	✓
BitFit	✗	✓	✓
WARP	✗	✗	✗
Ours	✓	✓	✓

Table 1: Parameter Efficient Finetuning Methods

WARP inserts trainable token embeddings around input, *i.e.*, task-specific parameters are inserted only in the input layer. Due to this, WARP is limited compared to other methods that insert trainable parameters in different layers (*i.e.*, *Multi-layer*), as the information may not propagate correctly to the deeper layers (Liu et al., 2022b). As such, our proposed method inserts task-specific information in each transformer block. In particular, we add a bias or shift vector to the output feed-forward layer's activation in each transformer block. All these shifts are derived from a single trainable vector, keeping the total trainable parameter count similar to WARP.

This is in contrast to BitFit (Ben Zaken et al., 2022), which updates all the bias parameters independently without sharing. Our proposed *parameter sharing* or joint reparametrization of task parameters drastically reduces the number of trainable parameters without significant performance degradation. On average, our method is within two points of BitFit on NLU tasks but uses 20x fewer parameters. Specifically, we achieve within 5% of full finetuning performance with only 4.1K parameters (see Figure 1), outperforming WARP which uses a similar number of parameters. Lastly, we show that parameter sharing and multi-layer tuning can also improve WARP.

WARP increases the effective sequence length, and Adapter inserts task-specific layers, incurring additional computational overhead. In contrast, our method is *efficient* in memory usage and run-time during training. Further, task-specific parameters learned by our approach can be fused with LM during inference, leading to no additional latency during inference, making it especially appealing for resource-constrained applications. Besides computational efficiency, our approach's parameter efficiency makes it an excellent private learner. Our approach's utility is competitive or outperforms the best differential private finetuning results (Yu et al., 2022) when training for similar levels of privacy.

2 Method

Model. Figure 2 summarizes our model, highlighting task-specific parameters with colored fonts. Specifically, we consider a trainable vector $\mathbf{z} \in \mathbb{R}^d$ to incorporate task-specific information in each transformer block. We do so by projecting $\mathbf{z}$ with random but fixed matrices $\mathbf{W}_l$ to obtain shift vectors $\mathbf{z}_l$ for the l^{th} transformer block ($\mathbf{z}_l \in \mathbb{R}^{d'_l}$, $\mathbf{W}_l \in \mathbb{R}^{d'_l \times d}$, and $l \in \{1 \dots L\}$). $\mathbf{z}_l$ is added to the output activations of the respective transformer block, as shown in Figure 2. $\mathbf{z}_l$ is of the same dimensionality as the activations of the output feed-forward layer in the l^{th} transformer block (d'_l), and $\mathbf{z}$ is shared between all the blocks. Hence, we call our approach Shared Layer Shift or *SLaSh*.

The random projection matrices, $\mathbf{W}_l$, are not trainable and are fixed throughout the training. We initialize $\mathbf{W}_l$ and $\mathbf{z}$ with zero-centered Gaussian or Uniform distribution for our experiments (See Appendix B.2 for ablations on initialization choices).

SLaSh is akin to training only bias parameters of the output feed-forward layers. However, the projection step decouples the dimensions of $\mathbf{z}$ and activations, providing the flexibility to change the number of trainable parameters and control the complexity of the model by varying d irrespective of the activation dimensions. Our choice of adding $\mathbf{z}_l$ to only output activations is inspired by Subramani and Suresh (2020), who use a similar setup to learn sentence representations. We also consider adding the shifts to other activations, such as intermediate activations or activations after the self-attention layer in Appendix B.1. In particular, adding shifts to output activations performs similarly or better than other choices. Adding shifts to intermediate layers performs similarly to adding shifts to the output layer. However, the dimensionality of intermediate activations is usually more than that of output activations which would increase the size of projection matrices, making it an inferior choice.

Classification Head. We experiment with token classification and sequence classification tasks with `BERT`-like models. To this end, we remove the decoder layer of the pretrained LM and attach a task-specific linear layer (Classifier) to predict the output from text representations. Verbalizers (Schick and Schütze, 2021) can also be used.

Number of Parameters. SLaSh only trains the task-specific vector ($\mathbf{z}$) and the prediction head (Classifier), usually a classification or regression

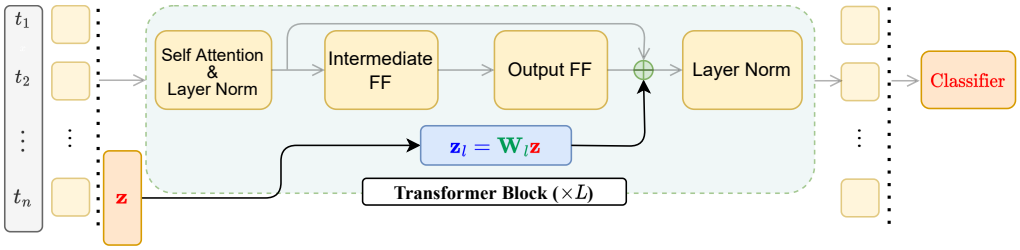

Figure 2: Shared Layer Shift or SLaSh for sequence classification tasks. The shifts are obtained by the projections $\mathbf{W}_l\mathbf{z}$ and added to the corresponding transformer block's output activation. $\mathbf{z}$ is shared across all the blocks. Red font indicates trainable modules, *i.e.*, parameters that are updated during finetuning. Other parameters remain unchanged during the finetuning. $[t_i]_{i=1}^n$ denotes the sequence of tokens.

layer. Suppose the number of class labels is C. SLaSh will only use $d + C \times (d'_L + 1)$ trainable parameters per task, where d'_L is the activation dimension of the last transformer block. In our implementation, we maintain additional $\sum_{l=1}^{L} d'_l \times d$ parameters for $\mathbf{W}_l$ matrices during training. However, these matrices can also be generated on the fly from the random seed or state of the random number generator for both backward and forward pass computation. More concretely, `RoBERTa-large` has $L = 24, d'_l = 1024 \quad \forall l \in \{1 \dots L\}$, and for GLUE tasks, the number of classes, C, can be 3 maximum. If d is set to 1,024, only 4,099 trainable parameters are required per task. In contrast, `RoBERTa-large` has 355M parameters.

The maximum size of $\mathbf{z}$ could be the sum of the dimensions of all the shift vectors, *i.e.*, $\sum_{l=1}^{L} d'_l$. Increasing the size beyond that is similar to training respective bias parameters independently without any sharing or reparametrization.

Inference. Pretrained LM parameters are shared across all the tasks. The projection weights remain unchanged during the training and can be reproduced from the random seed or random number generator's state. Hence, once the model is trained, only $\mathbf{z}$ and classifier parameters need to be preserved. Our approach maintains computational efficiency during inference as it does not require additional computations apart from the language model inference. Indeed, once the shift vectors $\mathbf{z}_l$ are computed, they can be combined with biases of the output feed-forward layers.

Improving Prompt-Tuning. These joint reparametrization of task parameters can also improve prompt-tuning methods such as WARP. We make two modifications — a) Insert prompts in different layers, and b) Prompts are derived from a single vector. We refer to this as JR-WARP (Jointly Reparametrized WARP. We provide more details about JR-WARP in Appendix A. Multilayer or deep-prompts have already been shown improve performance (Liu et al., 2022b; Li and Liang, 2021). Here we improve parameter efficiency while maintaining performance.

