# Sparsity May Be All You Need: Sparse Random Parameter Adaptation

**Jesus Rios, Pierre Dognin, Ronny Luss, Karthikeyan Natesan Ramamurthy**
IBM Research
{jriosal, pdognin, rluss, knatesa}@us.ibm.com

## Abstract

Full fine-tuning of large language models for alignment and task adaptation has become prohibitively expensive as models have grown in size. Parameter-Efficient Fine-Tuning (PEFT) methods aim at significantly reducing the computational and memory resources needed for fine-tuning these models by only training on a small number of parameters instead of all model parameters. Currently, the most popular PEFT method is the Low-Rank Adaptation (LoRA), which freezes the parameters of the model and introduces a small set of trainable parameters in the form of low-rank matrices. We propose simply reducing the number of trainable parameters by randomly selecting a small proportion of the model parameters to train on, while fixing all other parameters, without any additional prior assumptions such as low-rank structures. In this paper, we compare the efficiency and performance of our proposed approach to other PEFT methods as well as full parameter fine-tuning. We find our method to be competitive with LoRA when using a similar number of trainable parameters. Our findings suggest that what truly matters for a PEFT technique to perform well is not necessarily the specific adapter structure, but rather the number of trainable parameters being used.

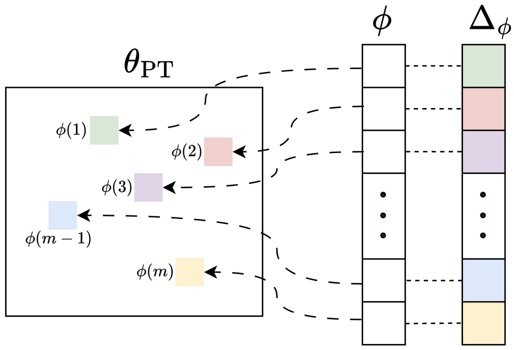


Figure 1: The proposed `SpaRTA` method which randomly chooses a small subset of parameters from $\theta_{\text{PT}}$, stores the indices of the selected parameters in $\phi$ and updates the model via adapter $\Delta_\phi$.

## 1 Introduction

It has become common practice to train application-ready language models in two phases (Radford et al., 2018; Kenton and Toutanova, 2019): first, the model is *pre-trained* on a very large and general corpus of (unlabeled) text; then further trained (or *fine-tuned*) on a smaller specific set of examples demonstrating the intended behavior for a particular application, such as instruction following (Ouyang et al., 2022).

Overall, supervised fine-tuning (SFT) requires less computational resources than pre-training (PT) due to the significantly smaller size of the training set as well as the typical use of *early stopping* to deal with overfitting. This means orders-of-magnitude less gradient computations and parameter updates are needed during SFT compared to PT. However, a major drawback is that memory requirements remain the same, unless a parameter-efficient fine-tuning (PEFT) technique is used. The main memory bottleneck during training is the number of trainable parameters, since additional memory must be allocated for their gradients and other per-parameter statistics needed by the optimizer. The idea behind PEFT (Lialin et al., 2023) is to significantly reduce the number of trainable parameters during fine-tuning while maintaining performance.

Low Rank Adaptation (`LoRA`), first introduced by Hu et al. (2022), currently remains the most popular PEFT technique. `LoRA` freezes all the pre-trained model parameters $\theta_{\text{PT}}$ and introduces trainable low-rank matrices (e.g., $B$, $A$) to represent the changes ($\Delta = BA$) needed for adapting the model to a new task. The adapted model parameters are given by $\theta_{\text{PT}} + \Delta$. Memory and computational efficiency are achieved by optimizing only over the parameters of these newly added, but significantly smaller, matrices.



*Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 18650–18666
November 4-9, 2025 ©2025 Association for Computational Linguistics

The success of LoRA begs one to ask what properties make this method perform well. Is the low-rank structure critical, i.e., does $\Delta$ need to be low-rank? Is it sufficient to constrain $\Delta$ to be low dimensional? A main goal of this paper is to investigate these questions. An abundance of research is going into new methods for structured $\Delta$ (see Section 3 below) and novel directions into unstructured methods for fine-tuning could open avenues in areas such as model merging (Wortsman et al., 2022; Matena and Raffel, 2022) or pluralistic alignment (Feng et al., 2024).

In this work, we propose a different approach where $\Delta$ is not factorized into low rank matrices but rather chosen to be a random subset of the model parameters. This Sparse Random parameTer Adaptation (SpaRTA) method imposes a sparsity constraint on the adapter that can be easily controlled. By changing the desired sparsity, one can change the number of adaptation parameters. Regardless of how the selected parameters are sampled from the model parameters, subsequent updates only affect these parameters. This sparsity constraint and randomness of selected parameters is in contrast to techniques such as LoRA that effectively affect all parameters in $\theta_{\text{PT}}$. See Figure 1 for an illustration of the method. Generally, one samples $m$ parameters from the pre-trained model $\theta_{\text{PT}}$, stores their indices in $\phi$, and uses adapter $\Delta_\phi$ to fine-tune $\theta_{\text{PT}}$.

To investigate the performance of SpaRTA, we build adapters with different sparsity levels and evaluate them on a wide range of natural language understanding benchmarks. SpaRTA is compared to other PEFT approaches including LoRA and DoRA (Liu et al., 2024), and found to be quite competitive compared to these methods given that it only modifies a small sparse number of model parameters.

## 2 Motivation

Yu et al. (2024) look at the differences between a language model's parameters before and after fine-tuning, and demonstrate empirically that it is possible to randomly drop (i.e., set to zero) up to 99% of these parameter changes, represented by $\Delta$, without significantly affecting model performance.

This motivates our approach, SpaRTA, which produces a performant fine-tuned model by directly adapting only a small percentage of the pre-trained model parameters. SpaRTA randomly selects the (scalar) parameters to train and freezes the remaining parameters (i.e., setting the corresponding $\Delta$ values to zero). This $\Delta$-sparsity also helps in reducing overfitting, as pre-trained models typically have more than enough capacity to learn the often limited amount of (labeled) data used for fine-tuning. There is no guarantee of $\Delta$-sparsity in LoRA, but this is a desired property since it reduces parameter interference (Yadav et al., 2023) when merging fine-tuned models.

SpaRTA, like LoRA, reduces the number of gradient computations during training (when compared with full fine-tuning), and ultimately has the same inference cost as the original model (after merging back the sparse $\Delta$ into the model). For an adaptation technique, having lower memory and computation needs during training as well as identical inference time to the original model are all quite desirable properties. SpaRTA has all these properties plus the unique added benefit of producing only sparse changes in a small number of the parameters of the original model that can be beneficial for merging multiple SpaRTA adapters.

### 2.1 Is Low-Rank Adaption Necessary?

In Appendix A, we show that the changes (i.e., $\Delta$) in weight matrix during full parameter fine-tuning are, in fact, not generally low-rank for capable models such as `gemma-2b-it` and `mistral-7b-it`. This indicates that LoRA works, not particularly because of its low-rank constraint, but rather due to the reduction in model capacity achieved by LoRA when fine-tuning on limited training data, as it is typically done in task adaption. Such insight also hints that any constraint reducing the capacity of the original model could perform competitively, motivating our approach which selects a small number of parameters from the original model to be updated during training.

## 3 Related Work

The last few years has seen many advances in PEFT methods. Perhaps the most well-known and used method in practice is LoRA (Hu et al., 2022), which has spurred many variants including: DoRA, which adapts only the directions of pre-trained weights (Liu et al., 2024); VeRA, which shares low-rank matrices across layers (Kopiczko et al., 2024); AdaLoRA, which adaptively allocates parameter budgets among weight matrices according to their importance (Zhang et al., 2023); and SoRA, which dynamically adjusts the intrinsic rank during adaptation (Ding et al., 2023). Each of these methods has a different structure to the $\Delta$ being optimized, with the commonality being the training of some function of low-rank matrices. AdaLoRA and SoRA are also dynamic methods, which adjust the function during adaptation.

Beyond adding structured parameters is the concept of fine-tuning a small subset of the total parameters of the model, i.e., sparse fine-tuning, where one must first decide which subset of parameters to fine-tune (similar to deciding which parameters in each layer to add adapters to in LoRA). Ansell et al. (2024) start with an initial set of parameters and offer procedures to *drop* (according to parameter magnitude) and *grow* (according to gradient) the set, i.e., they learn the set of parameters to train. Alternatively, Ma et al. (2024) focus on the sparsity of neuron activations during training by precomputing neuron importance scores and only including *important* neurons in computations during training. Ansell et al. (2022) first fine-tune on all parameters, select the parameters that change the most, and then fine-tune again from scratch on the selected parameters. Deng et al. (2025) proposes improvements via a modified pruning strategy to the random dropping and rescaling method of Yu et al. (2024) which motivates this work. In contrast to these more complex, sometimes dynamic approaches, our proposed SpaRTA method produces a performant fine-tuned model by directly adapting only a small percentage of the pre-trained model parameters chosen completely at random.

