Training Neural Networks with Fixed Sparse Masks

Anonymous Author(s)
Affiliation
Address
email

Abstract

During typical gradient-based training of deep neural networks, all of the model's parameters are updated at each iteration. Recent work has shown that it is possible to update only a small subset of the model's parameters during training, which can alleviate storage and communication requirements. In this paper, we show that it is possible to induce a *fixed* sparse mask on the model's parameters that selects a subset to update over many iterations. Our method constructs the mask out of the k parameters with the largest Fisher information as a simple approximation as to which parameters are most important for the task at hand. In experiments on parameter-efficient transfer learning and distributed training, we show that our approach matches or exceeds the performance of other methods for training with sparse updates while being more efficient in terms of memory usage and communication costs.

1 Introduction

Stochastic gradient descent (SGD) is a vital component of the modern pipeline for training deep neural networks. Along with the back-propagation algorithm, gradient descent allows for the efficient minimization of a loss function by gradually updating a model's parameters. SGD minimizes the loss over a small random subset of the dataset at each training iteration, which allows training over large datasets. In practice, minimizing an over-parametrized neural network's training loss using SGD often results in models that generalize well to new data [24, 15, 3], making it an invaluable tool.

While effective, SGD requires that all model parameters are updated at every iteration of training. As a result, communicating changes to the model requires communicating the updated value of every parameter. Since modern neural networks often have millions or billions of parameters [9, 7, 32], this communication can become excessively expensive. A concrete example of the negative impacts of these costs can be motivated in the setting of transfer learning. In transfer learning, a model's parameters are initialized from an existing pre-trained model before being fine-tuned (i.e. trained) on a task of interest. Pre-trained models can be fine-tuned a huge number of times – for example, the Hugging Face model repository[1] has thousands of fine-tuned variants of the BERT model [11]. Each of these fine-tuned variants requires a unique copy of the model's parameters, each of which takes up around 500MB of disk space. Relatedly, in distributed training [10] and federated learning [27], workers compute updates for a centralized model in parallel on different subsets of data. After a certain number of updates, the workers each communicate the newly-computed parameter values back to the centralized model. The communication step can cause a significant amount of overhead, particularly when the model is large, since the workers must communicate all parameter values when using standard SGD.

These issues could be mitigated by only updating a small subset of model parameters during training, while ideally still maintaining performance close to that of training all parameters. For example,

[1] `https://huggingface.co/models`

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

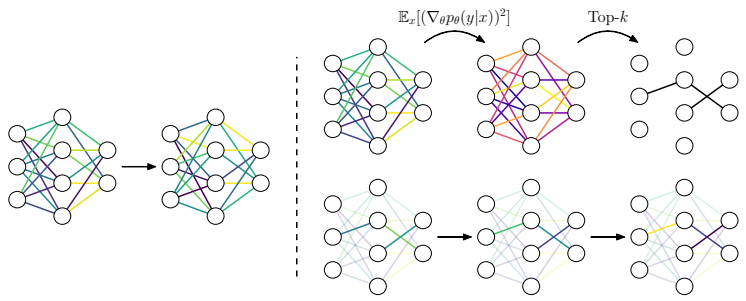

Figure 1: Diagram comparing our proposed method to standard SGD. In traditional gradient-based training (left), all of a model's parameters are updated at every iteration. We propose FISH Mask, a method for precomputing a sparse subset of parameters to update over many subsequent training iterations. To construct the FISH Mask, we find the k parameters with the largest Fisher Information (right, top). Then, we train the model with traditional gradient descent, but only update those parameters chosen by the mask (right, bottom).

Adapters [17, 33, 5] introduce additional parameters into a pre-trained model in the form of small task-specific modules that are fine-tuned while the rest of the model's parameters are kept fixed. Relatedly, diff pruning [14] and BitFit [6] demonstrate that it is possible to fine-tune a model while only updating a small subset of the existing parameters. In the distributed and federated learning settings, Aji and Heafield [2] and Konečnỳ et al. [21] have shown that it is possible for each worker to only update a sparse subset of a model's parameters, thereby reducing communication costs.

Existing methods for training with sparse updates typically work in one of three ways: they either add parameters to the model (Adapters), choose a hand-defined and heuristically-motivated subset of parameters (BitFit), or allow the subset of parameters to change over the course of training (Diff Pruning and methods in distributed and federated training). In this paper, we argue for *pre-computing* a sparse subset of existing parameters to update before training begins and keeping the subset *fixed* over the course of training. This approach yields various benefits: First, by updating a subset of existing parameters, we avoid any increase in the total size of the model. Second, by avoiding hand-defining the mask, we can ensure that our procedure is model-agnostic. Third, by pre-computing the mask, we avoid computational and memory overheads that are apparent when updating the mask over the course of training. It also allows workers in the distributed training setup to operate on strictly complementary subsets of parameters. Finally, keeping the mask fixed over many iterations, we can ensure that only a specific fixed number of parameters are updated. We are not aware of any existing techniques that satisfy these desiderata.

