Follow us
@icicleai

Stable Top-K: Exploiting Temporal Stability of Top-K Gradients

Zhao Zhang
Texas Advanced Computing Center

http://icicle.ai

Why Sparsity?

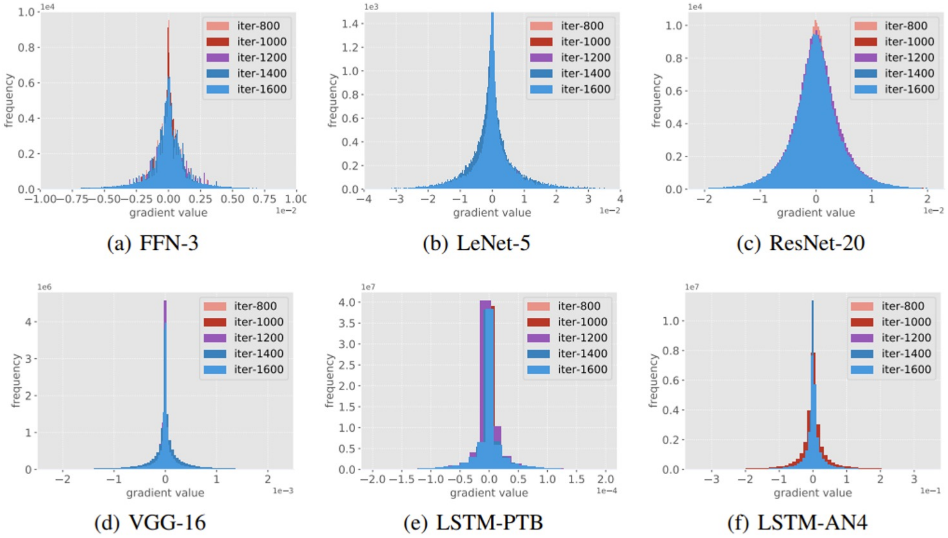
- **Communication latency scales with the size of gradient**
- **Size of the gradient scales with the size of the model parameters**
- Specifically,
  - In fp32, $\text{memory}_{\text{gradients}} = (4 \text{ bytes}/\text{param}) \cdot (\#\text{params})$
  - In fp16, $\text{memory}_{\text{gradients}} = (2 \text{ bytes}/\text{param}) \cdot (\#\text{params})$
- Large scale training require fp32 gradients, e.g., LARS and LAMB.
- Assuming fp32 gradients, the following models must communicate (every iteration):
  - BERT-Large: (4 bytes/param) * (345M params) = **1.28 GB**
  - GPT-NeoX 20B: (4 bytes/param) * (20B params) = **74 GB**
  - GPT-3 175B: (4 bytes/param) * (175B params) = **652 GB**

Why Sparsity?

- Communicating gradients requires expensive networking and bottlenecks training
- However, gradient values are noisy, and **most values are near zero**

- Only the gradients with large magnitudes matter for training convergence

- How many gradient values can be removed before convergence is affected? 90%, 99%, 99.9%?



SGD gradient distributions from: https://arxiv.org/pdf/1911.08772.pdf

What is Sparsity?

- Reduces communication volume by only propagating some gradient elements
- Compressor function $Comp_k$ (e.g. Top_K or $Rand_K$) keeps only *k* gradient elements, and sums + stores the remaining values as a 'residual' (ε) for the next iteration

Standard SGD:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta_t \frac{1}{P} \sum_{p=1}^{P} \boldsymbol{g}_t^p ,$$

Sparse SGD:

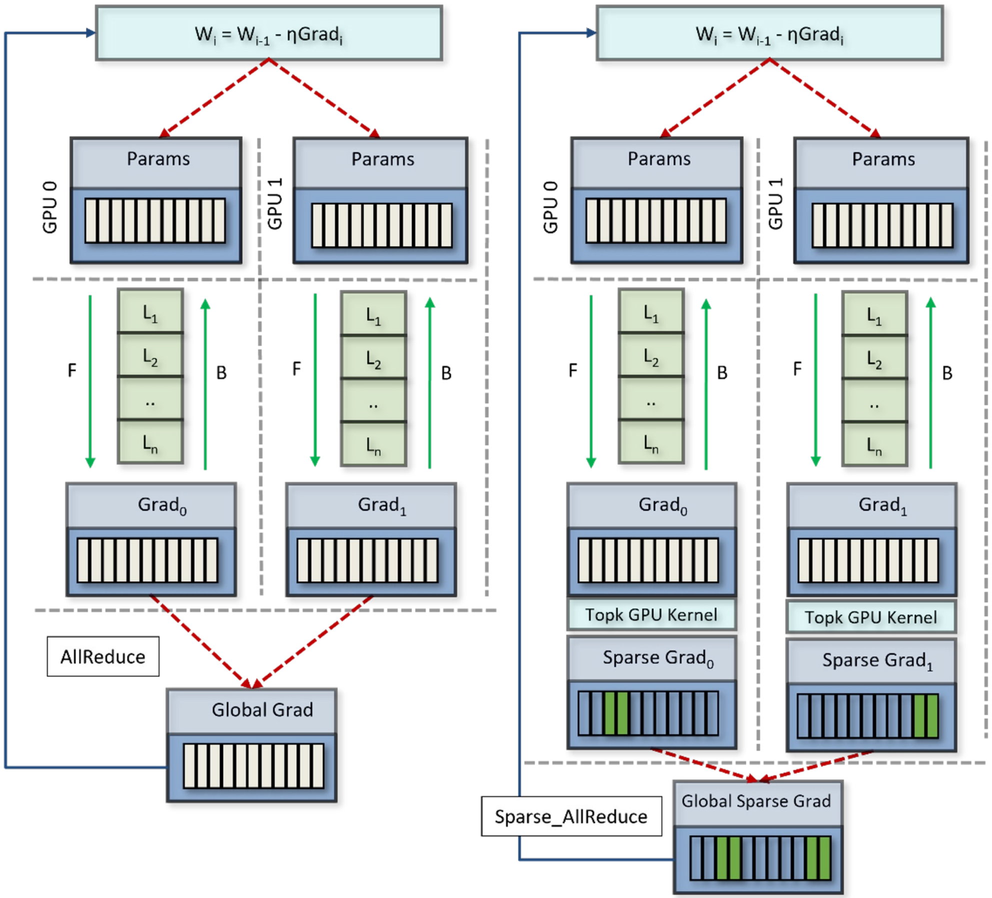
$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta_t \frac{1}{P} \sum_{p=1}^{P} Comp_k(\boldsymbol{g}_t^p + \boldsymbol{\epsilon}_t^p)$$

Top_k Sparsity

- The Top_k compressor function selects the top *k* largest elements (in terms of magnitudes) of the gradient and accumulates for all other elements.
- Top_k has commonly been implemented at the Python layer (except for SparCML), and has been added to native PyTorch
- Convergence has been proven and demonstrated for many model types, but requires careful hyperparameter tuning

Top_k Sparsity

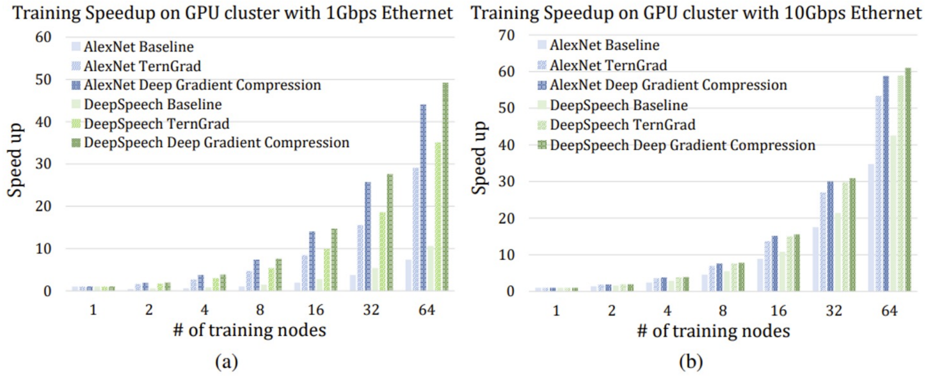
- In dense data-parallel training, the full gradients are averaged across all workers via an AllReduce operation
- TopK sparsity works by applying a sparsificiation GPU kernel on each worker, then communicating the positions and topk values via a Sparse_AllReduce operation



Previous SOTA: Deep Gradient Compression

- Attempts to resolve the hyperparameter tuning problem by:
  - Topk sparsification of gradients
  - Modify the optimizer and gradient update rules to correct sparsity's convergence effects. Use this to push sparsity to 99.9%

- DGC doesn't help much when interconnect is fast →
  - High GPU cost in selecting gradients
  - **DGC is not scalable**