Yet another related direction is that of compression which results in sparse models; algorithms in this genre take a dense fine-tuned model with the goal of compressing it while maintaining performance. Compression (see Zhu et al. (2024) for a survey) could be accomplished by quantization, low-rank approximation, pruning (i.e., removing neurons, attention heads, or even layers), or distillation. The focus of this paper, however, is on fine-tuning dense models rather than learning sparse models as in Liu et al. (2023).

## 4 SpaRTA: Adapting a Random Subset of Model Parameters

Suppose the parameters of a pre-trained language model are $\theta_{\text{PT}} \in \mathbb{R}^n$ where $n$ is the number of parameters, and full parameter fine-tuning (FT) is performed with a labeled dataset characterizing a task. FT typically updates $\theta_{\text{PT}}$ using a stochastic first-order gradient-based optimization algorithm, e.g., Adam from Kingma and Ba (2015), to maximize the conditional probability of the labels in the training dataset under the model. The FT model is then given by $\theta_{\text{FT}} = \theta_{\text{PT}} + \Delta_{\text{FT}}$ where $\Delta_{\text{FT}} \in \mathbb{R}^n$.

This has two drawbacks in terms of memory efficiency. First, the optimizer state can be large, e.g., the state of the Adam optimizer is 4 times as large as the parameter space as it includes current parameter values, their gradients as well as per parameter statistics of those gradients. Second, storing a new FT model takes as much memory as the PT model which could be an issue when many task-specific models are requested.

SpaRTA proposes to *randomly* select a small subset of the model parameters to optimize while freezing the rest. The model parameters $\theta \in \mathbb{R}^n$ are partitioned into trainable $\theta_{\text{T}} \in \mathbb{R}^m$ and frozen $\theta_{\text{F}} \in \mathbb{R}^{n-m}$ ones, with $m \ll n$ being the number of selected parameters. This allows our approach to have a drastically lower memory footprint than FT by reducing the size of the optimizer state as well as faster training by reducing the number of gradients to compute. This is similar with respect to LoRA in that both reduce the memory footprint by optimizing a small set of parameters (the adapter) while freezing most (SpaRTA) or all (LoRA) of the model parameters.

In our SpaRTA implementation[1], we introduce: (i) non-trainable *indices* $\phi \in \mathbb{R}^m$ containing the indices of the randomly selected elements in $\theta_{\text{PT}}$, and (ii) trainable parameters $\Delta_\phi \in \mathbb{R}^m$ representing the subset of $\Delta$ that SpaRTA learns at indices $\phi$. The pseudocode for SpaRTA is given in Algorithm 1.

Whereas optimizing LoRA requires computing gradients with respect to the $A$ and $B$'s used to compute $\Delta$ (which constitute additional parameters, independent from $\theta_{\text{PT}}$), SpaRTA requires computing gradients with respect to $\Delta_\phi$, the changes over a subset of parameters chosen from $\theta_{\text{PT}}$ and indexed by $\phi$.

Generating the index set $\phi$ in SpaRTA is done by sampling from a Bernoulli independently and including each scalar parameter in $\phi$ with probability $m/n$. Hence, SpaRTA uses $m$ trainable parameters out of $n$ in expectation. Finally, inference is performed, similarly to LoRA, by merging the fine-tuned $\Delta$ into $\theta_{\text{PT}}$ and then making the necessary forward passes. Thus, SpaRTA does not introduce any additional *inference* latency.

[1]Code is available at https://github.com/IBM/SpaRTA.

**Algorithm 1: SpaRTA**

**Input:** Pre-trained model with $\theta_{\text{PT}} \in \mathbb{R}^n$
**Input:** Task labeled dataset $\mathcal{D}$
**Sample** $\phi$**:** Indices of (scalar) parameters to be optimized
**Initialize:** $\Delta_\phi = 0$
**while** *validation loss has not converged* **do**
 **1. Merge:** $\theta_\phi = \theta_\phi + \Delta_\phi$
 **2. Compute loss:** Continue forward pass from $\theta_\phi$ using a batch of labeled data from $\mathcal{D}$
 **3. Compute gradients** with respect to $\Delta_\phi$ using backpropagation
 **4. Unmerge:** $\theta_\phi = \theta_\phi - \Delta_\phi$, without recording this operation in the computational graph
 **5. Update** $\Delta_\phi$ with the Adam Optimizer
**Output:** $\phi$, $\Delta_\phi$

## 5 Memory Usage

Recall that SpaRTA freezes $n - m$ ($m \ll n$) of the model parameters. We define sparsity as $s = 1 - m/n \in (0, 1)$, the percentage of frozen model parameters (e.g., if $1\%$ of parameters are trainable, then the sparsity is $99\%$). Subsequently, density is defined as $k = m/n = 1-s$, the percentage of trainable model parameters. In practice, for a chosen sparsity $s$, one can freeze a model parameter with probability $s$, expecting a total sparsity percentage of $s$ over all model parameters. Thus, in expectation, $k = m/n$ percent of the model parameters are chosen as trainable, for a total of $n\,k = m$ trainable parameters.

For SpaRTA, only $\Delta_\phi$ (of size $m$) is trainable, which is significantly smaller than the total number of model parameters since $m \ll n$. However, the indices of these randomly chosen parameters must be recorded into $\phi$, adding to the memory requirements. Indices can be stored in 16-bit integers for all the Transformer models (Vaswani et al., 2017) considered in this paper.

SpaRTA sparsifies neither the model *head* (which is kept fully trainable) nor the *embeddings* (which is kept frozen during training). The parameters in Transformer networks consist of bias vectors and two-dimensional weight matrices. Storing indices for all these trainable parameters would require at most $m\,(2 \times 16)$ bits of memory ($m$ parameters, two integers to index a 2-dimensional matrix, 16 bits per integer).

The values in $\Delta_\phi$ are of the same type as the model parameters, e.g., using 16-bit brain floating-point (`bfloat16`) tensors. Thus, SpaRTA requires up to $m\,(2 \times 16 + 16) = 3m \times 16$ bits of extra memory to specify the index $\phi$ and delta $\Delta_\phi$ tensors. That is $3\,k$ times more memory than the original model, which requires just $n \times 16$ bits for storing its parameters. For instance, using SpaRTA on a model with sparsity 80%, 90%, 95% and 99% would require up to 60%, 30%, 15% and 3% more memory, respectively.

We next analyze[2] SpaRTA's memory savings during training. Optimizing the full set of model parameters (FT) using Adam (Kingma and Ba, 2015) requires memory for the parameters, their gradients, and their (adaptive) first and second moments estimates. This requires $4\,n \times 16$ bits of memory when using a `bfloat16` representation.

In contrast, SpaRTA only optimizes $\Delta_\phi$, requiring a total of $m\,(4 \times 16 + 2 \times 16) + n \times 16$ bits of memory: (i) $m\,(4 \times 16)$ bits needed by Adam to optimize $\Delta_\phi$ of size $m$, (ii) $m\,(2 \times 16)$ bits for $\phi$ with the indices identifying the model parameters associated with $\Delta_\phi$, and (iii) the PT model parameters ($n \times 16$ bits). Memory savings appear then if and only if

$$\begin{aligned} m\,(4{\times}16 + 2{\times}16) + n \times 16 &< 4n{\times}16 \\ kn\,(6{\times}16) &< 3n{\times}16 \\ k &< 0.5, \end{aligned} \tag{1}$$

that is, SpaRTA is a real PEFT method iff $k < 0.5$, i.e., a sparsity higher than 50% is required ($s > 0.5$), making less than 50% of the model parameters trainable. For instance, using SpaRTA on a model with sparsity $s$ = 80%, 90%, 95%, and 99% requires 45%, 60%, 67.5% and 73.5% less memory than full parameter FT, respectively, see Table 1. Additional savings are possible regarding storing indices given a fixed random number generator; in this case, the random path of selected parameters to train can be derived given a fixed seed.

## 6 Experimental Setup

We next detail our experimental framework. Motivation is first given for the tasks followed by a description of models to be used and how they are adapted (with manipulations) to the tasks.

[2]This analysis does not include memory requirements associated with model buffers (e.g., those used to track running statistics for layer normalization) because they are relatively small and the same for both the Full FT and SpaRTA.

| Model | storage | Full FT | SpaRTA | | | | |
|---|---|---|---|---|---|---|---|
| $n$ | (on disk) | 0% | 50% | 80% | 90% | 95% | 99% |
| 2B | 4 | 16 | 16 | 8.8 | 6.4 | 5.2 | 4.2 |
| 7B | 14 | 56 | 56 | 30.8 | 22.4 | 18.2 | 14.8 |

Table 1: SpaRTA memory usage efficiency during training for $n$ = 2B and 7B parameter models. Memory in Gigabytes (GB) for storing the parameters on disk (storage) as well as for FT with Adam on (i) the full set of parameters (Full FT) equivalent to 0% sparsity (ii) a sparse random subset (SpaRTA) for different sparsity percentages from 50% to 99%.

