Can Limiting the Number of Layers Updated Help Accelerate Data-Parallel Fine-Tuning?

Abhash Kumar Singh, Chetna Sureka, Tarun Bedi

1 Introduction

With the growing complexity of Machine Learning models, large data sets and high training times, it becomes imperative to use distributed model training. This can be done with two different approaches - data parallel and model parallel. While the former splits the data set and concurrently trains the same model present on different nodes, the latter method focuses on splitting the model over different nodes and concurrently training all parts of the model using the entire data set. In this project, we focus on training models over very large data sets for which we will focus on the data parallel setting.

The data parallel setting brings with it large network overheads in communicating computed gradients after every iteration to the parameter server and aggregating these values and updating the worker nodes again. For models with large number of layers, not all the weights in the inner layers are updated due to vanishing gradients [4]. Thus, sending gradients for these layers to the parameters server, updating weights for those layers in the parameter server and sending the updated weights back to the worker nodes may introduce overheads both in terms of network load and latency. These can be enormous depending on the size of the model and the number of worker nodes used.

The use of deep learning models for specialized applications has accelerated by the rapid adoption of fine-tuning and transfer learning. In transfer learning, we use a pre-trained model and train only the last few layers for our specialized application, while in fine-tuning we train the whole pre-trained model on the new dataset. As expected, the use of fine-tuning leads to a significantly faster convergence and computation time, primarily due to the transfer of common parts of the learning task from the pre-trained model. However, on the use of large models and datasets, even fine-tuning requires significant amount of time. For example, fine tuning BERT even on the relatively small IMDB dataset [7], takes around 3 hours on a single GPU.

This strongly motivates us to experiment on ways to reduce training time by avoiding parts of the training that do not significantly improve accuracy, and to experiment on how we can avoid backward and forward passes in some parts of our data parallel training process. Essentially, if we have an n-layered model and we freeze upto the k^{th} layer, then we would back-propagate the gradients to update the weights only from the n^{th} layer uptil the k^{th} layer, and the remaining weights would be left unchanged. Moreover, by the use of caching, we can avoid recomputing the forward passes from layers 1 to k in the next iteration.

We thus propose the following experiments to fine-tune data parallel model training:

1. **Training time benefit**: Analyze the training time benefit in data parallel systems with freezing, while varying the number of layers frozen. [1]

2. **Accuracy benefit**: Analyze the validation accuracy while varying the number of layers frozen.

3. **Optimal freezing layer**: Experiment thresholding/gradient-norm test/other heuristics to find the optimal number of layers to freeze.

4. **Weighted votes to improve accuracy**: Experiment on weighted votes from each worker node in the data parallel setup, to find the optimal freezing layer for better performance overall.

5. **Caching benefit**: Experiment benefit of caching with freezing, to avoid re-computation of forward passes till the frozen layers.

6. **Communication overheads**: Benchmark the communication overhead of the data parallel setup with freezing and caching weights of the optimal layers in the model, and compare it to the single node setting.

2 Background

In this section, we provide background about the dataset and model used in the project. We chose to study freezing layers in deep neural network in the context of vision tasks.

2.1 Resnet-18

Residual neural networks (ResNet) [3] are artificial neural networks that utilize skip connections, or shortcuts to jump over some layers. These models are implemented with double or triple layer skips that contain nonlinearities (ReLU) and batch normalization in between. They were designed to avoid the problem of vanishing gradients, by reusing activations from a previous layer until the adjacent layer learns its weights. This speeds up learning by reducing the impact of vanishing gradients, as there are fewer layers to propagate through. The network then gradually restores the skipped layers as it learns the feature space. [Residual neural network, Wikipedia]

We have used Resnet-18 as our model and its architecture is described in Figure 1. We used the `torch` package in PyTorch library for our experiments.

Layer Name	**Output Size**	**ResNet-18**
`conv1`	$112 \times 112 \times 64$	7×7, 64, stride 2
`conv2_x`	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
`conv3_x`	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
`conv4_x`	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
`conv5_x`	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
`average pool`	$1 \times 1 \times 512$	7×7 average pool
`fully connected`	1000	512×1000 fully connections
`softmax`	1000	

Figure 1: ResNet-18 Architecture. [9]

2.2 CIFAR-10

The CIFAR-10 dataset [6] (Canadian Institute For Advanced Research) is a widely used datasets for machine learning research. It has 60,000 32x32 color images in 10 different classes.The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

2.3 Model Fine-Tuning

Fine-tuning of large pre-trained models help in transferring them to specialized tasks. In fine-tuning, a pre-trained model is trained on a new task. This has many advantages like training with scarce data, transferring common features among related tasks and faster convergence.

