## EECE695D: Efficient ML Systems

# Distributed Training

## (part 2)

# Recap: Data-Parallel Training

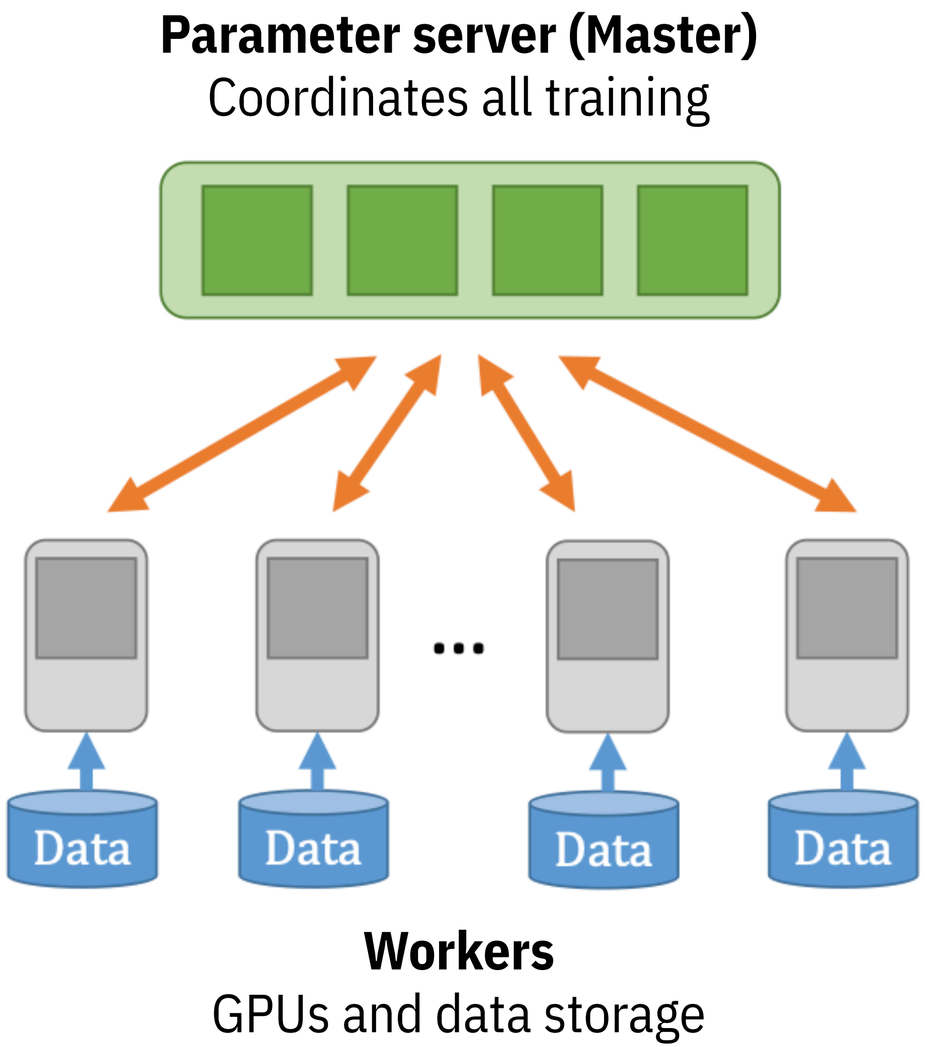
Training a model using multiple computing devices:

- Model parameters are shared over all devices.
- Dataset is split into multiple pieces.

**Pros:** Easy to realize.

**Cons:** Not too good for large models

High all-reduce overhead (bandwidth, rather than packet delay)