Motivated by these benefits, we introduce a new method for pre-computing fixed sparse masks. Our approach first estimates the importance of each parameter using an empirical approximation of its Fisher information (i.e. squared gradient norm). Then, we construct the mask by choosing the k parameters with the largest Fisher information. The resulting mask, which we deem a "FISH (**F**isher-**I**nduced **S**parse unc**H**anging) mask", can be re-used for many subsequent training iterations. We demonstrate the effectiveness of using a FISH Mask in a wide variety of settings, including parameter-efficient transfer learning, distributed training with long delays across workers, and reducing checkpoint size. Broadly speaking, FISH Mask training can dramatically reduce memory and communication requirements, while sacrificing minimal performance compared to standard gradient descent and outperforming relevant prior work on training with sparse updates.

2 The Fisher-Induced Sparse uncHanging (FISH) Mask

The main contribution of this paper is a method for pre-computing a sparse subset of a model's parameters to update over many subsequent training iterations. To construct such a subset, we use an approximation of each parameter's Fisher Information as a signal of how important the parameter is for a given task. We refer to the resulting mask (i.e. binary array indicating which parameters are included in the subset) as a FISH (**F**isher-**I**nduced **S**parse unc**H**anging) mask. In this section, we provide the necessary background and detail the steps necessary for computing a FISH Mask. This process is diagrammed in fig. 1.

2.1 Fisher Information

Our goal is to select a subset of parameters to update that are the most important in some sense. One way to measure a parameter's importance is to consider how much changing the parameter will change the model's output. If we represent the output distribution over y produced by a model with parameter vector $\theta \in \mathbb{R}^{|\theta|}$ given input x as $p_\theta(y|x)$, then we might be interested in measuring $\mathrm{D}_{\mathrm{KL}}(p_\theta(y|x) \mid\mid p_{\theta+\delta}(y|x))$, where $\delta \in \mathbb{R}^{|\theta|}N$ is a small perturbation. It can be shown [26, 29] that as $\delta \to 0$, to second order,

$$\mathbb{E}_x \mathrm{D}_{\mathrm{KL}}(p_\theta(y|x) \mid\mid p_{\theta+\delta}(y|x)) = \delta^{\mathrm{T}} F_\theta \delta + O(\delta^3) \tag{1}$$

where $F_\theta \in \mathbb{R}^{|\theta|\times|\theta|}$ is the Fisher Information matrix [12, 4], defined as

$$F_\theta = \mathbb{E}_{x\sim p(x)} \left[\mathbb{E}_{y\sim p_\theta(y|x)} \nabla_\theta \log p_\theta(y|x) \nabla_\theta \log p_\theta(y|x)^{\mathrm{T}} \right] \tag{2}$$

Given this relation, it can be seen that the Fisher Information matrix is closely connected to how much each parameter affects the model's predictions. Indeed, this has led the Fisher Information matrix to be widely used in modern machine learning, e.g. as a measure of parameter importance [19], as a preconditioner in gradient descent [4, 29, 26], or as a way to measure the amount of "information" in each parameter of a neural network [1].

When applied to large neural networks, the $|\theta| \times |\theta|$ Fisher Information matrix is intractable to compute. Prior work therefore frequently approximates F_θ as a diagonal matrix, or equivalently, as a vector in $\mathbb{R}^{|\theta|}$. Separately, when training machine learning models we seldom have the ability to draw samples $x \sim p(x)$; instead, we are given a finite training set of such samples. Furthermore, it is rarely necessary to compute the expectation over x in eq. (2) over the full training set; instead, it can often be well-approximated over a N samples $x_1, \ldots, x_N$. These constraints result in the following common approximation:

$$\hat{F}_\theta = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{y\sim p_\theta(y|x_i)} (\nabla_\theta \log p_\theta(y|x_i))^2 \tag{3}$$

where $F_\theta \in \mathbb{R}^{|\theta|}$. This approximation also has an intuitive interpretation: A given entry in $\hat{F}_\theta$ corresponds to the average of the gradient of the model's output with respect to a given parameter. If a given parameter heavily affects the model's output, then its corresponding entry in $\hat{F}_\theta$ will be large, so we can reasonably treat $\hat{F}_\theta$ as an approximation of the importance of each parameter.

