HOW FINE CAN FINE-TUNING BE? LEARNING EFFICIENT LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

State-of-the-art performances on language comprehension tasks are achieved by huge language models pre-trained on massive unlabeled text corpora, with very light subsequent fine-tuning in a task-specific supervised manner. It seems the pre-training procedure learns a very good common initialization for further training on various natural language understanding tasks, such that only few steps need to be taken in the parameter space to learn each task. In this work, using Bidirectional Encoder Representations from Transformers (BERT) as an example, we verify this hypothesis by showing that task-specific fine-tuned language models are highly close in parameter space to the pre-trained one. Taking advantage of such observations, we further show that the fine-tuned versions of these huge models, having on the order of 10^8 floating-point parameters, can be made very computationally efficient. First, fine-tuning only a fraction of critical layers suffices. Second, fine-tuning can be adequately performed by learning a binary multiplicative mask on pre-trained weights, *i.e.* by parameter-sparsification. As a result, with a single effort, we achieve three desired outcomes: (1) learning to perform specific tasks, (2) saving memory by storing only binary masks of certain layers for each task, and (3) saving compute on appropriate hardware by performing sparse operations with model parameters.

1 INTRODUCTION

One very puzzling fact about overparameterized deep neural networks is that sheer increases in dimensionality of the parameter space seldom make stochastic gradient-based optimization more difficult. Given an effective network architecture reflecting proper inductive biases, deeper and/or wider networks take just about the same, if not a lower, number of training iterations to converge, a number often by orders of magnitude smaller than the dimensionality of the parameter space. For example, ResNet-18 (parameter count 11.7M) and ResNet-152 (parameter count 60.2M) both train to converge, at similar convergence rates, in no more than 600K iterations on Imagenet (He et al., 2015). Meaningful optimization seems to happen in only a very low-dimensional parameter subspace, *viz.* the span of those relatively *few* weight updates, with its dimensionality not ostensibly scaling with the model size. In other words, the network seems already perfectly converged along most of the parameter dimensions at initialization, suggesting that training only marginally alters a high-dimensional parameter configuration.

This phenomenon is epitomized in fine-tuning of pre-trained models. Pre-training is a, often unsupervised, learning procedure that yields a good common initialization for further supervised learning of various downstream tasks. The *better* a pre-trained model is, the *fewer* iterations are required on average to fine-tune it to perform specific tasks, resulting in fine-tuned models hypothetically *closer* [1] to the pre-trained one in parameter space. However, *better* pre-trained models are, almost always, *larger* models (Hestness et al., 2017), and nowhere is this trend more prominent than recent pre-trained language models that achieved state-of-the-art natural language understanding performance, *e.g.* GPT-2 (Radford et al., 2019) has 1.5B parameters.

Thus, a problem naturally arises hand-in-hand with an obvious hint to its solution: as pre-trained models get larger, on the one hand, computation of each fine-tuned model becomes more expensive

[1] This vague notion of *closeness*, *viz.* separation by *few* gradient update steps in the parameter space, will be made explicit later in the text.

in terms of both memory and compute for inference, while on the other hand, greater *closeness* between the pre-trained and fine-tuned models in the parameter space prescribes a higher degree of computational redundancy that could be potentially avoided. Additionally, there might exist more computationally efficient fine-tuned networks that are not necessarily *close* to, but *cheaply attainable* from, the pre-trained parameters, which are shared across all tasks.

In this study, we seek to address these questions, using Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018) and the General Language Understanding Evaluation (GLUE) benchmark tasks (Wang et al., 2018) as a working example.

We first found that the fine-tuned and pre-trained parameters are both L_1-close and angular-close in parameter space, consistent with the small number of fine-tuning iterations separating them. Next, we demonstrated that there also exist good fine-tuned models that are L_0-close (*i.e.* having a small number of different components) to the pre-trained one. Further, we showed that there exist good fine-tuned parameters that are L_0-small (*i.e. sparse*, or having a large fraction of zero components). Finally, we successfully found fine-tuned language models that are both L_0-small and L_0-close to the pre-trained models.

We remark the practical implications of these constraints. By forcing fine-tuned parameters to be L_0-close to the pre-trained ones, one only needs to store a small number of different weights per task, in addition to the common pre-trained weights, substantially saving parameter memory. By forcing fine-tuned parameters to be sparse, one potentially saves memory and compute, provided proper hardware acceleration of sparse linear algebraic operations.

Surprisingly, our findings also reveal an abundance of good task-specific parameter configurations within a sparse L_0-vicinity of large pre-trained language models like BERT: a specific task can be learned by simply masking anywhere between 1% to 40% of the pre-trained weights to zero. See Figure 1 for an explanation of the L_0- and sparse L_0- vicinities.