3 Experiments

We evaluate our approach for sequence classification tasks in Section 3.1 with the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019) and token classification tasks with named entity recognition (NER) on CoNLL 2003 dataset (Tjong Kim Sang and De Meulder, 2003) in Section 3.2. We report memory and training time requirements to quantify the computational efficiency in Section 3.3. Finally, we demonstrate the utility of our approach for differential private finetuning of LMs in Section 3.4.[1]

Baselines. We compare against full-finetuning and several prominent parameter-efficient finetuning techniques. Specifically, we compare with Adapter (Houlsby et al., 2019), Low-Rank Adaptation (LoRA, Hu et al. (2022)), BitFit (Ben Zaken et al., 2022), and Word Adversarial Reprogramming (WARP, Hambardzumyan et al. (2021)).

Adapter introduces task-specific feed-forward layers in each transformer block. Adapter typically trains down-project and up-project feed-forward layers in pairs for each transformer block. The dimensions of the down-projection (denoted as m) govern the per-task trainable parameters.

Low-rank adaptation, or *LoRA* learns the change in the pretrained weights, *i.e.*, ΔW, for

[1]Details about training, hyperparameter search, and best hyperparameters for all the experiments are in Appendix C. The code is available at `https://github.com/umgupta/jointly-reparametrized-finetuning`.

Method	# Params	MNLI (392,702)	QQP (363,846)	QNLI (104,743)	SST-2 (67,349)	CoLA (8,551)	STS-B (5,749)	MRPC (3,668)	RTE (2,490)	Avg.
Finetuning	355M	90.2	92.2	94.7	96.4	68.0	92.4	90.9	86.6	88.9
Adapter	3M	90.4	88.5	94.7	96.3	67.4	92.5	92.9	83.4	88.3
Linear Classifier	3.1K	70.9	77.1	78.8	89.8	48.9	73.8	83.8	72.2	74.4
LoRA	800K	90.8	88.8	94.9	96.2	68.2	92.6	93.6	87.4	89.0
WARP_1	4.1K	83.9	81.6	87.6	93.8	46.1	80.4	84.7	72.6	78.8
WARP_8	11K	87.6	83.8	93.0	95.4	57.4	81.0	85.6	72.9	82.1
WARP_{20}	25K	88.2	84.5	93.5	96.0	60.6	88.6	90.8	75.8	84.8
$\text{WARP}_{\text{MNLI}}$	25K	-	-	-	-	-	91.0	91.2	86.3	86.4
LoRA [rank = 1]	101K	90.0	87.1	94.3	95.9	63.3	91.9	92.9	85.6	87.6
Adapter [m = 1]	150K	90.4	88.0	94.7	95.9	68.0	92.1	92.6	85.6	88.4
BitFit	276K	90.4	87.3	94.5	95.4	66.0	92.1	93.3	83.4	87.8
Ours [d = 1,024]	4.1K	$85.8_{\pm 0.23}$	$83.2_{\pm 0.15}$	$92.2_{\pm 0.24}$	$94.7_{\pm 0.57}$	$59.6_{\pm 2.43}$	$90.4_{\pm 0.41}$	$91.1_{\pm 0.56}$	$81.5_{\pm 2.18}$	84.8
Ours [d = 2,048]	5.1K	$87.4_{\pm 0.08}$	$84.1_{\pm 0.09}$	$92.9_{\pm 0.28}$	$94.9_{\pm 0.34}$	$60.7_{\pm 2.11}$	$90.7_{\pm 0.30}$	$91.3_{\pm 0.84}$	$83.5_{\pm 1.67}$	85.7
Ours [d = 10K]	13.1K	$89.0_{\pm 0.14}$	$85.5_{\pm 0.10}$	$93.4_{\pm 0.19}$	$95.2_{\pm 0.36}$	$62.8_{\pm 1.43}$	$91.5_{\pm 0.24}$	$89.5_{\pm 4.17}$	$84.1_{\pm 1.10}$	86.4
JR-WARP_1 [d = 10K]	13.1K	$86.8_{\pm 1.26}$	$84.2_{\pm 0.52}$	$93.2_{\pm 0.20}$	$95.3_{\pm 0.37}$	$57.3_{\pm 2.61}$	$89.1_{\pm 0.69}$	$89.7_{\pm 1.41}$	$79.6_{\pm 1.32}$	84.4
Ours [d = 24,576] (max)	27.7K	89.5	86.5	93.4	95.6	64.0	91.5	92.1	87.7	87.5

Table 2: Results of finetuning `RoBERTa-large` with different methods on GLUE Development set. The bracketed numbers in the heading are training set sizes. # Params are per-task trainable parameters. Rows with very few (< 10K) parameters are highlighted in gray to facilitate comparison. Finetuning results are from Liu et al. (2019), and Adapter (3M) and WARP results are from Hambardzumyan et al. (2021). Linear results are the best of linear classifier and WARP_0 performance from Hambardzumyan et al. (2021).[2] $\text{WARP}_{\text{MNLI}}$ used an additional intermediate step of supervised training on the MNLI dataset. LoRA (800K) results are adapted from Hu et al. (2022).[3] The standard deviations are computed over 5 training runs with different seeds. Due to computational limitations, we report error bars for our methods only.

the downstream tasks. ΔW is parameterized as the product of low-rank matrices, which requires much fewer parameters than full-finetuning. The rank of the matrices determines per-task parameters.

WARP$_n$ introduces n learnable input tokens by adding trainable embeddings to the input. It is the continuous version of prompt-tuning and a special case of PrefixTuning (Li and Liang, 2021), with prefixes introduced only in the embedding layer. The learned tokens do not necessarily correspond to an existing token from the vocabulary.

Finally, we compare with *BitFit*, which finetunes only all the bias parameters. Indeed, BitFit finetunes a superset of parameters considered by our approach. Further, SLaSh shares trainable parameters across all the blocks, which is more efficient.

3.1 Sequence Classification Tasks

Datasets. We use the GLUE benchmark for sequence classification. We consider 2 single-sentence tasks and 6 sentence pair tasks from the GLUE benchmark. Corpus of Linguistic Acceptability (CoLA) and Stanford Sentiment Treebank (SST-2) are the single sentence tasks, and the task is to predict grammatical acceptability and sentiment. Microsoft Research Paraphrase Corpus (MRPC), Semantic Textual Similarity Benchmark (STS-B), and Quora Question Pairs (QQP) are the sentence similarity tasks. Multi-genre Natural Language Inference (MNLI), Question-Answering NLI (QNLI), and Recognizing textual entailments (RTE) are textual entailment prediction tasks. Similar to Devlin et al. (2019); Houlsby et al. (2019), we omit results on Winograd Schema Challenge (WNLI) as LMs do not outperform random prediction baselines.

All the tasks except STS-B are considered supervised classification tasks. Labels for STS-B are similarity scores from 1-5, and thus it is considered a regression task. We report accuracy on matched validation set for MNLI, Matthew's correlation and Pearson correlation on CoLA and STS-B, F1-score for MRPC and QQP, and accuracy for the rest of the tasks on the development set. Model selection is also performed based on these metrics.

`[CLS]` vs. `[MASK]` Representations. We consider two sentence-level representations for se-

[2] WARP_0 feeds `[MASK]` representations to the classifier head, whereas the linear classifier uses `[CLS]` representations.

[3] Since they report different metrics, we evaluated LoRA from the provided checkpoints on MNLI, STS-B, and QQP.