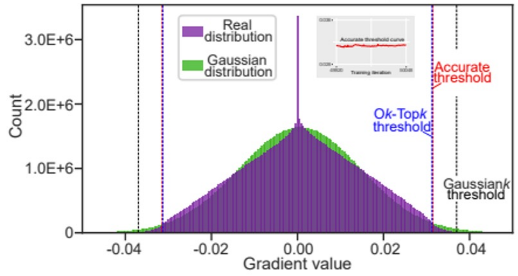
Current SOTA: OkTop_k Sparsity

- Sparsification overhead scales with number of processes *P*
  - (Cost of sending message of size L) = α + βL
  - Where (α = Latency) and (β = Bandwidth)
- **Dense**: Standard Allreduce
- **Top*k*A**: Allgather + local sparse reduction
- **Top*k*DSA**: SparCML's sparse reduce-scatter + allgather
- **gTop*k***: reduction tree + broadcast tree
- **Gaussian*k***: Same as TopkA with gaussian fitting
- **Ok-Top*k***: Split buffers via isend/irecv, sparse reduction, allgatherv

Table 1. Communication overhead of dense and sparse allreduces (n is the number of gradient components and $n \gg k$)

Algorithms	Bandwidth	Latency
Dense [12]	$2n\frac{P-1}{P}\beta$	$2(\log P)\alpha$
Top*k*A [36, 47]	$2k(P\text{-}1)\beta$	$(\log P)\alpha$
Top*k*DSA [36]	$[4k\frac{P-1}{P}\beta, (2k+n)\frac{P-1}{P}\beta]^1$	$(P + 2\log P)\alpha$
gTop*k* [42]	$4k(\log P)\beta$	$2(\log P)\alpha$
Gaussian*k* [41]	$2k(P\text{-}1)\beta$	$2(\log P)\alpha$
O*k*-Top*k*	$[2k\frac{P-1}{P}\beta, 6k\frac{P-1}{P}\beta]^1$	$(2P + 2\log P)\alpha$

Secondary result: BERT Gradients are also sparse ⟶



(c) BERT [13] on Wikipedia [13] with density = 1.0%.

Shortcomings of Current SOTA

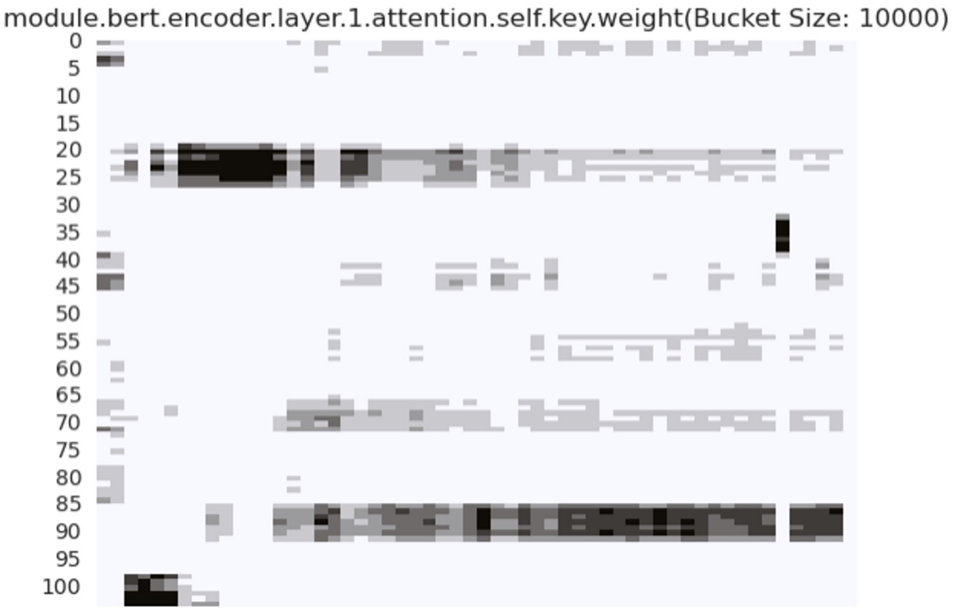
- While OkTopk is scalable, it hurts convergence.
- Language models have two measures of accuracy:
  - **Training (perplexity) loss:** Accuracy while the model is training on general language data
  - **Downstream evaluations:** Effectiveness on the model on specific tasks (e.g. Q&A)
- While the OkTopk paper demonstrated reasonable training loss, our experiments show poor downstream evaluation accuracy

Model	SQuAD	GLUE
BERT-Large (Baseline)	90.40	0.802
BERT-Large (OkTopK)	88.10	0.770