Note that both eq. (2) and eq. (3) include an expectation over $y \sim p_\theta(y|x)$. When the number of classes is small, this expectation can be computed exactly. For tasks with many possible classes, it is common to approximate the expectation with a few samples from $p_\theta(y|x)$. In supervised learning settings, we have access to the ground-truth label y_i for each sample x_i in our training set. This leads to the possibility of replacing $\mathbb{E}_{y\sim p_\theta(y|x_i)}(\nabla_\theta \log p_\theta(y|x_i))^2$ in eq. (3) with $\nabla_\theta \log p_\theta(y_i|x_i)^2$. Performing this approximation is referred to as the "empirical Fisher". It has been shown that using the empirical Fisher can lead to degenerate behavior when used as a preconditioner in an optimizer [26, 23]. Since our use of the Fisher Information is largely based on a heuristically-motivated notion of parameter importance, we experimented with both the empirical and standard (eq. (3)) Fisher approximations and found that the empirical Fisher approximation was faster to compute than the standard Fisher approximation, while giving similar performance. We discuss this further in section 4.4.1.

2.2 Computing Fixed Sparse Masks

Recall that our goal is to select a subset of parameters that can be updated (while the remainder of parameters remain fixed) over many iterations of training. Having established the Fisher Information as a useful tool for estimating the importance of a model's parameters, to compute a FISH Mask we therefore first compute the approximate Fisher information (as described in the previous section) for all of a model's parameters. Then, to construct the mask, we simply choose the k parameters with the largest Fisher Information, where k is set according to the desired sparsity level. Specifically, a FISH Mask comprises the parameters $\{\theta_i \mid \hat{F}_{\theta_i} \geq \texttt{sort}(\hat{F}_\theta)_k\}$. Computing the FISH Mask is cheap because $\hat{F}_\theta$ can be computed efficiently using backpropagation, and (as we will show in **??**) we can obtain a reliable mask for relatively small values of N. Further, the fact that we re-use the mask for

many iterations prevents us from having to compute $\hat{F}_\theta$ frequently. As we will show in section 4, we find that this simple procedure is sufficient to produce a mask that can be reused for many iterations (over 100,000 iterations in some cases) in a wide variety of settings without sacrificing substantial performance compared to standard gradient-based training.

3 Related Work

Our approach bears similarity and takes inspiration from existing approaches for parameter-efficient transfer learning and distributed training of machine learning models. In this section, we outline related methods, some of which we will compare to directly in section 4.

3.1 Parameter-Efficient Transfer Learning

Transfer learning [28], where a model is initialized from a pre-trained checkpoint before being fine-tuned on a related downstream task, can dramatically improve performance and speed up convergence on the downstream task [11, 8, 32]. Standard practice is to update all of the model's parameters during fine-tuning, though in some cases reasonable performance can be attained by only fine-tuning the output layer of the model [18, 8, 30]. Training only the output layer has the benefit that adapting a given pre-trained model to a downstream task only requires adding a relatively small number of new parameters, but typically results in worse performance compared to training the full model.

Various methods have been proposed that endeavor to match the performance of fine-tuning the full model while only updating or adding a small amount of parameters. Adapters [17, 33, 5] are small subnetworks that are added between a pre-trained neural networks layers. Various works have shown that, when appropriately designed, updating only the parameters in the adapters and the output layer can approach the performance of fine-tuning all parameters. The main drawback of this approach is that the adapters themselves can add a non-negligible number of parameters to the model - for example, Houlsby et al. [17] add on average 3.6% more parameters to adapt a pre-trained BERT model [11] to tasks in the GLUE benchmark [35].

More closely related to our approach are methods for choosing a small subset of the model's existing parameters to update. Guo et al. [14] propose "Diff Pruning", where a sparse binary mask is found over the course of training that denotes which parameters should be updated or fixed at the value from the pre-trained model. Sparsity in the binary mask is enforced through a smooth approximation of the L_0 norm introduced by Louizos et al. [25]. Guo et al. [14] also show improved performance by enforcing that the mask has a structured sparsity according to which parameters correspond to a given weight matrix or bias vector. Ultimately, Diff Pruning is shown to outperform Adapters in terms of both sparsity and performance when applied to fine-tuning BERT on the GLUE benchmark. However, using Diff Pruning requires significantly more memory during training in order to store and update the mask. A more recent result by Ben-Zaken et al. [6] demonstrated that simply updating the bias parameters in BERT can attain competitive performance with Diff Pruning. While this provides a simple and strong baseline, it is not universally applicable; for example, the pre-trained T5 model [32] does not have any bias vectors. In section 4, we show that using a FISH Mask outperforms all of these approaches parameter-efficient fine-tuning of BERT on GLUE.

