2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems

Entropy-based Gradient Compression For Distributed Deep Learning

Di Kuang, Mengqiang Chen, Danyang Xiao
School of data and Computer Science
Sun Yat-sen University
Guangzhou, China
{kuangd, chenmq9, xiaody}@mail2.sysu.edu.cn

Weigang Wu
School of data and Computer Science
Sun Yat-sen University
Guangzhou, China
wuweig@mail.sysu.edu.cn

***Abstract*—Nowadays, with the increasing amount of data and scale of network models, distributed deep learning, i.e., training a deep neural network using multiple workers distributed across different computing nodes, is becoming more and more popular. One of the major challenges of distributed deep learning lies in the frequent communication of gradients among workers, because it may cause severe bottlenecks, in terms of time latency and bandwidth. In this paper, we propose a novel approach named Entropy-based Gradient Compression (EGC) to reduce communication overhead. The major components of EGC are two algorithms: the entropy-based threshold selection algorithm and the automatic learning rate correction algorithm. To improve the accuracy, EGC also includes two commonly used algorithms: gradient residual and momentum correction. To evaluate the performance of EGC, we conduct experiments based on applications of image classification and language modeling using public datasets including Cifar10, Tiny ImageNet and Penn Treebank. The experiment results show that, compared with existing works, EGC can achieve a gradient compression ratio about 1000× while keeping the accuracy similar or even higher.**

***Index Terms*—distributed training, deep learning, entropy, gradient compression**

I. Introduction

In the past few decades, Deep Learning has become the essential machine learning algorithm and achieved remarkable success in various application fields ranging from machine translation [1], image processing [2], speech recognition [3] and many others. With increase of the scale and complexity of data sets and neural network models, the training of deep neural networks also becomes more and more difficult. For example, the number of parameters of a single network model can be as large as a billion [4]. Training such complex neural networks is very costly, in both time and computing resources. Therefore, how to reduce training time while keeping satisfactory accuracy has become a hotspot in deep learning research.

Distributed deep learning is an effective way to speed up deep neural network training process. There are roughly two paradigms for distributed deep learning: model parallelism [5] and data parallelism [6]. Under model parallelism, different workers/machines in a distributed system are responsible for training different parts of the nerual network model. That is, a network model is divided into layers and/or blocks and different machines are assigned to train different layers/blocks. During training, the workers need to exchange the local parameters/gradients to the next workers to ensure normal training of the model. Under data parallelism, each worker keeps a full copy of the whole network model, and different workers are assigned different shards of the data set. During training, the workers need to exchange gradients/parameters with each other to aggregate training results and get a converged global model. Compared with model parallelism, data parallelism is more effective and more widely used, because the division of data is easier than division of network model.

In data parallelism, the major challenge comes from communications to exchange model parameters among workers, which may delay the computation at workers and also consume network bandwidth.

Different methods [7] [8] [9] [10] have been proposed to reduce communication cost and speed up data parallel training of distributed deep learning. Gradient compression is one of such methods, and it only operates on the communication content, without changing the model structure and communication process.

Gradients compression based solutions can be categorized into two major classes: gradient quantization and gradient sparsification. Quantization-based approaches include 1-bit SGD [11], TernGrad [12] and QSGD [13] and so on. Although they achieve a small loss of accuracy by using at least one bit for each parameter, the compression ratio is limited. Sparsification-based approaches include the threshold quantization of Strom [14] and more. They mainly choose a threshold and send gradient elements which are larger than the threshold. Although they can achieve high compression ratios, they will harm the accuracy of the model. And another important issue is difficult to choose a suitable threshold.

In this paper, we propose a new gradient compression approach, Entropy-based Gradient Compression (EGC). EGC uses an entropy-based adaptive gradient sparsification method. More precisely, EGC uses an entropy-based threshold selection algorithm to calculate the entropy of the gradient for each neural network layer in each epoch. This entropy value, together with a hyper parameter K, is used to determine the threshold of gradient sparsification. To avoid the loss of

This research is partially supported by National Natural Science Foundation of China (U1711263, U1801266), and Guangdong Natural Science Foundation (2018B030312002, 2018A030313492).

