SparsifyGradients - Reducing the Network Load by Dropping Gradient Values

Max Lam, Edward Look, Jesslynn Whittell

1 Abstract

We present SparsifyGradients, a simple algorithm to drop 90% of gradient values during distributed machine learning training. We show that the increase in network performance by dropping these values outweighs the loss in accuracy, leading to an overall speedup in training. We show up to 2x speedup over non-sparsified training on 14 c3.xlarge EC2 machines for training a cuda-convnet neural network for the cifar10 image classification task.

2 Introduction

Machine learning models, in particular deep neural networks, have achieved state of the art results in tasks such as object detection, speech recognition, image classification, game playing and more. Due to increasing amounts of data and the computationally intensive nature of training deep neural networks, research has been conducted to distribute the training of these models across multiple machines. Various algorithms to do this have been proposed such as bulk synchronous parallel training (BSP), stale synchronous parallel training (SSP) and asynchronous training. A main issue inherent to all these methods is data communication: gradients need to be communicated between machines every iteration of the algorithm. Even with a few machines, distributed training can be bottlenecked by the network bandwidth of the cluster.

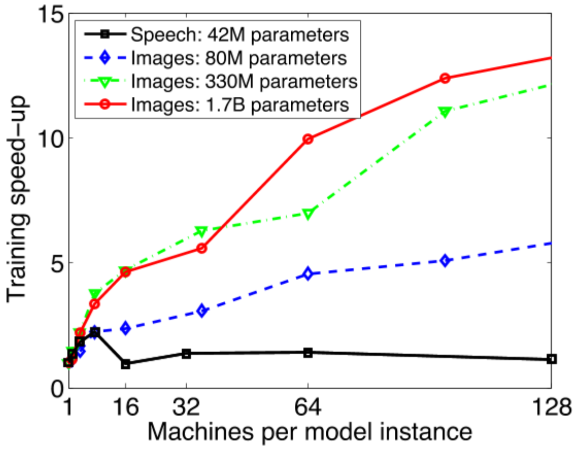


Figure 1: Speedup versus number of machines from "Large Scale Distributed Deep Networks" [Dean et al., NIPS 2012]. Scaling is limited by the network bandwidth with larger clusters.

In this project, we focus on tackling this communication issue. We focus on the bulk synchronous parallel training algorithm in a cluster setting. The following diagram describes the three phases of the bulk synchronous parallel algorithm.

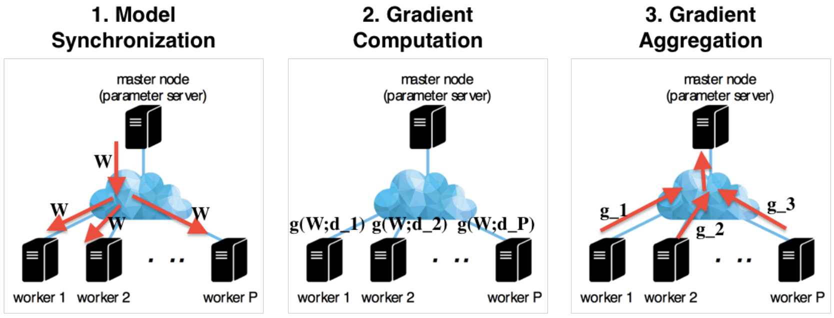


Figure 2: The bulk synchronous parallel training (BSP) algorithm. In phase 1 the model is distributed by the master to each worker in the cluster. In phase 2 each worker computes gradients given a shard of the overall data set with the model received from the master. In phase 3 gradients are sent by the workers back to the master to be aggregated and applied to the model. The process repeats.

3 Sparsify Gradients

We call our proposed method SparsifyGradients, which does exactly that – zero out gradient values so that the overall gradient takes up less memory and hence is faster to transmit across the network. Our belief is that 90% of the smaller gradient values have little effect on convergence and can be discarded without much penalty. During the gradient aggregation phase, SparsifyGradients is performed on each gradient being communicated to the master, zeroing out values of the gradient that fail to meet a certain threshold. Specifically, gradient values are sorted by their absolute values and dropped if they are below a specified percentile. For example, if the percentile is specified to be 90, then 90% of the gradient values with the smallest magnitudes will be zeroed out. After applying SparsifyGradients to each gradient being sent, the resulting gradients are then converted to sparse format and communicated to the master.

Sparsify Gradients Algorithm

```
def SparsifyGradient(gradient, percentile=90)
    threshold = calculate_percentile_value(abs(gradient), percentile)
    return sparse_format(gradient > threshold)
```

4 Experimental Setup

We tested our algorithm on 14 c3.xlarge (13 workers, 1 master) Amazon EC2 machines in a LAN setting, on a cuda-convnet style neural network for the cifar10 task. The c3.xlarge machines are CPU instances with 7.5 GiB of memory and 4 virtual cpus, connected by a 500 Mbps network channel. We use Tensorflow as a blackbox to evaluate and train the neural network and MPI for communication between machines, which uses the TCP protocol underneath. Model synchronization uses the MPI broadcast routine, while gradient aggregation uses the MPI gather routine. Since the MPI broadcast is implemented using an optimized tree-based communication algorithm, while the MPI gather uses a naive algorithm, the gradient aggregation phase accounts for most of the network bottleneck. Our experiments compare sparsifying gradients with a 90 percentile cutoff against the default non-sparsified setting. We use a batch size of 8 per worker when training the cuda convnet neural network.