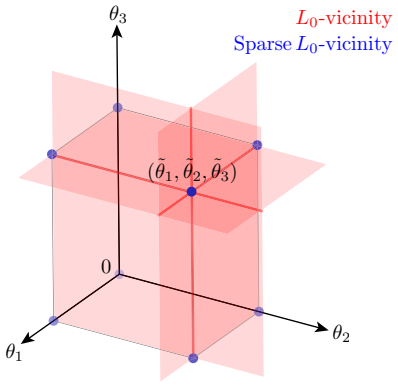

Figure 1: An illustration of the L_0-vicinity and the sparse L_0-vicinity of a pre-trained parameter in a three-dimensional parameter space. The L_0-vicinity is continuous and contains parameters that are L_0-close, whereas the sparse L_0-vicinity is a discrete subset of L_0-close parameters that are also L_0-small.

2 Related work

Our search for L_0-close fine-tuning solutions is motivated by the observation that sensitivities of the optimization objective to different layers in a network are highly variable (Zhang et al., 2019). Zhou et al. (2019) trained fine-grain sparse connectivity patterns over randomly initialized network parameters, termed *supermasks*, suggesting a similar and complementary role model sparsification plays to gradient-based learning of the objective. This is also related to network architecture search (NAS). The most similar study to ours is *piggyback* and its variants (Mallya et al., 2018; Mancini et al., 2019), where in a multi-task visual object classification scenario, the authors trained task-specific binary masks on top of a shared set of pre-trained parameters. In this work, we not only applied similar techniques further to larger pre-trained language models, but also studied the trade-

Table 1: Task-specific model information of $\mathrm{BERT}_{\mathrm{BASE}}$ (parameter count 109M).

GLUE TASK	MNLI	QQP	QNLI	SST-2	COLA	STS-B	MRPC	RTE
ADDITIONAL PARAMETER COUNT	2,304	1,536	1,536	1,536	1,536	768	1,536	1,536
FINE-TUNING ITERATION COUNT	36,816	34,113	9,822	6,315	804	540	345	234

off between L_0-closeness and sparseness in a systematic way. Also, note that randomly generated high-dimensional masks can also support multi-task learning, *e.g.* Cheung et al. (2019).

A large body of literature is concerned with sparsification of large networks for efficient inference. Here, we employed iterative pruning (Zhu & Gupta, 2017) during fine-tuning to produce high-performance fine-grain sparse models. To enforce parameter sparsity differentiably in combination with the L_0-closeness constraint, instead of principled approaches to imposing L_0-regularization (Louizos et al., 2017), we used simpler straight-through estimator, much like binary quantization techniques (Courbariaux et al., 2015; Courbariaux & Bengio, 2016); note that this is also used by Mallya et al. (2018) and Zhou et al. (2019).

3 METHODS

3.1 NOTATIONS AND MODEL ARCHITECTURE

Consider a pre-trained network $F_{\boldsymbol{\theta}} : \boldsymbol{x} \mapsto F(\boldsymbol{x}; \boldsymbol{\theta})$, parameterized by $\boldsymbol{\theta}$, noted as subscript for convenience. The fine-tuning procedure to perform a task $t \in \mathcal{T}$ can be described as a supervised training procedure of model $G^{(t)}_{\boldsymbol{\phi}} \circ F_{\boldsymbol{\theta}} : \boldsymbol{x} \mapsto \boldsymbol{y}$ on fine-tuning set $\left\{(\boldsymbol{x}^{(t)}_i, \boldsymbol{y}^{(t)}_i)\right\}$, where $G^{(t)}$ is a task-specific last layer unique to task t, and $\circ$ denotes function composition.

In the case of BERT, we have a stack of modules

$$F_{\boldsymbol{\theta}} = \mathrm{BERT}_{\boldsymbol{\theta}} \triangleq P_{\boldsymbol{\theta}_{L+1}} \circ B_{\boldsymbol{\theta}_L} \circ \cdots \circ B_{\boldsymbol{\theta}_1} \circ E_{\boldsymbol{\theta}_0} \quad (\boldsymbol{\theta} \triangleq [\boldsymbol{\theta}_l]_0^{L+1}), \tag{1}$$

among which E is the embedding layer, P a final pooling layer and each B is a transformer block

$$B_{\boldsymbol{\vartheta}} : \boldsymbol{x} \mapsto \mathcal{LN}(\boldsymbol{x} + \mathcal{DO}(\boldsymbol{W}_O \mathrm{GeLU}(\boldsymbol{W}_I \mathcal{LN}(\boldsymbol{x} + \mathcal{DO}(\boldsymbol{W}_D \mathcal{A}(\boldsymbol{W}_Q \boldsymbol{x}, \boldsymbol{W}_K \boldsymbol{x}, \boldsymbol{W}_V \boldsymbol{x})))))), \tag{2}$$