978-1-7281-2058-4/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCC/SmartCity/DSS.2019.00046

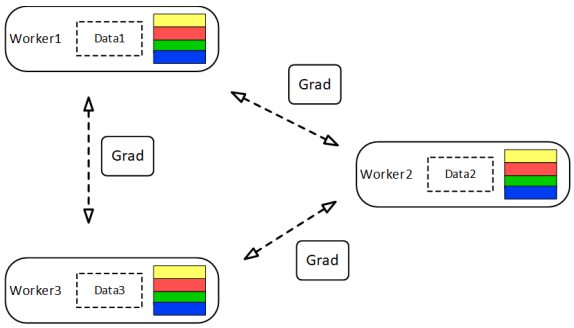

Fig. 1. The All-Reduce mode implemented by NCCL.

accuracy, EGC employs three other components: automatic learning rate correction algorithm, gradient residual algorithm and momentum correction algorithm.

To simplify the communication of gradients, EGC adopts the All-Reduce [15] [16] paradigm implemented by NCCL, a collection communication library for efficient GPUs developed by Nvidia, as shown in Fig. 1.

We conduct experiments to examine the performance of EGC, based on several network models and datasets: CNN [17] for image classification(with Cifar10 [18] and Tiny ImageNet), RNN [19] [20] for language modeling(with Penn Treebank [21]). We also implemented a baseline approach without gradient compression and Gradient Dropping proposed by Aji & Heafield [22] for comparison purpose. The experimental results show that, with EGC, gradient can be compressed up to 1000× while maintaining the accuracy of the resulting model.

The rest of the paper is organized as follows. Section II reviews related work on gradient compression for distributed deep learning and Section III presents the formulation of the problem. Section IV presents the details of the proposed EGC method. The experiments for performance evaluation are reported in V. We finally conclude this paper and introduce future work in Section VI.

II. Related Work

Researchers have proposed different methods to reduce communication cost in distributed deep learning. Among others, gradient quantization and gradient sparsification are effective and widely studied.

A. Gradient Quantization

Gradient quantization quantifies gradients to reduce the precision of values, so that the size of gradient values is can be reduced and fewer Bytes need to be communicated. The major challenge of gradient quantization is how to achieve satisfactory model accuracy with low precise gradients. Different approaches using gradient quantization have been designed. Seide et al. proposed 1-bit quantization [11] approach which quantizes each gradient parameter to a sign bit. That is, in communications among workers, each parameter is quantized to one bit, so that the size of communication data can be significantly reduced. To avoid significant loss in model accuracy, the quantization error is retained and added to the gradients calculated in the next step.This approach can achieve 10× speedup in typical speech recognition applications. Alistarh et al. proposed another approach called QSGD [13]. QSGD stochastically rounds gradients to linearly quantized values and allows the user to trade off accuracy and gradient precision. Meanwhile QSGD justify the effective of gradient quantization and the convergence of this approach. Wen et al. proposed an approach called TernGrad [12] .By using 3-level gradients, TernGrad quantizes gradients with 2 bits per parameter.

B. Gradient Sparsification

Gradient sparsification reduces communication cost by sending a small number of parameters. Strom [14] developed an approach called threshold quantization. In this approach, only gradients which are greater than a predefined threshold will be sent, so that only a small part of the gradients need to be communicated. The quantization errors will add to gradients in the next step, similar as in the 1-bit quantization. However, it is not trivial to determine the threshold for gradients. Dryden et al. [23] proposed to fix the proportion of positive and negative gradients to send, rather than fix the threshold of value. Aji & Heafield [22] proposed the approach of gradient dropping, which sparsifies gradients by removing the gradient of R% (R% is a fixed percentage) with the smallest absolute value. To guarantee convergence, Gradient Dropping uses the layer normalization [24] with one global threshold. This approach can drop 99% of the gradient and achieve 11% speed improvement for a neural machine translation, without impacting the translation quality. Concurrently, Chen et al. proposed an approach named AdaComp [25] which is based on the localized selection of gradient residues and automatically tunes the compression rate depending on local activity.

C. Summary

In this section, we introduce two methods to reduce communication cost, such as gradient quantization and gradient sparsification.Gradient quantization reduces the bit widths of gradients, while gradient sparsification reduces the number of gradients that need to be sent. In this paper, we propose a novel gradient sparsification approach, which uses an entropy-based threshold to select gradients to be sent. Also, our approach does not use layer normalization [24], so we do not change the model structure. More importantly, our approach can avoid the loss of accuracy to some extent.