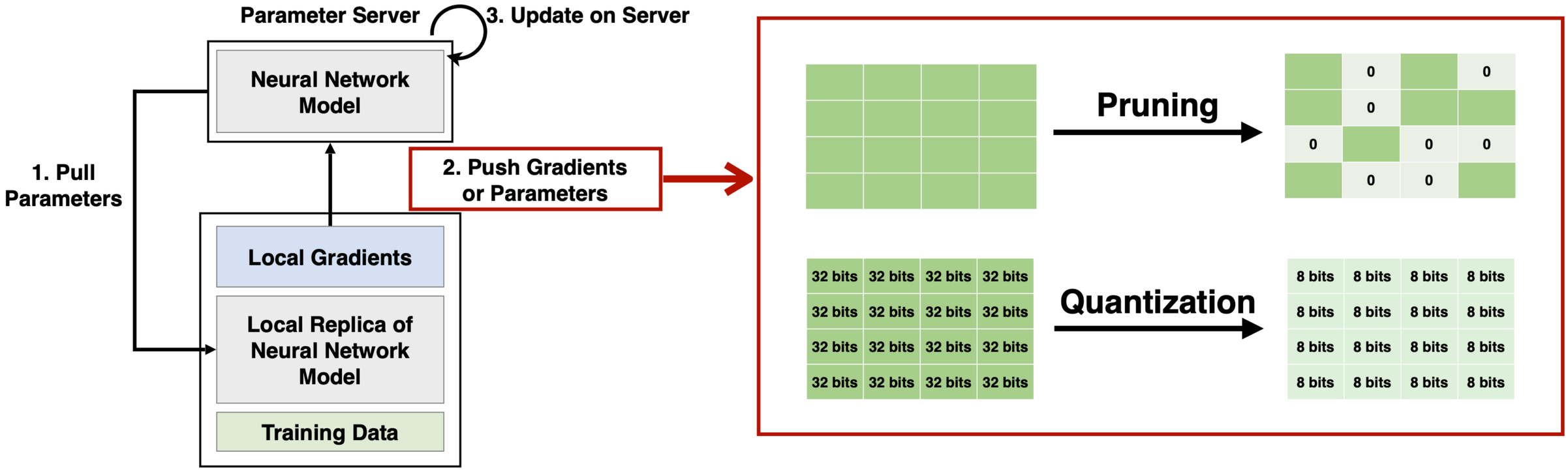

*Brainteaser.*

Suppose that we have $K$ devices.
Each device sends the gradient after a random delay, where the delay is i.i.d. with respect to some distribution with unit variance.

Then, the total delay would be $\sim \sqrt{\log K}$ ⋯ Gaussian.

$$\sim \sum_{i=1}^{K} \frac{1}{i} \approx \log K \quad \cdots \quad \text{Exponential}$$

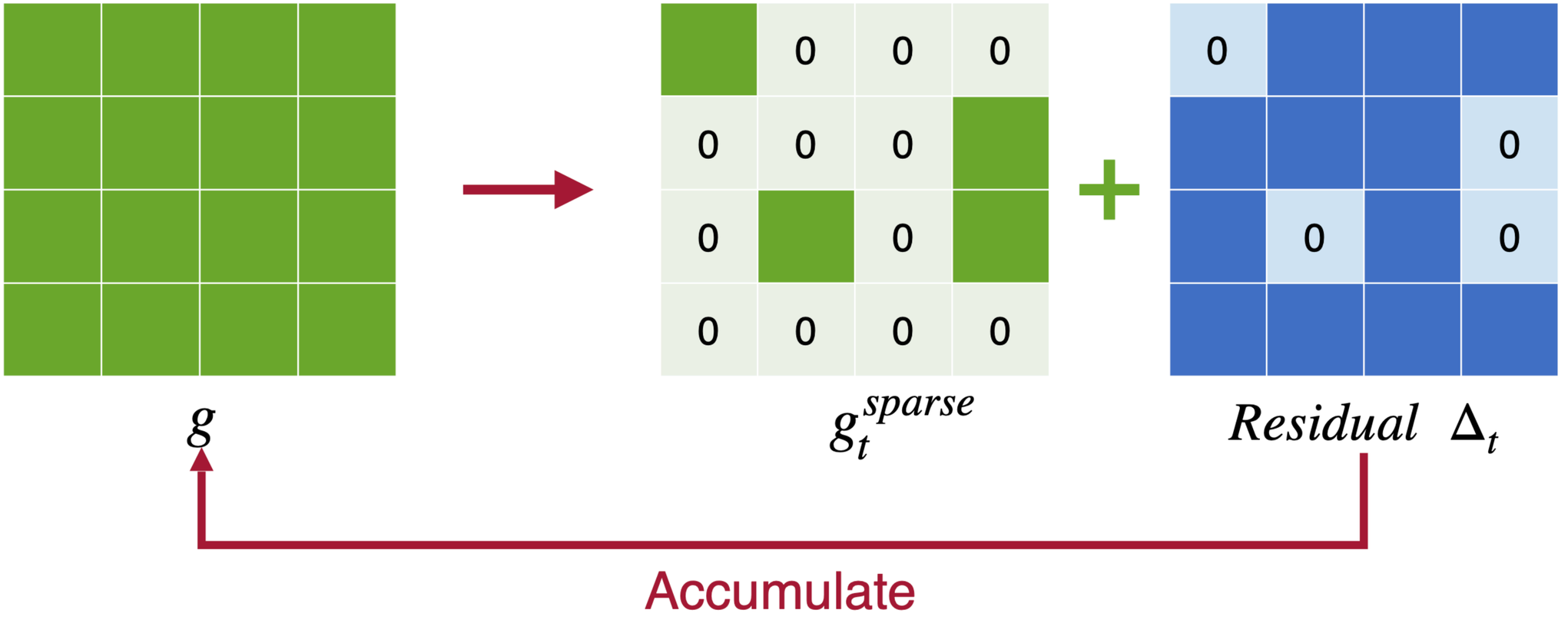
**Question.** Can we reduce the communication load?
**Idea.** Maybe we can use model compression techniques... (pruning / quantization)