where $\boldsymbol{\vartheta} \triangleq [\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V, \boldsymbol{W}_D, \boldsymbol{W}_I, \boldsymbol{W}_O]$ collects all the learnable parameter matrices in the block. $\mathcal{A}(\cdot, \cdot, \cdot)$ represents scaled dot-product attention (Vaswani et al., 2017), $\mathcal{DO}(\cdot)$ dropout, $\mathcal{LN}(\cdot)$ layer normalization and $\mathrm{GeLU}(\cdot)$ the Gaussian error linear unit activation function (Hendrycks & Gimpel, 2016). We experimented with the $\mathrm{BERT}_{\mathrm{BASE}}$ model (Devlin et al., 2018), for which $L = 12$, with total parameter count of 109M [2]. See Table 1 for additional task-specific parameter counts, all 5 orders of magnitude smaller than the total parameter count. Optimization of them alone fails to fine-tune (see Appendix A).

3.2 GLUE BENCHMARK

The GLUE benchmark is a collection of diverse natural language understanding tasks (Wang et al., 2018). We fine-tune on these tasks and report the evaluation performances. We exclude the problematic WNLI set [3]. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for all other tasks.

3.3 CONSTRAINED FINE-TUNING PROCEDURES

For all fine-tuning procedures, we use the exact hyperparameters as described in the original paper (Devlin et al., 2018) unless specified otherwise, with additional constraints described as follows. No constraints are imposed on task-specific last layers $G^{(t)}$.

L_0-close fine-tuning To search for fine-tuned solutions that are L_0-close to the pre-trained parameters, we selectively fix certain parameter matrices at pre-trained values and perform fine-tuning optimization on a lower-dimensional parameter space.

[2] Pre-trained parameters obtained from `https://github.com/google-research/bert`.

[3] See (12) in `https://gluebenchmark.com/faq`.

Table 2: Fine-tuned parameters are L_1-close and angular-close to pre-trained ones. We compare measured distance metrics with expected distances between two independent random initializations, either uniformly or normally distributed from $-\frac{1}{\sqrt{H}}$ to $\frac{1}{\sqrt{H}}$ where $H = 768$ is the hidden dimension.

DISTANCE METRIC	BETWEEN UNIFORM INITIALIZATIONS	BETWEEN NORMAL INITIALIZATIONS	BETWEEN FINE-TUNED AND PRE-TRAINED ($[\min, \max]$)
L_1-DISTANCE	20.0 ± 0.1	16.7 ± 0.1	$[0.1, 3.3]$
ANGULAR DISTANCE	0.500	0.500	$[0.001, 0.027]$

Sparse (L_0-small) fine-tuning We use iterative pruning (Zhu & Gupta, 2017) during fine-tuning to produce sparse models. Pruning is based on weight magnitudes in each layer and is performed periodically during fine-tuning with sparsity gradually increasing from 0% to a final level according to a cubic schedule. Iterative pruning successfully ensures that parameters are L_0-small (see Appendix B).

Supermask training as fine-tuning (sparse and L_0-close) In order to search for fine-tuned networks that are both sparse and L_0-close to the pre-trained one, we reparameterize the model by a multiplicative binary mask

$$\boldsymbol{\theta} = \tilde{\boldsymbol{\theta}} \odot \boldsymbol{\mu}, \tag{3}$$

where $\tilde{\boldsymbol{\theta}}$ is the pre-trained parameters, and $\boldsymbol{\mu} \in \{0, 1\}^N$ the mask, N being the dimensionality of the parameter space and $\odot$ the Hadamard product.

If learning is purely through optimizing the mask $\boldsymbol{\mu}$ while holding $\tilde{\boldsymbol{\theta}}$ constant, the mask is called a *supermask* (Zhou et al., 2019). Since $\boldsymbol{\mu}$ is discrete-valued and thus not differentiable, we reparameterize $\boldsymbol{\mu}$ as

$$\boldsymbol{\mu} = \mathrm{Bern}(\sigma(\boldsymbol{\nu})), \tag{4}$$

where $\mathrm{Bern}(p)$ denotes an element-wise independent Bernoulli sampler with probability p, and $\sigma(\cdot)$ the sigmoid function, applied element-wise on $\boldsymbol{\nu} \in \mathbb{R}^N$, the continuous *mask parameter*. We treat gradient backpropagation through $\boldsymbol{\mu}$ as a straight-through estimator, similar to the techniques used in Mallya et al. (2018); Zhou et al. (2019). Same fine-tuning hyperparameters were used except for the learning rate (see Appendix C).

Control over the final sparsity is exerted by initialization of $\boldsymbol{\mu}$ for fine-tuning. We initialize $\boldsymbol{\nu}$ according to a soft magnitude-based pruning mask: a fraction of small-magnitude values are initialized to $\nu = -5$ and the rest to $\nu = 5$. We found that the initial sparsity directly controls the final sparsity (see Appendix D), allowing us to produce masks with sparsity levels ranging from 1% to 89%.