III. Problem Formulation

In deep learning, the most common optimization method is Gradient Descent, which uses all the samples to optimize the weights in a neural network model. As the number of samples increases, the amount of calculation will increase greatly. To simplify the optimization procedure, researchers propose Stochastic Gradient Descent (SGD) [26], which should be the most widely used method in deep learning.

With the typical SGD, model parameters are update as:

$$x_{t+1} = x_t - \gamma \bigtriangledown f_t(x_t) \tag{1}$$

where x_t and $\bigtriangledown f_t(x_t)$ are model parameters and calculated gradients in step t.f_t is a loss function and γ is the learning rate.

In data parallel distributed SGD, all workers have identical copies of the same model and calculate gradients using different subsets of training data. Gradients ($\bigtriangledown f_t(x_t)$) are shared across all workers, and each worker updates its local model using the shared gradients.

The communication between workers is a bottleneck in the training process. Gradients consist of tens of millions of floating point values, so the total size of exchanged data can be very large and cost much time and network bandwidth.

IV. The Proposed Approach

In this section, we will give a comprehensive introduction to EGC. Firstly, we will introduce the entropy-based threshold selection algorithm used to dynamically determine gradients to sent out. Then, we will introduce our automatic learning rate correction algorithm which can reduce the learning rate with the test indicator. Finally, we introduce gradient residual and momentum correction to ensure convergence.

A. Entropy-based Threshold Selection Algorithm

In this algorithm, we calculate the entropy of the gradient of each layer in each epoch to evaluate the gradient's importance. Entropy is a commonly used metric to measure the disorder or uncertainty in information theory. A larger entropy value means more information in the gradient. In EGC, when the gradient of a layer contains less information, we will send fewer gradients.

To compute the entropy, we need to collect gradient matrices that can be obtained using an evaluation set. In this algorithm, we use all of the gradient matrices of each layer as the evaluation set. In the first step, we divide the elements into N different bins. The second step counts the number of corresponding gradient elements in each bin, and uses the number to calculate the probability of the number of gradients in each bin. The third step calculates the entropy of this layer by using the probability of each bin. The formula is as follows:

$$H_j = -\sum_{i=1}^{N} p_i \log p_i \tag{2}$$

where p_i is the probability of bin i, H_j is the entropy of the gradient of the layer j. From (2), we know that the entropy is related to the value of N. So, in next section, we will choose a good value of N by experiment.

In general, when the gradient of a layer contains little information, its entropy will be quite small. Therefore, our approach can be used to estimate the amount of information contained in each layer of the gradient.

The next challenge is how to use the obtained entropy value to determine the threshold of gradients to be sent. Here we are divided into two steps. Firstly, A fixed ratio $\frac{1}{K}$ is used to indicate the proportion of the gradient to be sent after each minibatch. Since the unit of entropy we calculated is bit, our choice of K value is a form 2^N , for example 2^8, 2^9, 2^{10}, 2^{11} etc. Based on experience, we learn that K provides good results when it is equal to 2^{10}. In the next section, we will show experiments that support this conclusion. Finally, we will determine the threshold τ. We get the τ value by using a selection algorithm(e.g. Quickselect). The algorithm will return the $\frac{H_j}{K}$th largest gradient w, which is the threshold τ. After finding the threshold τ, we only need to send the gradient with an absolute value greater than the threshold.

Algorithm 1 describes the entropy-based threshold selection algorithm. The output of the algorithm is the threshold of each layer.

Algorithm 1 Entropy-based Threshold Selection Algorithm

Input:
Gradients G_l, $l = 1..L$ for each of the L layers in the net, K, N
Output:
τ_l, $l = 1..L$

for layer $l = 1..L$ **do**
  Find G^l_{max} = $max_{g \in G_l} g$
  Find G^l_{min} = $min_{g \in G_l} g$
  Compute the interval: $H = \frac{abs(G^l_{max} - G^l_{min})}{N}$
  for bin $i = 1..N$ **do**
    Compute Number $h(i)$ = Estimate Number(G_l, i, H, G^l_{min})
  end for
  for $k = 1..N$ **do**
    Compute Probability $p(k)$ = $\frac{h(k)}{\sum_k h(k)}$
  end for
  Entropy $H = -\sum_{i=1}^{N} p(i) \log p(i)$
  The ratio $t = \frac{H}{K}$
  Threshold $\tau_l = Quickselect(G_l, t)$
end for
Function Estimate Number(G_l, i, H, G^l_{min})
$num = 0$
for $k = 1, 2, \cdots$ **do**
  if $G_l[k] \geq G^l_{min} + (i-1) * H$ and $G_l[k] < G^l_{min} + i * H$ **then**
    $num = num + 1$
  end if
end for
return num

B. Automatic Learning Rate Correction Method

Learning rate is an important parameter in a training model. Adjusting the learning rate is one of the critical factors to influence both convergence and training speed. Therefore, automatically correcting the learning rate can help improve model accuracy and reduce training time.