5 Results and Discussion

We achieve around a 2x speedup in time to .995 training accuracy on 14 c3.xlarge ec2 machines to train a cuda-convnet style neural network.

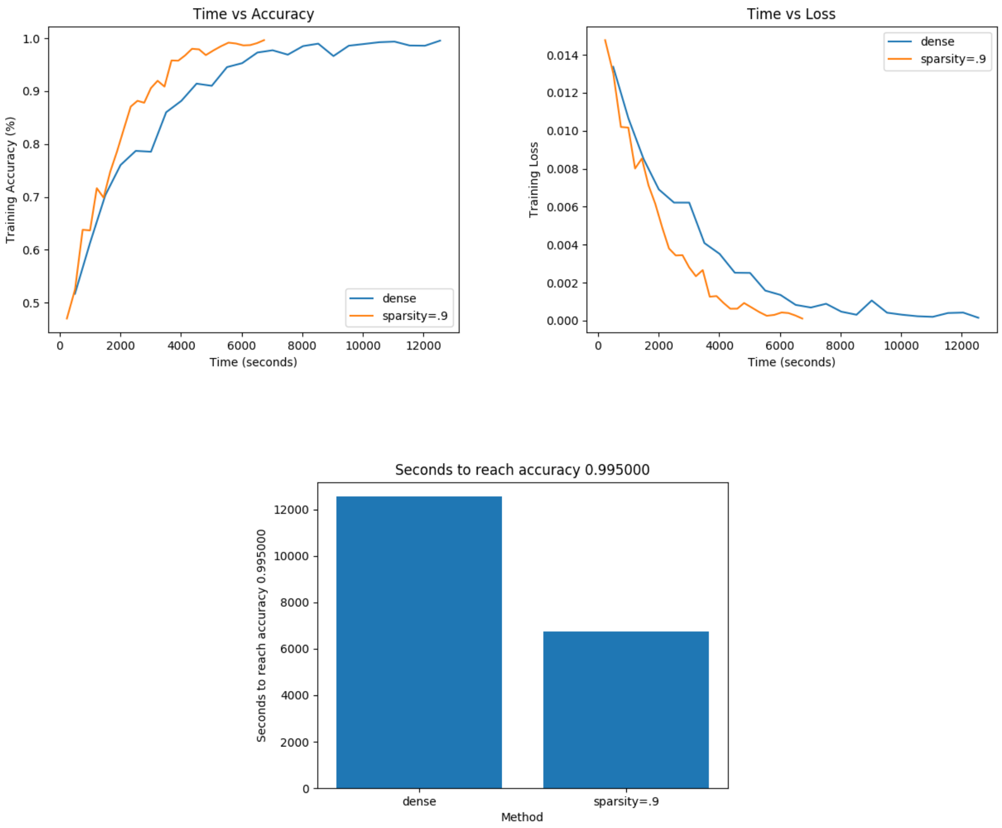
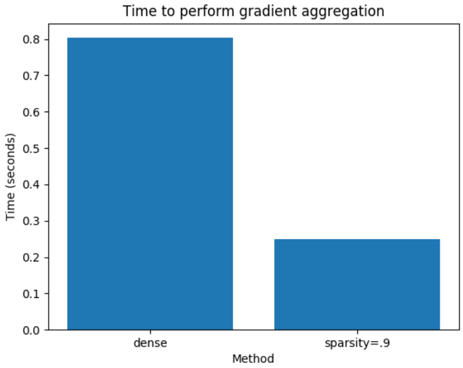
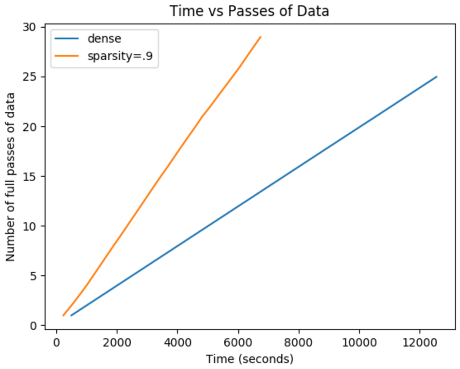


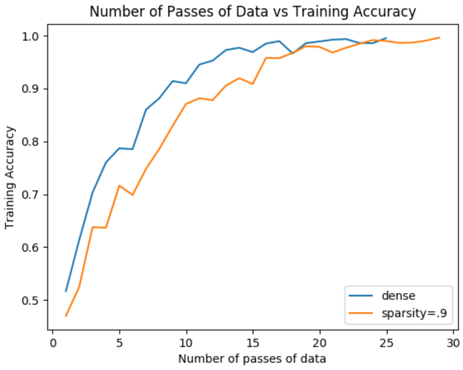
Figure 3: SparsifyGradients achieves around a 2x speedup in time to 99.5% training accuracy while training a cuda-convnet neural network.