3.2 Distributed Training

As models and datasets grow, it becomes efficient or impossible to train a model on a single machine. This has motivated the need for distributed training strategies where computation for training a model is shared across many machines (called workers) [10]. A major consideration in distributed training are computation costs, since workers need to regularly communicate parameter updates with one another. To minimize communication costs, workers can compute multiple updates on their copy of the model before communicating their changes, which gives rise to the "stale gradient" problem where workers are operating on an out-of-date copy of the model. The standard and straightforward approach to dealing with stale gradients it to simply apply updates in the "wrong" order, which can be effective in practice [10, 34]. An orthogonal approach to reducing communication costs explored by Aji and Heafield [2] is to have workers only update a small subset of the model's parameters. Aji and Heafield [2] simply have workers communicate only those updates corresponding to the gradients with top-k largest magnitude at each step. This bears a similar motivation to FISH Mask, but results

in a "mask" that changes at every iteration and therefore requires workers communicate after each update. In contrast, pre-computing a FISH Mask allows workers to perform multiple iterations before communicating their updates, thereby further reducing communication costs.

An extreme variant of distributed training is federated learning [20, 27]. In federated learning, asynchronous workers perform many updates on private data before communicating the changes back to a centralized model. The training involves one server and multiple clients, and the server model's gradient is the combination of which of client models. As with any form of asynchronous training, communication costs and stale gradients are significant issues. McMahan et al. [27] demonstrated that averaging the updates computed by individual workers is an effective approach to dealing with stale gradients and Konečnỳ et al. [21] investigated techniques for significantly reducing communication costs. Our method is complementary to the techniques for reducing communication proposed by Konečnỳ et al. [21].

4 Experiments

We evaluate the efficacy of the FISH Mask in three settings: parameter-efficient transfer learning, distributed training, and training with efficient checkpointing. For parameter-efficient transfer learning, we demonstrate that our approach matches the performance of standard gradient-based training on the GLUE benchmark [35] while updating only 0.5% of the model's parameters parameters per task. For distributed training, we evaluate FISH Mask training for both transfer learning on GLUE and training from scratch on CIFAR-10 [22]. In both settings, we find that we can dramatically reduce communication without sacrificing significant performance, though from-scratch training on CIFAR requires a higher sparsity level. Finally, we demonstrate a novel application of training with sparse updates: Minimizing the size of checkpoints over training. We show that using a FISH Mask while training on CIFAR-10 with a sparsity level of 10% can shrink checkpoint size on disk by a factor of 10 while sacrificing minimal performance. Throughout our experiments, we report results with varying sparsity to get a sense of the savings induced by the FISH Mask. For ease of comparison, we report sparsity in terms of the total percentage of parameters that are updated. This percentage can be converted to a value of k used for the top-k operation when constructing the mask simply by multiplying it against the total number of parameters in the model. We also include ablation studies to measure the impact of the number of samples used to estimate the Fisher Information as well as the choice of true or empirical Fisher. All experiments for GLUE are run with the $\text{BERT}_{\text{LARGE}}$ variant of BERT, which contains 16 attention heads, 24 layers, and 330 million parameters in total [11], and most experiments are run on a RTX 3090 GPU. For experiments on CIFAR-10, we use a ResNet-18 [16] with various optimizations for fast convergence.[2]

4.1 Parameter-Efficient Transfer Learning

In parameter-efficient transfer learning, the goal is to fine-tune a pre-trained model while only updating as few parameters as possible. We focus on fine-tuning $\text{BERT}_{\text{LARGE}}$ on the GLUE benchmark [35], which is the primary setting used for evaluation in prior work. For all experiments, we fine-tune for 7 epochs and perform a hyper-parameter search across learning rate $\in \{1 \times 10^{-4}, 5 \times 10^{-5}, 1 \times 10^{-5}\}$ and batch size $\in \{8, 16\}$ for each GLUE task. We find the learning rate of 5×10^{-5} and batch size of 16 to be effective for most tasks, with the exception of batch size $= 8$ used for RTE. Additional hyper-parameters, such as choice of optimizer, sequence length, and others, follow from the default configuration for $\text{BERT}_{\text{LARGE}}$ presented in the Hugging Face library [36]. Test set results are reported by submitting to the GLUE benchmark test server results are reported using the final model checkpoint following a hyper-parameter search on validation results, unless otherwise noted.