In this paper, we change the learning rate dynamically based on test indicators. In each epoch, we use the training set to train the model and then verify it with the validation set. After training the validation set, we will get the loss value. The change of the loss value can respond well to the training

situation of the entire model. So, we identify the loss value as the test indicators. Before training, we set the ratio t of the rate of attenuation of the learning rate, the number n of non-standards and the lower limit of the learning rate min_lr.

In the first epoch of training, we set the initial value of the learning rate(γ) and the loss value(val_loss). In the next epoch, we will compare the loss value with val_loss . If the loss value does not decrease for n consecutive times, model parameters are update as:

$$x_{t+1} = x_t - \frac{\gamma}{1 + t * epochs} * \bigtriangledown f_t(x_t) \tag{3}$$

Where $\bigtriangledown f_t(x_t)$ is calculated gradients in step t.f_t is a loss function, and γ is the learning rate. t is the rate of attenuation of the learning rate. $epochs$ is the number of the epoch. From (3), we can see that the learning rate will reduce quickly when the epoch increases. If the learning rate drops to min_lr, we will set the learning rate to min_lr and do not change it again.

In this algorithm, we can set the learning rate to a large initial value to accelerate convergence at the start of training. With increasing the epoch, the learning rate will automatically reduce to make accuracy close to the best.

Algorithm 2 describes the automatic learning rate correction algorithm.

Algorithm 2 Automatic Learning Rate Correction Algorithm

Input:

Initial parameters γ and val_loss, the number of epochs N , n, t, min_lr, evaluate function Evaluate(), dataset $test\_loader$

Output:

γ

$k = 0$
for epoch $i = 1$..... **do**
  val_loss_new = Evaluate($test\_loader$)
  if $val\_loss\_new > val\_loss$ **then**
    $k = k + 1$
    val_loss = val_loss_new
  end if
  if $k \geqslant n$ **then**
    $\gamma = \frac{\gamma}{1+t*i}$ and $k = 0$
  end if
  if $\gamma < min\_lr$ **then**
    $\gamma = min\_lr$
  end if
end for
return γ

C. Gradient residual and momentum correction

With using the entropy-based threshold selection algorithm, the number of gradients sent in each epoch will be much less than the original gradient matrix. If we discard unsent gradient elements, it will have a great impact on convergence. So, on each worker, after each mini-batch, gradient values are aggregated in gradient residuals [14]. In the gradient residual, we will add the gradient element not sent in the previous epoch as the residual to the gradient element calculated by the next epoch and then judge the threshold. If it is greater than the threshold, it will be sent. Otherwise, it will continue to accumulate. So, we ensure that a gradient with an absolute value less than the threshold can be sent.

At the same time, we have learned that the DGC [27] has made several minor modifications to the ordinary gradient descent method to improve the convergence speed and ensuring convergence. So in this paper, we use the momentum correction method proposed in DGC, which mainly uses momentum correction gradient instead of the original gradient to add to the residual to ensure convergence.

V. Peroformance Evaluation

In this section, we experimentally evaluate our approach based on the model convergence and the compression ratio. Specifically, we demonstrate that the proposed approach can reduce communication overhead while maintaining accuracy.

A. Experiment settings

The experiments use GPU nodes in the Tianhe-2 supercomputer system. Each GPU node is equipped with two 10-core Intel Xeon E5-2660 v3 CPUs and two NVIDIA Tesla K80 GPUs. The memory capacity of each GPU node is 256GB. We conduct our experiments on PyTorch, which is a deep learning experimental platform that provides a high degree of flexibility and efficiency. The communication mode between workers is the All-Reduce mode implemented by NCCL.