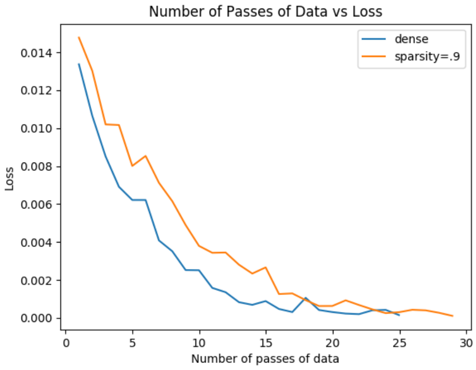
As expected, dropping 90% of the gradient values and converting to sparse format greatly reduces the time needed to perform the gradient aggregation phase, which is mostly network bandwidth bottlenecked. This in turn allows data to be processed faster.





It is important to note that dropping 90% of values only decreases communication time by a factor of around 2-3 due to the format of the sparse matrix, which stores indices along with its values.





The figures above compare the convergence properties of sparsified vs non-sparsified distributed training. As expected, dropping gradient values hurts convergence, though this loss in accuracy is compensated by the speedup in network communication. It is interesting to note that although 90% of gradients' values are zeroed out, convergence to 99.5% training accuracy is still achieved in a comparable number of passes of data. We believe there are a couple of reasons for this. Firstly, we believe that most gradient values do not change model parameters too much and can be seen as noise, and that training is propelled by a few select gradient values. Histograms of the absolute values of the gradients for the cuda convnet model show that gradient values follow a normal distribution, with many values close to 0.

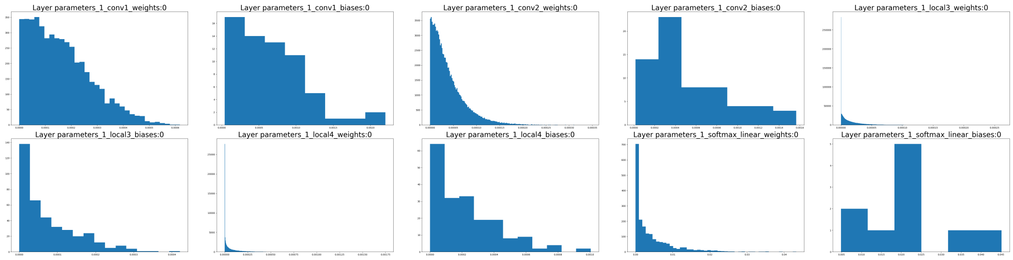


Figure 4: Histograms of gradient magnitudes for each layer of the cuda-convnet model. The histograms follow a normal distribution, implying that only a few percentage of gradient values make a significant impact on training. We believe that the top 10 percent of gradient values account for most of the training convergence, while the rest can be seen as noise.

Secondly, we believe that if a gradient value falls below the threshold in one iteration, but is essential to an accurate model, then in some subsequent iteration the magnitude of that gradient value will rise above the threshold to compensate. Further investigation into these two points is future research.

6 Conclusion

We present SparsifyGradients, a method which drops gradient values to reduce network communication load. We tested our algorithm on a cluster of 14 machines and trained a cuda-convnet model in a process where 90% of the lowest magnitude gradient weights are dropped. Doing so achieves around a 2x speedup over the default in time to 99.5% training accuracy. To explain the relative intactness of convergence while dropping 90% of gradient weights, we propose two explanations. Firstly, that most gradient values can be seen as noise and that only a few select gradient values are responsible for convergence. Secondly, that important gradient values that fail the threshold limit eventually succeed due to a delayed compensation mechanism.

7 Future Work

It is very interesting that despite dropping 90% of gradient values, training still converges to a reasonable training accuracy (99.5%). We believe that this phenomena is the same as that which allows quantized gradients to perform well despite the loss of data; it would be interesting to investigate how much data loss can be tolerated while also guaranteeing convergence. Furthermore, it would be interesting to measure the impact of certain gradient values as they are propagated among different weights. Investigating this might yield insight into which gradient values are worth keeping or discarding.

On the topic of quantization, further work can be done to reduce the gradient network transfer overhead. Such extensions might include quantizing the remaining values to 8-bits, or even enforcing some sort of discretization.

We also neglected the network communication burden of the model synchronization phase, which is in theory as expensive as the gradient aggregation phase. Efficient communication of the model during this phase is another topic for future work.

On the implementation side, it would be more efficient to use Tensorflow's native distributed communication mechanism to test SparsifyGradient, as various scheduling optimizations (like prefetching) are built into their system. Due to time and flexibility constraints we opted to use MPI for communication and Tensorflow as an optimization black box.

Finally, as we have demonstrated that gradient values might be dropped without much penalty, it would be interesting to implement the full training procedure using UDP instead of TCP and simply ignore the lost packets. Due to time constraints and the complexity of managing a lower level of the network model, we were not able to explore this.

8 Update

As of 4/14/2017, we found out that this idea has been explored in `https://arxiv.org/abs/1704.05021` with similar results.

9 Code

`https://github.com/agnusmaximus/SparsifyGradients`