Method	% params	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
Finetuning	100%	86.4	88.0	92.3	94.2	61.1	90.6	92.5	77.4	85.3
BitFit	0.09%	85.8	85.2	91.9	93.7	60.1	90.6	91.9	71.8	83.9
LoRA [rank = 1]	0.04%	86.3	85.6	92.7	94.3	60.1	90.1	91.3	76.2	84.6
Adapter [m = 1]	0.05%	86.7	86.1	92.0	94.3	61.4	91.0	92.3	78.3	85.3
Ours [d = 1,024]	0.003%	$80.6_{\pm 0.26}$	$80.9_{\pm 0.09}$	$89.1_{\pm 0.53}$	$92.6_{\pm 0.27}$	$55.5_{\pm 1.99}$	$89.4_{\pm 0.19}$	$90.4_{\pm 0.76}$	$76.9_{\pm 1.87}$	81.9
Ours [d = 5K]	0.007%	$83.6_{\pm 0.16}$	$83.2_{\pm 0.11}$	$90.6_{\pm 0.21}$	$93.1_{\pm 0.45}$	$59.1_{\pm 1.74}$	$89.9_{\pm 0.28}$	$90.7_{\pm 0.88}$	$76.7_{\pm 1.84}$	83.4
JR-WARP$_1$ [d = 5K]	0.007%	$81.9_{\pm 0.78}$	$81.6_{\pm 0.66}$	$88.2_{\pm 1.24}$	$92.5_{\pm 0.60}$	$43.4_{\pm 9.12}$	$86.3_{\pm 1.75}$	$82.5_{\pm 3.45}$	$69.5_{\pm 1.36}$	78.2
Ours [d = 9,216] (max)	0.011%	84.4	83.9	90.5	93.7	58.8	90.1	90.8	79.4	83.9

Table 3: Results of finetuning `RoBERTa-base` with different methods on GLUE Development set. Finetuning results are taken from Ben Zaken et al. (2022). `RoBERTa-base` has 108 million parameters. The standard deviations are computed over 5 training runs with different seeds. Due to computational limitations, we report error bars for our methods only.

quence classification tasks — `[CLS]` and `[MASK]` token representations. Masked language models (MLMs) such as `BERT` and `RoBERTa` are pretrained by attaching a `[CLS]` token to the beginning of the input text. The `[CLS]` token representation is trained with the next sentence prediction loss and thus touted as the sentence-level representation. To this end, most previous works use `[CLS]` token representations. However, Hambardzumyan et al. (2021) suggested that `[MASK]` tokens representations, *i.e.*, inserting the `[MASK]` token at the end of input for single-sentence or between the sentences for tasks involving sentence pairs, produces better results than using `[CLS]` token representation.

We also find that the `[MASK]` representations are better than `[CLS]` representations generally and report results with `[MASK]` representations in the paper. We compare the two in Appendix B.3.

Training. We use `RoBERTa` (Liu et al., 2019) as the pretrained model to compare with previous works. For SLaSh, we vary the number of parameters by varying the size of the $\mathbf{z}$ vector. The output activation and embedding dimensions are 1,024 in `RoBERTa-large`. So, we train with d = 1,024 and 2,048 to compare head-to-head with WARP. We report results with d = 5K and 10K for `RoBERTa-base` and `RoBERTa-large`, which improves the results further. To demonstrate the capabilities of tuning only output activation's biases, we train with the maximum possible d, *i.e.*, the total number of activations, 9,216 and 24,576 for `RoBERTa-base` and `RoBERTa-large`. We also train LoRA and Adapter with minimum parameter configurations (rank = 1 and m = 1) as the results reported in their papers use a larger number of parameters than those concerning this work. We demonstrate that parameter sharing can also improve WARP by introducing JR-WARP and training it with d = 5K and 10K for respective `RoBERTa` models.

Results. Tables 2 and 3 summarize the results of finetuning with different methods using pretrained `RoBERTa` models. Parameter-efficient finetuning approaches aim to achieve performance at par with full finetuning while using fewer parameters. To this end, Figure 1 provides a visual summary of the parameter vs. performance trade-offs.

SLaSh's average performance is already within 4 points of full finetuning for both `RoBERTa-base` and `-large` models with d = 1,024. This gap is further reduced by increasing the dimension of the $\mathbf{z}$ vector. Even though the best models from our approach do not match the full-finetuning performance overall, for smaller datasets such as STS-B, MRPC, and RTE, SLaSh is competitive with full-finetuning. In the case of `RoBERTa-large`, we have 92.4 vs. 91.5 for STS-B, 90.9 vs. 91.3 for MRPC, and 86.6 vs. 84.1 for RTE with finetuning and SLaSh, respectively.[4] The parameter sharing reduces the per-task parameters considerably (4 orders of magnitude less) and is faster and more efficient to train (Section 3.3). All these make our approach suitable for low-resource, low-data applications such as training on edge devices or learning personalized models.

Most efficient tuning techniques tune a few hundred thousand parameters, except for WARP. It adds trainable parameters around input embeddings, which facilitates training with a few thou-

[4]Note that we consider the average performance of SLaSh across different training runs, whereas, for baselines, performance from a single training run with fixed seed is reported. This can slightly exaggerate baseline numbers.

sand parameters and is most comparable to our approach in per-task parameters. Our approach with d = 2,048 (*i.e.*, 5.1K parameters) outperforms WARP with 25K parameters on all datasets with less than 10K training samples. Further, SLaSh outperforms the best results of WARP while using less than 60% of parameters (13K vs. 25K). These observations do not change even with WARP pretraining on the MNLI task to improve the performance on smaller datasets ($\text{WARP}_{\text{MNLI}}$). We do not require this supervised pretraining trick. These results validate the intuition that instead of introducing task parameters closer to the input layer as in WARP, it may be more effective to introduce the parameters throughout the layers as in SLaSh.

Armed with this intuition, we improve WARP's performance by introducing prompts in all transformer blocks derived from a single vector (JR-WARP). On average, it underperforms SLaSh, and the variance among different training runs is higher. Nevertheless, JR-WARP performs comparably to WARP_{20} (84.4 vs. 84.8) while using fewer parameters (13K vs. 25K), suggesting that reusing parameters across layers improves parameter efficiency but does not deteriorate performance.

Next, we compare with LoRA and Adapter, arguably the most prominent language transformer finetuning approaches. We note that the Adapter (rank = 1) has a slightly better average performance than LoRA (m = 1) (Tables 2 and 3). SLaSh performs comparably to these methods for smaller datasets, using 5x fewer parameters and being roughly 2x faster to train for `RoBERTa-base` and 7x fewer parameters and roughly 1.25x faster to train for `RoBERTa-large` (Tables 2, 3 and 5). For example, in the case of `RoBERTa-base`, we have 91.0 vs. 89.9 for STS-B, 92.3 vs. 90.7 for MRPC, and 78.3 vs. 76.7 for RTE with Adapter and SLaSh, respectively.

Finally, SLaSh performs comparably to BitFit while tuning much fewer parameters. As with the other baselines, it is only for the larger datasets that BitFit considerably outperforms SLaSh. Further, we observe that tuning only output activation's biases, which used fewer than 15% of BitFit's parameters, performs comparably to BitFit on average (last row of Tables 2 and 3).

Another interesting result is the performance of BitFit vs. Adapter and LoRA with a similar number of trainable parameters. We observe that Adapter and LoRA outperform BitFit on most tasks with fewer trainable parameters. For instance, BitFit outperforms LoRA on QNLI, CoLA, STS-B, MRPC with `RoBERTa-large`, and only STS-B and MRPC with `RoBERTa-base`. Adapter outperforms BitFit on all the tasks with both pretrained models except MRPC with `RoBERTa-large`. These results contradict Ben Zaken et al. (2022), suggesting that while tuning bias parameters may achieve close to finetuning, LoRA or Adapter may yield better performance with fewer parameters.

