How fine can fine-tuning be? Learning efficient language models

Evani Radiya-Dixit
Stanford University

Xin Wang
Cerebras Systems

Abstract

State-of-the-art performance on language understanding tasks is now achieved with increasingly large networks; the current record holder has billions of parameters. Given a language model pre-trained on massive unlabeled text corpora, only very light supervised fine-tuning is needed to learn a task: the number of fine-tuning steps is typically five orders of magnitude lower than the total parameter count. Does this mean that fine-tuning only introduces small differences from the pre-trained model in the parameter space? If so, can one avoid storing and computing an entire model for each task? In this work, we address these questions by using Bidirectional Encoder Representations from Transformers (BERT) as an example. As expected, we find that the fine-tuned models are close in parameter space to the pre-trained one, with the closeness varying from layer to layer. We show that it suffices to fine-tune only the most critical layers. Further, we find that there are surprisingly many good solutions in the set of sparsified versions of the pre-trained model. As a result, fine-tuning of huge language models can be achieved by simply setting a certain number of entries in certain layers of the pre-trained parameters to zero, saving both task-specific parameter storage and computational cost.

1 Introduction

Modern deep neural networks operate in a regime where the generalization gap diminishes with growing model capacity, defying the classical bias-variance trade-off (Belkin et al., 2018). Increased model capacity always leads to better generalization, a trend highly prominent in the natural language understanding domain. From BERT (340M parameters, Devlin et al., 2018), to GPT-2 (1.5B parameters, Radford et al., 2019), and to Megatron-LM (8.3B parameters, Shoeybi et al., 2019), state-of-the-art performance in language comprehension tasks keeps improving with larger model capacity.

These huge language models are first pre-trained on large text corpora. Pre-training is a learning procedure, often unsupervised, that yields a good common initialization for further supervised learning of various downstream tasks. This further learning, called fine-tuning, is an additional optimization of model parameters jointly with a very small number of extra task-specific parameters (e.g. Table 1).

Though larger models generalize better, they are more expensive computationally, and the costs grow with the number of tasks learned. The high computational cost is usually addressed by network compression techniques that produce compact models for efficient inference (e.g. Zhao et al., 2019). To reduce task-specific cost, transfer learning and continual learning methods are useful to maximize sharing of parameters across tasks (e.g. Houlsby et al., 2019; Liu et al., 2019). In this work, we attempt to achieve these two desirable outcomes with a single effort. We use Bidirectional Encoder Representations from Transformers (BERT, Devlin et al., 2018) and the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) as a working example.

Our intuition comes from the observation that the amount of fine-tuning necessary to learn each task is very small (five orders of magnitude smaller than the dimensionality of the parameter space, Table 1). This is not surprising: the high quality of a pre-trained model should naturally lead to rather few iterations needed to fine-tune it to perform specific tasks. But importantly, such light fine-tuning might result in fine-tuned models hypothetically closer to the pre-trained one in parameter space. This suggests a potentially high degree of computational redundancy across tasks that

1 Work done during internship at Cerebras Systems (current email: evanir@stanford.edu).
2 Correspondence to XW (poincare.disk@gmail.com).

Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).
How fine can fine-tuning be? Learning efficient language models

Figure 1: An illustration of \(L_0\)-close and sparsification constraints on a pre-trained parameter in a three-dimensional parameter space. Here \(\tilde{\theta}\) is the pre-trained parameters. Individual fine-tuning procedures for different tasks send the pre-trained parameters to distinct optimized parameters in a close \(L_1\)-neighborhood, e.g. \(\theta^{(1)}\), \(\theta^{(2)}\) and \(\theta^{(3)}\), of which each component, viz. \(\theta_1\), \(\theta_2\) or \(\theta_3\), is subject to change. The \(L_0\)-closeness constraint (red with its saturation encoding closeness) forces optimization within those parameter configurations that share a certain fraction of components with \(\tilde{\theta}\), i.e. having a small number of different components. The sparsification constraint (blue with its saturation encoding density) further confines optimization to a discrete subset of \(L_0\)-close parameters, where all changed components have to be set to zero.