We compare our approach with the baseline and Gradient Dropping, in terms of training loss, accuracy and compression ratio. The baseline is the basic approach without gradient compression. Gradient Dropping is proposed by Aji et al [22]. The accuracy is defined as the test accuracy at the last epoch, and the compression ratio is defined as the number of the total parameters of networks divided by the average number of parameters sent.

We validate our approach via two types of machine learning tasks: image classification on Cifar10 and Tiny ImageNet, language modeling on Penn Treebank dataset.

- Image Classification: We studied AlexNet [28], VGG-19 [29] and ResNet-50 [30] on Cifar10 [18], ResNet-18 [30] on Tiny ImageNet. Cifar10 [18] consists of 50000 training images and 10000 validation images in 10 classes. The dataset is divided into five training batches and one test batch, and there are 10000 images per batch.

TABLE I
ACCURACY AND COMPRESSION RATIO WITH DIFFERENT K

Model	K	Compression Ratio	Accuracy
AlexNet	256	269×	0.7457
	512	524×	0.7446
	1024	1041×	0.7462
	2048	2077×	0.7056

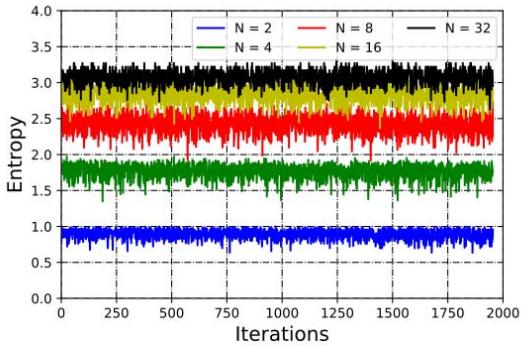

Fig. 2. The entropy with different N

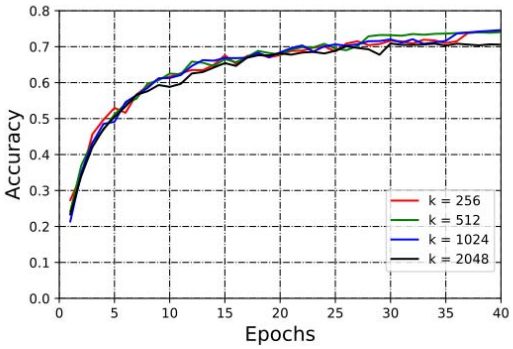

Fig. 3. Accuracy of AlexNet on Cifar10 with different k

Tiny ImageNet Challenge is the default course project for Stanford CS231N. Tiny ImageNet has 200 classes. Each class has 500 training images, 50 validation images and 50 test images.

- Language Modeling: The Penn Treebank(PTB) [21] dataset is an English corpus, which consists of 923000 training, 73000 validation and 82000 test words. We adopt the 2-Layer LSTM language model architecture with 200 hidden units per layer.

B. Establishing Optimal Parameter Values

The proposed approach has two hyper-parameters: (a) N, the number of bins used for the preliminary entropy estimate, (b) $\frac{1}{k}$, the fixed rate of sending gradient.

For the value of N, we find that different N influences the entropy. On the single GPU node, we study AlexNet on Cifar10 with different N to choose a good value of N. Fig. 2 shows that the entropy is smallest when N equals 2. So we initialize the value of N as 2.

For the value of K, we experiment with some fixed values to determine a better value of k. Since the unit of entropy is bit, we choose the value of the form 2^N. Fig. 3 shows the convergence curve of AlexNet under 4 GPU nodes and different values of k. When k equals 256, 512 and 1024, the learning curve converges similar. The learning curve is worse than others when k is equal to 2048.

TABLE.I shows the detailed accuracy and compression ratio. From TABLE.I, when k equals 256, 512 and 1024, the accuracy is similar which is about 0.745. When k is 2048, the accuracy is 0.7056. Although the compression ratio is the highest when k is equivalent to 2048, its accuracy is the smallest. So we choose the value of k as 1024 by considering the compression ratio and accuracy.

C. Image Classification

We first examine EGC on image classification task. For the experiment, we study AlexNet,VGG-19 and ResNet-50 on Cifar10, ResNet-18 on Tiny ImageNet. TABLE.II lists compression rates and final accuracies achieved by different compression methods when applied to the training of neural networks on 2 different datasets.