Method	# params	Test	Validation
Finetuning	108M	91.35	94.97
Linear Classifier	7K	82.02	85.94
LoRA [rank = 1]	44K	89.50	93.38
Adapter [m = 1]	63K	90.09	93.55
BitFit	109K	89.83	93.62
WARP_{20}	22.3K	86.03	89.89
Ours [d = 1,024]	8K	86.49	89.37
Ours [d = 5K]	12K	88.30	91.38
JR-WARP_1 [d = 5K]	12K	87.08	90.93

Table 4: Results of finetuning `BERT-base-cased` for NER task on CoNLL-2003 (English) dataset.

3.2 Token Classification Task

Next, we evaluate our method on more complex token classification tasks such as NER. We consider the CoNLL-2003 (English) dataset. We use `BERT-base-cased` as the pretrained-LM and finetune it to predict the 9 entity classes. We use the validation set for model selection and report micro-F1 on the test and validation sets.

Results. Table 4 reports the results of finetuning with `BERT-base-cased` for the NER task. We see similar trends in performance as the sequence classification task. However, owing to the complexity of the NER task, all the methods underperform full-finetuning significantly (91.35 F1 score). SLaSh with 8K parameters underperforms full-finetuning by more than 4 points (86.49). The performance is improved to 88.30 by increasing the number of trainable parameters. However, LoRA, Adapter, and BitFit outperform the best results from SLaSh by roughly 1.5 points but use more than 3.5x parameters compared to SLaSh. Among the parameter-efficient techniques, Adapter performed the best while using fewer parameters than BitFit. Similar to Section 3.1, SLaSh and JR-WARP outperform WARP. Hyperparameter tuning (*e.g.*, increasing the sequence length) can improve JR-WARP results further. Overall, SLaSh

Method	Time (s)	Memory (GB)
Finetuning	3291	15.6
BitFit	2083	8.6
LoRA [rank = 1]	2019	13.0
Adapter [m = 1]	2289	13.1
WARP_{20}	1869	9.0
Ours [d = 10K]	1764	9.3

(a) `RoBERTa-large`

Method	Time (s)	Memory (GB)
Finetuning	1227	5.8
BitFit	819	3.3
LoRA [rank = 1]	1026	4.9
Adapter [m = 1]	1385	4.8
WARP_{20}	635	3.5
Ours [d = 5K]	558	3.3

(b) `RoBERTa-base`

Table 5: Memory and execution time for training 1 epoch on QNLI dataset (104,743 samples) with batch size 8. We report the maximum memory allocated during the training on a Quadro RTX 8000 GPU.

is suitable for extremely low-parameter applications, even the token classification tasks, but it may degrade performance.

3.3 Time & Memory Requirements

One of the goals of parameter-efficient tuning is to achieve as much utility as possible while being efficient with memory and computing. To this end, we report memory and time for training 1 epoch on the QNLI dataset in Table 5. Full finetuning requires longer execution time and more memory than any other approach, making a clear case for parameter-efficient approaches. SLaSh requires considerably less time and memory than LoRA and Adapter — 40% less time and 33% less memory for `RoBERTa-base` and 12% less time and 30% less memory for `RoBERTa-large`. The gains are less pronounced for `large` models than `base` because relatively more resources are utilized for transformer computations than tuning-specific computations. Compared to BitFit, SLaSh trains faster, but the memory requirements are similar due to SLaSh maintaining projection matrices during training.

We maintained projection matrices in memory instead of generating them on the fly for our experiments, and Table 5 uses this implementation. However, matrices can be generated on the fly for both forward and backward passes from the state of the random number generator, leading to a further reduction in memory usage. With this improvement, the memory usage comes down to 8.3 GB and 3.1 GB for the large and base model without significantly impacting training time. Finally, WARP's memory utilization is identical to SLaSh, but has slightly higher training time due to increased sequence length. SLaSh is much more resource-efficient during training than other methods without too much compromise on performance.

Inference times for all the methods were similar. The time to perform inference over the QNLI validation set (5,463 examples) varied between 13.9-14.5 seconds for `RoBERTa-base` and 39.7-40.8 seconds for `RoBERTa-large`.

3.4 Differential Private Finetuning

As machine learning is beginning to be applied in commercial settings and on user data, ensuring the privacy of training data is becoming crucial. Neural networks trained without safeguards can easily leak information about their private training data (Carlini et al., 2021, 2022). To mitigate these issues, neural networks can be trained with a strong notion of privacy, Differential Privacy (DP), which limits the influence of a single training example on the result (Dwork et al., 2014).

Differential privacy is formally characterized by ϵ and δ and denoted as $(\epsilon, \delta) - \text{DP}$. Lower ϵ and δ imply more privacy. The standard procedure to train neural networks with DP is Differential Private SGD (DPSGD, Abadi et al. (2016)). DPSGD is a private variant of SGD in which per-sample parameter gradients are clipped, and Gaussian noise is added before the update step. The noise magnitude depends on ϵ, δ, and model size and drastically impacts utility (Tramer and Boneh, 2021).

Recently, Yu et al. (2022); Li et al. (2022) demonstrated that the utility of differential private finetuning is at par with non-private training. One of the key insights is that the parameter-efficient methods are better private learners than full finetuning. Intuitively, the amount of noise scales with parameters and fewer parameters implies less noise is added during training. Naturally, this encouraged us to evaluate SLaSh and JR-WARP for private learning. To this end, we use the same setup as Yu et al. (2022). In particular, we consider the tasks with more than 10K samples in the GLUE benchmark and train to achieve $(\epsilon = 6.7, \delta = 10^{-6}) - \text{DP}$. Different from Section 3.1, we report accuracy for

	MNLI	QQP	QNLI	SST-2
Non-Private Training				
Finetuning	90.2	92.2	94.7	96.4
Ours $_{[d=10K]}$	89.1	89.1	93.5	95.9
JR-WARP$_1$ $_{[d=10K]}$	89.0	88.9	93.5	95.5
Private Training				
Ours $_{[d=10K]}$	**88.0**	86.9	**91.2**	94.5
JR-WARP$_1$ $_{[d=10K]}$	87.7	86.3	91.1	94.4
RGP	86.1	86.7	90.0	93.0
Adapter	87.7	86.3	90.7	93.9
Compacter	87.5	86.2	90.2	94.2
LoRA	87.8	**87.4**	90.8	**95.3**

(a) Finetuning with `RoBERTa-large`

	MNLI	QQP	QNLI	SST-2
Non-Private Training				
Finetuning	87.6	91.9	92.8	94.8
Ours $_{[d=5K]}$	83.6	87.4	90.8	93.7
JR-WARP$_1$ $_{[d=5K]}$	83.4	87.2	90.7	93.3
Private Training				
Ours $_{[d=5K]}$	83.0	84.9	87.6	92.4
JR-WARP$_1$ $_{[d=5K]}$	81.3	84.7	**87.9**	92.0
RGP	80.1	85.5	87.2	91.6
Adapter	83.4	85.6	87.5	**92.5**
Compacter	82.6	84.7	85.1	92.3
LoRA	**83.5**	**85.7**	87.3	92.2

(b) Finetuning with `RoBERTa-base`

Table 6: Results of differential private finetuning on GLUE Development set. Non-private finetuning and Private training results for RGP, Compacter, Adapter, and LoRA are from Yu et al. (2022). Private models were trained to achieve $\epsilon = 6.7$ for all datasets and $\delta = 10^{-6}$ for MNLI, QQP, and QNLI and $\delta = 10^{-5}$ for SST-2. For our method, privacy parameters are $\epsilon = 6.7$ and $\delta = 10^{-6}$ for all datasets (*i.e.*, identical or stricter than the baselines).

all the tasks here. We compare against the methods reported by Yu et al. (2022), which include LoRA, Adapter, and Compacter (Karimi Mahabadi et al., 2021). Compacter is an improved and efficient version of the Adapter. RGP updates all the parameters, *i.e.*, it is similar to full-finetuning but uses a different parametrization.