### 6.1 Tasks

The main experimental focus is on Natural Language Understanding (NLU) tasks, specifically *sequence classification*, which involves classifying natural language sequences into a given number of classes. NLU tasks are easier to evaluate than Natural Language Generation (NLG) tasks, as one can use simple metrics (Accuracy, F1-scores, etc.) that can avoid the inherent ambiguity of evaluating more general generative tasks. While SpaRTA is also applicable to NLG tasks, they are not used in the following demonstrations due to the challenges associated with their evaluation, typically requiring human judgments as the gold standard measure of performance. Human evaluations can be expensive and time consuming, do not scale easily, and are not as *objective* as NLU evaluations.

### 6.2 Language Models

Starting with available open-weight language models, the goal is to adapt them to perform a selected NLU task. Two types of trained models are used: *base* and *instruction*-tuned models, where the latter have additionally been trained to follow users' intent when prompted, such as answering questions or responding to instructions. Specifically, we consider the following language models: `gemma-2b` and `gemma-2b-it` from the Gemma family (Team et al., 2024), and `mistral-7b` and `mistral-7b-it` from the Mistral[3] family (Jiang et al., 2023). The `"it"` suffix refers to instruction-tuned models. They are of particular interest for our experiments as they will show results for models with two different numbers of parameters. All models are readily available text-to-text, decoder-only transformer models with open weights, downloadable from Hugging Face.

Note that when using an *instruction*-following (`it`) model, inputs are formatted according to the conventions established when training the model on instructions.

[3]We use v0.3 for both models.

### 6.3 Task Adaptation

When using a *base* model for a sequence classification task, the raw text to be classified is input directly into the model as a sequence of tokens (with some special token if necessary to deal with structured inputs). However, when using an *instruction* model, these sequences are first wrapped into a classification-specific instruction. For example, a possible instruction could be: "Determine if the following sentence has a positive sentiment. Respond Yes or No.", followed by the sequence itself to be classified.

A (generative) pre-trained Transformer model with a decoder-only architecture has a head that transforms the final hidden state of each token in the input sequence into vocabulary logits. Efficiently adapting this model for sequence classification requires the swap of this head for a sequence classification head, which uses only the final hidden state of the last token in the input sequence $h \in \mathbb{R}^d$ to do the classification. This reduces the parameter size of the head, which is just a linear layer applied to $h$, from a weight matrix $W \in \mathbb{R}^{v\times d}$ to $W \in \mathbb{R}^{c\times d}$, where $d$ is the dimension of the model's hidden states (e.g., 2,048 and 4,096 for the `gemma-2b` and `mistral-7b` models respectively), $v$ is the number of tokens in the vocabulary (e.g., 256,000 for `gemma-2b` or 32,768 for `mistral-7b`), and $c$ is the number of classes (e.g. 2 to 4 in the experiments that follow). With this small change, the model outputs classification probabilities for each input sequence through

$$p = \text{softmax}(h\, W^T) \in \mathbb{R}^c. \qquad (2)$$

The classification heads of our *base* pre-trained models (i.e., `gemma-2b` and `mistral-7b`) are initialized with random weights. The weights of the the original vocabulary heads of instruction-tuned models (i.e., `gemma-2b-it` and `mistral-7b-it`) are rather reused when initializing their classification heads. To do so requires to first identify the tokens in the vocabulary that the model is expected to use for classification following the instruction. For example, these could be the tokens associated with a "Yes" or "No" response. The embeddings in the original model (head) associated with those classification tokens are extracted and used to initialize the classification head. While many models

| Dataset | Classes | Train | Dev | Test |
|---|---|---|---|---|
| IMDB | 2 | 25,000 | 5,000 | 20,000 |
| COLA | 2 | 7,551 | 1,000 | 1,043 |
| MNLI | 3 | 100,000 | 10,000 | 19,647 |
| MRPC | 2 | 3,668 | 408 | 1,725 |
| QNLI | 2 | 99,725 | 5,000 | 5,463 |
| QQP | 2 | 100,000 | 5,000 | 40,430 |
| RTE | 2 | 2,182 | 300 | 277 |
| SST-2 | 2 | 66,349 | 1,000 | 872 |
| BoolQ | 2 | 9,427 | 1,270 | 2,000 |
| MMLU | 4 | 99,842 | 1,531 | 14,042 |

Table 2: Sequence classification datasets. Training sets limited to 100K samples. Training samples with > 256 tokens are removed (here using `gemma-2b` tokenizer, with `mistral-7b` tokenizer in Appendix B).

tie their vocabulary heads to their tokens embedding matrices, these new classification heads are never tied to the model's input embedding matrix.

## 7 Experimental Results

Efficacy of `SpaRTA` is demonstrated empirically by comparing it to two baselines, `LoRA` (only the additional matrices representing low-rank adapters are optimized) and `DoRA` (magnitude is optimized in addition to low-rank adapters). We also explore the performance of `SpaRTA` on a range of sparsity levels, varying the number of trainable parameters.

We consider several NLU benchmarks, including IMDB (Maas et al., 2011), GLUE (Wang et al., 2019), BoolQ (Clark et al., 2019) and MMLU (Hendrycks et al., 2020). See Appendix B for detailed descriptions. Table 2 summarizes these datasets and our splits for training, development, and testing. A detailed description of the training setup can be found in Appendices C and E. All results are averaged over 3 random seeds.

### 7.1 IMDB

Table 4 shows the results for IMDB, where each model is asked to classify a review as positive or negative. Each model is fine-tuned using the following adaptation methods: (i) Full parameter fine-tuning (Full FT) where all model parameters are optimized; (ii) `SpaRTA` for different density levels 5%, 0.5%, 0.05%, with the last allowing `SpaRTA` to have approximately the same number of trainable parameters as `LoRA`; (iii) `LoRA` with rank $r = 8$, equivalent to about $0.05\%$ of trainable parameters compared to the model full parameter size.

| Sparsification targets | Loss | Accuracy |
|---|---|---|
| Wq, Wv (like LoRA) | 0.117 | 96.3% |
| **Wv, Wo** | **0.109** | **96.7%** |
| Wq, Wk, Wv | 0.123 | 95.9% |
| Wq, Wk, Wo | 0.112 | 96.0% |
| Wq, Wq, Wv, Wo | 0.110 | 96.5% |
| MLP | 0.150 | 94.7% |
| Wq, MLP | 0.150 | 94.5% |
| Wk, MLP | 0.150 | 95.1% |
| Wv, MLP | 0.145 | 95.2% |
| Wo, MLP | 0.140 | 95.6% |
| Wq, Wk, MLP | 0.150 | 94.6% |
| Wv, Wo, MLP | 0.136 | 95.8% |
| W, MLP, norm | 0.140 | 95.4% |

Table 3: Loss and accuracy on the SST-2 task after applying our `SpaRTA` approach to different types of parameters in the `gemma-2b` model, given that the same number of trainable parameters are selected. Results are averaged across 10 random seeds.

In Table 4, results for adaptation methods (rows) are sorted in order of descending density as to show the impact of decreasing the number of trainable parameters on the overall performance. For `gemma-2b` and `gemma-2b-it`, Full FT adaptation gives (practically always) the best test loss and accuracy numbers. This is expected since all model parameters are fine-tuned. `SpaRTA` results at 5% density are close to Full FT, even matching them for `gemma-2b-it`. As the density decreases by orders of magnitude, the results slowly degrade. For 0.05% density, `gemma` models' performances match or slightly lag performance from `LoRA`.

For `mistral` models, the trend is similar with Full FT showing best (or close to best) loss and accuracy numbers. `SpaRTA` performs well, matching and even improving upon Full FT with a density of 0.05%. `SpaRTA` even improves over `LoRA` for both `mistral-7b` and `mistral-7b-it` at this low density. Overall these results are encouraging; `SpaRTA` is competitive, provides similar performance to `LoRA` for low densities, and does not degrade the performance of full parameter fine-tuning.

### 7.2 Which Parameters Should `SpaRTA` Target?

So far, `SpaRTA` has chosen trainable parameters from any layer of the transformer with equal probability. In contrast, `LoRA` concentrates adaption on the key and value self-attention weight matrices.

Now, we investigate whether it is better to con-

| | | gemma-2b | | gemma-2b-it | | mistral-7b | | mistral-7b-it | |
|---|---|---|---|---|---|---|---|---|---|
| method | density (%) | loss | acc. | loss | acc. | loss | acc. | loss | acc. |
| **PT (zero-shot)** | | – | – | 0.425 | 92.0% | – | – | 0.435 | 86.1% |
| **Full FT** | 100% | 0.092 | 96.9% | 0.107 | 96.3% | 0.080 | 97.4% | 0.080 | 97.3% |
| **SpaRTA** | 5% | 0.096 | 96.7% | 0.105 | 96.3% | 0.084 | 97.2% | 0.081 | 97.5% |
| **SpaRTA** | 0.5% | 0.096 | 96.7% | 0.103 | 96.4% | 0.087 | 96.9% | 0.087 | 97.0% |
| **SpaRTA** | 0.05% | 0.106 | 96.3% | 0.114 | 96.2% | 0.080 | 97.3% | 0.076 | 97.4% |
| **LoRA** | ≈0.05% | 0.101 | 96.5% | 0.113 | 96.2% | 0.086 | 97.1% | 0.081 | 97.1% |

Table 4: Test loss and accuracy of models adapted to the IMDB dataset with different fine-tuning methods. We also report zero-shot performance of instruction following PT models. For training details, see Table 19.