In TABLE.II, the accuracy of EGC can compare to the baseline and Gradient Dropping in all experiments and even higher in some experiments, such as ResNet-50 on Cifar10 and ResNet-18 on Tiny ImageNet. For the compression ratio, EGC compresses the gradient by 1000×, but Gradient Dropping only compresses the gradient by 100×. For the baseline and Gradient Dropping, EGC can achieve a higher compression ratio, with comparable accuracy.

To reflect the performance changes of our approach when we expand the node, we experiment with AlexNet on 2, 4,

TABLE II
ACCURACY AND COMPRESSION RATIO ON IMAGE CLASSFICATION

Method		Baseline	Gradient Dropping	EGC
AlexNet @Cifar10	Accuracy	0.713	0.7122	0.7262
	Compression	1×	100×	1041×
VGG-19 @Cifar10	Accuracy	0.8922	0.8915	0.8913
	Compression	1×	100×	1153×
ResNet-50 @Cifar10	Accuracy	0.8608	0.8732	0.8647
	Compression	1×	100×	1052×
ResNet-18 @Tiny ImageNet	Accuracy	0.3575	0.3611	0.3681
	Compression	1×	100×	1030×

TABLE III
ALEXNET TRAINED ON CIFAR10 DATASET

GPUs	Method(Compression Ratio)	Accuracy
2	Baseline(1×)	72.92%
	Gradient Dropping(100×)	73.34% +0.38%
	EGC(1038×)	74.40% +1.48%
4	Baseline(1×)	71.30%
	Gradient Dropping(100×)	71.22% -0.08%
	EGC(1041×)	72.62% +1.32%
8	Baseline(1×)	69.17%
	Gradient Dropping(100×)	69.96% +0.79%
	EGC(1044×)	70.03% +0.86%

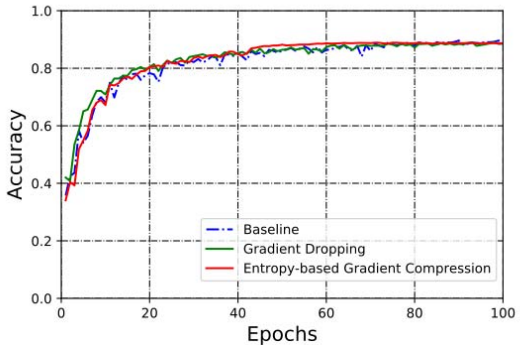

Fig. 4. Accuracy of VGG-19 on Cifar10

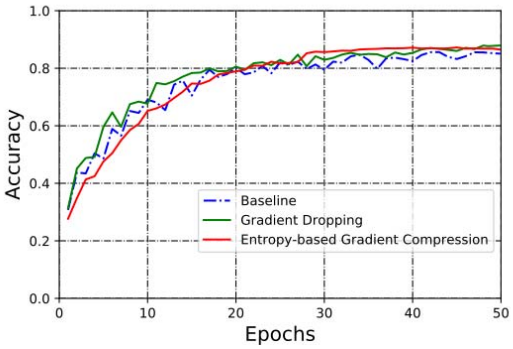

Fig. 6. Accuracy of ResNet-50 on Cifar10

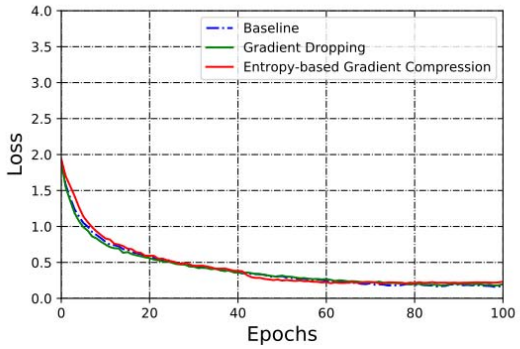

Fig. 5. Training loss of VGG-19 on Cifar10

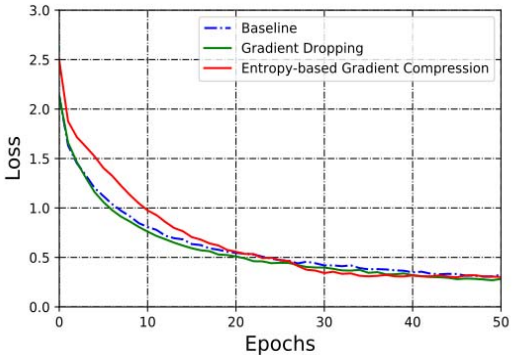

Fig. 7. Training loss of ResNet-50 on Cifar10

8 GPU nodes. TABLE.III shows the result of the experiment. The accuracy of the baseline, Gradient Dropping and EGC will decline with the increase of GPU nodes. But the accuracy of EGC is always higher than others. Meanwhile, we observe that EGC's compression ratio is the highest in three approaches, which can achieve approximately 1040×. So, when the number of GPU nodes increases, EGC has the higest accuracy and compression ratio in three approaches.