A major contribution of the present work is the demonstration that fine-tuning can be realized by sparsification, which has favorable practical implications. 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 when there are many tasks to perform. By forcing fine-tuned parameters to be sparse, one potentially saves both memory and compute, because each task only requires a binary mask on top of the common pre-trained parameters and sparse linear algebraic operations could be used instead of dense ones.

2 Related work

A large body of literature is concerned with sparsification of large networks for efficient inference (e.g. Zhu and Gupta, 2017). 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 sparse connectivity patterns over randomly initialized network parameters, termed supermasks, suggesting that sparsification plays a role similar and complementary 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 to larger pre-trained language models, but also studied the sparsity-accuracy tradeoff in a systematic way. Houlsby et al. (2019) added adapter modules to pre-trained language models, achieving parameter sharing across multiple tasks, but not reducing the computational cost of the resultant fine-tuned networks. Also, note that randomly generated high-dimensional masks can also support multi-task learning, e.g. Cheung et al. (2019).

To impose parameter sparseness differentially in combination with the \(L_0\)-closeness constraint, instead of principled approaches to imposing \(L_0\)-regularization (Louizos et al., 2017), we used the simpler straight-through estimator, much like binary quantization techniques (Courbariaux et al., 2015; Courbariaux and 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 \(F_0 : x \mapsto F(x; \theta)\) that transforms input sequence \(x\) into a good representation. It
is parameterized by $\theta$, noted as subscript for convenience. The fine-tuning procedure to perform a task $t \in T$ ($T$ being the set of tasks) can be described as a supervised training procedure of model $G^{(t)}_\phi \circ F_\theta : x \mapsto y$ on fine-tuning set $\{(x^{(i)}, y^{(i)})\}$. $G^{(t)}_\phi$ is a task-specific last layer unique to task $t$ and is parameterized by $\phi$; $\circ$ denotes function composition.

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

$$F_\theta = \text{BERT}_\theta \triangleq F_{\theta_{L+1}} \circ B_{\theta_L} \circ \cdots \circ B_{\theta_0} \circ E_{\theta_0},$$

(1)

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

$$B_\theta : x \mapsto \mathcal{LN}(x + DO(W_Q \text{GeLU}(W_I \mathcal{LN}(x) + DO(W_D A(W_Q x, W_K x, W_V x))))),$$

(2)

where $\theta \triangleq [W_Q, W_K, W_V, W_D, W_I, W_O]$ collects all the learnable parameter matrices in the block. $W_Q$, $W_K$, $W_V$ and $W_D$ are the query, key, value, and dense self-attention projection matrices, respectively. $W_I$ and $W_O$ are the intermediate and output feed-forward matrices, respectively. $\mathcal{A}(\cdot, \cdot, \cdot)$ represents multi-head scaled dot-product attention \cite{vaswani2017attention}, $DO(\cdot)$ dropout, $\mathcal{LN}(\cdot)$ layer normalization, and GeLU(·) the Gaussian error linear unit activation function \cite{hendrycks2016gaussian}. We experimented with the BERT$_{BASE}$ model \cite{devlin2018bert}, for which $L = 12$, with total parameter count of 109M\textsuperscript{4}. See Table\textsuperscript{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\textsuperscript{A}).

### 3.2 GLUE benchmark

The GLUE benchmark is a collection of diverse natural language understanding tasks \cite{wang2018glue, warstadt2018neural, socher2013semantic, dolan2005automatically, cer2018glue, williams2018broad, raij2016squad, dagan2006pascual, bar2006wisconsin, gianniccolo2007ubiversity, bentivogli2009sources}. We exclude the problematic WNLI set\textsuperscript{5} \cite{levesque2012winograd}. We fine-tune on these tasks and report the evaluation performances. F1 is reported for QQP and MRPC, Matthews correlation for CoLA, Pearson and Spearman correlation for STS-B, and accuracy 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 \cite{devlin2018bert} unless specified otherwise, with additional constraints described as follows. No constraints are imposed on task-specific last layers $G^{(t)}_\phi$.