Baselines We compare GLUE task performance of the FISH Mask to several other baselines and methods focused on parameter-efficient transfer learning. In **Dense Fine-tuning**, we fine-tune all parameters of a pre-trained model, as is typical in standard transfer learning. In the **Random Mask** baseline, we randomly select and fix k parameters to update at the start of training. To compare to prior work, we reproduce **Bit-Fit** [6], in which only the bias parameters of the BERT model are updated across training. Our reproduction follows the original paper and performs a hyper-parameter search with learning rates in the $[1 \times 10^{-3}, 1 \times 10^{-4}]$ range. We also reproduce results from **Diff**

[2] Our implementation is based on `https://github.com/davidcpage/cifar10-fast`

Method	Updated Params/Task	QNLI	SST-2	MNLI_m	MNLI_mm	CoLA	MRPC	STS-B	RTE	QQP	AVG
Dense Fine-tuning	100%	93.4	94.9	87.0	86.1	61.0	86.6	86.5	70.9	80.5	82.5
Random Mask	0.50%	89.8	93.4	83.7	84.0	43.2	77.8	87.7	61.3	77.2	76.8
Bit-Fit [6]	0.08%	90.4	94.5	85.0	84.8	60.3	86.3	85.0	69.6	78.5	81.2
Diff Pruning [14]	0.50%	91.9	93.8	86.0	85.5	61.0	86.2	85.6	67.5	80.1	81.5
Adapters (8-256) [17]	3.60%	92.4	94.3	85.4	85.0	59.2	86.5	86.7	71.6	80.5	82.4
FISH	0.50%	93.1	94.7	86.5	85.9	61.6	87.1	86.5	71.2	80.2	82.6

Table 1: GLUE test server evaluation results with BERT_LARGE. MRPC and QQP are reported as an average of F1-score and accuracy, and STS-B is reported as an average of Pearson and Spearman correlation. Accuracy is reported for all other tasks. All results are reproduced locally, with the exception of Adapters [17] taken from the official GLUE benchmark server. Training with the FISH surpasses (82.6) other methods and equals dense fine-tuning performance (82.5) whilst updating only 0.5% of model parameters.

Pruning [14], which updates the sparse mask over the course of training. Our reproduction of Diff Pruning at 0.5% mask sparsity follows the paper's code-base[3] and training settings, and reports the GLUE test set results using the best validation checkpoint. Finally, we compare to **Adapters** [17], an approach that trains task-specific "adapter" layers that are inserted into a pre-trained model for fine-tuning. We report the results on Adapters with 8-256 bottleneck layers from the GLUE test server, which adds 3.6% additional parameters per task. Due to restrictions on the number of permissible submissions to the GLUE test server, we are only able to report results with a sparsity of 0.5% for those methods where we can control the sparsity level. We therefore include additional validation set results for varying sparsity levels when using a FISH Mask.

Results Our results on parameter-efficient transfer learning with the FISH Mask can be seen in table 1. FISH Mask training results in effectively the same performance (82.6%) as standard "dense" fine-tuning (82.5%), despite updating just 0.5% of of BERT_LARGE parameters. The results of the Random Mask baseline achieves a significantly lower average GLUE score, which demonstrates the value of using the Fisher Information in selecting parameters to update. Finally, FISH Mask training is competitive with all other parameter-efficient transfer learning approaches; the next-best score is achieved by Adapters [17], which uses 3.6% additional parameters per task vs. FISH Mask's 0.5%.

Figure 2 shows the change in performance on the GLUE validation set as we change the sparsity level of the FISH Mask. While we effectively matched the test set performance of fine-tuning all parameters when using a sparsity level of 0.5%, our validation set score was slightly worse (84.3% vs. 85%). Increasing the sparsity level to 2.1% effectively closes this gap. These results demonstrate that the FISH Mask can be a useful tool in mitigating the storage costs of saving many fine-tuned models, requiring only the updated parameters and their respective indices to be saved.

4.2 Distributed Training

Now, we turn to using FISH Mask to reducing communication costs in distributed training settings. We consider the setting where distributed workers compute many subsequent updates to a local copy of the model before transmitting their changes back to a central server. In our experiments, we assume all workers sample data i.i.d. from the same dataset and that all workers compute the same number of updates between each communication step, though our results should carry over to settings where workers use different datasets and varying amounts of updates (e.g. in Federated Learning [27]).

Let θ denote the parameter vector stored on the central server and δ_i represent the update computed by worker i. After each time updates are communicated, the server must transmit the updated θ back to all workers. When standard gradient-based training is used, all parameters are updated, so $|\delta_i| = |\theta|$. If there are M workers, the communication costs for normal training are then $M|\delta_i| = M|\theta|$ for worker-to-server communication and $M|\theta|$ for server-worker communication, for a total communication cost of $2M|\theta|$.

