A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is a light greenish-blue. They are positioned diagonally, with the blue one partially covering the green one.


Term Paper Presentation - Gradient Compression for Distributed Training

NAMAN MAKKAR (nbm49)



Need for gradient compression -

- Distributed model training suffers from communication bottlenecks due to the frequent large gradient updates that have to be transmitted across different compute nodes when utilising synchronous data-parallel SGD.
- Due to the fact that the gradient vectors of the modern over-parameterized models are large, this affects the scalability of distributed deep learning training.
- An effective way to address this issue is to make use of gradient compression or communication compression techniques and algorithms which seek to reduce the size of the gradient updates to ensure scalability.



Traditional Lossy Gradient Compression Techniques

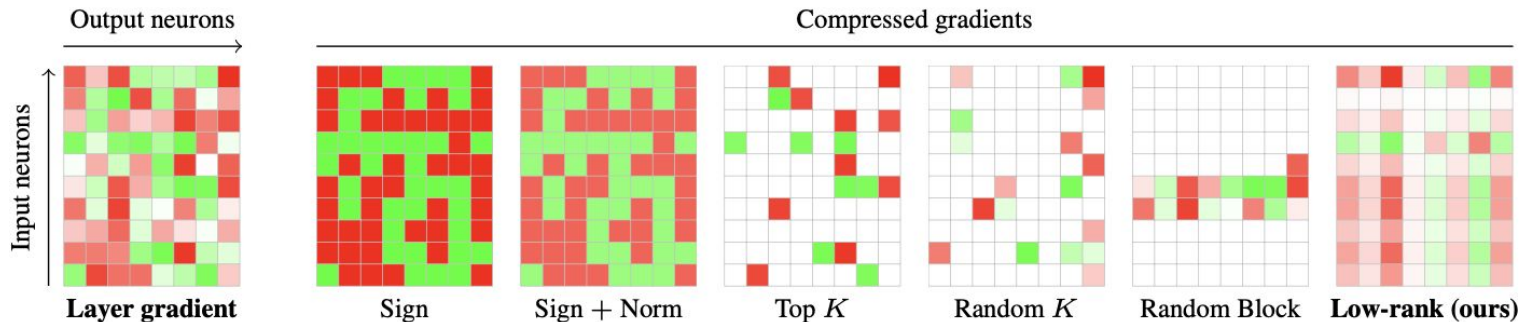
- Quantization - In this approach gradients are quantized to a low bit precision. Can only reduce the traffic by 32x (1 bit in 32 bits) and is not practical for large-scale models or low-bandwidth networks.
- Sparsification - This approach selects a subset of the gradient elements and removes the non-significant gradient elements with little impact on convergence.

Problems with Traditional Lossy Gradient Compression -

- Non-linearity - Majority of the lossy gradient compression methods like Top K sparsification for example are non-linear compressors and the gradients compressed by them cannot be added hierarchically hence preventing the current SGD algorithms from making use of ***all-reduce*** (which is utilised in most deep learning frameworks), but instead require a ***gather*** operation. This reduces the scalability of these approaches.
- Use of static compression ratio - Due to the fact that traditional lossy gradient compression techniques make use of a static compression ratio throughout the entire training regime forcing the users to balance the trade-off between model accuracy and per-iteration speedup.

PowerSGD - An algorithm for low rank gradient approximation

- PowerSGD comes up with a computationally lightweight linear compressor, which calculates a low rank (rank- r) approximation of the gradients.
- Calculating a low rank approximation of gradients is equivalent to carrying out a spectral regularization and can be expected to have good generalization ability compared to sparsification and quantization.
- It has been shown that the eigenspectrum of the stochastic gradients of deep learning models decays, which means that low rank approximation techniques can get away with aggressive gradient compression without sacrificing convergence.