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

**Sparsification as fine-tuning** In order to search for fine-tuning networks that are both sparse and $L_0$-close to the pre-trained one, we reparameterize the model by a multiplicative binary mask

$$\theta = \tilde{\theta} \odot \mu,$$

(3)

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

If learning is purely through optimizing the mask $\mu$ while holding $\tilde{\theta}$ constant, the mask is called a super-mask \cite{zhou2019supermask}. Since $\mu$ is discrete-valued and thus not differentiable, we reparameterize $\mu$ as

$$\mu = \text{Bern}(\sigma(\nu)),$$

(4)

where $\text{Bern}(\rho)$ denotes an element-wise independent Bernoulli sampler with probability $\rho$, and $\sigma(\cdot)$ the sigmoid function, applied element-wise on $\nu \in \mathbb{R}^N$, the continuous mask parameter that is task-specific. We treat gradient backpropagation through $\mu$ as a straight-through estimator, similar to the techniques used in \cite{mallya2018structured, zhou2019supermask}. Same fine-tuning hyperparameters as described in \cite{devlin2018bert} were used except for the learning rate (see Appendix\textsuperscript{B}).

Control over the final sparsity\textsuperscript{6} is exerted by initialization of $\mu$ for fine-tuning. We initialize $\nu$ according to a

---

\textsuperscript{4} Pre-trained parameters obtained from \url{https://github.com/google-research/bert}.

\textsuperscript{5} See (12) in \url{https://gluebenchmark.com/faq}.

\textsuperscript{6} Defined as the fraction of zero components, equal to one minus density.
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. $10^9$. Thus, we first asked 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 all self-attention layers, of size $768 \times 768$ where 768 is the hidden state dimension. We report the minimum and maximum values across GLUE tasks: RTE showed the smallest values, and MNLI showed the largest 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, or that between an initialization and the pre-trained paremeters. This confirms 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 D).

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 dense projection matrices ($W_V$ and $W_D$) change considerably more than query and key projection matrices ($W_Q$ and $W_K$) during fine-tuning.

**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. $10^9$. Thus, we first asked 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 all self-attention layers, of size $768 \times 768$ where 768 is the hidden state dimension. We report the minimum and maximum values across GLUE tasks: RTE showed the smallest values, and MNLI showed the largest 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, or that between an initialization and the pre-trained paremeters. This confirms 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 D).

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 dense projection matrices ($W_V$ and $W_D$) change considerably more than query and key projection matrices ($W_Q$ and $W_K$) during fine-tuning.

**4.2 $L_0$-close fine-tuning**

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 substantially 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 on the fine-tuning process.
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, as well as those between the pre-trained and a random initialization. Statistics presented in the rightmost column are across all GLUE tasks.

| Distance metric | Between uniform initializations | Between normal initializations | Between uniform initialization and pre-trained | Between normal initialization and pre-trained | Between fine-tuned and pre-trained ([min, max]) |
|-----------------|--------------------------------|-------------------------------|---------------------------------------------|---------------------------------------------|------------------------------------------------|
| \(L_1\)-distance | 20.0 ± 0.1                      | 16.7 ± 0.1                    | 41.9 ± 15.2                                 | 40.9 ± 15.4                                 | [0.1, 3.3]                                    |
| Angular distance | 0.500                           | 0.500                         | 0.500                                       | 0.500                                       | [0.001, 0.027]                                |

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); other GLUE tasks show similar patterns. 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 |

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

to further reduce computational cost. Though \(L_0\)-close fine-tuning obviates optimization of a substantial 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. 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 C). We note that there is little task performance degradation between 1% and 40% parameter sparsity, very close to sparse networks produced by iterative pruning [Zhu and Gupta, 2017] but underperforming it at high sparsity levels (see Appendix E). Layer-wise sparsity levels of supermasks also demonstrate systematic trends (Figure 4): (1) across GLUE tasks, \(W_Q\), \(W_K\) and \(W_I\) tend to be sparser than \(W_V\), \(W_D\) and \(W_O\), 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 performance with little degradation from baseline (Table 3).