**Model: gemma-2b-it**

| | | QNLI | | RTE | | SST2 | | QQP | | MNLI | | MRPC | | COLA | | BoolQ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| method | targets | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | mcc | loss | acc. | avg. |
| **PT (zero-shot)** | | 1.34 | 59.5 | 1.77 | 56.3 | 0.75 | 60.7 | 1.41 | 48.3 | 2.91 | 30.9 | 0.91 | 64.6 | 0.85 | 0.7 | 0.88 | 68.5 | 44.7 |
| **SpaRTA (5%)** | ALL | 0.17 | 93.7 | 0.54 | 81.0 | 0.14 | 95.0 | 0.26 | 89.0 | 0.33 | 87.2 | 0.35 | 85.3 | 0.39 | 56.2 | 0.36 | 84.4 | 84.0 |
| **SpaRTA (0.5%)** | | 0.16 | 94.0 | 0.43 | 80.7 | 0.15 | 94.7 | 0.24 | 89.7 | 0.32 | 87.8 | 0.34 | 86.5 | 0.45 | 54.9 | 0.37 | 84.7 | 84.1 |
| **SpaRTA (0.037%)** | | 0.18 | 93.1 | 0.48 | 76.9 | 0.16 | 94.5 | 0.26 | 88.8 | 0.35 | 86.4 | 0.36 | 84.5 | 0.45 | 54.8 | 0.40 | 83.6 | 82.8 |
| **LoRA (0.037%)** | Q,V | 0.18 | 93.4 | 0.44 | 78.7 | 0.14 | 95.1 | 0.26 | 89.1 | 0.33 | 87.8 | 0.33 | 85.4 | 0.41 | 55.0 | 0.36 | 84.0 | 83.6 |
| **DoRA (0.037%)** | | 0.18 | 93.4 | 0.45 | 78.0 | 0.15 | 95.6 | 0.26 | 89.0 | 0.33 | 87.8 | 0.33 | 85.5 | 0.41 | 57.1 | 0.36 | 84.4 | 83.8 |
| **SpaRTA (0.037%)** | | 0.19 | 92.8 | 0.46 | 77.6 | 0.16 | 94.8 | 0.26 | 88.6 | 0.36 | 86.3 | 0.36 | 84.2 | 0.45 | 54.5 | 0.39 | 83.4 | 82.8 |
| **SpaRTA (0.037%)** | O,V | 0.17 | 93.5 | 0.44 | 80.7 | 0.15 | 94.9 | 0.26 | 88.9 | 0.34 | 87.2 | 0.35 | 85.7 | 0.42 | 55.9 | 0.37 | 85.0 | 84.0 |

Table 5: Test loss and accuracy of gemma-2b-it adapted to GLUE and BoolQ datasets with different fine-tuning methods. Standard errors can be found in Table 15, see Appendix D. For training details see Appendix E.2.

centrate the selection of sparse, trainable parameters on a few model layers or equally across all. Specifically, given a budget for the number of parameters that can be trained, which type of parameters should we target for sparsification (while freezing the remaining) to achieve best task performance when adapting the sparsified parameters within the targeted types?

We conduct an ablation study to answer this question. The gemma-2b model is adapted to the SST-2 task using our SpaRTA approach targeting different combinations of parameter types. We set a budget of 1.25M trainable parameters within the gemma-2b model so that sparsity is $s = 99.95\%$.

Results of these experiments are shown in Table 3. Concentrating the selection of trainable parameters through sparsification in the self-attention value (Wv) or/and output (Wo) weight matrices yields the best performance under the given budget. In contrast, distributing our sparsification across different combinations of weight types may lead to significantly lower performance.

Corresponding results confirming, as done in Hu et al. (2022), that targeting Wo and Wv is suboptimal for LoRA can be found in Appendix D. Note that concentrating our sparsification over targeted parameter tensors, although proven beneficial in terms of single-task adaption performance, can decrease performance when merging since it increases the chance of parameter interference.

### 7.3 GLUE and BoolQ

We now focus on comparing SpaRTA and LoRA methods using the 7 NLU tasks in the GLUE benchmark as well as BoolQ. Table 5 presents the results for adapting gemma-2b-it. An equivalent set of results for the gemma-2b model is given in Table 13, see Appendix D. Adaptation results (rows) are grouped by which type of parameters are targeted, with SpaRTA further ordered in descending order of density. SpaRTA with 0.037% density has approximately the same number of trainable parameters as LoRA, for which a rank $r = 8$ was used.

When targeting all parameters, SpaRTA overall exhibits better performance with higher density, i.e, when more parameters are chosen to be trained, with a few exceptions. Targeting the Wq and Wv self-attention matrices shows very similar performance between LoRA, DoRA, and SpaRTA. Targeting the Wo and Wv self-attention matrices within SpaRTA, which was shown to be the optimal in Section 7.2, yields better performance for SpaRTA when compared to both LoRA and DoRA on four of

**Model: `mistral-7b-it`**

| | | QNLI | | RTE | | SST2 | | QQP | | MNLI | | MRPC | | COLA | | BoolQ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| method | targets | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | mcc | loss | acc. | avg. |
| **PT (zero-shot)** | | 1.38 | 76.6 | 1.54 | 66.4 | 0.66 | 66.3 | 0.85 | 57.5 | 2.42 | 35.5 | 0.70 | 70.0 | 0.72 | 25.2 | 0.39 | 85.2 | 49.6 |
| **SpaRTA (5%)** | ALL | 0.11 | 96.0 | 0.26 | 89.8 | 0.12 | 95.9 | 0.22 | 90.7 | 0.23 | 91.6 | 0.32 | 87.3 | 0.36 | 62.6 | 0.24 | 90.8 | 88.1 |
| **SpaRTA (0.5%)** | | 0.11 | 95.8 | 0.30 | 90.5 | 0.12 | 96.0 | 0.23 | 90.4 | 0.24 | 91.2 | 0.32 | 88.0 | 0.35 | 67.9 | 0.24 | 90.5 | 88.8 |
| **SpaRTA (0.048%)** | | 0.12 | 95.7 | 0.31 | 89.2 | 0.12 | 96.1 | 0.23 | 90.5 | 0.24 | 91.1 | 0.31 | 88.2 | 0.34 | 67.3 | 0.26 | 89.5 | 88.4 |
| **LoRA (0.048%)** | Q,V | 0.12 | 95.6 | 0.24 | 91.2 | 0.12 | 96.0 | 0.23 | 90.2 | 0.24 | 91.0 | 0.29 | 89.3 | 0.32 | 69.9 | 0.24 | 90.5 | 89.2 |
| **DoRA (0.048%)** | | 0.13 | 95.0 | 0.27 | 90.9 | 0.11 | 96.4 | 0.23 | 90.2 | 0.25 | 90.9 | 0.28 | 88.7 | 0.32 | 68.2 | 0.24 | 90.5 | 88.8 |
| **SpaRTA (0.048%)** | | 0.12 | 95.6 | 0.31 | 88.3 | 0.13 | 95.6 | 0.23 | 90.1 | 0.25 | 90.9 | 0.30 | 88.6 | 0.35 | 65.2 | 0.26 | 89.7 | 88.0 |
| **SpaRTA (0.048%)** | O,V | 0.11 | 95.9 | 0.31 | 88.4 | 0.14 | 95.8 | 0.23 | 90.4 | 0.24 | 91.2 | 0.30 | 88.4 | 0.34 | 64.9 | 0.26 | 89.7 | 88.1 |

Table 6: Test loss and accuracy of the `mistral-7b-it` model adapted to GLUE and BoolQ datasets with different fine-tuning methods. Standard errors can be found in Table 16, see Appendix D. For training details see Appendix E.2.

| **method** | **targets** | **`gemma-2b-it`** | **`mistral-7b-it`** | **`gemma-2b`** | **`mistral-7b`** |
|---|---|---|---|---|---|
| LoRA | Q, V | 83.6 | **89.2** | 81.7 | **87.3** |
| SpaRTA | Q, V | 82.8 | 88.0 | 78.3 | 86.7 |
| SpaRTA | O, V | **84.0** | 88.1 | **83.2** | 85.6 |

Table 7: Average accuracy (%) per adapted model. Averages are taken across 8 tasks (7 GLUE and BoolQ) based on Tables 5, 6, 13, and 14. SpaRTA is shown to be competitive with LoRA, particularly when targeting Wo (O) and Wv (V) self-attention matrices, which were previously shown to be optimal for SpaRTA.

the eight datasets (QNLI, RTE, MRPC, BoolQ).