Fig. 4 and 5 display the accuracy and training loss of VGG-19 on Cifar10 with 4 GPU nodes. On the figure, the blue line, green line and red line signify the baseline, Gradient Dropping and our approach, respectively. We observe that the learning curve converges fastest with EGC. For accuracy, three approaches have similar accuracy, and the baseline will be slightly better. For the training loss, learning curves of Gradient Dropping and EGC are close to the baseline. The eventual loss of EGC is little better than the baseline and Gradient Dropping. Combined with TABLE.II, EGC can increase the compression ratio from 100× to 1153× with a slight reduction in accuracy.

Fig. 6 and 7 show the accuracy and training loss of ResNet-50 on Cifar10 with 4 GPU nodes. From Fig. 6, we can observe that the convergence speed of EGC is similar to the baseline and Gradient Dropping. They all start to converge from the 27th epoch. Combined with TABLE.II, the accuracy of Gradient Dropping is the best in three approaches, and EGC is better than the baseline. In Fig. 7, the training loss of Gradient Dropping is the lowest and our approach closely match the baseline. To sum up, for ResNet-50, EGC can get a high compression ratio with a little increase in accuracy.

When scaling to the large-scale dataset, Fig. 8 and 9 show the learning curve of ResNet-18 on Tiny ImageNet. Compared to the baseline and Gradient Dropping, the accuracy of EGC is improved, and increases by one percent. The learning curve of EGC is much better than others. Simultaneously, the training loss of EGC is the smallest, and the training loss curve converges fastest. Besides, combine the data in TABLE.II, EGC gives 10× better compression than Gradient Dropping. As a result, EGC gains higher accuracy than others and compresses the gradient by 1030×.

D. Language Modeling

For language modeling, Fig. 10 shows the perplexity of the language model trained with 4 GPU nodes on PTB dataset. The perplexity with EGC closely matches the baseline and

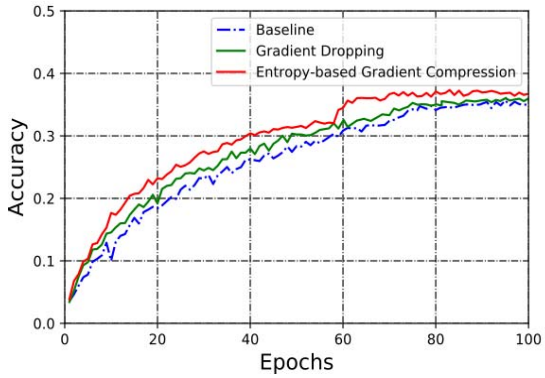

Fig. 8. Accuracy of ResNet-18 on Tiny ImageNet

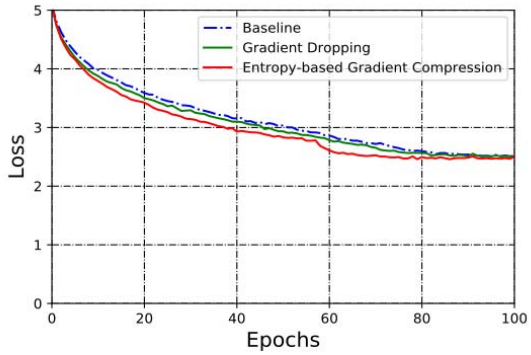

Fig. 9. Training loss of ResNet-18 on Tiny ImageNet

Gradient Dropping. From TABLE.IV, EGC compresses the gradient by 1098× with a slight reduction in perplexity.

VI. Conclusion And Future Work

In this paper, we propose EGC, an entropy-based adaptive gradient compression approach to solve communication overhead. We calculate the entropy of the gradient of each layer to dynamically determine the threshold at which the gradient needs to be sent, where the entropy changes with the number of iterations and the network layer. To reduce the impact of gradient sparsity on convergence, EGC employs automatic learning rate correction algorithm, gradient residual and momentum correction.