However, various deep neural networks take a lot of time to train and even fine-tuning on them is expensive. To reduce the computational cost, we can only fine-tune a subset of the layers. Avoiding gradient computation for certain layers (freezing) can significantly reduce training time. [1]

For the freezing schemes followed in our experiments, please refer to section 3.

3 Design

3.1 Analyzing the Benefits of Freezing

To better understand the speedup benefits of freezing different layers while training a model and seeing the training time vs. accuracy trade-offs, we start by training the ResNet-18 model on the CIFAR-10 data set. In Figure 1, we see that multiple layers are clubbed together in the ResNet-18 architecture keeping room only to freeze 7 different blocks. By varying the number of frozen layers from 0 to 6, we run 50 epochs freezing different layers each time and compare the training time benefit with the accuracy achieved for each case.

3.2 Adaptive Freezing for Fine Tuning

From our early results from analyzing the benefits of freezing, we realize that statically determining which layers to freeze is difficult and can lead to reduced accuracy. Hence, we present an online test to determine which layer should be frozen at a given time. We call this the Gradient Norm Test.

Our intuition in designing this test is that the rate of change of the gradient values for a layer can be used to determine how fast the model weights are being updated for a particular layer. Consider that we accumulate gradients for each layer in the model (Δ) and perform our test at fixed intervals (T). Then we define the gradient norm change for layer l, η_l, as:

$$\eta_l = \left| \frac{\|\Delta_{T-1,l}\| - \|\Delta_{T,l}\|}{\|\Delta_{T-1,l}\|} \right|$$

We next rank the layers in the order of η_l to determine the layer that is changing slowest. Given our earlier observation about how layers change in order, we can designate a layer to be frozen if all layers preceding it are frozen and it is the slowest changing layer. Algorithm 1 describes the above procedure.

Algorithm 1 Adaptive Freezing Module

```
Input: List of layers that are not frozen activeLayers
Input: accumulated gradients for current interval $\eta_{T,l}$
    and previous interval $\eta_{T-1,l}$
1: for $layer_l$ in $activeLayers$ do
2:     if $\eta_l$ is min($\eta$) then
3:         freeze $layer_l$
4:         break
5:     else
6:         continue
7:     end if
8: end for
```

Putting the above description and algorithm together, our adaptive freezing module essentially makes a decision on the set of layers to freeze at different intervals of the fine-tuning procedure. By using a gradient norm-based rate of change metric for each layer, we hope to incrementally freeze layers optimally, by freezing layers with a smaller rate of change first.

4 Experimental Setup

We make use of 4 C8220 nodes on CloudLab[11] and setup Python's Pytorch library in CPU only mode on all 4 of these. These nodes contain Two Intel E5-2660 v2 10-core CPUs at 2.20 GHz and 256GB ECC Memory. Nodes 1, 2 and 3 are worker nodes. Node 0 serves as both the master node and the worker node.

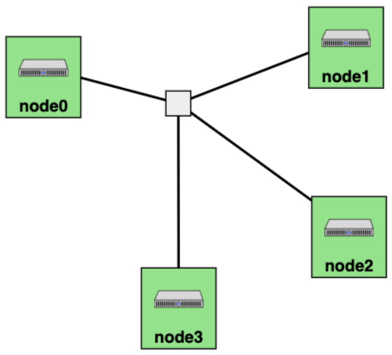

Figure 2: Node Topology Used

4.1 Distributed Training

For the distributed training, we used the `torch.distributed` package from PyTorch library. We used distributed data parallel training(DDP) for the single-program multiple-data training paradigm with `gloo` as the communication backend.

4.2 Training parameters

In all our experiments, we have used Stochastic Gradient Descent(SGD) as our optimizer with learning rate 0.1, momentum 0.9 and weight decay 0.0001. We used Cross Entropy Loss as the loss function. All the training set images were normalized before being applied with transformations like random crop of size 32 and random horizontal flip.