A similar set of results on GLUE and BoolQ is provided for `mistral-7b-it` in Table 6. Again, results for the `mistral-7b` model can be found in Table 14 in Appendix D. SpaRTA exhibits comparable performance to LoRA and DoRA for a low density of 0.0048%, where all methods have comparable numbers of trainable parameters. And once again, with SpaRTA targeting Wv and Wo typically outperforming the targeting of Wq and Wv.

To further illustrate the competitiveness of SpaRTA with LoRA, Table 7 summarizes accuracy results across the 8 datasets while Table 8 offers another perspective by considering win-rates among the competing methods for each model. Note that LoRA exhibits best performance overall when targeting the Wq and Wv self-attention matrices as mentioned in the original LoRA paper (Hu et al., 2022), and hence we target these parameters when using LoRA. While we have already explored optimal targets for SpaRTA, we include results targeting Wq and Wv for completeness. Across both tables, SpaRTA (Wo, Wv) outperforms SpaRTA (Wq, Wv) in six out of the eight scenarios. LoRA and SpaRTA are equally performant on average across the two tables, demonstrating the claim that SpaRTA is competitive with LoRA.

### 7.4 MMLU

We have chosen MMLU because it is known for being a very challenging NLU task. Table 9 compares SpaRTA against LoRA on adapting each of our models to the MMLU multiple-choice question answering task, where each model must predict the correct answer to a set of questions. Solving this task requires a high level of world knowledge and problem solving skills. In this experiment, we restrict both methods to use approximately the same number of trainable parameters. See Table 20 in the Appendix for training details. Here again, the results show that SpaRTA is competitive with LoRA, providing similar performance.

### 7.5 Remarks

Our results establish that SpaRTA is a viable adaptation technique that can be competitive with LoRA, especially for larger LMs. These results demonstrate that a simple sparsifying scheme can offer a valid adaptation technique. This opens the possibility for further investigating sparse adaptation as the low-rank approximation of a model's parameter changes is not the only mechanism to provide a performant adaptation method. This indicates that what matters is not necessarily the adapter structure used in PEFT techniques, but rather the number of trainable parameters, that is relevant to the adaptation task at hand.

| method | targets | gemma-2b-it | mistral-7b-it | gemma-2b | mistral-7b |
|---|---|---|---|---|---|
| LoRA | Q, V | 37.58 | **62.5** | 37.5 | **50** |
| SpaRTA | Q, V | 0 | 0 | 0 | 31.3 |
| SpaRTA | O, V | **62.5** | 37.5 | **62.5** | 18.8 |

Table 8: Win rates (%) per adapted model, computed across 8 tasks (7 GLUE tasks and BoolQ) based on Tables 5, 6, 13, and 14. SpaRTA is shown to be competitive with LoRA, particularly when targeting Wo (O) and Wv (V) self-attention matrices, which were previously shown to be optimal for SpaRTA.

| MMLU | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | gemma-2b | | gemma-2b-it | | mistral-7b | | mistral-7b-it | |
| | loss | acc. | loss | acc. | loss | acc. | loss | acc. |
| **PT (zero-shot)** | - | - | 5.284 | 35.3% | - | - | 1.839 | 59.3% |
| **SpaRTA** | 1.249 | 45.1% | 1.250 | 45.1% | 0.987 | 61.5% | 0.930 | 63.1% |
| **LoRA** | 1.233 | 46.9% | 1.271 | 45.1% | 0.981 | 62.8% | 0.928 | 63.0% |

Table 9: Test loss and accuracy of models adapted to the MMLU dataset with SpaRTA and LoRA. We also report zero-shot performance of instruction following PT models. For training details see Table 20 in Appendix E.

## 8 Conclusion

As PT language models have grown in size, PEFT has become crucial for enabling fine-tuning large PT language models on limited hardware and financial resources. We have introduced SpaRTA, an approach that sharply decreases the set of trainable parameters, reducing the GPU memory used by the optimizer and speeding up training. We have demonstrated on a variety of task adaptation scenarios that our fine-tuning approach is parameter-efficient and competitive with LoRA, the current PEFT standard. Experiments with 2B and 7B parameter pre-trained models demonstrate good performance, and as per Yu et al. (2024), we expect larger models to allow for higher levels of sparsity in training, meaning that efficiency of SpaRTA should get better with larger model sizes as also suggested in Table 1.

Regarding future directions, while SpaRTA has been applied to *supervised learning*, it is also amenable to *reinforcement learning* often used for model alignment (Ouyang et al., 2022). We also plan to explore the merging of SpaRTA adapters due to their potential for little interference.

## 9 Limitations

We have demonstrated various benefits of SpaRTA, including low memory and high performance. Regarding limitations, questions remain about how to best deal with overfitting, though we have some insights. We have observed in our experiments that as we increase the sparsity level, and reduce, in turn, the number of trainable parameters, there is less overfitting. Indeed, there is a point in which both the training and validation losses converge together without significantly diverging from each other, eliminating the need for explicit overfitting mitigation techniques. Moreover, further increasing of the sparsity level beyond this point results in underfitting. Thus, we can think of our approach as a technique to improve generalization by limiting a model's capacity to overfit to the training data. However, finding the breaking point at which this happens requires expensive experimentation. We leave as future work the investigation of how such an optimal sparsity level depends on the model, dataset sizes, and task complexity. Knowing this relation will allow us to determine in advance how much sparsity in the trainable parameters is needed for reducing the capacity of a large model to a point where it learns a new task on a relatively small amount of examples without overfitting.

## References

Alan Ansell, Edoardo Ponti, Anna Korhonen, and Ivan Vulić. 2022. Composable sparse fine-tuning for cross-lingual transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Alan Ansell, Ivan Vulić, Hannah Sterz, Anna Korhonen, and Edoardo M. Ponti. 2024. Scaling sparse fine-tuning to large language models. *arXiv preprint arXiv:2401.16405*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*.

Wenlong Deng, Yize Zhao, Vala Vakilian, Minghui Chen, Xiaoxiao Li, and Christos Thrampoulidis. 2025. DARE the extreme: Revisiting delta-parameter pruning for fine-tuned models. In *The Thirteenth International Conference on Learning Representations*.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. 2023. Sparse low-rank adaptation of pre-trained language models. In *EMNLP*.

Shangbin Feng, Taylor Sorensen, Yuhan Liu, Jillian Fisher, Chan Young Park, Yejin Choi, and Yulia Tsvetkov. 2024. Modular pluralism: Pluralistic alignment via multi-LLM collaboration. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

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. *International Conference on Learning Representations (ICLR)*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, volume 1, page 2.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. 2024. Vera: Vector-based random matrix adaptation. In *ICLR*.

Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. 2023. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. In *ICML*.

Shiwei Liu, Tianlong Chen, Zhenyu Zhang, Xuxi Chen, Tianjin Huang, Ajay Kumar Jaiswal, and Zhangyang Wang. 2023. Sparsity may cry: Let us fail (current) sparse neural networks together! In *ICLR*.

Da Ma, Lu Chen, Pengyu Wang, Hongshen Xu, Hanqi Li, Liangtai Sun, Su Zhu, Shuai Fan, and Kai Yu. 2024. Sparsity-accelerated training for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*.

Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150. Association for Computational Linguistics.

Michael Matena and Colin Raffel. 2022. Merging models with fisher-weighted averaging. In *NeurIPS*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *Technical Report, OpenAI*.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30.

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 *ICLR*.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. 2023. TIES-merging: Resolving interference when merging models. In *Advances in Neural Information Processing Systems*, volume 36, pages 7093–7115.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2024. Language models are Super Mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and

Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *ICLR*.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. 2024. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12.

## A Ranks of Differences in Fine-Tuned Weight Matrices

`LoRA` is based on the idea that the changes in the model parameters after adapting it to a new task can have a low-rank approximation. To see if this is the case, we can easily check this presumption in a few well-known fine-tuned models by comparing their weight matrices before and after fine-tuning. Thus, for every weight matrix $W_{\text{PT}}$ (i.e. a 2-dimensional trainable tensor) in the *pre-trained* model, we compute the *delta* matrix $\Delta = W_{\text{FT}} - W_{\text{PT}}$, where $W_{\text{FT}}$ is the corresponding weight matrix after *fine-tuning* the model over all the original model parameters using standard supervised (or reinforcement) learning. `LoRA` assumes that these matrices don't change or their differences given by $\Delta$ are low-rank.