**Note.** We are compressing the gradients only... but broadcasting parameters can be implemented efficiently.

**Sparse Communication.** Send gradients with top-k magnitude.
Also, accumulate the unpruned part to the gradient.

(Note: For momentum, accumulate real update, not the gradients itself)
(Note: For gradient clipping, apply it before adding it to the residual)
(Note: Warm up both learning rate and sparsity)



**Sparse Communication.** Send gradients with top-k magnitude.
Also, accumulate the unpruned part to the gradient.

| Task | | Baseline | Deep Gradient Compression |
|---|---|---|---|
| **ResNet-50 On ImageNet** | Top-1 Accuracy | 75.96% | **76.15% (+0.19%)** |
| | Top-5 Accuracy | 92.91% | **92.97% (+0.06%)** |
| | Gradient Compression Ratio | 1 × | **277 ×** |
| **5-Layer GRU On LibriSpeech** | Word Error Rate (WER) | 9.45% | **9.06% (-0.39%)** |
| | Word Error Rate (WER) | 27.07% | **27.04% (-0.03%)** |
| | Gradient Compression Ratio | 1 × | **608 ×** |
| **2-Layer LSTM Language Model On Penn Treebank** | Perplexity | 72.30 | **72.24 (-0.06)** |
| | Gradient Compression Ratio | 1 × | **462 ×** |

**Figure 10: Deep Gradient Compression can compress the gradient exchange by 277× to 608× without losing any accuracy**

*Lin et al., "Deep gradient compression: Reducing the communication bandwidth for distributed training" ICLR 2018.*

**Sparse Communication.** Send gradients with top-k magnitude.
Also, accumulate the unpruned part to the gradient.

## Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training

Yujun Lin, Song Han, Huizi Mao, Yu Wang, William J. Dally

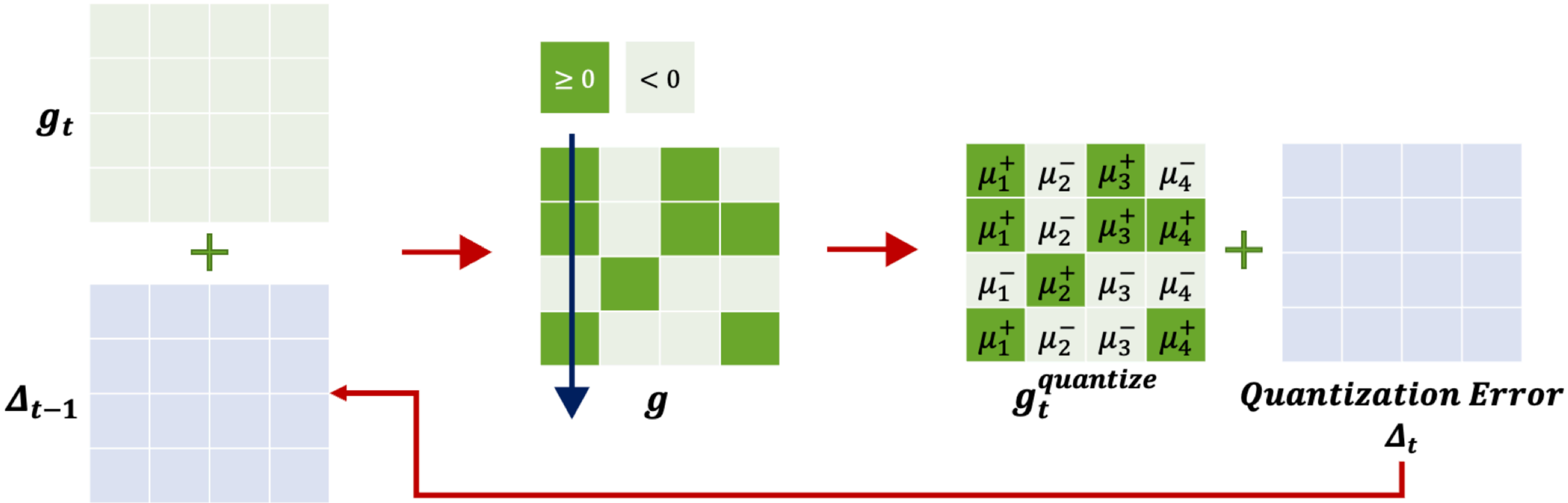
Large-scale distributed training requires significant communication bandwidth for gradient exchange that limits the scalability of multi-node training, and requires expensive high-bandwidth network infrastructure. The situation gets even worse with distributed training on mobile devices (federated learning), which suffers from higher latency, lower throughput, and intermittent poor connections. In this paper, we find 99.9% of the gradient exchange in distributed SGD is redundant, and propose Deep Gradient Compression (DGC) to greatly reduce the communication bandwidth. To preserve accuracy during compression, DGC employs four methods: momentum correction, local gradient clipping, momentum factor masking, and warm-up training. We have applied Deep Gradient Compression to image classification, speech recognition, and language modeling with multiple datasets including Cifar10, ImageNet, Penn Treebank, and Librispeech Corpus. On these scenarios, Deep Gradient Compression achieves a gradient compression ratio from 270x to 600x without losing accuracy, cutting the gradient size of ResNet-50 from 97MB to 0.35MB, and for DeepSpeech from 488MB to 0.74MB. Deep gradient compression enables large-scale distributed training on inexpensive commodity 1Gbps Ethernet and facilitates distributed training on mobile. Code is available at: this https URL.