5 Evaluation

5.1 Analyzing the Benefits of Freezing

To get a better sense of the training time speedups due to freezing and its trade-off with accuracy, we begin by training the ResNet-18 model on the CIFAR-10 dataset, and benchmark the training times (i.e. forward pass, backward pass and optimizer step) and accuracy by freezing varying number of layers each time. By varying the number of frozen layers from 0 to 6, we run 50 epochs freezing different layers each time.

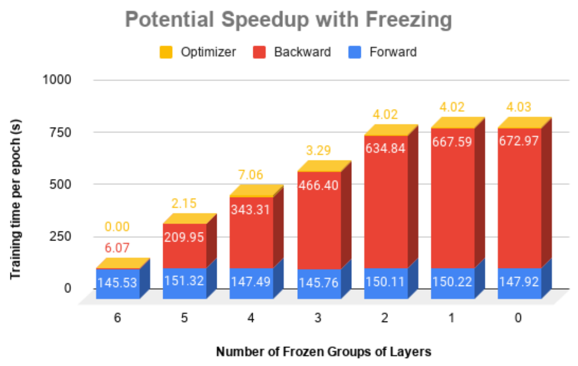

Figure 3: Potential time benefit of freezing, by varying the number of groups frozen

As expected, from Figure 3, we observe a larger time saving when more number of groups are frozen. This is because a larger part of the computationally-heavy backward pass step is avoided, leading to larger time savings. Further, from Figure 4, we observe a larger dip in accuracy when the number of frozen groups in higher.

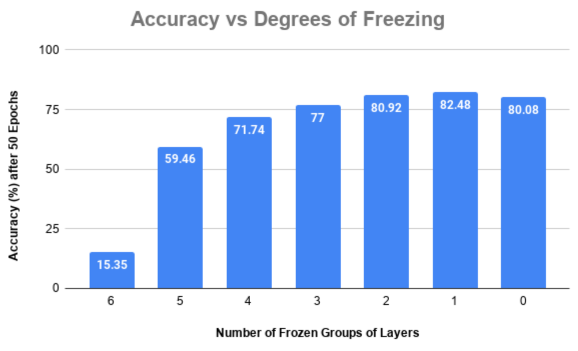

Figure 4: Analyzing the impact of freezing on accuracy, by varying the number of groups frozen

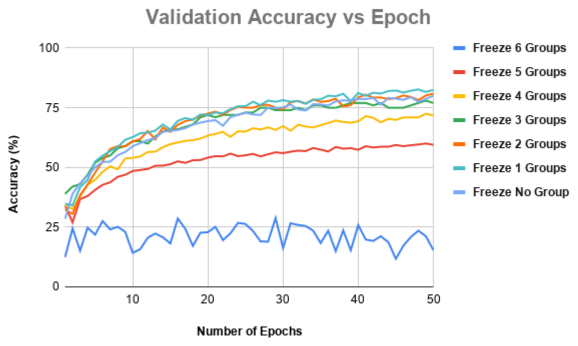

Figure 5: Analyzing the epoch-wise impact of freezing on accuracy, by varying the number of groups frozen

We believe that a larger part of the backward pass being avoided leads to poorer model performance. Finally, from these graphs, we believe that choosing 4 as the number of frozen groups is a good trade-off in this case, where we observe a $\approx 65\%$ saving in total training time with a negligible $\approx 8\%$ dip in accuracy. Also, we observe a similar epoch-wise accuracy trend and convergence pattern while choosing 4 as the number of frozen groups, as can be seen in Figure 5.

5.2 Adaptive Freezing for Fine Tuning

By beginning with a pre-trained ResNet-18 model, we first analyze the speedups observed by using the adaptive freezing approach. In Table 1, we report the time spend in the different phases of training, and calculate our speedup relative to full fine-tuning. As expected, we observe the highest speedup in the backward pass step of the training process, since backward passes are computationally heavy, and multiple backward passes were avoided because of freezing. We noted a $\approx 22.6\%$ speedup in average backward pass duration. Overall, we observed a speedup of $\approx 20.8\%$ in total training time.