We compute all delta weight matrices for two well-known *instruction-following* fine-tuned models: `gemma-2b-it` and `mistral-7b-it`, see Section 6.2. As we can see in Tables 10 and 11, the feedforward (MLP) *delta* $\Delta$-matrices associated with each layer of the transformer network are all full rank[4] in both models. This is also the case for all self attention key (k) and value (v) projection matrices. However, the self attention query (q) and output (o) $\Delta$-matrices all show relatively small rank deficiencies: between $9$ and $2$ for the query projection matrices and between $46$ and $3$ for the output projection matrices, out of a potential maximum rank of $2,048$ for the `gemma-2b-it` model; and between $1,788$ and $28$ for the query projection matrices and between $246$ and $10$ for the output projection matrices, out of a potential maximum rank of $4,096$ for the `mistral-7b-it` model. The token embedding $\Delta$-matrices of both the `gemma-2b-it` and `mistral-7b-it` models are full rank. Of the two models, `mistral-7b-it` is the only one that does not tie its token embeddings' weights to its head; thus the `mistral-7b-it`'s model head (untied lm head) is updated independently during fine-tuning, and the resulting delta change of this matrix is also full rank. Basically, we observe that the fine-tuning changes in the weight matrices of these well known models are all full rank, with the only exception being the changes in the query (q) and output (o) projection matrices that show small rank deficiencies.

The observed ranks suggest that constraining the (delta) changes in weight matrices to be low-rank is not essential for fine-tuning models efficiently.

[4]We compute the rank of a matrix as the number of singular values that are greater than zero, given a specified tolerance, using `torch.linalg.matrix_rank`. We did some sensitivity analysis to determine such tolerance.

## B Additional Dataset Details

**IMDB** contains a sample of "highly polar" movie reviews obtained from the online Internet Movie Database (IMDb) website. IMDb registered users provide a rating (from $1$ to $10$) with each review. The reviews are binary (positive/negative) labeled with their sentiment, defined from user ratings (Maas et al., 2011). Reviews with a rating higher or equal than 7 are given a positive label; and a negative label if the rating is lower or equal than $4$. No reviews with ratings beyond these ranges are present in the dataset, which was constructed to have an equal number of positive and negative reviews, so guessing randomly yields 50% accuracy.

**GLUE** datasets are described in details in Wang et al. (2019).

**BoolQ** (Clark et al., 2019) is a reading comprehension dataset with binary yes/no questions. Each example is a triplet of (passage, question, answer). It represents a natural language inference task in which a passage and a question are given as input, and a yes or no answer should be predicted.

**MMLU** (Hendrycks et al., 2020) is a benchmark designed to measure the knowledge a model acquires during pre-training. To facilitate the elicitation of a model's knowledge using multiple-choice questions, MMLU also comes with a dataset of examples on multiple-choice question answering. Thus, this training set is not designed to increase the model knowledge about the world, but to teach a model to answer knowledge questions in multiple-choice format in a zero-shot setting.

## C Training Details

We adapted each (generative) pre-trained model in Section 6.2 to do sequence classification as per the examples in any of the datasets from Table 2, using different supervised fine-tuning methods, including `SpaRTA`. Base models were adapted by switching their vocabulary heads to randomly initialized sequence classification heads with $c$ output classifica-

| Weight matrix | dims | Layer(s) | Rank(s) | Rank deficiencies |
|---|---|---|---|---|
| tokens embed | [ 256000, 2048 ] | - | 2048 | 0 (full rank) |
| self attn q proj | [ 2048, 2048 ] | 0 - 17 | 2046 - 2039 | 2 - 9 |
| self attn k proj | [ 256, 2048 ] | 0 - 17 | 256 | 0 (full rank) |
| self attn v proj | [ 256, 2048 ] | 0 - 17 | 256 | 0 (full rank) |
| self attn o proj | [ 2048, 2048 ] | 0 - 17 | 2045 - 2002 | 3 - 46 |
| mlp gate proj | [ 16384, 2048 ] | 0 - 17 | 2048 | 0 (full rank) |
| mlp up proj | [ 16384, 2048 ] | 0 - 17 | 2048 | 0 (full rank) |
| mlp down proj | [ 2048, 16384 ] | 0 - 17 | 2048 | 0 (full rank) |

Table 10: Rank of the change differences in `gemma-2b-it` model weight matrices after full fine-tuning.

| Weight matrix | dims | Layer(s) | Rank(s) | Rank deficiencies |
|---|---|---|---|---|
| tokens embed | [ 32768, 4096 ] | - | 4096 | 0 (full rank) |
| self attn q proj | [ 4096, 4096 ] | 0 - 31 | 4068 - 2308 | 28 - 1788 |
| self attn k proj | [ 1024, 4096 ] | 0 - 31 | 1024 | 0 (full rank) |
| self attn v proj | [ 1024, 4096 ] | 0 - 31 | 1024 | 0 (full rank) |
| self attn o proj | [ 4096, 4096 ] | 0 - 31 | 4086 - 3850 | 10 - 246 |
| mlp gate proj | [ 14336, 4096 ] | 0 - 31 | 4096 | 0 (full rank) |
| mlp up proj | [ 14336, 4096 ] | 0 - 31 | 4096 | 0 (full rank) |
| mlp down proj | [ 4096, 14336 ] | 0 - 31 | 4096 | 0 (full rank) |
| untied lm head | [ 4096, 32768 ] | - | 4096 | 0 (full rank) |

Table 11: Rank of the change differences in `mistral-7b-it` model weight matrices after full fine-tuning.

tion tokens. For instruction models, we re-used the vocabulary heads as described in Section 6.3.

The examples (e.g., text extracts, sentences) to be classified were converted into sequences of tokens before passing them as inputs to a model. We wrapped each example into an instruction to take advantage of instruction-following models, adding to the token length of the model inputs. We tried to keep such instructions as short as possible while still achieving a good initial performance before fine-tuning. For instance, after tokenization, the maximum token length of a training input from the SST-2 dataset was

- 67 (without) and 87 (with instruction) for the `gemma-2b-it`;
- 72 (without) and 95 (with instruction) for `mistral-7b-it`.

We observe here how the Gemma's tokenizer compresses more the input than Mistral's because of its larger vocabulary size; 256,000 (Gemma) vs. 32,767 (Mistral).

We tokenized all training examples before starting the fine-tuning and looked at the histogram of their token lengths. To avoid batches with too much padding and improve training efficiency, we dropped those training examples with a disproportionate large token length, i.e., corresponding to a tail of extreme values in the histogram. We only do this for the training data since evaluation (on the development or test data) requires much less compute and memory (no gradients need to be calculated and stored) and is performed less frequently. We indicate in Tables 2 the final splits after filtering the training data this way, with the final number of training examples used for the fine-tuning.

With `SpaRTA`, we froze the token embeddings layer, made fully trainable the classification head, and randomly chose a sparse proportion of (scalar) parameters to be trained in all the other layers of a model. We demonstrated our method with varying density levels. The total average number of trainable parameters in each case was approximately: 99M (5% density), 10M (0.5%) and 800k (0.037%) for the (base and instruct) Gemma 2B models; and 349M (5% density), 35M (0.5%) and 3.5M (0.048%) for the (base and instruct) Mistral 7B models.

For `LoRA`, we factorized the changes in the query and value self-attention projection weight matrices with rank $r = 8$ decomposition matrices, which were optimized while keeping all other model parameters frozen. The number of new trainable parameters introduced by the `LoRA` approach in the Gemma 2B and Mistral 7B models were approximately 925k and 3.5M, respectively. Also, we set $\alpha = 16$ to scale the `LoRA` adapters.

We observed overfitting when training with Full parameter FT: the development loss started deteriorating after a few epochs (e.g., approximately 2 for SST-2) while the training loss went quickly to zero. We used a combination of early stopping, dropout and weight decay to deal with overfitting. We noticed that `SpaRTA` is a natural regularizer: increasing sparsity resulted in less overfitting, to a point in which there was no more overfitting (e.g., this was achieved at a sparsity $s > 99\%$ for all models under consideration). In general, overfitting was more noticeable with the Mistral 7B models; as expected since they are larger than the Gemma 2B models. We also noticed that larger models require higher sparsity levels to eliminate overfitting given the same training data.

## D Additional Experiment Results

Table 12 confirms targeting Wo and Wv is not the best choice for `LoRA` (96.2% accuracy for Wo and Wv versus 96.4% for Wq and Wv which is the `LoRA` default). These results can be compared with the corresponding `SpaRTA` results in Section 7.2. Tables 13 and Table 14 present the complete set of results on GLUE and BoolQ for the `gemma-2b` and `mistral-7b` models, respectively. Tables 15 and 16 present standard errors from corresponding experiments.

## E Training Hyper-Parameters

The hyper-parameters used for investigating `SpaRTA` and other adaptation methods are summarized next.