Results. Table 6 reports the results of private finetuning `RoBERTa` under a fixed privacy budget ($\epsilon = 6.7, \delta = 10^{-6}$). Due to using only a tiny number of parameters, the gap in the non-private and private utility of SLaSh and JR-WARP is small. Further, SLaSh outperforms all the other methods on MNLI and QNLI tasks and is only second to the best (LoRA) on QQP and SST-2 with `RoBERTa-large`. Similarly, JR-WARP and SLaSh outperform all the other methods on the QNLI task with `RoBERTa-base`; however, JR-WARP's utility is lower on MNLI. SLaSh's utility is generally comparable to other methods for all the tasks. Our approaches (SLaSh and JR-WARP) may be more effective for larger models as those are easier to tune with fewer parameters (Lester et al., 2021).

4 Related Work

Prompt tuning and task-specific finetuning are standard ways to prime LMs for downstream tasks (Liu et al., 2022a; Treviso et al., 2022). Prompt tuning inserts task-specific information or parameters around the input. Various versions exist, such as manual prompt-tuning, discrete prompt search (Shin et al., 2020), and continuous search (Hambardzumyan et al., 2021). Prompt tuning is highly parameter efficient but is generally only effective for larger LMs (Lester et al., 2021; Yang et al., 2022). Due to joint reparametrization, our method uses a similar number of parameters as prompt-tuning methods but outperforms them.

Several parameter-efficient LM finetuning methods have been proposed, such as Adapter (Houlsby et al., 2019), LoRA (Hu et al., 2022), Prefix-Tuning (Li and Liang, 2021), and Parallel Adapters (He et al., 2022). Further improvements try to maintain the utility while reducing the parameters such as Compacter (Karimi Mahabadi et al., 2021) that parameterizes weight matrices via the sum of Kronecker products, pruning adapter layers (Rücklé et al., 2021; Pfeiffer et al., 2021) and gating mechanisms to choose the best modules (Mao et al., 2022). These methods outperform prompt tuning but use more parameters. In contrast, we outperform prompt tuning while using similar number of parameters and are competitive with other finetuning approaches.

Our approach could be of independent interest for understanding intriguing properties of pretrained language models, the role of different parameters, and sharing parameters across layers. Ben Zaken et al. (2022); Cai et al. (2020) have shown that pretrained models can be finetuned by only updating the bias parameters, but unlike us, they do not share parameters. Gheini et al. (2021) finetune only cross attention layers for machine translation. Zhou et al. (2022b) share only output layers across tasks, but parameters across different

layers are not shared. Zhou et al. (2022a) have shown that task embeddings can be derived from task-specific finetuned parameters. The $\mathbf{z}$ in our approach can also be helpful as a task-embedding.

Parameters derived by fixed random transformations a few parameters have previously been used to study the task's intrinsic dimensionality (Li et al., 2018; Aghajanyan et al., 2021). Those works focus on weight matrices. While insightful, these are cumbersome to train for real-world deployment. Instead, we focus on bias or embeddings, providing a tractable operationalization for regular training and finetuning while using similar order of parameter count. For example, Aghajanyan et al. (2021) show that the intrinsic dimension of the QQP dataset with `RoBERTa-large` is 774, *i.e.*, at least 774 parameters are required to achieve within 90% of full finetuning performance. SLaSh achieves an F1-score of 83.2, more than 90% of full finetuning performance on QQP with 4.1K parameters ($92.2 \times 0.9 = 83.0$).

5 Conclusion

We introduce a multilayer LM finetuning technique where task-specific parameters are derived from a single vector. We show two instantiations of this technique — SLaSh and JR-WARP. SLaSh introduced shifts in the output activation of each transformer block, whereas JR-WARP inserted prompts in each transformer block. These methods require only a tiny fraction of the original language model parameters (similar to prompt-tuning) and outperform previous methods that use a similar number of per-task parameters. Despite the drastic reduction in the number of parameters, we demonstrate that these perform just as well as full finetuning for sentence and token classification tasks (only at max a 5% difference in performance). The high parameter efficiency leads to better training speed and resource utilization and improves private training.

6 Limitations

Experiments. In this work, we propose new methods for finetuning language models. We acknowledge that similar to previous approaches, our experiments are limited to English datasets and specific supervised tasks. However, our method does not use language- or task-specific tricks and should apply to other languages and tasks.

Method. As demonstrated in Section 3, SLaSh is computationally efficient and performs comparably to the full finetuning for small datasets. Moreover, its parameter and memory efficiency makes it an excellent private learner. However, it may underperform by a few points compared to full-finetuning larger datasets with higher intrinsic dimensionality due to using very few parameters. For example, SLaSh struggles with generative tasks such as text summarization, as generative tasks are more complex and involve making predictions over the whole vocabulary. In contrast, classification tasks have relatively fewer output labels. In our initial experiments, SLaSh reached a ROUGE-2 score of 12.93 on the XSum summarization task (Narayan et al., 2018) with pretrained `BART`, whereas full finetuning achieves a score of 21.94 (He et al., 2022).

The limitations of SLaSh are due to the small number of parameters it updates. Since shift is applied to only certain biases, the number of parameters can not be increased beyond a limit. However, we show that SLaSh is a more efficient and performant alternative to the methods that use a similar number of per-task parameters. Moreover, we showed that joint reparametrization improves parameter efficiency of other methods. As such, this principle can be extended to methods that are not restricted by a maximum limit on the number of parameters. For example, JR-WARP's parameters can be naturally increased by increasing the prompt length, which should improve the results further (details in Appendix A).

7 Ethics Statement

We propose a parameter-efficient method to tune transformer-based language models. The ethical implications are similar to the finetuning methods proposed before us. Our method improves parameter and computational efficiency, which should have an overall positive impact by reducing costs and enabling low-resource applications. Further, the positive private training results should encourage its adoption in real-world setups.

References

Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 308–318, New York, NY, USA. Association for Computing Machinery.

Armen Aghajanyan, Sonal Gupta, and Luke Zettle-

moyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328, Online. Association for Computational Linguistics.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. TinyTL: Reduce memory, not parameters for efficient on-device learning. In *Advances in Neural Information Processing Systems*, volume 33.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. 2022. Quantifying memorization across neural language models. *arXiv preprint arXiv:2202.07646*.

Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom B. Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. 2021. Extracting training data from large language models. In *USENIX Security Symposium*, pages 2633–2650.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407.