Using a sparse mask will effectively reduce the size of each δ_i to k. Without loss of generality, we assume that $|\delta_i|$ is the same across all workers. Furthermore, assuming updates are aggregated on the central server by summing them together, the server can either communicate the full updated parameter

[3] `https://github.com/dguo98/DiffPruning`

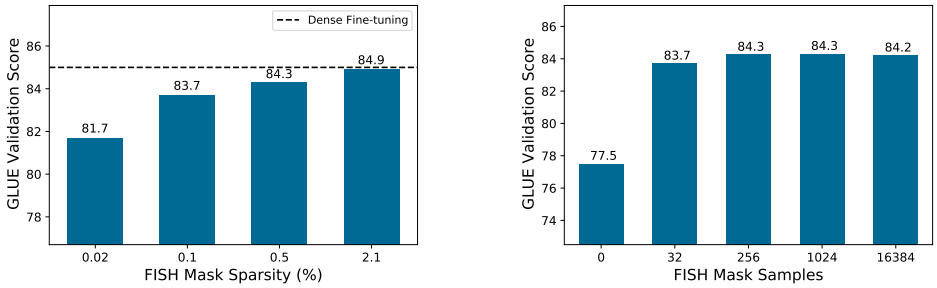

Figure 2: (Left) GLUE validation performance of the FISH Mask at varying levels of sparsity. Compared to the densely fine-tuned baseline score of 85%, training with the FISH Mask is competitive at 0.5% sparsity. (Right) GLUE validation performance at varying levels of dataset samples used to compute the FISH Mask. Few samples are needed to effectively compute the FISH Mask and obtain good performance.

vector back to all workers, or the server can communicate all *other* workers' updates to a given worker. These two options amount to either a communication cost of $M|\theta|$ or $M(M-1)\|\delta_i| \approx M^2|\delta_i|$. In general, we expect M to be small and $|\theta|$ to be large, so we typically have $M^2|\delta_i| < M|\theta|$. Therefore, we can achieve significant communication savings by using sparse updates. For simplicity, we set $M = 2$ in all of our experiments; this makes the savings in communication equal to the sparsity level.

To apply FISH in distributed training, $\hat{F}_\theta$ is first computed on the central server. Then, the server segments $\hat{F}_\theta$ into M nonoverlapping masks $\hat{F}_\theta^{(1)}, \ldots, \hat{F}_\theta^{(M)}$ by letting $\hat{F}_\theta^{(i)}[n] = \hat{F}_\theta[Mn + i]$. This ensures that the workers update completely complementary sets of parameters and that the total amount of Fisher Information "contained" in each mask is roughly balanced across workers.

Baselines We experiment with distributed fine-tuning of $\text{BERT}_{\text{LARGE}}$ on GLUE and distributed training (from scratch) of a ResNet18 on CIFAR-10. We compare FISH Mask training to two baselines: First, in **standard training**, we tune all the parameters in a single machine (i.e. standard, non-distributed training). This likely forms an upper bound on performance for distributed training techniques. Second, in **densely-updated distributed training**, workers use standard gradient descent to compute updates over all parameters, and all of the worker's updates are added together on the centralized server. For a fair comparison, we keep the overall training batches the same for all methods, so the training iterations of a worker are half of that of the single-machine baseline. On GLUE, we follow the transfer setting to fine-tune the model for 7 epochs (3.5 for each worker). On CIFAR-10, the model is trained from scratch for 100 epochs (50 for each worker). All the models and hyper-parameters follow those mentioned in section 4.1, and we fix the sparsity to be 0.5% in GLUE experiments. We searched for the best initial learning rate in {0.4, 0.2, 0.08, 0.04, 0.02} and found that a learning rate of 0.4 worked best in all settings. For both sets of experiments, we report results using a varying number of updates computed by each worker between each worker-server communication step (referred to as "worker updates"). We also include results at varying sparsity levels on CIFAR-10.

Results Table 2 shows our results for distributed fine-tuning of $\text{BERT}_{\text{LARGE}}$ on GLUE. Across varying numbers of worker updates, FISH Mask training at a sparsity level of 0.5% achieves comparable performance to both standard training and a densely-updated distributed training baseline. At high delays between worker communication, we find that performance begins to degrade both for densely-updated and FISH Mask-based training. This is likely due to the "stale gradient" problem discussed in section 3.

Results from training a ResNet18 on CIFAR-10 are shown in table 3. We find that FISH Mask training results in a significant degradation in performance compared to standard training and densely-updated

Table 2: Average GLUE score on the validation set attained when fine-tuning $\text{BERT}_{\text{LARGE}}$ with distributed training. Standard (non-distributed) fine-tuning achieves a score of 85.0%. FISH Mask training uses a sparsity level of 0.5%.

	Worker updates		
	10	30	100
Densely-updated	85.0%	85.0%	84.5%
FISH Mask	84.7%	84.1%	83.9%

Table 3: CIFAR-10 validation set accuracy achieved by a ResNet18 through distributed training. Standard (non-distributed) training achieves an accuracy of 95.2%. Percentages in FISH Mask rows denote sparsity levels.