- **We seek to find a gradient sparsity scheme that's scalable and preserves downstream task accuracy**

Stable TopK

- We hypothesize that gradient elements are temporally stable, since:
  - Pre-training should lead to the creation of circuits that are comprised of nearby neurons
  - Such circuits should gradually adapt over many training iterations
- We find this hypothesis to be true for CNNs (ResNet-50) and transformers (BERT-Large, OpenFold, and ViT)



Stable TopK

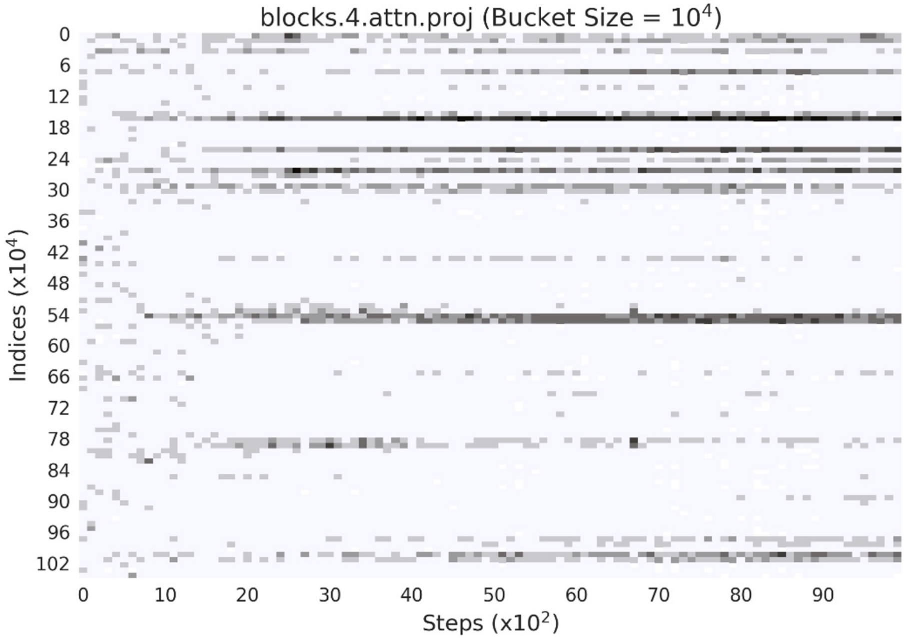
- While regions are stable, individual gradient positions are not (see figures below for **BERT**)
- Such behavior necessitates the use of a "TopK bucket" instead of specific elements
- These insights introduces the key idea of our work: **Instead of communicating the exact TopK elements every iteration, only communicate TopK buckets every N iterations**

Bucketed Indices

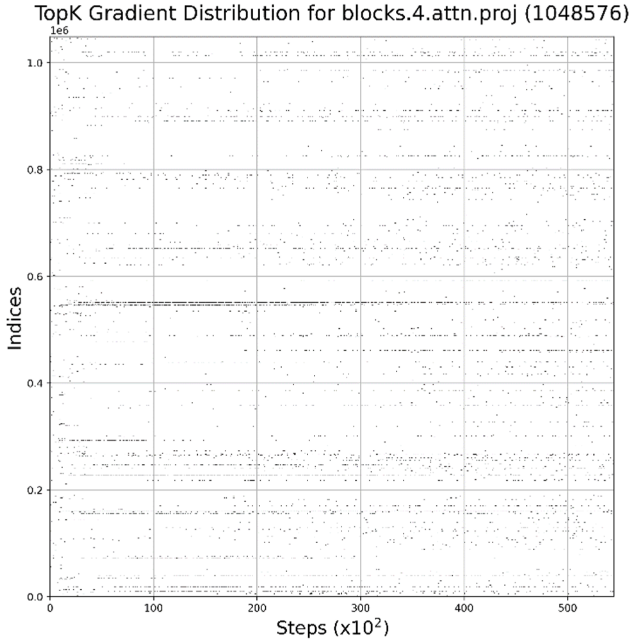
Individual Indices

Stable TopK

- Same insights hold for masked autoencoder (MAE) vision models (see below)