4 EXPERIMENTAL RESULTS

4.1 FINE-TUNED AND PRE-TRAINED PARAMETERS ARE L_1-CLOSE AND ANGULAR-CLOSE

We observe that the original fine-tuning procedures for GLUE tasks all take 10^2 to 10^4 parameter update steps (Table 1), negligible compared to the dimensionality of the parameter space, *viz.* 10^8. Thus, we first questioned whether fine-tuned parameters are indeed *close* to the pre-trained ones in parameter space. We measured the L_1-distances, *i.e.* L_1-norm of parameter difference, and angular distances (Table 2). Specifically, we inspect the weight matrices in self-attention layers, of size 768×768 where 768 is the hidden state dimension. We report the minimum and maximum values across GLUE tasks: MNLI showed the largest values, and RTE showed the smallest values. Evidently, we see a significantly higher L_1- and angular-closeness between fine-tuned and pre-trained parameters as compared to the expected distance between two independent random initializations. This suggests that, during the course of fine-tuning, the very *few* model parameter updates traversed a very *short* distance in the parameter space. Comparing the parameter distance across GLUE tasks, we find that it scales with the number of fine-tuning iterations (see Appendix E).

Further, we inspect the closeness in parameter subspaces for each layer. We found that, though all layers change very little during fine-tuning, there is nevertheless a high degree of variability across different parameter matrices (Figure 2). Blocks deeper in the encoder stack are less L_1-close but more angular-close than shallower ones. In all self-attention modules, value and output projection

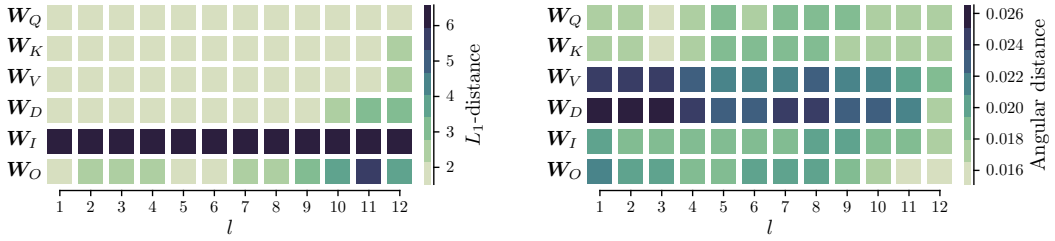

Figure 2: L_1- and angular distances in parameter subspaces between pre-trained and fine-tuned weights. Shown are metrics across the 12 encoder stack layers for the self-attention projection matrices ($\boldsymbol{W}_Q$, $\boldsymbol{W}_K$, $\boldsymbol{W}_V$ and $\boldsymbol{W}_D$) and feed-forward matrices ($\boldsymbol{W}_I$ and $\boldsymbol{W}_O$). The results presented here are for MNLI fine-tuning, but similar patterns are observed across all GLUE tasks.

Table 3: L_0-close fine-tuning results: layers excluded from fine-tuning, corresponding number of parameters remaining to fine-tune, and the fine-tuning performance on the MRPC task (F1 score). We report the mean and standard deviation across 10 independent runs.

LAYERS EXCLUDED FROM FINE-TUNING	TASK-SPECIFIC PARAMETER STORAGE	F1 SCORE
NONE (BASELINE)	109M FLOAT (100%)	89.4 ± 0.7
(1) KEY PROJECTION LAYERS IN SELF-ATTENTION	102M FLOAT (94%)	89.1 ± 0.8
(2) DEEPEST ENCODER STACK LAYERS	95M FLOAT (87%)	88.8 ± 0.9
(3) WORD EMBEDDING LAYER	86M FLOAT (78%)	89.3 ± 0.8
(1), (2), AND (3)	66M FLOAT (60%)	88.7 ± 0.9
(1), (2), AND (3) WITH 30% SPARSE FINE-TUNING	66M BINARY (60%)	87.4 ± 2.2
(1), (2), AND (3) WITH 40% SPARSE FINE-TUNING	66M BINARY (60%)	86.6 ± 2.2

matrices ($\boldsymbol{W}_V$ and $\boldsymbol{W}_D$) change significantly more than query and key projection matrices ($\boldsymbol{W}_Q$ and $\boldsymbol{W}_K$) during fine-tuning.

4.2 L_0-CLOSE FINE-TUNING

Next, inspired by the high degree of variability in each layer's parameter change during fine-tuning, we ask whether effective fine-tuning can be achieved by optimizing only a fraction of layers while having others fixed at pre-trained values, resulting in fine-tuned models L_0-close in parameter space.