Mozhdeh Gheini, Xiang Ren, and Jonathan May. 2021. Cross-attention is all you need: Adapting pretrained Transformers for machine translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1754–1765, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Sivakanth Gopi, Yin Tat Lee, and Lukas Wutschitz. 2021. Numerical composition of differential privacy. In *Advances in Neural Information Processing Systems*, volume 34, pages 11631–11642. Curran Associates, Inc.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 2021. WARP: Word-level Adversarial ReProgramming. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4921–4933, Online. Association for Computational Linguistics.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Xuechen Li, Florian Tramer, Percy Liang, and Tatsunori Hashimoto. 2022. Large language models can be strong differentially private learners. In *International Conference on Learning Representations*.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2022a. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.* Just Accepted.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022b. P-tuning: Prompt tuning can be comparable to fine-tuning

across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68, Dublin, Ireland. Association for Computational Linguistics.

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

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Scott Yih, and Madian Khabsa. 2022. UniPELT: A unified framework for parameter-efficient language model tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6253–6264, Dublin, Ireland. Association for Computational Linguistics.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. AdapterHub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54, Online. Association for Computational Linguistics.

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

Jae Hun Ro, Theresa Breiner, Lara McConnaughey, Mingqing Chen, Ananda Theertha Suresh, Shankar Kumar, and Rajiv Mathews. 2022. Scaling language model size in cross-device federated learning. In *ACL 2022 Workshop on Federated Learning for Natural Language Processing*.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2021. AdapterDrop: On the efficiency of adapters in transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7930–7946, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Timo Schick and Hinrich Schütze. 2021. It's not just size that matters: Small language models are also few-shot learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2339–2352, Online. Association for Computational Linguistics.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online. Association for Computational Linguistics.

Nishant Subramani and Nivedita Suresh. 2020. Discovering useful sentence representations from large pretrained language models. *arXiv preprint arXiv:2008.09049*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Florian Tramer and Dan Boneh. 2021. Differentially private learning needs better features (or much more data). In *International Conference on Learning Representations*.

Marcos Treviso, Tianchu Ji, Ji-Ung Lee, Betty van Aken, Qingqing Cao, Manuel R. Ciosici, Michael Hassid, Kenneth Heafield, Sara Hooker, Pedro H. Martins, André F. T. Martins, Peter Milder, Colin Raffel, Edwin Simpson, Noam Slonim, Niranjan Balasubramanian, Leon Derczynski, and Roy Schwartz. 2022. Efficient methods for natural language processing: A survey. *arXiv preprint arXiv:2209.00099*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Mingbin Xu, Congzheng Song, Ye Tian, Neha Agrawal, Filip Granqvist, Rogier van Dalen, Xiao Zhang, Arturo Argueta, Shiyi Han, Yaqiao Deng, et al. 2022. Training large-vocabulary neural language models by private federated learning for resource-constrained devices. *arXiv preprint arXiv:2207.08988*.

Hao Yang, Junyang Lin, An Yang, Peng Wang, Chang Zhou, and Hongxia Yang. 2022. Prompt tuning for generative multimodal pretrained models. *arXiv preprint arXiv:2208.02532*.

Ashkan Yousefpour, Igor Shilov, Alexandre Sablayrolles, Davide Testuggine, Karthik Prasad, Mani Malek, John Nguyen, Sayan Ghosh, Akash Bharadwaj, Jessica Zhao, Graham Cormode, and Ilya Mironov. 2021. Opacus: User-friendly differential privacy library in PyTorch. *arXiv preprint arXiv:2109.12298*.

Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, Sergey Yekhanin, and Huishuai Zhang. 2022. Differentially private fine-tuning of language models. In *International Conference on Learning Representations*.

Wangchunshu Zhou, Canwen Xu, and Julian McAuley. 2022a. Efficiently tuned parameters are task embeddings. *arXiv preprint arXiv:2210.11705*.

Xin Zhou, Ruotian Ma, Yicheng Zou, Xuanting Chen, Tao Gui, Qi Zhang, Xuanjing Huang, Rui Xie, and Wei Wu. 2022b. Making parameter-efficient tuning more efficient: A unified framework for classification tasks. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 7053–7064, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Supplementary: Jointly Reparametrized Multi-Layer Adaptation for Efficient and Private Tuning

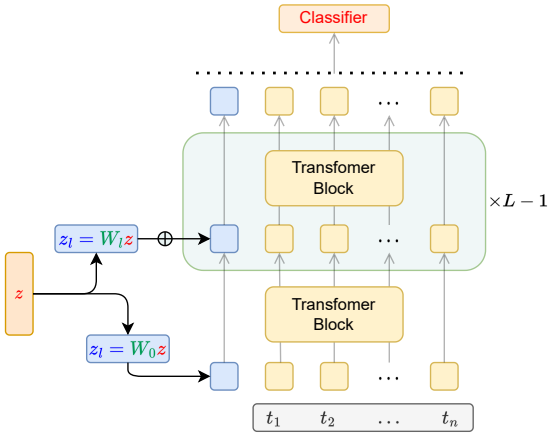

Figure 3: JR-WARP for sequence classification tasks with prompt length 1. Similar to Figure 2, trainable modules are indicated in red. $[t_i]_{i=1}^n$ denotes the sequence of tokens. The prompts are obtained by the projections $\mathbf{W}_l\mathbf{z}$. For the first layer, the prompt is appended to the token embeddings. We add the prompt to the transformed prompt from the previous block for the rest of the layers. $\mathbf{z}$ is shared across all the blocks.

A JR-WARP: Improved Prompt Tuning

Figure 3 summarizes JR-WARP with prompt length 1. We introduce prompts or embeddings in each transformer block, similar to Liu et al. (2022b). However, in our case, the prompts are reparametrized as random projections of a single vector $\mathbf{z} \in \mathbb{R}^d$.[5] The prompt is appended to the token embeddings for the first layer, *i.e.*, the embedding layer. Previous multi-layer prompt tuning approaches discard the transformed prompt from the previous layers and insert a new prompt at each layer (Lester et al., 2021; Liu et al., 2022b). Instead, from the second transformer block onwards, we do not discard previous representations and add the prompt to the resulting representation (or the transformed prompt) from the previous layer. $\mathbf{W}_l$ and $\mathbf{z}$ are initialized similarly to SLaSh.

WARP appends prompt only to the token embeddings, and in Figure 3, this can be achieved by keeping only the lower arm emitting from $\mathbf{z}$ block and setting $\mathbf{W}_0$ as the identity matrix. Figure 3 shows prompt length 1, but it can be extended to prompts longer than length 1. However, our main aim is to evaluate performance while using parameters similar to WARP. Therefore, we keep the prompt length to 1, and d is 10K and 5K in our experiments. When extending the prompt length to more than one, there are multiple ways to reparametrize prompts. For example, reparametrize prompts within the same layer from a single $\mathbf{z}$ or reparametrize prompts within the same index or time step from a single $\mathbf{z}$, as we have done in this work.

[5]This reparametrization differs from the generally suggested reparametrization of using an MLP encoder to transform the prompts.

B Ablations

Here we evaluate alternative hyperparameter choices for SLaSh by performing ablation studies concerning the position of shifts, initialization of parameters, and using `[MASK]` vs. `[CLS]` representations. Overall, our results are relatively less sensitive to these choices.

B.1 Adding shifts to other activations

In the main paper, we showed the results of adding shifts, *i.e.*, random projections from a trainable vector to the output layer's activation. These shifts can also be added to other activations, such as the activations after attention layers, intermediate feed-forward layers, or a combination of these. We evaluate these choices on tasks from the GLUE benchmark, and Table 7 summarizes our findings.