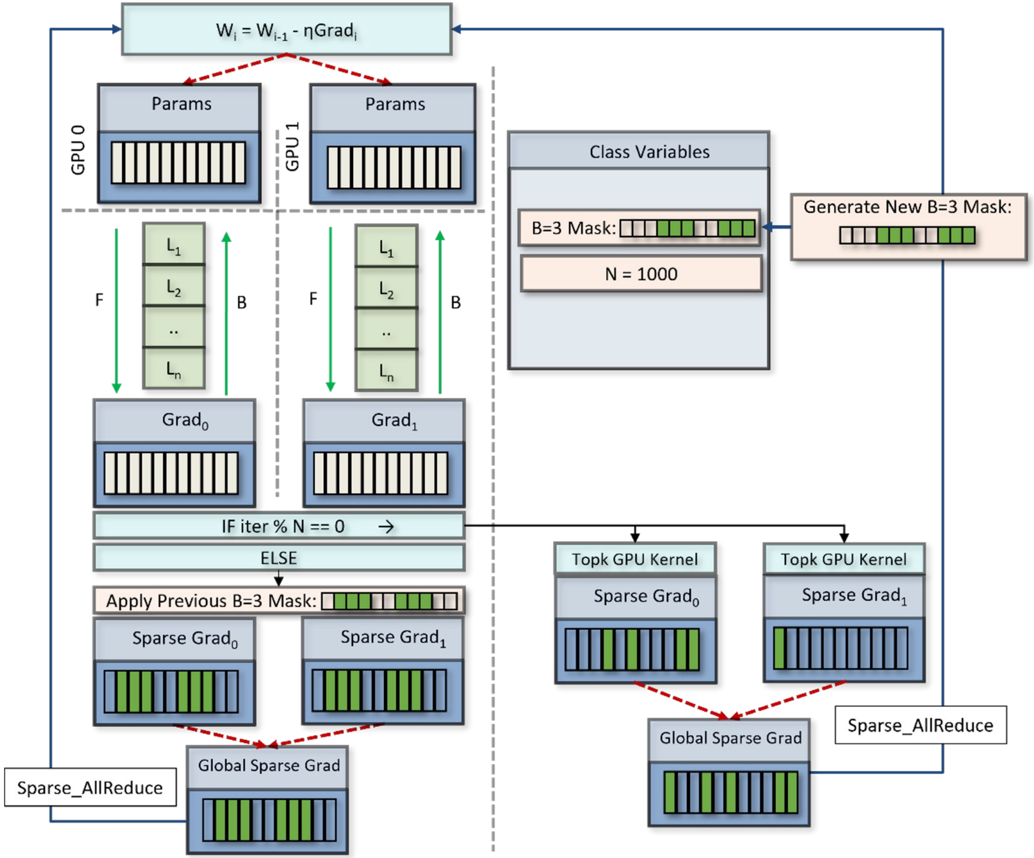
Bucketed Indices



Individual Indices

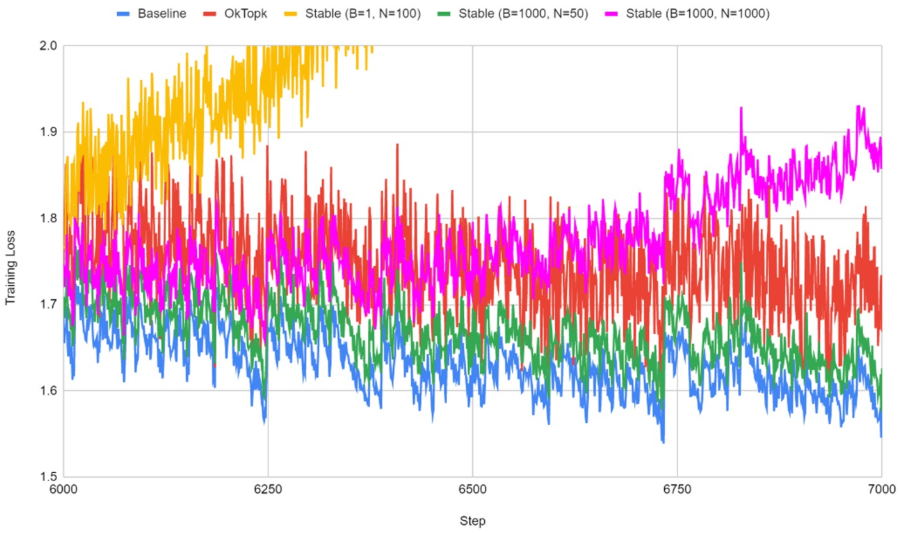
Stable TopK

- In Stable TopK, the sparsification kernel is only applied every N iterations
- Communicate buckets of indices and their values instead of specific indices
- For all other iterations, simply apply the bucketed mask from the last recomputation