Our results suggest this is indeed feasible (Table 3). Informed by different layers' sensitivity to fine-tuning, we performed fine-tuning experiments by progressively excluding: (1) key projection layers in self-attention across all encoder stack layers, (2) the penultimate and ultimate encoder stacks, and (3) the word embedding layer. Each of these exclusions independently or all three combined do not significantly degrade performance, while reducing the number of parameters to fine-tune by up to 40% (from 109M to 66M).

4.3 SPARSIFICATION AS FINE-TUNING

Encouraged by these results, we ask whether more aggressive constraints can be imposed to the fine-tuning process to further cut computational cost. Though L_0-close fine-tuning obviates optimization of a large fraction of parameters, avoiding full storage of all parameters for each fine-tuned task, all operations still need to be performed at inference time. In order to reduce operations, we seek to sparsify parameters. Combined with strict L_0-closeness, this amounts to a search over a binary mask in a high-dimensional parameter space. We adopt *supermask* training (see Section 3) to this end.

Figure 3 shows fine-tuned model performance across GLUE tasks obtained by supermask training. Final sparsity level of the supermask is controlled by its initialization (see Section 3 and Appendix D). We note that there is little task performance degradation between 1% and 40% of parameter sparsity. Layer-wise sparsity levels of supermasks also demonstrate systematic trends (Figure 4): (1) across GLUE tasks, $\boldsymbol{W}_Q$, $\boldsymbol{W}_K$ and $\boldsymbol{W}_I$ tend to be sparser than $\boldsymbol{W}_V$, $\boldsymbol{W}_D$ and $\boldsymbol{W}_O$, and (2) shallower encoder stack layers are sparser than deeper ones. Moreover, we show that supermask fine-tuning of only a fraction of sensitive layers could also achieve high performance (Table 3).

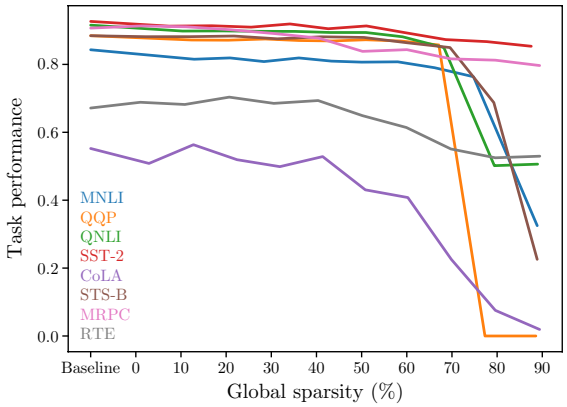

Figure 3: Performance of supermask fine-tuned models across GLUE tasks. We show the mean of performance metrics across 10 independent Bernoulli samplings. Note the baseline performance for each task marked by the leftmost end of each curve.

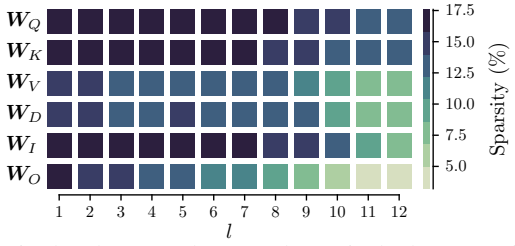

Figure 4: Supermask sparsity levels across layers. Shown is the low-sparsity MNLI supermask with a global sparsity level of 12.9%, but similar patterns are observed across all GLUE tasks.

4.4 Fine-tuned supermasks are nontrivial

How does the learning of a supermask actually work? Does a supermask trivially learn to prune away the weights with smallest magnitudes? To address these questions, we inspect the magnitudes of the pre-trained weights zeroed by the supermasks (Figure 5, Table 4). These weights turn out to have remarkably higher magnitudes than the next smallest entries, suggesting the learning of supermaks is mechanistically distinct from trivial magnitude-based pruning.

Table 4: Comparison between supermask pruned weights and magnitude-based pruned weights. Specifically, we compare the weights pruned with low-sparsity supermasks (initialized at 0% sparsity) and weights pruned with one-shot magnitude-based pruning at the same final sparsity. We report the maximum and mean magnitude of the pruned weights. The last row shows percentages of the overlap between the supermask and the magnitude-based pruning mask, *i.e.* the percentages of weights zeroed by the supermask that are also the smallest weights.