PowerSGD Algorithm - Power Iteration step

- Each layer in the model is approximated independently. \mathbf{M} is the gradient of each parameter given below.
- Compress + Aggregate calculates a rank- r approximation of the gradient matrix \mathbf{M} with the help of matrix \mathbf{Q} which is initialized from an i.i.d standard normal distribution. The matrix \mathbf{P} is calculated by an all-reduce mean of the matrix multiplication between \mathbf{M} and \mathbf{Q} over W workers.
- The matrix \mathbf{P} which is an $n \times r$ matrix is then orthogonalized using the Gram-Schmidt process, this is the most expensive part of the computation though much cheaper than using SVD.
- The orthogonalized \mathbf{P} matrix is used for estimating \mathbf{Q} using another all-reduce on matrix multiplication over W workers. \mathbf{P} and \mathbf{Q} matrices can then be used as a compressed, rank- r representation of gradient matrix \mathbf{M} .
- This yields the factorization $\mathbf{M} \sim \mathbf{P}\mathbf{Q}^\top$ with the similar performance as the best rank- r approximation using SVD.
- The algorithm makes use of a warm-start, i.e the compressed representation - \mathbf{Q} is reused for approximating the next gradient, this improves the accuracy of the gradient approximation

Algorithm 1 Rank- r POWERSGD compression

```
1: The update vector  $\Delta_w$  is treated as a list of tensors corresponding to individual model parameters.
   Vector-shaped parameters (biases) are aggregated uncompressed. Other parameters are reshaped
   into matrices. The functions below operate on such matrices independently. For each matrix
    $M \in \mathbb{R}^{n \times m}$ , a corresponding  $Q \in \mathbb{R}^{m \times r}$  is initialized from an i.i.d. standard normal distribution.
2: function COMPRESS+AGGREGATE(update matrix  $M \in \mathbb{R}^{n \times m}$ , previous  $Q \in \mathbb{R}^{m \times r}$ )
3:    $P \leftarrow MQ$ 
4:    $P \leftarrow \text{ALL REDUCE MEAN}(P)$   $\triangleright$  Now,  $P = \frac{1}{W}(M_1 + \dots + M_W)Q$ 
5:    $\hat{P} \leftarrow \text{ORTHOGONALIZE}(P)$   $\triangleright$  Orthonormal columns
6:    $Q \leftarrow M^\top \hat{P}$ 
7:    $Q \leftarrow \text{ALL REDUCE MEAN}(Q)$   $\triangleright$  Now,  $Q = \frac{1}{W}(M_1 + \dots + M_W)^\top \hat{P}$ 
8:   return the compressed representation  $(\hat{P}, Q)$ .
9: end function
10: function DECOMPRESS( $\hat{P} \in \mathbb{R}^{n \times r}$ ,  $Q \in \mathbb{R}^{m \times r}$ )
11:   return  $\hat{P}Q^\top$ 
12: end function
```

PowerSGD - Accommodating the power iteration step in the SGD algorithm

- The PowerSGD's power iteration scheme is a biased compressor, i.e. compression followed by decompression does not yield the original gradient
- It therefore, makes use of an error feedback mechanism in the SGD algorithm
- Error calculated by subtracting the rank-r approximated gradient from the original gradient
- The momentum term is updated with the aggregated rank-r approximated gradients across W workers, which are aggregated with the help of all-reduce thanks to the linearity of the compressor algorithm.

Algorithm 2 Distributed Error-feedback SGD with Momentum

```
1: hyperparameters: learning rate  $\gamma$ , momentum parameter  $\lambda$ 
2: initialize model parameters  $\mathbf{x} \in \mathbb{R}^d$ , momentum  $\mathbf{m} \leftarrow \mathbf{0} \in \mathbb{R}^d$ , replicated across workers
3: at each worker  $w = 1, \dots, W$  do
4:   initialize memory  $\mathbf{e}_w \leftarrow \mathbf{0} \in \mathbb{R}^d$ 
5:   for each iterate  $t = 0, \dots$  do
6:     Compute a stochastic gradient  $\mathbf{g}_w \in \mathbb{R}^d$ .
7:      $\Delta_w \leftarrow \mathbf{g}_w + \mathbf{e}_w$  ▷ Incorporate error-feedback into update
8:      $\mathcal{C}(\Delta_w) \leftarrow \text{COMPRESS}(\Delta_w)$ 
9:      $\mathbf{e}_w \leftarrow \Delta_w - \text{DECOMPRESS}(\mathcal{C}(\Delta_w))$  ▷ Memorize local errors
10:     $\mathcal{C}(\Delta) \leftarrow \text{AGGREGATE}(\mathcal{C}(\Delta_1), \dots, \mathcal{C}(\Delta_W))$  ▷ Exchange gradients
11:     $\Delta' \leftarrow \text{DECOMPRESS}(\mathcal{C}(\Delta))$  ▷ Reconstruct an update  $\in \mathbb{R}^d$ 
12:     $\mathbf{m} \leftarrow \lambda \mathbf{m} + \Delta'$ 
13:     $\mathbf{x} \leftarrow \mathbf{x} - \gamma (\Delta' + \mathbf{m})$ 
14:  end for
15: end at
```