We find that the performance of shifting attention activations is similar to shifting output activations in most cases except for RTE and CoLA. Similar observations hold for intermediate activations. Shifting activations from intermediate feed-forward layers performed similarly for all tasks compared to output activations. These observations do not change when we increase the trainable parameters. Shifting output activations performed slightly better in terms of average performance computed across all tasks. Moreover, the intermediate activations have a higher dimension than the output activation (3,072 vs. 768 for `RoBERTa-base`). Therefore, intermediate activations required maintaining bigger random projection matrices ($\mathbf{W}_l$) during training.

In summary, other choices can perform similarly. We chose output activations due to their smaller dimension and transformers using layer norm immediately after it, which can take care of sudden

	Position	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
d = 1,024	`attention`	80.3	81.0	88.7	93.2	57.9	89.5	91.1	73.6	81.93
d = 1,024	`intermediate`	80.0	81.2	88.9	93.2	59.6	89.7	92.3	76.2	82.64
d = 1,024	`output`	80.4	80.9	89.3	93.1	59.5	89.3	91.7	77.6	82.72
d = 5K	`intermediate`	83.7	83.7	90.2	93.2	58.4	89.9	92.1	78.0	83.65
d = 5K	`output`	83.4	83.4	90.6	93.2	59.3	90.4	91.9	77.6	83.74

Table 7: Effect of adding shifts at different position on sequence classification tasks (GLUE Development set) with `RoBERTa-base` as the pretrained model. All the results are with `[CLS]` representations.

Initialization	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
$\mathbf{z}, \mathbf{W}_l \in \{\mathcal{N}, \mathcal{U}\}$	95.0	63.6	90.8	92.1	84.8	85.27
$\mathbf{z} = \mathbf{0}, \mathbf{W}_l \in \{\mathcal{N}, \mathcal{U}\}$	95.2	66.0	90.4	91.7	83.8	85.42
$\mathbf{z} \in \{\mathcal{N}, \mathcal{U}\}, \mathbf{W}_l = \mathbf{I}$	95.1	62.7	90.4	92.6	83.8	84.92

Table 8: Effect of different initialization of SLaSh parameters on sequence classification tasks (GLUE Development set) with `RoBERTa-large` as the pretrained model. All the results use `[MASK]` representations and d = 1,024.

drifts in activations, etc.

B.2 Initialization

Regarding the initialization of $\mathbf{z}$ and $\mathbf{W}_l$, we have several choices. $\mathbf{z}$ can be initialized randomly or with all zeros. Like Hambardzumyan et al. (2021), we report results with random initialization for $\mathbf{z}$ in the main paper. In particular, it is initialized as $\mathcal{N}(0, \sigma = \frac{1}{\sqrt{d}})$ or $\mathcal{U}(-\frac{1}{\sqrt{12d}}, \frac{1}{\sqrt{12d}})$. The projection matrices, $\mathbf{W}_l$, are also initialized randomly with identical distributions as $\mathbf{z}$. With these initialization choices, the variance of $\mathbf{z}_l$ is $\frac{1}{d}$ in each dimension. We consider the choice of Gaussian or Uniform initialization as a hyperparameter.

Table 8 shows the effect of different initialization on performance for sequence classification tasks. The results are relatively less sensitive to initialization. When both $\mathbf{z}$ and weight matrices are randomly initialized, the performance is better on STS-B, MRPC, and RTE than when $\mathbf{z}$ is initialized as all zeros. However, the average performance of all zeros is higher due to its performance being much higher on CoLA.

For the particular case of d = 1024, *i.e.*, the dimension of $\mathbf{z}$ is the same as the activations, we can initialize $\mathbf{W}_l$ as identity. In this case, all the blocks are shifted with the same vector. This performed similarly or worse on all tasks except MRPC. Random projections allow the model to select different parts of $\mathbf{z}$ for each transformer block. The above-mentioned result partly demonstrates the utility of using random projection matrices.

B.3 `[MASK]` vs. `[CLS]` Representations

As discussed in Section 3.1, we can use `[CLS]` or `[MASK]` representation for classification tasks. Table 9 compares this with `RoBERTa-base` and `RoBERTa-large` models. In terms of average performance, we find that `[MASK]` token representations are better or similar to `[CLS]` token representations.

The choice of representations mattered very little for bigger datasets (>10K samples), with the performance being similar for both choices. For smaller datasets, however, we do not see any clear patterns. On average, `[MASK]` token representation performed slightly better than `[CLS]` representation, echoing the observation of Hambardzumyan et al. (2021). So we use `[MASK]` representation for all the results in the main paper.

C Hyperparameters

Our implementation is based on the Hugging Face Transformers library (Wolf et al., 2020) and PyTorch 1.10 and 1.13. We use AdapterHub (Pfeiffer et al., 2020) for training LoRA and Adapter models. We use PyTorch-Opacus (Yousefpour et al., 2021) for private training. We mainly vary the learning rate and training epochs for all the methods. For SLaSh and JR-WARP, we consider one additional hyperparameter — Gaussian or Uniform initialization and disable all the dropout layers.

We use a similar training setup for *sequence classification* tasks as Hambardzumyan et al. (2021). We tune the learning rate in $\{1e^{-4}, 3e^{-4}, 1e^{-3}, 3e^{-3}, 1e^{-2}, 3e^{-2}\}$ and use

Model	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
				RoBERTa-base					
d = 1,024, [MASK]	80.8	80.9	89.8	92.9	57.6	89.5	91.0	78.7	82.65
d = 1,024, [CLS]	80.4	80.9	89.3	93.1	59.5	89.3	91.7	77.6	82.72
d = 5K, [MASK]	83.6	83.2	90.8	93.7	61.3	90.3	91.3	79.4	84.21
d = 5K, [CLS]	83.4	83.4	90.6	93.2	59.3	90.4	91.9	77.6	83.74
				RoBERTa-large					
d = 1024, [MASK]	86.2	83.3	92.2	95.0	63.6	90.8	92.1	84.8	86.01
d = 1024, [CLS]	86.3	83.1	92.3	95.1	61.6	90.5	92.6	82.3	85.47
d = 10K, [MASK]	89.1	85.6	93.6	95.9	65.5	91.8	91.8	85.6	87.33
d = 10K, [CLS]	89.1	85.7	93.6	95.8	64.0	91.7	91.8	86.6	87.29

Table 9: Comparing SLaSh with [MASK] and [CLS] token representation on sequence classification tasks (GLUE Development set).

Task	d = 1,024			d = 2,048			d = 10K		
	Initialization	LR	# Epoch	Initialization	LR	# Epoch	Initialization	LR	# Epoch
RTE	$\mathcal{N}$	$3e^{-2}$	10	$\mathcal{U}$	$1e^{-2}$	20	$\mathcal{U}$	$1e^{-2}$	10
MRPC	$\mathcal{U}$	$1e^{-2}$	20	$\mathcal{U}$	$1e^{-2}$	10	$\mathcal{U}$	$1e^{-2}$	20
STSB	$\mathcal{U}$	$3e^{-3}$	10	$\mathcal{U}$	$1e^{-2}$	10	$\mathcal{U}$	$3e^{-3}$	10
CoLA	$\mathcal{N}$	$3e^{-3}$	20	$\mathcal{U}$	$1e^{-2}$	10	$\mathcal{N}$	$1e^{-2}$	10
SST-2	$\mathcal{N}$	$3e^{-3}$	10	$\mathcal{N}$	$1e^{-3}$	20	$\mathcal{N}$	$3e^{-3}$	10
QNLI	$\mathcal{U}$	$3e^{-3}$	20	$\mathcal{N}$	$3e^{-3}$	20	$\mathcal{U}$	$1e^{-3}$	10
QQP	$\mathcal{U}$	$3e^{-3}$	20	$\mathcal{N}$	$1e^{-3}$	20	$\mathcal{N}$	$1e^{-3}$	20
MNLI	$\mathcal{N}$	$3e^{-4}$	10	$\mathcal{N}$	$1e^{-3}$	20	$\mathcal{U}$	$1e^{-3}$	20

Table 10: Hyperparameters of best-performing SLaSh models for sequence classification with RoBERTa-large. Results shown in Table 2.