GLUE task	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE
Pruned max	0.0093	0.0093	0.0075	0.0059	0.0022	0.0018	0.0009	0.0007
Supermask max	1.7	6.4	2.5	1.7	1.1	2.8	1.8	2.8
Pruned mean	0.0033	0.0032	0.0026	0.0020	0.0008	0.0006	0.0003	0.0002
Supermask mean	0.032	0.033	0.033	0.035	0.037	0.036	0.038	0.036
Overlap	11.1%	10.0%	6.7%	3.6%	0.7%	0.7%	0.7%	0.7%

4.5 Occurrences and uniqueness of good, sparse fine-tuned supermasks

One surprising finding of this study is the many occurrences of *good* fine-tuned parameters among the 2^N configurations in the set $\left\{\boldsymbol{\theta} : \boldsymbol{\theta} = \tilde{\boldsymbol{\theta}} \odot \boldsymbol{\mu}, \boldsymbol{\mu} \in \{0,1\}^N\right\}$, *viz.* vertices of an N-dimensional hypercube. First, there exist supermasks up to 40% sparse without significant performance degrada-

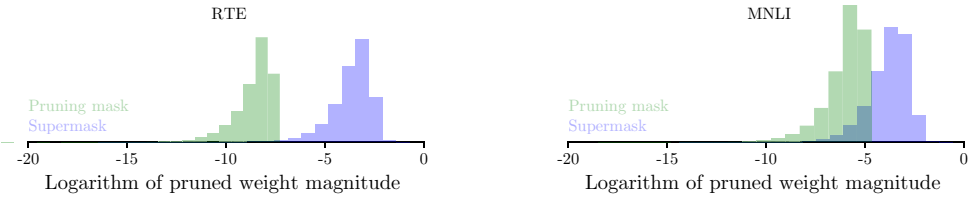

Figure 5: Pruned weight distributions, compared between supermask and magnitude-based pruning.

Table 5: Low-sparsity supermask performance. We report the sparsity levels achieved when the supermasks were initialized at 0% sparsity. For several tasks, fine-tuning is achieved with less than 3% of pre-trained weights pruned. For the supermask evaluation results, we include the mean and standard deviation of 10 Bernoulli samplings of a single run.

GLUE TASK	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE
BASELINE	84.3/85.6	88.5	91.6	92.7	55.2	88.5	90.7	67.1
SUPERMASK	81.5/82.9 ± 0.1	87.2 ± 0.1	89.8 ± 0.1	91.3 ± 0.2	50.8 ± 0.8	88.2 ± 0.1	91.3 ± 0.4	68.8 ± 1.0
FINAL SPARSITY	12.9%	12.6%	10.3%	7.4%	2.9%	2.2%	1.3%	1.0%

tion for all GLUE tasks, for some tasks even sparser (Figure 3). It is natural that performance drops as the mask becomes sparser; however, it is rather counterintuitive that there exist good supermasks at the dense end (Figure 3), because we know that the pre-trained model with only the task-specific last layer fine-tuned utterly fails to learn any task (Appendix A).

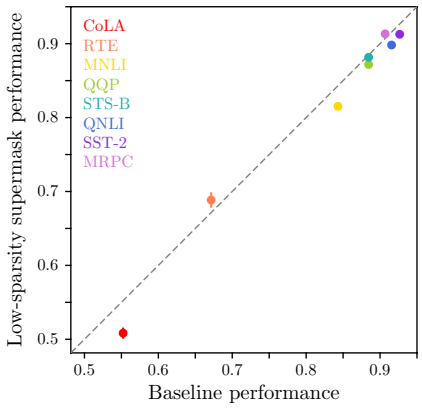

Figure 6: Low-sparsity supermask performance, *i.e.* task performance of super-masks initialized at 0% sparsity, compared against baseline.

To shed more light on this phenomenon, we study the supermasks trained with all-dense initialization (Figure 6). Surprisingly, these low-sparsity supermasks successfully learned to perform all the tasks without significant degradation from baseline. Essentially, complicated tasks like MNLI and QQP can be learned by clamping $12 - 13\%$ of the pre-trained weights to zero, whereas for simple tasks like MRPC and RTE, setting only $1 - 2\%$ of the pre-trained weight entries to zero suffices to learn the task (Table 5). Fine-tuning can indeed be very *fine*, suggesting relative frequent occurrences of *good* solutions within a sparse L_0-neighborhood of the pre-trained parameters.

Finally, we question whether the supermasks learned to perform different tasks share commonalities. Specifically, we quantify the amount of overlapping zeros in learned supermasks across different tasks (Figure 7). It seems the overlaps are not substantially larger than what would have been caused by chance, suggesting that, even though there seem to be many good supermasks for each task, these masks are largely distinct from each other, each unique to the task it learns.