### E.1 IMDB

The sets of best parameters used in training the various adaptation methods for each model are given in Table 19. To improve training efficiency, we excluded training examples exceeding 384 tokens in length, resulting in the use of only 19,306 examples for `gemma-2b`, 18,744 examples for `gemma-2b-it`, 18,373 examples for `mistral-7b`, and 17,672 examples for `mistral-7b-it` for training. That is 77%, 75%, 73%, and 71% of the original training data of 25,000 examples, respectively.

| LoRA targets | rank | Loss | Accuracy |
|---|---|---|---|
| Wq | 16 | 0.155 | 94.8 % |
| Wk | 24 | 0.182 | 94.5% |
| Wv | 24 | 0.116 | 96.2% |
| Wo | 16 | 0.114 | 96.2% |
| Wq, Wk | 8 | 0.152 | 94.8% |
| Wq, Wv (default) | 8 | 0.110 | 96.4% |
| Wq, Wo | 6 | 0.107 | 96.6% |
| Wk, Wv | 12 | 0.110 | 96.7% |
| Wk, Wo | 8 | 0.110 | 96.9% |
| Wv, Wo | 8 | 0.114 | 96.2% |
| Wq, Wv, Wo | 4 | 0.108 | 96.7% |
| Wk, Wv, Wo | 6 | 0.113 | 96.7% |
| Wq, Wk, Wv, Wo | 3 | 0.108 | 96.8% |
| MLP | 1 | 0.112 | 96.2% |

Table 12: Test loss and accuracy on SST-2 after applying LoRA to different types of parameters in the gemma-2b model, given that the same number of trainable parameters are selected. Results averaged across 10 random seeds.

### E.2 GLUE and BoolQ

For GLUE and BoolQ, training was done using a simple grid search over the learning rates: [1e-3, 5e-4, 2e-4, 1e-4, 5e-5, 1e-5, 5e-6]. The optimal learning rates found are given in Tables 17 and 18. Weight decay was set to 0. Dropout was set to 0.0 for SpaRTA and 0.1 for LoRA and DoRA. Batch sizes were set according to what could fit in GPU memory (different per dataset and model). The number of epochs was set to 3 for instruct models and 2 for base models. Similarly to IMDB, any sample longer than 256 tokens (which is dependnent on the tokenizer used) is discarded from the dataset to avoid having few minibatch with one very long sample compared to the rest, as seen in Table 2.

### E.3 MMLU

As with other datasets, we started approximating the deltas of the self-attention matrices Wq and Wv with rank $r = 8$ matrices, resulting in approximately 1M trainable parameters for the Gemma 2B models and 3.5M for the Mistral 7B models under LoRA ($\alpha = 16$). We choose the sparsity of our SpaRTA adapter accordingly so both methods end up with the same number of parameters to train. Thus, our SpaRTA approach uses a $s = 99.96\%$ for the Gemma models (i.e. making only $0.04\%$ of the original parameters trainable); and $s = 99.95\%$ for the Mistral models.

Since we observed that gemma-2b struggled to learn with both SpaRTA and LoRA methods, we decided to increase the number of trainable parameters when adapting gemma-2b to MMLU. Specifically, we increased the rank of the LoRA adaption matrices to $r = 16$, which lead to approximately 2M trainable parameters; and chose accordingly a sparsity $s = 99.92\%$ for SpaRTA.

The MMLU dataset has a small fraction of training examples that are extremely long. We enforce a maximum input token length of $520$ for training efficiency. This reduces the number of training examples from $99,842$ to $84,296$ and $74,100$ for the Gemma and Mistral instruction models; and to $91,321$ and $85,820$ for their respective base models. Test and validations sets are not affected by this decision. The training parameters used for adapting each model to MMLU are shown in Table 20.

Model: gemma-2b

| Method | Targets | QNLI | | RTE | | SST2 | | QQP | | MNLI | | MRPC | | COLA | | BoolQ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | mcc | loss | acc. |
| SpaRTA (5%) | ALL | 0.15 | 94.5 | 0.50 | 80.5 | 0.12 | 96.6 | 0.23 | 90.2 | 0.39 | 85.5 | 0.38 | 84.6 | 0.36 | 62.6 | 0.32 | 86.7 |
| SpaRTA (0.5%) | | 0.15 | 94.5 | 0.51 | 77.3 | 0.12 | 96.1 | 0.23 | 90.1 | 0.42 | 84.1 | 0.37 | 84.3 | 0.36 | 62.3 | 0.34 | 85.4 |
| SpaRTA (0.037%) | | 0.20 | 91.9 | 0.62 | 66.3 | 0.13 | 95.4 | 0.27 | 88.3 | 1.34 | 49.3 | 0.51 | 76.1 | 0.49 | 50.7 | 0.39 | 82.8 |
| SpaRTA (0.037%) | Q,V | 0.18 | 93.0 | 0.58 | 70.4 | 0.13 | 95.7 | 0.26 | 88.8 | 1.04 | 56.8 | 0.43 | 80.3 | 0.41 | 56.9 | 0.36 | 84.5 |
| LoRA (0.037%) | | 0.15 | 94.2 | 0.62 | 65.5 | 0.11 | 96.8 | 0.26 | 88.9 | 0.42 | 84.0 | 0.47 | 77.3 | 0.40 | 60.9 | 0.34 | 85.7 |
| DoRA (0.037%) | | 0.15 | 94.4 | 0.62 | 66.5 | 0.11 | 96.6 | 0.25 | 89.0 | 0.41 | 84.4 | 0.47 | 77.6 | 0.39 | 60.6 | 0.43 | 78.9 |
| SpaRTA (0.037%) | O,V | 0.16 | 93.8 | 0.55 | 73.2 | 0.11 | 96.5 | 0.25 | 89.3 | 0.51 | 80.4 | 0.39 | 84.3 | 0.39 | 61.9 | 0.34 | 86.1 |

Table 13: Test loss and accuracy of model gemma-2b adapted to GLUE and BoolQ datasets with different fine-tuning methods. Results are averaged over 3 random seeds. For training details see Appendix E.2.

Model: mistral-7b

| Method | Targets | QNLI | | RTE | | SST2 | | QQP | | MNLI | | MRPC | | COLA | | BoolQ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | mcc | loss | acc. |
| SpaRTA (5%) | ALL | 0.11 | 95.9 | 0.36 | 85.0 | 0.11 | 96.8 | 0.38 | 82.1 | 0.30 | 89.2 | 0.33 | 86.7 | 0.35 | 64.6 | 0.25 | 90.1 |
| SpaRTA (0.5%) | | 0.12 | 95.8 | 0.44 | 84.4 | 0.11 | 96.7 | 0.22 | 90.6 | 0.26 | 90.4 | 0.32 | 87.0 | 0.35 | 66.8 | 0.26 | 89.9 |
| SpaRTA (0.048%) | | 0.13 | 95.3 | 0.41 | 83.6 | 0.11 | 96.7 | 0.23 | 90.1 | 0.40 | 86.7 | 0.34 | 86.6 | 0.36 | 64.6 | 0.26 | 89.8 |
| SpaRTA (0.048%) | Q,V | 0.14 | 94.5 | 0.37 | 84.0 | 0.11 | 96.6 | 0.24 | 89.7 | 0.33 | 88.6 | 0.36 | 85.5 | 0.36 | 65.2 | 0.28 | 89.3 |
| LoRA (0.048%) | | 0.12 | 95.4 | 0.39 | 85.4 | 0.12 | 96.5 | 0.23 | 90.1 | 0.28 | 89.7 | 0.32 | 87.0 | 0.38 | 64.5 | 0.27 | 89.6 |
| DoRA (0.048%) | | 0.12 | 95.6 | 0.41 | 85.3 | 0.12 | 96.8 | 0.24 | 90.1 | 0.28 | 89.7 | 0.33 | 86.4 | 0.34 | 65.8 | 0.26 | 89.9 |
| SpaRTA (0.048%) | O,V | 0.15 | 94.5 | 0.41 | 85.6 | 0.21 | 91.7 | 0.24 | 89.6 | 0.33 | 87.8 | 0.35 | 86.4 | 0.39 | 59.8 | 0.29 | 89.3 |

Table 14: Test loss and accuracy of model mistral-7b adapted to GLUE and BoolQ datasets with different fine-tuning methods. Results are averaged over 3 random seeds. For training details see Appendix E.2.

Model: gemma-2b-it

| Method | targets | QNLI | | RTE | | SST2 | | QQP | | MNLI | | MRPC | | COLA | | BoolQ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | mcc | loss | acc. |
| SpaRTA (5%) | ALL | 0.00 | 0.14 | 0.03 | 1.54 | 0.01 | 0.51 | 0.00 | 0.21 | 0.01 | 0.26 | 0.01 | 0.48 | 0.00 | 0.88 | 0.01 | 0.54 |
| SpaRTA (0.5%) | | 0.00 | 0.16 | 0.03 | 0.64 | 0.01 | 0.46 | 0.00 | 0.17 | 0.00 | 0.13 | 0.01 | 0.07 | 0.01 | 0.75 | 0.01 | 0.25 |
| SpaRTA (0.037%) | | 0.00 | 0.12 | 0.01 | 0.55 | 0.00 | 0.17 | 0.00 | 0.18 | 0.00 | 0.14 | 0.00 | 0.45 | 0.01 | 0.44 | 0.00 | 0.09 |
| LoRA (0.037%) | Q,V | 0.00 | 0.11 | 0.01 | 1.10 | 0.00 | 0.35 | 0.00 | 0.03 | 0.00 | 0.11 | 0.00 | 0.22 | 0.02 | 1.40 | 0.01 | 0.12 |
| DoRA (0.037%) | | 0.00 | 0.11 | 0.01 | 0.55 | 0.00 | 0.20 | 0.00 | 0.03 | 0.00 | 0.09 | 0.00 | 0.32 | 0.01 | 1.31 | 0.01 | 0.29 |
| SpaRTA (0.037%) | | 0.00 | 0.08 | 0.00 | 1.04 | 0.01 | 0.11 | 0.00 | 0.00 | 0.00 | 0.26 | 0.00 | 0.22 | 0.02 | 2.83 | 0.00 | 0.41 |
| SpaRTA (0.037%) | O,V | 0.00 | 0.13 | 0.01 | 0.64 | 0.00 | 0.27 | 0.00 | 0.16 | 0.00 | 0.17 | 0.02 | 0.22 | 0.01 | 4.36 | 0.00 | 0.22 |

Table 15: Standard errors of test loss and accuracy of model gemma-2b-it adapted to GLUE and BoolQ datasets with different fine-tuning methods.