	Adaptive Fine Tuning	**Full Fine Tuning**	***Speedup***
Avg. Forward Pass	15.75 s	15.96 s	1.3%
Avg. Backward Pass	197.49 s	242.04 s	22.6%
Avg. Optimizer Step	3.69 s	3.99 s	8.1%
Total Training Time	180.77 min	218.33 min	20.8%

Table 1: Observed speedups of the forward, backward and optimizer steps, as well as the total training time of adaptive fine tuning compared with full fine tuning with no freezing

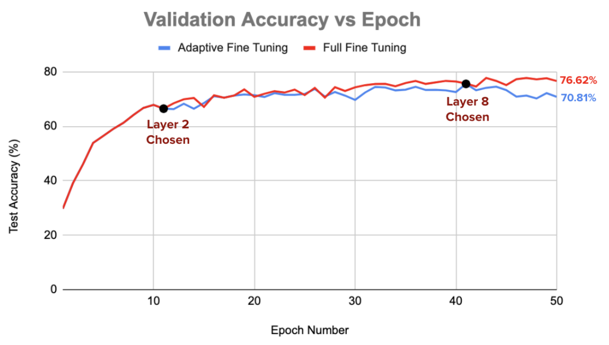

Figure 6: Analyzing the epoch-wise impact of freezing on accuracy, by comparing our adaptive fine tuning algorithm with full fine tuning

The objective of our experiments was to improve training time by using data parallel training in context of vision tasks without losing out much on the test accuracy. In Figure 6, we compare the epoch-wise test accuracy achieved during our adaptive fine tuning approach and full fine tuning. We observe that our adaptive freezing module chose its layers to freeze as 2 and 8 at epochs 11 and 41 respectively. As expected, we observed larger time savings after layer 8 was chosen, since this helped avoid a larger part of the computationally-heavy backward pass operation.

Finally, we obtained an accuracy of $\approx 70.81\%$ with our adaptive freezing approach as compared to full fine tuning approach, which had an accuracy of $\approx 76.62\%$. This is within $\approx 8\%$ of the full fine tuning approach and is a good trade-off in terms of $\approx 21\%$ less training time required.

6 Related Work

While doing fine-tuning, a whole pre-trained model is trained on new task. This can have several advantages over training models from scratch, like, making use of common features among related tasks and getting faster convergence reducing the computation time. In some recent works, like [10] and [5], it was found that language models have better performance when fine tuned. However, large models like BERT [2] require significant amount of time even when fine tuned. Fine tuning BERT on the relatively small IMDB dataset [8], takes up to 3 hours on a single P100 GPU. This motivates a need to speedup the fine tuning process.

In order to reduce the computation cost, it is intuitive to fine-tune only a subset of layers. We expect that avoiding gradient computation for certain layers via freezing can help significantly reduce training time, without much effects on accuracy - because of the vanishing gradients involved in the frozen layers. [1] follows a similar approach of freezing early layers of a model which converge rapidly and caching the output of forward pass up to the layer that has been frozen. We plan to expand this approach from a single-node to the data parallel setup, and experimentally study the effect of freezing on computation times and accuracy. Most of the prior work in this domain has also been done only on NLP tasks. Through our work, we hoped to extend the above mentioned strategy on Computer Vision tasks with a CNN based model. We used the CIFAR-10 dataset with a deep learning model like ResNet-18.

7 Future work

While we were able to analyze the benefits of freezing and come up with an adaptive freezing algorithm, we outline our plan of future work below:

1. **Using GPUs for training**: Due to resource constraints on our CloudLab account, we were only able to train on CPUs. We plan to run the experiments on GPUs which are usually suited for deep learning workloads and report our results.

2. **Data distribution across nodes and weighted votes**: In order to study the effects of data distribution across worker nodes, we plan to experiment on weighted votes from each worker node in the data parallel setup. This is done so as to better find the optimal freezing layer for better performance overall, by accounting for differences in data distribution across nodes.

3. **Caching benefit**: We also plan to experiment the benefit of caching with freezing, to avoid recomputation of forward passes till the frozen layers.

4. **Communication overheads**: Finally, we plan to benchmark the communication overhead of the data parallel setup with freezing and caching weights of the optimal layers in the model, and compare it to the single node setting.

References

[1] Anonymous. Autofreeze: Automatically freezing model blocks to accelerate fine-tuning. 2020.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[4] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. 1998.

[5] J. Howard and S. Ruder. Universal language model fine-tuning for text classification, 2018.

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

[7] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[8] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[9] P. Napoletano, F. Piccoli, and R. Schettini. Anomaly detection in nanofibrous materials by cnn-based self-similarity. *Sensors (Basel, Switzerland)*, 18, 01 2018.

[10] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations, 2018.

[11] R. Ricci, E. Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *USENIX*, 39(6), Dec. 2014.