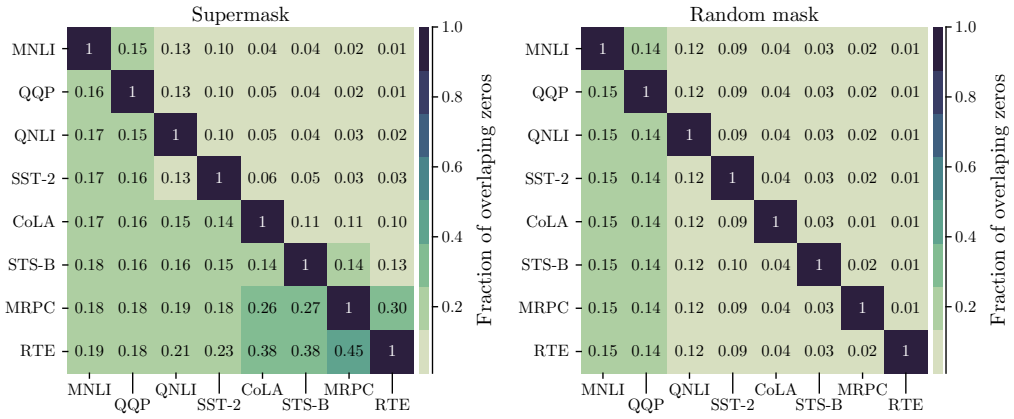

Figure 7: Fractions of overlap of zero elements in supermasks across GLUE tasks, compared to randomly generated masks. Each value in the grid shows the fraction of pruned elements in one task (horizontal axis) that are also pruned in the other (vertical axis). Here, we show low-sparsity supermasks (initialized at 0% sparsity) and compare the masks in the value layer of the first encoder, which is one of the most sparse layers in the entire model.

5 Conclusions

We show that, due to surprisingly frequent occurrences of *good* parameter configurations in the sparse L_0-vicinity of large pre-trained language models, two techniques are highly effective in producing efficient fine-tuned networks to perform specific language understanding tasks: (1) optimizing only the most sensitive layers and (2) learning to sparsify parameters. In contrast to commonly employed *post-training* compression methods that have to trade off with performance degradation, our procedure of generating sparse networks is by itself an optimization process that learns specific tasks.

References

Brian Cheung, Alex Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno Olshausen. Superposition of many models into one. feb 2019. URL http://arxiv.org/abs/1902.05522.

Matthieu Courbariaux and Yoshua Bengio. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv*, pp. 9, 2016. URL http://arxiv.org/abs/1602.02830.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. *Nips*, pp. 1–9, 2015. ISSN 10495258. doi: arXiv:1412.7024. URL http://arxiv.org/abs/1511.00363.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. oct 2018. URL http://arxiv.org/abs/1810.04805.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Arxiv.Org*, 7(3):171–180, 2015. ISSN 1664-1078. doi: 10.3389/fpsyg.2013.00124. URL http://arxiv.org/pdf/1512.03385v1.pdf.

Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). jun 2016. URL http://arxiv.org/abs/1606.08415.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep Learning Scaling is Predictable, Empirically. dec 2017. URL http://arxiv.org/abs/1712.00409.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning Sparse Neural Networks through L_0 Regularization. dec 2017. URL http://arxiv.org/abs/1712.01312.

Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a Single Network to Multiple Tasks by Learning to Mask Weights. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11208 LNCS, pp. 72–88, jan 2018. ISBN 9783030012243. doi: 10.1007/978-3-030-01225-0_5. URL http://arxiv.org/abs/1801.06519.

Massimiliano Mancini, Elisa Ricci, Barbara Caputo, and Samuel Rota Bulò. Adding new tasks to a single network with weight transformations using binary masks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11130 LNCS, pp. 180–189, may 2019. ISBN 9783030110116. doi: 10.1007/978-3-030-11012-3_14. URL http://arxiv.org/abs/1805.11119.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. 2019.

Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. (Nips), 2017. URL http://arxiv.org/abs/1706.03762.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. apr 2018. URL https://arxiv.org/abs/1804.07461.

Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are All Layers Created Equal? feb 2019. URL http://arxiv.org/abs/1902.01996.

Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask. may 2019. URL http://arxiv.org/abs/1905.01067.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. 2017. URL http://arxiv.org/abs/1710.01878.

APPENDIX A OPTIMIZATION OF TASK-SPECIFIC LAST LAYERS ALONE FAILS TO FINE-TUNE

Optimization of only task-specific layers does not lead to successful fine-tuning. For instance, for the MRPC task, freezing parameter weights in the pre-trained model and optimizing the task-specific last layer alone yields a low-performing model. Across 10 independent runs, the model consistently predicts all 1's for the paraphrase classification task, yielding an F1 score of 81.2. This is a significant degradation compared to the baseline performance of 89.4 ± 0.7 across multiple runs (Table 3). Thus, it is critical to fine-tune layers in the pre-trained model and not just the task-specific layers alone.

APPENDIX B SUPERMASK TRAINING VERSUS ITERATIVE PRUNING

Iterative pruning during fine-tuning (Figure 8) outperforms supermask training (Figure 3) at higher sparsity levels. While supermask training remains successful up to 40% sparsity, iterative pruning produces binary masks up to 50% sparse and for some tasks even sparser without significant performance degradation. However, while iterative pruning saves compute on appropriate hardware by performing sparse operations with model parameters, supermask training further saves memory by storing only binary masks of certain layers for each task.