Model: mistral-7b-it

| Method | targets | QNLI | | RTE | | SST2 | | QQP | | MNLI | | MRPC | | COLA | | BoolQ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | acc. | loss | mcc | loss | acc. |
| SpaRTA (5%) | ALL | 0.00 | 0.09 | 0.01 | 0.24 | 0.01 | 0.26 | 0.00 | 0.03 | 0.00 | 0.08 | 0.01 | 0.47 | 0.01 | 1.91 | 0.00 | 0.16 |
| SpaRTA (0.5%) | | 0.00 | 0.05 | 0.01 | 1.07 | 0.01 | 0.18 | 0.00 | 0.10 | 0.00 | 0.09 | 0.02 | 0.25 | 0.01 | 1.28 | 0.00 | 0.09 |
| SpaRTA (0.048%) | | 0.00 | 0.10 | 0.03 | 0.55 | 0.00 | 0.20 | 0.00 | 0.03 | 0.00 | 0.11 | 0.01 | 0.32 | 0.02 | 1.87 | 0.00 | 0.20 |
| LoRA (0.048%) | Q,V | 0.00 | 0.16 | 0.01 | 0.60 | 0.01 | 0.44 | 0.00 | 0.06 | 0.00 | 0.13 | 0.01 | 0.33 | 0.01 | 1.16 | 0.00 | 0.17 |
| DoRA (0.048%) | | 0.02 | 0.76 | 0.02 | 0.87 | 0.00 | 0.17 | 0.00 | 0.03 | 0.00 | 0.07 | 0.00 | 0.17 | 0.01 | 1.22 | 0.00 | 0.16 |
| SpaRTA (0.048%) | | 0.00 | 0.06 | 0.01 | 0.73 | 0.00 | 0.23 | 0.00 | 0.04 | 0.00 | 0.05 | 0.01 | 0.43 | 0.01 | 1.17 | 0.00 | 0.27 |
| SpaRTA (0.048%) | O,V | 0.00 | 0.02 | 0.01 | 0.42 | 0.01 | 0.34 | 0.00 | 0.07 | 0.00 | 0.13 | 0.02 | 0.24 | 0.01 | 0.97 | 0.00 | 0.23 |

Table 16: Standard errors of test loss and accuracy of model mistral-7b-it adapted to BLUE and BoolQ datasets with different fine-tuning methods.

**Model: gemma-2b-it**

| Method | QNLI | RTE | SST2 | QQP | MNLI | MRPC | COLA | BoolQ |
|---|---|---|---|---|---|---|---|---|
| **SpaRTA (5%)** | 1e-5 | 5e-4 | 5e-5 | 5e-5 | 1e-5 | 2e-4 | 5e-5 | 5e-5 |
| **SpaRTA (0.5%)** | 2e-4 | 1e-3 | 2e-4 | 2e-4 | 2e-4 | 5e-4 | 1e-3 | 2e-4 |
| **SpaRTA (0.037%)** | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-3 |
| **LoRA (0.037%)** | 2e-4 | 1e-3 | 5e-4 | 5e-4 | 5e-4 | 2e-4 | 5e-4 | 5e-4 |
| **DoRA (0.037%)** | 2e-4 | 1e-3 | 5e-4 | 5e-4 | 5e-4 | 2e-4 | 5e-4 | 5e-4 |
| **SpaRTA (0.037%)** | 5e-4 | 1e-3 | 5e-4 | 5e-4 | 5e-4 | 5e-4 | 1e-3 | 5e-4 |
| **SpaRTA (0.037%)** | 5e-4 | 1e-3 | 5e-4 | 5e-4 | 5e-4 | 5e-4 | 1e-3 | 5e-4 |

Table 17: Optimal learning rates for gemma-2b-it model for different fine-tuning methods.

**Model: mistral-7b-it**

| Method | QNLI | RTE | SST2 | QQP | MNLI | MRPC | COLA | BoolQ |
|---|---|---|---|---|---|---|---|---|
| **SpaRTA (5%)** | 1e-5 | 5e-6 | 5e-6 | 1e-5 | 1e-5 | 5e-5 | 5e-5 | 5e-6 |
| **SpaRTA (0.5%)** | 2e-4 | 1e-4 | 5e-5 | 5e-4 | 2e-4 | 5e-4 | 2e-4 | 5e-5 |
| **SpaRTA (0.048%)** | 5e-4 | 5e-4 | 5e-4 | 1e-3 | 5e-4 | 1e-3 | 1e-3 | 1e-3 |
| **LoRA (0.048%)** | 2e-4 | 1e-4 | 2e-4 | 2e-4 | 1e-4 | 2e-4 | 2e-4 | 1e-4 |
| **DoRA (0.048%)** | 2e-4 | 1e-4 | 2e-4 | 2e-4 | 1e-4 | 2e-4 | 2e-4 | 1e-4 |
| **SpaRTA (0.048%)** | 2e-4 | 1e-4 | 2e-4 | 5e-4 | 2e-4 | 1e-3 | 1e-3 | 5e-4 |
| **SpaRTA (0.048%)** | 2e-4 | 1e-4 | 2e-4 | 5e-4 | 2e-4 | 1e-3 | 1e-3 | 5e-4 |

Table 18: Optimal learning rates for mistral-7b-it model for different fine-tuning methods.

| | Parameter | gemma-2b | gemma-2b-it | mistral-7b | mistral-7b-it |
|---|---|---|---|---|---|
| Full FT | batch size | 32 | 32 | 36 | 36 |
| | num epochs | 2 | 2 | $*$ | $*$ |
| | learning rate | 1e-5 | 1e-5 | 3e-6 | 3e-6 |
| | max grad norm | 10 | 50 | 120 | 120 |
| | dropout | 0.1 | 0.1 | 0.15 | 0.15 |
| | weight decay | 0.1 | 0.1 | 0.01 | 0.01 |
| SpaRTA | batch size | 40 | 40 | 16 | 16 |
| | num epochs. | 2 | 2 | $*$ | $*$ |
| $d = 5\%$ | learning rate | 1.5e-5 | 8e-6 | 2e-6 | 2e-6 |
| $d = 0.5\%$ | learning rate | 1e-4 | 5e-5 | 3e-6 | 3e-6 |
| $d = 0.05\%$ | learning rate | 6e-5 | 6e-5 | 6e-5 | 6e-5 |
| LoRA | batch size | 40 | 40 | 20 | 20 |
| | num epochs | 3 | 3 | 3 | 3 |
| | learning rate | 2e-4 | 2e-4 | 5e-6 | 5e-6 |
| | max grad norm | 15 | 15 | - | - |
| | dropout | 0.1 | 0.1 | 0.1 | 0.1 |
| | $r$ | 8 | 8 | 8 | 8 |
| | $\alpha$ | 16 | 16 | 16 | 16 |
| Head | batch size | 40 | 40 | 16 | 16 |
| | num epochs | 4 | 4 | 3 | 3 |
| | learning rate | 2e-4 | 2e-4 | 1e-4 | 1e-4 |

Table 19: (IMDB) Training parameters used with each fine-tuning method and model in Table 4. An $*$ in number of epochs indicates early stopping was used. For SpaRTA, parameters for density 5%, 0.5% 0,05% are reported.

| | Training parameter | `gemma-2b` | `gemma-2b-it` | `mistral-7b` | `mistral-7b-it` |
|---|---|---|---|---|---|
| `SpaRTA` / `LoRA` | epochs | 14 / 6 | 1 | 1 | 1 |
| | batch size | 40 | 40 | 40 | 40 |
| | learning rate | 1e-4 | 1e-4 | 5e-5 | 5e-5 |
| | dropout | 0.0 | 0.05 | 0.1 | 0.1 |
| | weight decay | 0.0 | 0.0 | 0.0 | 0.0 |

Table 20: Training parameters used with both SpaRTA and LoRA for each of the pre-trained models in Table 9 (MMLU).