4.4 Many good, sparse fine-tuned supermasks exist, but for pre-trained parameters only

One surprising finding of this study is the many occurrences of good fine-tuned parameters among the \(2^N\) configurations in the set \(\{\theta : \theta = \theta \odot \mu \mid \mu \in \{0, 1\}^N\}\) viz. vertices of an \(N\)-dimensional hypercube, even though most of them are quite distant from the pre-trained parameters by \(L_1\)-metric.

First, there exist supermasks up to 40% sparse without remarkable performance degradation for all GLUE tasks, for some tasks even sparser (Figure 3 right end). Second, for any task, below this maximum sparsity, we found good masks at any sparsity level (Figure 3), which can be controlled by initialization of the supermask (see Appendix C). Finally, while it is natural that performance drops as the mask becomes extremely sparse (Figure 3 right end), it is rather counterintuitive that there exist good supermasks at the dense extreme (Figure 3 left end), since we observe that the pre-trained model with only the task-specific last layer fine-tuned utterly fails to learn any task (Appendix A). Noticeably, good supermasks selectively prune important weights of large magnitudes (Appendix E).

To understand this phenomenon better, we study the su-
permarks trained with all-dense initialization (Figure 5). Surprisingly, these low-sparsity supermasks successfully learn to perform all the tasks without noticeable degradation from baseline. Essentially, complicated tasks like MNLI and QQP can be learned by clamping a small fraction of the pre-trained weights to zero (see Appendix A for how model performance improves with sparsity), 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 4). Fine-tuning can indeed be very fine, suggesting relatively frequent occurrences of good solutions within a sparse $L_0$-neighborhood of the pre-trained parameters.

Moreover, we ask whether such frequent occurrences of good sparsified versions of parameters is a unique property of the pre-trained weights. In other words, can one also obtain good supermasks on parameters that are not pre-trained? To answer this question, we perform supermask fine-tuning on pre-trained parameters with components shuffled (thusly norm-preserved). Performance degrades significantly, for instance, for the MRPC task, from an F1 score of 91.3 to 81.2 with shuffled pre-trained parameters. It is clear that one cannot obtain any good masks by doing so, suggesting that having high-performance sparsified versions is unique to pre-trained parameters.

### 4.5 Task-uniqueness of fine-tuned supermasks

Finally, we ask 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 6). 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.

### 5 Discussion

One very puzzling fact about modern deep neural networks is that, despite the large number of parameters, overparameterization helps both generalization and optimization. On the one hand, given an effective network architecture reflecting proper inductive biases, better generalizing models are always larger models (Hestness et al., 2017). On the other hand, sheer increases in dimensionality of the parameter space seldom make stochastic gradient-based optimization more difficult: deeper and/or wider networks take just about the same, if not a lower number of training iterations to converge. For example, ResNet-18 (11.7M parameters) and ResNet-152 (60.2M parameters) both train to converge, at similar convergence rates, within no more than 600K iterations on ImageNet (He et al., 2015). Thus, 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.
sparsity supermasks (initialized at 0% sparsity) and 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-sparse masks in the value layer of the first encoder, which is one of the most sparse layers in the entire model.

Figure 6: 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-sparse masks (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.

We show that, due to surprisingly frequent occurrences of good parameter configurations in a close $L_0$-neighborhood and in the set of sparsified 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 pre-trained parameters as fine-tuning. In contrast to commonly employed post-training sparsification methods which always incur performance degradation, our procedure of sparsifying pre-trained networks (similar to Mallya et al., 2018; Mancini et al., 2019) is by itself an optimization process that learns specific tasks.