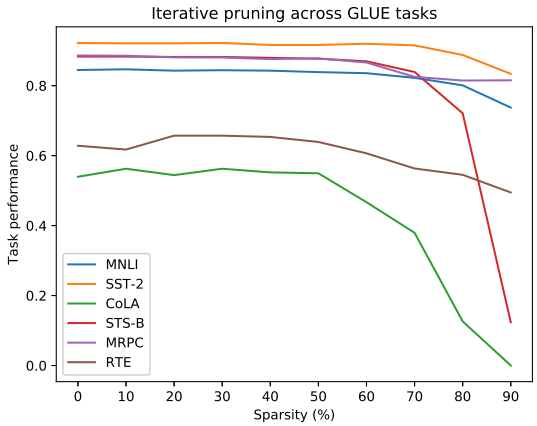

Figure 8: Iterative pruning during fine-tuning. We plot the evaluation performance at sparsity levels from 10% to 90% across GLUE tasks. Note the baseline performance for each task marked by the leftmost end of each curve (0% sparsity).

APPENDIX C LEARNING RATE OF SUPERMASK TRAINING

Supermask training requires a much larger learning rate compared to typical training (Zhang et al., 2019). While a learning rate of 2×10^{-5} is used for optimizing weights, a learning rate of 2×10^{-1} is used for optimizing masks. We notice a degradation in performance at smaller learning rates for supermask training (Table 6).

Table 6: MRPC low-sparsity supermask performance at learning rates from 2×10^{-5} and 2×10^{-1}. We note that supermask training requires a much higher learning rate than typical parameter training. At low learning rates, the model significantly degrades in performance and predicts all 0's for the paraphrase classification task, yielding an F1 score of 0.0. This pattern holds true across GLUE tasks.

LEARNING-RATE	2×10^{-1}	2×10^{-2}	2×10^{-3}	2×10^{-4}	2×10^{-5}
F1 SCORE	91.3 ± 0.4	82.0 ± 0.2	0.0	0.0	0.0

APPENDIX D CORRELATION BETWEEN INITIAL AND FINAL SPARSITIES OF SUPERMASKS

There is no straightforward to control the amount of weights pruned when training supermasks (Zhang et al., 2019). We find that setting the initial sparsity through a soft magnitude-based pruning mask controls the final sparsity level. Figure 9 shows this correlation between initial and final sparsities of supermasks across GLUE tasks.

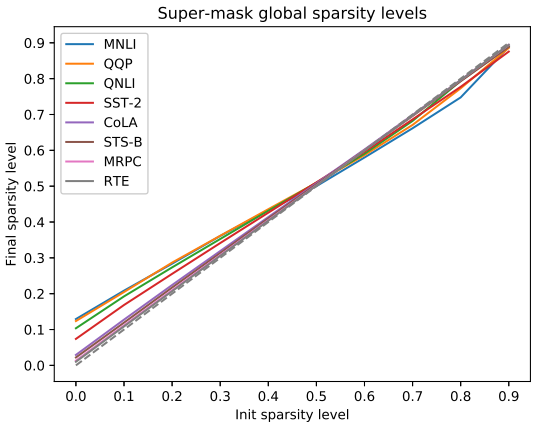

Figure 9: The supermasks are initialized to a soft magnitude-based pruning mask and the sparsity level shifts during supermask training. This figure shows the initial sparsity level plotted against the final sparsity level. We note that, at lower initial sparsity levels, the supermask is pushed to a greater sparsity level, whereas at higher sparsity levels, the supermask is pushed to a lower sparsity level. This pattern is similar across GLUE tasks but is most prominent in the MNLI task, scaling with the number of fine-tuning steps (Table 1).

APPENDIX E CORRELATION OF PARAMETER DISTANCE WITH FINE-TUNING STEPS

We find that parameter distance scales with the number of fine-tuning steps (Figure 10).

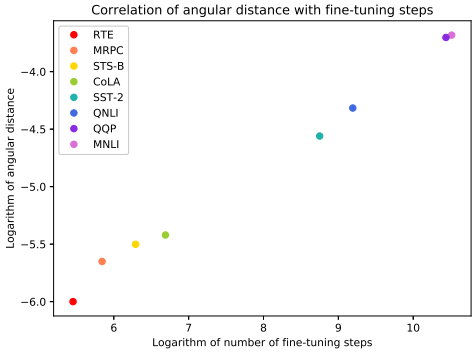

Figure 10: Scaling of parameter distance with the number of fine-tuning iterations. We find that angular distance correlates with the amount of fine-tuning (Table 1). Each data point corresponds to a different GLUE task.