Experimental Results of PowerSGD and observations

- It was observed that Rank 2 and Rank 4 approximation provide comparable or even better results to the original SGD algorithm with no gradient compression while reducing the communication per epoch by 136x and 72x respectively on an experimental setup consisting of 16 GPUs.
- This can be explained by the fact that in modern over-parameterized neural networks, the final learnt model has a low stable rank. Due to this, a low rank approximation of the gradients provided spectral regularization to the model allowing it to perform better on the test set.
- When compared with gradient sparsification techniques and gradient quantization techniques, Rank 2 and Rank 7 approximations of PowerSGD provided the best balance between test accuracy and time per batch.
- However, one of the drawbacks of this algorithm is that it too makes use of a static compression rate like the lossy gradient compression techniques.

Table 3: PowerSGD with varying rank. With sufficient rank, PowerSGD accelerates training of a ResNet18 and an LSTM by reducing communication, achieving test quality on par with regular SGD in the same number of iterations. The time per batch includes the forward/backward pass (constant). See Section 5 for the experimental setup.

Image classification — RESNET18 on CIFAR10					
Algorithm	Test accuracy	Data sent per epoch		Time per batch	
SGD	94.3%	1023 MB	(1×)	312 ms	+0%
Rank 1	93.6%	4 MB	(243×)	229 ms	−26%
Rank 2	94.4%	8 MB	(136×)	239 ms	−23%
Rank 4	94.5%	14 MB	(72×)	260 ms	−16%

Language modeling — LSTM on WIKITEXT-2					
Algorithm	Test perplexity	Data sent per epoch		Time per batch	
SGD	91	7730 MB	(1×)	300 ms	+0%
Rank 1	102	25 MB	(310×)	131 ms	−56%
Rank 2	93	38 MB	(203×)	141 ms	−53%
Rank 4	91	64 MB	(120×)	134 ms	−55%

Table 4: Comparing different compression operators for Error-feedback SGD in a unified setting; running 300 epochs of Error-feedback SGD with Momentum (Algorithm 2) with a learning rate tuned for full-precision SGD on 16 GPUs for CIFAR10. Note that the variations of PowerSGD with ranks 2 and 7 strikes the best balance between the achieved test accuracy and time per batch (total time for forward, backward, compression, decompression, and gradient aggregation).

		Test accuracy	Sent/epoch	All-reduce	Time/batch
No compression		94.3%	1023 MB	✓	312 ms
Medium	Rank 7	94.6%	24 MB	✓	285 ms
	Random Block	93.3%	24 MB	✓	243 ms
	Random K	94.0%	24 MB	✓	540 ms
	Sign+Norm	93.9%	32 MB	✗	429 ms
	Top K	94.4%	32 MB	✗	444 ms
High	Rank 2	94.4%	8 MB	✓	239 ms
	Random Block	87.8%	8 MB	✓	240 ms
	Random K	92.6%	8 MB	✓	534 ms
	Top K	93.6%	8 MB	✗	411 ms

Accordion - An adaptive gradient compression algorithm

- This algorithm addresses the problem of static compression rates used for the purpose of gradient compression.
- Accordion is an adaptive gradient compression algorithm which identifies critical regimes of model training and makes use of a low compression rate for these critical regimes of training while utilising a higher compression rate for non-critical regimes
- The identification of critical regimes is carried out by measuring the change in the accumulated gradient norm in the current and previous epoch and making use of a threshold used for declaring critical regimes. The threshold is initialized to 0.5 in the algorithm.