6 Acknowledgements

We thank Sofia Samaniego de la Fuente for help with the experiments. We also wish to thank Robert S. Schreiber, Urs Körster, Jorge Albericio, Natalia S. Vassilieva, and Marcel Nassar for discussions and feedback on the manuscript.

References

Bar-Haim, R., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B., and Szpektor, I. (2006). The Second PASCAL Recognising Textual Entailment Challenge.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2018). Reconciling modern machine learning and the bias-variance trade-off.

Bentivogli, L., Clark, P., Dagan, I., and Giampiccolo, D. (2009). The Sixth PASCAL Recognizing Textual Entailment Challenge.

Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., and Specia, L. (2018). SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation. pages 1–14. Association for Computational Linguistics (ACL).

Cheung, B., Terekhov, A., Chen, Y., Agrawal, P., and Olshausen, B. (2019). Superposition of many models into one.

Courbariaux, M. and Bengio, Y. (2016). BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. arXiv, page 9.

Courbariaux, M., Bengio, Y., and David, J.-P. (2015). BinaryConnect: Training Deep Neural Networks with binary weights during propagations. Nips, pages 1–9.

Dagan, I., Glickman, O., and Magnini, B. (2006). The PASCAL Recognising Textual Entailment Challenge. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 3944 LNAI, pages 177–190.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

Dolan, W. B. and Brockett, C. (2005). Automatically Constructing a Corpus of Sentential Paraphrases. Proceedings of the Third International Workshop on Paraphrasing (IWP2005), pages 9–16.

Frankle, J. and Carbin, M. (2018). The Lottery Ticket Hypothesis: Finding Small, Trainable Neural Networks.

Giampiccolo, D., Magnini, B., Dagan, I., and Dolan, B. (2007). The Third PASCAL Recognizing Textual Entailment Challenge. Technical report.
How fine can fine-tuning be? Learning efficient language models

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. *Arxiv.org*, 7(3):171–180.

Hendrycks, D. and Gimpel, K. (2016). Gaussian Error Linear Units (GELUs).

Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M. M. A., Yang, Y., and Zhou, Y. (2017). Deep Learning Scaling is Predictable, Empirically.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-Efficient Transfer Learning for NLP.

Levesque, H. J., Davis, E., and Morgenstern, L. (2012). The Winograd Schema Challenge. Technical report.

Liu, X., He, P., Chen, W., and Gao, J. (2019). Multi-Task Deep Neural Networks for Natural Language Understanding.

Louizos, C., Welling, M., and Kingma, D. P. (2017). Learning Sparse Neural Networks through $L_0$ Regularization.

Mallya, A., Davis, D., and Lazebnik, S. (2018). 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, pages 72–88.

Mancini, M., Ricci, E., Caputo, B., and Bulò, S. R. (2019). 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, pages 180–189.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100,000+ questions for machine comprehension of text. In *EMNLP 2016 - Conference on Empirical Methods in Natural Language Processing. Proceedings of the Conference*, pages 2383–2392. Association for Computational Linguistics (ACL).

Shankar Iyer, Dandekar, N., and Csernai, K. (2017). First Quora Dataset Release: Question Pairs.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. (2019). Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism.

Socher, R., Perelygin, A., Wu, J., Y., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP 2013 - 2013 Conference on Empirical Methods in Natural Language Processing. Proceedings of the Conference*, pages 1631–1642. Association for Computational Linguistics.

Vaswani, A., Uszkoreit, J., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention Is All You Need. (Nips).

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018). GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding.

Warstadt, A., Singh, A., and Bowman, S. R. (2018). Neural Network Acceptability Judgments.

Williams, A., Nangia, N., and Bowman, S. (2018). A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. pages 1112–1122. Association for Computational Linguistics (ACL).

Zhang, C., Bengio, S., and Singer, Y. (2019). Are All Layers Created Equal?