Task	d = 1,024			d = 5K		
	Initialization	LR	# Epoch	Initialization	LR	# Epoch
RTE	$\mathcal{U}$	$1e^{-2}$	20	$\mathcal{U}$	$1e^{-2}$	10
MRPC	$\mathcal{N}$	$3e^{-2}$	10	$\mathcal{N}$	$1e^{-2}$	10
STSB	$\mathcal{N}$	$1e^{-2}$	10	$\mathcal{N}$	$1e^{-2}$	20
CoLA	$\mathcal{U}$	$3e^{-3}$	10	$\mathcal{N}$	$1e^{-2}$	10
SST-2	$\mathcal{N}$	$1e^{-3}$	10	$\mathcal{U}$	$1e^{-2}$	10
QNLI	$\mathcal{U}$	$3e^{-3}$	20	$\mathcal{U}$	$3e^{-3}$	20
QQP	$\mathcal{N}$	$1e^{-3}$	20	$\mathcal{N}$	$3e^{-3}$	20
MNLI	$\mathcal{N}$	$1e^{-3}$	10	$\mathcal{N}$	$1e^{-3}$	20

Table 11: Hyperparameters of best-performing SLaSh models for sequence classification with RoBERTa-base. Results shown in Table 3.

Task	RoBERTa-base (d = 5K)			RoBERTa-large (d = 10K)		
	Initialization	LR	# Epoch	Initialization	LR	# Epoch
RTE	$\mathcal{N}$	$1e^{-2}$	10	$\mathcal{U}$	$1e^{-2}$	20
MRPC	$\mathcal{U}$	$3e^{-3}$	10	$\mathcal{N}$	$1e^{-2}$	10
STSB	$\mathcal{N}$	$1e^{-2}$	20	$\mathcal{N}$	$1e^{-2}$	20
CoLA	$\mathcal{N}$	$1e^{-2}$	10	$\mathcal{N}$	$1e^{-2}$	20
SST-2	$\mathcal{U}$	$3e^{-3}$	10	$\mathcal{U}$	$1e^{-2}$	20
QNLI	$\mathcal{U}$	$3e^{-3}$	20	$\mathcal{N}$	$1e^{-2}$	20
QQP	$\mathcal{U}$	$3e^{-3}$	20	$\mathcal{N}$	$3e^{-3}$	20
MNLI	$\mathcal{N}$	$3e^{-3}$	20	$\mathcal{N}$	$1e^{-3}$	20

Table 12: Hyperparameters of best-performing JR-WARP models for sequence classification with RoBERTa. Results shown in Tables 2 and 3.

Task	RoBERTa-base			RoBERTa-large		
	Initialization	LR	Grad. Clip Threshold	Initialization	LR	Grad. Clip Threshold
SST-2	$\mathcal{U}$	$3e^{-3}$	0.1	$\mathcal{U}$	$1e^{-3}$	1.0
QNLI	$\mathcal{U}$	$1e^{-2}$	1.0	$\mathcal{N}$	$1e^{-2}$	0.1
QQP	$\mathcal{N}$	$3e^{-3}$	1.0	$\mathcal{N}$	$3e^{-3}$	1.0
MNLI	$\mathcal{U}$	$3e^{-3}$	1.0	$\mathcal{U}$	$3e^{-3}$	1.0

Table 13: Hyperparameters of best-performing SLaSh models for private training. Results shown in Table 6.

Task	RoBERTa-base			RoBERTa-large		
	Initialization	LR	Grad. Clip Threshold	Initialization	LR	Grad. Clip Threshold
SST-2	$\mathcal{N}$	$1e^{-2}$	0.1	$\mathcal{N}$	$1e^{-2}$	1.0
QNLI	$\mathcal{U}$	$1e^{-2}$	1.0	$\mathcal{N}$	$1e^{-2}$	1.0
QQP	$\mathcal{N}$	$1e^{-2}$	1.0	$\mathcal{U}$	$1e^{-2}$	1.0
MNLI	$\mathcal{U}$	$1e^{-2}$	1.0	$\mathcal{U}$	$1e^{-2}$	0.1

Table 14: Hyperparameters of best-performing JR-WARP models for private training. Results shown in Table 6.

a linear learning rate scheduler with a warmup ratio of 0.06. We train for 10 or 20 epochs with a batch size of 8, and the gradient magnitudes are clipped to 1.0. Tables 10 and 11 and Table 12 list the best hyperparameters for each task for SLaSh and JR-WARP, respectively. We find the best hyperparameters based on the performance on the validation set from a single training run. Then to report the error bars (in the main paper), we train several models with those best-found hyperparameters but with different random seeds.

For *token classification* tasks, we tune the learning rate in $\{1e^{-4}, 3e^{-4}, 1e^{-3}, 3e^{-3}, 1e^{-2}, 3e^{-2}\}$ with a linear learning rate scheduler and use a warmup ratio of 0.1. We train for 5 epochs with a batch size of 32. The best result for SLaSh is obtained with uniform initialization and a learning rate of 0.01. The best result for JR-WARP is obtained with normal initialization and a 0.03 learning rate.

For *private training*, we replicated the setup of Yu et al. (2022) as much as possible. In particular, we tune the learning rate in $\{1e^{-3}, 3e^{-3}, 1e^{-2}\}$ without any scheduler and train for 20 epochs. We used a batch size of 2048 and considered two per sample gradient clipping thresholds — 0.1 and 1.0. We use the PRV accountant of Gopi et al. (2021) for privacy accounting, the same as Yu et al. (2022), to keep the results comparable. Based on this accountant, the Gaussian noise magnitudes for MNLI, QQP, QNLI, and SST-2 were 0.643, 0.651, 0.831, and 0.925. Table 13 and Table 14 list the best hyperparameters for SLaSh and JR-WARP.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Section 6

☑ A2. Did you discuss any potential risks of your work?
Section 7

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Abstract & Section 1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

- We have mentioned the main Python libraries used in this work in Appendix C. - The code to reproduce our results is available at https://github.com/umgupta/jointly-reparametrized-finetuning.

☑ B1. Did you cite the creators of artifacts you used?
We have cited the methods, datasets, and libraries used in this work throughout the paper,

☒ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
We use software commonly used by researchers in this field and is freely available. Researchers are generally familiar with these, so their licensing needs no discussion.

☒ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
- Our code is released with the appropriate license. - We use software commonly used by researchers in this field and is freely available. Researchers are generally familiar with these, so their licensing needs no discussion.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Not applicable. We used publicly available standard datasets in our work.

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Sections 3 and 6

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Section 3

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

C ☑ Did you run computational experiments?

Section 3

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
- Section 3, specifically Section 3.3 - Appendix C

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 3 and Appendix C

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
- For sequence classification tasks, we report the results of a single run for baselines and the mean performance of 5 training runs for our methods. - We report the results of a single training run for other experiments and tasks.

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Appendix C and Section 3

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.