	Worker updates		
	10	30	100
Densely-updated	93.0%	94.7%	88.9%
FISH Mask, 0.5%	84.8%	83.2%	65.4%
FISH Mask, 2%	90.4%	90.5%	59.0%
FISH Mask, 10%	93.3%	93.3%	83.7%

distributed training for low sparsity levels. However, using a FISH Mask with a sparsity level of 10% attains comparable performance to using dense updates when using 10 or 30 worker updates. For 100 worker updates, performance of any form of distributed training degrades significantly. As a whole, our results show significant promise for dramatically reducing computational costs in distributed training settings.

4.3 Efficient Checkpointing

Over the course of training a machine learning model, it is common to save intermediate *checkpoint* files that store the model's parameter values. These checkpoints can be useful for restarting training from a given iteration rather than starting from scratch in the event that the training job crashes or is otherwise stopped. They are also commonly used for post-hoc analysis, for example for evaluating a model's performance on new datasets or metrics over the course of training. Since checkpoints store a copy of all of the model's parameters, they can take up a significant amount of space on disk. Furthermore, depending on the checkpointing frequency, hundreds of checkpoints are often written to disk over the course of a training run. The development cycle of a machine learning model can result in hundreds of different model variants being trained. Combining these factors with the on-disk space needed to store the parameters of modern models (around 1 gigabyte for $\text{BERT}_{\text{LARGE}}$) results in potentially massive storage costs.

Training with sparse updates can significantly reduce these storage costs. Specifically, if only a small subset of the parameters are updated between checkpoint saves, then the checkpoint only needs to store the updated parameter values. Note that this allows the "mask" to change over the course of training, and is therefore a relaxation of the setting in section 4.1, where the requirement is that the same subset of parameters is updated over the entire fine-tuning run. It follows from our results in section 4.1 that FISH mask training could be readily applied to reducing checkpoint size in parameter-efficient transfer learning. However, we found that the strict requirement of fixing the mask over the entire training run resulted in a significant degradation in performance. This is in line with past work demonstrating the difficulty of identifying fixed sparse subnetworks to train before training beings [13]. We therefore allow the mask to change every time a checkpoint is written, which does not increase storage requirements over using the same fixed mask from the start of training.

We focus on training a ResNet18 on CIFAR-10 as a testbed for determining wither using a FISH mask could result in smaller checkpoints. Overall, we use the same experimental setup as in section 4.2. We compare both allowing the mask to change at each checkpoint save and forcing the mask to stay fixed. For simplicity, we assume that the checkpoint saving interval corresponds to a single epoch, which is a common choice in practice. We performed a new search over learning rates in {0.4, 0.2, 0.08, 0.04, 0.02} and found that the learning rate needs to be smaller for the model to converge in the fixed mask setting. We compare to a baseline of standard training, which serves as the upper bound on the performance.

Results In table 4, we show the results of FISH Mask training when keeping the mask fixed over the course of training or updating it at each epoch. We find that updating the FISH Mask every epoch can nearly match the performance of normal training ((94.4% vs. 95.2% accuracy) at a sparsity level of 10%, which would reduce storage requirements by a factor of 10. At lower sparsity levels, we

see some degradation in performance. As mentioned earlier, we also found that using a fixed mask significantly degraded performance, though only by a few percent at a sparsity level of 10%. This suggests that the FISH Mask could also be useful for identifying sparse subnetworks to train before training begins, as conjectured by the Lottery Ticket Hypothesis [13]. We leave exploration of this possibility for future work.

4.4 Ablations

Having established the effectiveness of training with a FISH mask, we now ablate a few design choices to help demonstrate the robustness of our approach. We perform all ablation experiments in the parameter-efficient transfer learning setup described in section 4.1.

Table 4: CIFAR-10 validation set accuracy when using the FISH Mask to reduce checkpoint sizes. "Epoch" refers to allowing the mask to change each epoch. Standard training achieves an accuracy of 95.2%.

	Sparsity level		
	0.5%	2%	10%
Fixed	60.5%	78.6%	91.3%
Epoch	86.4%	92.5%	94.4%

4.4.1 True Fisher vs. Empirical Fisher

In section 2, we note that past work has approximated the Fisher Information matrix (eq. (3)) using either the expectation over $y \sim p_\theta(y|x)$ ("true Fisher") or ground-truth labels ("empirical Fisher"). While past work has shown that using the empirical Fisher can be detrimental in optimization settings [26, 23], we mainly use the Fisher Information as a signal of parameter importance. The empirical Fisher also has the benefit that it avoids marginalizing over or sampling from $p_\theta(y|x)$ and only requires computing the gradient for a single value of y, making it more computationally efficient. When comparing the performance of using the true or empirical Fisher to compute a 0.5%-sparse FISH mask for parameter-efficient transfer learning, we observe that both methods achieve near-identical performance with an average validation-set GLUE score of 82.5 in both cases. Since computing the empirical Fisher can be more computationally efficient, we used the empirical Fisher for all experiments.