Zhao, S., Gupta, R., Song, Y., and Zhou, D. (2019). Extreme Language Model Compression with Optimal Subwords and Shared Projections.

Zhou, H., Lan, J., Liu, R., and Yosinski, J. (2019). Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask.

Zhu, M. and Gupta, S. (2017). To prune, or not to prune: exploring the efficacy of pruning for model compression.
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 non-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  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 $2 \times 10^{-5}$ is used for optimizing weights, a learning rate of $2 \times 10^{-1}$ is used for optimizing masks. We notice a degradation in performance at smaller learning rates for supermask training (Table 5). This pattern holds true across GLUE tasks.

Table 5: MRPC low-sparsity supermask performance at learning rates from $2 \times 10^{-5}$ and $2 \times 10^{-1}$.

| Learning-rate | F1 score    |
|---------------|-------------|
| $2 \times 10^{-1}$ | 91.3 ± 0.4 |
| $2 \times 10^{-2}$ | 82.0 ± 0.2 |
| $2 \times 10^{-3}$ | 0.0         |
| $2 \times 10^{-4}$ | 0.0         |
| $2 \times 10^{-5}$ | 0.0         |

Appendix C  Correlation between initial and final sparcities of supermasks

There is no straightforward control of the amount of weights pruned in previous reports of supermask training (Zhang et al., 2019; Mallya et al., 2018). We find that setting the initial sparsity through a soft magnitude-based pruning mask controls the final sparsity level, which we use to produce supermasks of varied sparsity levels. Figure 7 shows this correlation between initial and final sparcities of supermasks for different GLUE tasks. 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 D  Correlation of parameter distance with fine-tuning steps

In order to understand how distance in parameter space increases as a function of fine-tuning steps, we study this relationship across GLUE tasks. We find that parameter distance scales with the number of fine-tuning steps by a power law with exponent close to 0.5 (Figure 8).

Appendix E  Fine-tuning with iterative pruning

We also use iterative pruning (Zhu and 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 during fine-tuning (Figure 9) 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%
How fine can fine-tuning be? Learning efficient language models

Figure 9: 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).

Figure 10: Pruned weight distributions, compared between supermask and magnitude-based pruning. Shown for the RTE and MNLI fine-tuning tasks.

sparse and for some tasks even sparser without significant performance degradation. Though iterative pruning produces sparse models, the fine-tuned models do not share parameters—one still needs to store all parameters for each task. Fine-tuned supermasks, on the other hand, store only a binary mask of certain layers for each task, with all tasks sharing a same set of underlying pre-trained weights.

Appendix F Fine-tuned supermasks are not trivial

How does the learning of a supermask actually work? Does a supermask simply learn to prune away the weights with smallest magnitudes? Since pure magnitude-based pruning of pre-trained weights does not perform any task-specific learning, we reason that the weight entries being set to zero by the supermask must have significant values. Here, we inspect the magnitudes of the pre-trained weights zeroed by the supermasks (Figure 10 Table 6). These weights turn out to have remarkably higher magnitudes than the smallest entries, suggesting the learning of supermasks is not trivial magnitude-based pruning.

Appendix G Learning curves of low-sparsity supermask fine-tuning

Our results suggest that supermask fine-tuning, if initialized at 0% sparsity, gradually increases sparsity during optimization, reaching a final sparsity level that correlates with the number of fine-tuning steps (Table 4). For MNLI, the GLUE task with the most fine-tuning steps, the sparsity level reaches 12.9%. We ask how prediction accuracy grows with sparsity during fine-tuning. As shown in Figure 11 like model performance, sparsity rapidly grows during the initial phase of fine-tuning. This makes model performance increase roughly linearly with sparsity.

Figure 11: Learning curves of MNLI low-sparsity supermask fine-tuning.
Table 6: Comparison between weights pruned with low-sparsity supermasks (initialized at 0% sparsity) and weights pruned with 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, \textit{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% |