$$\frac{|\|\Delta_{\text{old}}\| - \|\Delta_{\text{curr}}\||}{\|\Delta_{\text{old}}\|} \geq \eta$$

Algorithm 1 ACCORDION for Gradient Compression

HyperParameters: compression levels $\{\ell_{\text{low}}, \ell_{\text{high}}\}$ and detection threshold η

Input: accumulated gradients in the current epoch (Δ_{curr}) and in the previous epoch (Δ_{prev})

Input: learning rate of the current epoch (γ_{curr}) and of the next epoch (γ_{next})

Output: compression ratio to use ℓ

if $\|\Delta_{\text{prev}}\| - \|\Delta_{\text{curr}}\| / \|\Delta_{\text{prev}}\| \geq \eta$ **or** $\gamma_{\text{next}} < \gamma_{\text{curr}}$ **then**

return ℓ_{low}

else

return ℓ_{high}

end if

Accordion with PowerSGD

- Accordion was implemented with PowerSGD by performing adaptive switching between Rank 1 and Rank 2,4 gradient approximations with the higher rank approximations being utilised for the critical regimes.
- It was observed that Training ResNet-18 on CIFAR-100 using Rank-1 approximation only achieved a test accuracy of 70% while training using Rank-2 approximation gave an accuracy of 71.7%. While switching between Rank-1 and Rank-2 approximation using the Accordion adaptive gradient compression algorithm by identification of the critical regimes yielded a test accuracy of 71.8%.
- It can be observed that switching between different Rank-r approximations of the gradients not only ensured similar performance to that of the higher rank approximation but also ensured a reduction in the data communicated per epoch.
- One of the drawbacks that can be observed for the Accordion algorithm is that when adaptive gradient compression with switching between Rank-1 and Rank-4 approximations was carried out on the VGG-19, the resulting test accuracy only matched that of the Rank-2 approximated PowerSGD, only when adaptive switching was carried out between Rank-2 and Rank-4 approximations did it perform as well as the Rank-4 approximation. This means that the lower rank number for the rank-r approximation has to be chosen carefully.

Table 1. ACCORDION with POWERSGD on CIFAR-10

Network	Rank	Accuracy	Data Sent (Million Floats)	Time (Seconds)
Resnet-18	Rank 2	94.5%	2418.4 (1×)	3509 (1×)
	Rank 1	94.1%	1350.4 (1.7×)	3386 (1.03×)
	ACCORDION	94.5%	1571.8 (1.5 ×)	3398 (1.03 ×)
VGG-19bn	Rank 4	93.4%	6752.0 (1×)	3613 (1×)
	Rank 1	68.6%	2074.9 (3.25×)	3158 (1.14×)
	ACCORDION	92.9%	2945.1 (2.3 ×)	3220 (1.12 ×)
Senet	Rank 4	94.5%	4361.3 (1×)	4689 (1×)
	Rank 1	94.2%	1392.6 (3.1×)	4134 (1.13×)
	ACCORDION	94.5%	2264.4 (1.9 ×)	4298 (1.09 ×)

Table 2. ACCORDION with POWERSGD on CIFAR-100

Network	Rank	Accuracy	Data Sent (Million Floats)	Time (Seconds)
Resnet-18	Rank 2	71.7%	2426.3 (1×)	3521 (1×)
	Rank 1	70.0%	1355.7 (1.8×)	3388 (1.04×)
	ACCORDION	71.8%	1566.3 (1.6 ×)	3419 (1.03 ×)
DenseNet	Rank 2	72.0%	3387.4 (1×)	13613 (1×)
	Rank 1	71.6%	2155.6 (1.6×)	12977 (1.04×)
	ACCORDION	72.5%	2284.9 (1.5 ×)	13173 (1.03 ×)
Senet	Rank 2	72.5%	2878.1 (1×)	5217 (1×)
	Rank 1	71.5%	1683.1 (1.7×)	4994 (1.04×)
	ACCORDION	72.4%	2175.6 (1.3 ×)	5074 (1.03 ×)