4.4.2 Sample Ablation

We also ablate the number of samples, N, used to compute the FISH Mask to study if more samples are beneficial. The results for parameter-efficient transfer learning on GLUE are shown in fig. 2, right. At a sample count of 0, the FISH Mask is equivalent to the Random Mask baseline presented in section 4.1 in which parameters to update are selected at random instead of informed by the Fisher Information. We observe that FISH Mask performance on the GLUE validation set is surprisingly stable across many values of samples, with just 32 samples needed to achieve the highest-possible performance. These suggest that using the approximate Fisher Information is a data-efficient approach of computing parameter importance, and we therefore ran all experiments with sample count $N = 1024$.

5 Conclusion

In this work, we proposed FISH Mask training as a novel method for pre-computing fixed sparse masks of a model's parameters to update over many subsequent iterations. The FISH Mask estimates the importance of model parameters by first approximating the Fisher Information of all of a model's parameters and then selecting the k parameters with the largest Fisher Information to include in the mask. We demonstrate the usefulness of FISH Mask training in several settings, including parameter-efficient transfer learning, distributed training, and reducing storage requirements of model checkpoints. In future work, we hope to explore the performance of FISH Mask training at lower sparsity levels and other measures of parameter importance other than the Fisher Information. We also hope to further demonstrate the efficacy of FISH Mask in real-world settings where the benefits of sparse parameter updating are even more pronounced, such as in federated learning. The integration of FISH Masks across tasks and sharing amongst practitioners could also be a useful line of inquiry, as recent frameworks such as AdapterHub [31] have enabled for Adapter modules [17].

References

[1] Alessandro Achille, Giovanni Paolini, and Stefano Soatto. Where is the information in a deep neural network? *arXiv preprint arXiv:1905.12213*, 2019.

[2] Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021*, 2017.

[3] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. *CoRR*, abs/1811.04918, 2018. URL `http://arxiv.org/abs/1811.04918`.

[4] SI Amari. Neural learning in structured parameter spaces-natural riemannian gradient. *Advances in neural information processing systems*, pages 127–133, 1997.

[5] Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. Simple, scalable adaptation for neural machine translation. *arXiv preprint arXiv:1909.08478*, 2019.

[6] Elad Ben-Zaken, Shauli Ravfogel, and Yoav Goldberg. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language models. 2021.

[7] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL `https://arxiv.org/abs/2005.14165`.

[8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020. URL `https://arxiv.org/abs/2002.05709`.

[9] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E. Hinton. Big self-supervised models are strong semi-supervised learners. *CoRR*, abs/2006.10029, 2020. URL `https://arxiv.org/abs/2006.10029`.

[10] J. Dean, G. Corrado, Rajat Monga, Kai Chen, M. Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng. Large scale distributed deep networks. In *NIPS*, 2012.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://www.aclweb.org/anthology/N19-1423`.

[12] Ronald A Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 222(594-604):309–368, 1922.

[13] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[14] Demi Guo, Alexander M. Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning. *CoRR*, abs/2012.07463, 2020. URL `https://arxiv.org/abs/2012.07463`.

[15] Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *International Conference on Machine Learning*, 2016.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[17] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. *CoRR*, abs/1902.00751, 2019. URL `http://arxiv.org/abs/1902.00751`.

[18] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *CoRR*, abs/1801.06146, 2018. URL `http://arxiv.org/abs/1801.06146`.

[19] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13), 2017.

[20] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[21] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[22] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[23] Frederik Kunstner, Lukas Balles, and Philipp Hennig. Limitations of the empirical fisher approximation for natural gradient descent. *arXiv preprint arXiv:1905.12558*, 2019.

[24] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[25] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

[26] James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[27] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2017.

[28] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[29] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

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

[31] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulic, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. *CoRR*, abs/2007.07779, 2020. URL `https://arxiv.org/abs/2007.07779`.

[32] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21, 2020.

[33] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. *arXiv preprint arXiv:1705.08045*, 2017.

[34] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL `https://proceedings.neurips.cc/paper/2011/file/218a0aefd1d1a4be65601cc6ddc1520e-Paper.pdf`.

[35] 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. *CoRR*, abs/1804.07461, 2018. URL `http://arxiv.org/abs/1804.07461`.

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

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Yes, will be provided in supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Experiments included are only run with one random seed due to computational requirements.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Yes, included in the beginning of the experiments section.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Yes, we note where code is taken from in footnotes.
   (b) Did you mention the license of the assets? [Yes] These will be mentioned in the supplementary codebase.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Our data uses standard NLP and vision benchmarks.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]