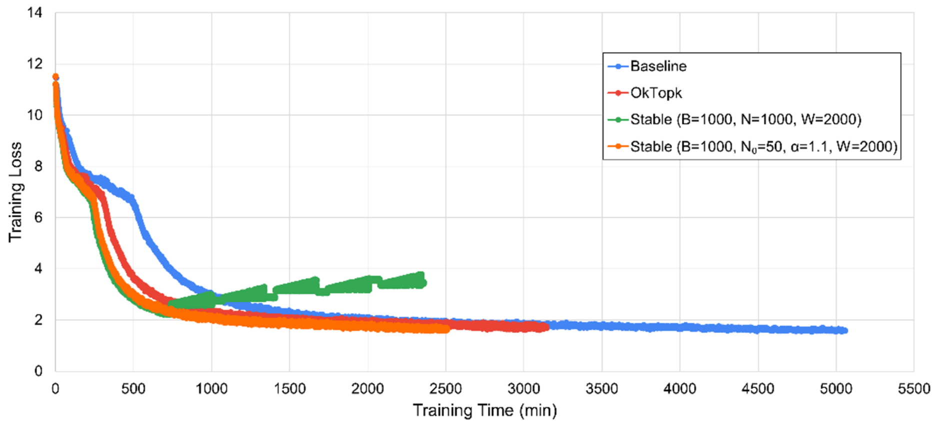
Stable TopK

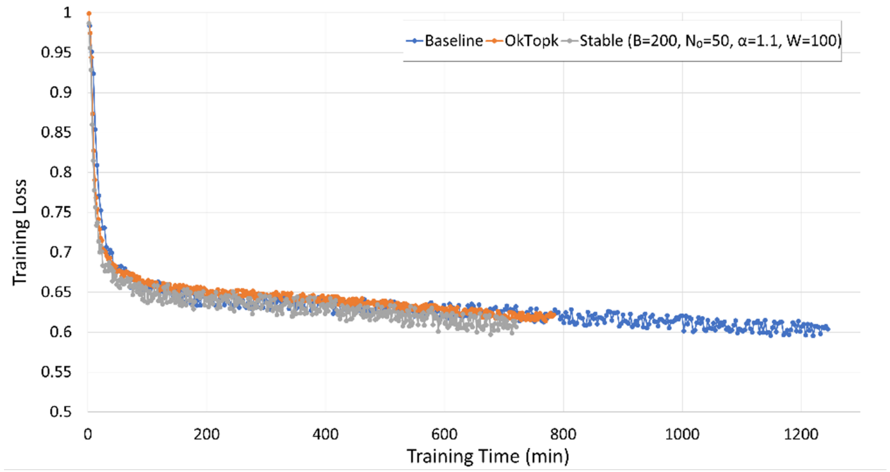
- By varying the S-TopK bucket size and sample frequency, we gain some valuable insights into the stable scheme
- **If S-TopK bucket size is too small**, the model quickly diverges because individual positions are not stable
- **If S-TopK sample frequency is too high**, the stable region may decay before the S-TopK bucket indices are updated
- If the bucket size and sample frequencies are chosen correctly, S-TopK nearly matches baseline loss



Stable TopK

- For both BERT (top) and MAE (bottom), stable TopK trains in the shortest time
- Again, new hyperparameters must be tuned to achieve convergence
- Higher values of N and lower values of B lead to lower sparsification and communication overheads, respectively





Stable TopK

- Since S-TopK doesn't compute the TopK indices every iteration, its throughput is higher than OkTopK
- In addition to maintaining a lower training loss than OkTopK, S-TopK preserves downstream evaluation performance

BERT-Large	SQuAD	GLUE	Time (hrs)
Baseline	90.4	0.802	84.3
Ok-TopK	88.10	0.770	52.4
S-TopK	89.96	0.802	41.8

MAE	ImageNet	Time (hrs)
Baseline	84.1%	20.3
Ok-TopK	81.3%	13.3
S-TopK	83.8%	11.0

Gradient Sparsification Summary and Future Work

- It’s challenging to ensure convergence for existing methods (e.g. OkTopK)
- Gradient indices **are not** stable over time, but regions of gradient elements **are** stable
- Stable TopK exploits this property by communicating sparse gradient regions periodically
- Stable TopK converges much closer to baseline than competing sparse methods in less time for both BERT and MAE
- Continuing on convergence

Thank you

zzhang@tacc.utexas.edu
zhao.zhang@rutgers.edu