We evaluate the proposed approach on two machine learning tasks: image classification and language modeling. On image classification, the experimental results demonstrate that EGC can compress the gradient by approximately 1000× with no loss of accuracy in most instances. In addition, when the number of nodes increases, EGC still has a good effect. Moreover, changes in the network model have little effect on EGC. Meanwhile, the experiments show that EGC also has good results in a large-scale dataset. On language modeling, EGC can compress the gradient by 1100× with a slight reduction in perplexity. In summary, in some fields, EGC can

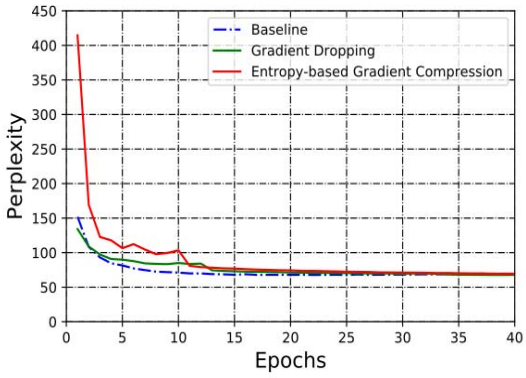

Fig. 10. Perplexity of LSTM language model on PTB dataset

TABLE IV
TRAINING RESULT OF LANGUAGE MODELING WITH 2 NODES

Method	Perplexity	Compression Ratio
Baseline	69.48	1×
Gradient Dropping	67.84 (-1.64)	100×
EGC	69.40 (-0.08)	1098×

compress the gradient by approximately 1000× while keeping the accuracy similar or even higher.

Although our approach has achieved good results in the experiment, there are several things for future improvements. On the one hand, due to limitations of the experimental environment, we cannot experiment with more GPU nodes. So, some works will be done in a large-scale cluster. On the other hand, we will study how to apply the entropy of the gradient to the gradient quantization to achieve the combination of quantization and sparsification.

References

[1] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[3] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, "The microsoft 2017 conversational speech recognition system," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5934–5938.

[4] J. Yuan, F. Gao, Q. Ho, W. Dai, J. Wei, X. Zheng, E. P. Xing, T.-Y. Liu, and W.-Y. Ma, "Lightlda: Big topic models on modest computer clusters," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2015, pp. 1351–1361. [Online]. Available: https://doi.org/10.1145/2736277.2741115

[5] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[6] S. Gupta, W. Zhang, and F. Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 171–180.

[7] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[8] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[9] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[10] P. Sun, Y. Wen, T. N. B. Duong, and S. Yan, "Timed dataflow: Reducing communication overhead for distributed machine learning systems," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2016, pp. 1110–1117.

[11] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[12] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Advances in neural information processing systems*, 2017, pp. 1509–1519.

[13] D. Alistarh, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Randomized quantization for communication-optimal stochastic gradient descent," *arXiv preprint arXiv:1610.02132*, 2016.

[14] N. Strom, "Scalable distributed dnn training using commodity gpu cloud computing," in *INTERSPEECH*, 2015.

[15] R. Rabenseifner, "Optimization of collective reduction operations," in *Computational Science-iccs, International Conference, Kraków, Poland, June*, 2004.

[16] J. Bruck, C. T. Ho, S. Kipnis, E. Upfal, and D. Weathersby, "Efficient algorithms for all-to-all communications in multi-port message-passing systems," *Parallel & Distributed Systems IEEE Transactions on*, vol. 8, no. 11, pp. 1143–1156, 1997.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, pp. 84–90, 2012.

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

[19] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, 2015.

[20] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, "Recurrent neural network based language model," in *INTERSPEECH*, 2010.

[21] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," *Computational Linguistics*, vol. 19, pp. 313–330, 1993.

[22] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *EMNLP*, 2017.

[23] N. Dryden, T. Moon, S. A. Jacobs, and B. V. Essen, "Communication quantization for data-parallel training of deep neural networks," *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)*, pp. 1–8, 2016.

[24] J. Ba, R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016.

[25] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan, "Adacomp : Adaptive residual gradient compression for data-parallel distributed training," in *AAAI*, 2018.

[26] L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*, 2012.

[27] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *CoRR*, vol. abs/1712.01887, 2018.

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.