Comments: we find 99.9% of the gradient exchange in distributed SGD is redundant; we reduce the communication bandwidth by two orders of magnitude without losing accuracy. Code is available at: this https URL

*Lin et al., "Deep gradient compression: Reducing the communication bandwidth for distributed training" ICLR 2018.*

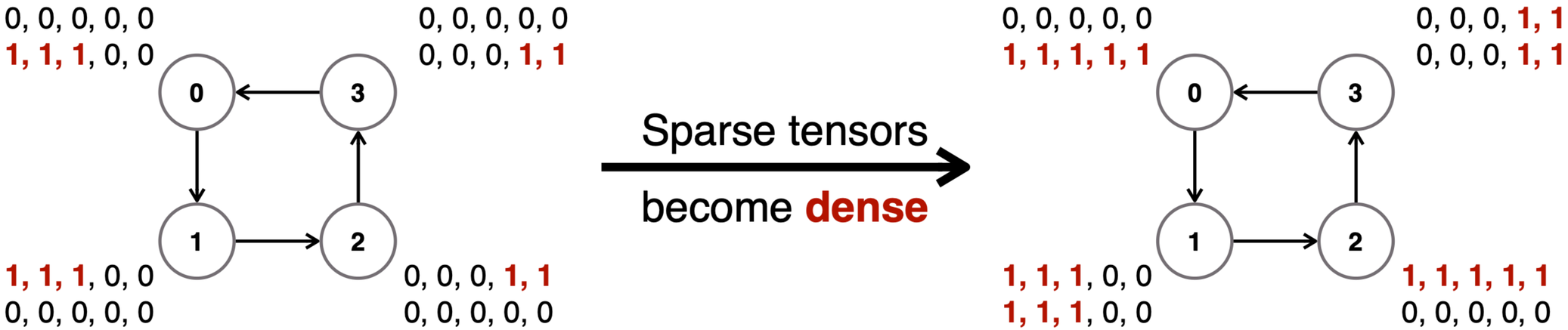
**Quantized Communication.** Similar idea, but with quantization.

**1-bit SGD.** (1) Column-wise scaling factor
(2) Accumulate quantization error




*Seide et al., "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs" Interspeech 2014.*

**Problem.** If we use allreduce, the gradients become dense!




We can use quantization, but it also has similar issues… (do you see why?)

*Vogels et al., "PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization" NeurIPS 2019.*

**Alternative.** Use low-rank approximation instead of sparse one (native in PyTorch as well).

**Algorithm 1** Rank-$r$ POWERSGD compression

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.

**function** COMPRESS+AGGREGATE(update matrix $M \in \mathbb{R}^{n \times m}$, previous $Q \in \mathbb{R}^{m \times r}$)

$P \leftarrow MQ$

$P \leftarrow$ ALL REDUCE MEAN$(P)$ ▷ Now, $P = \frac{1}{W}(M_1 + \ldots + M_W)Q$

$\hat{P} \leftarrow$ ORTHOGONALIZE$(P)$ ▷ Orthonormal columns

$Q \leftarrow M^\top \hat{P}$

$Q \leftarrow$ ALL REDUCE MEAN$(Q)$ ▷ Now, $Q = \frac{1}{W}(M_1 + \ldots + M_W)^\top \hat{P}$

**return** the compressed representation $(\hat{P}, Q)$.

**end function**

**function** DECOMPRESS($\hat{P} \in \mathbb{R}^{n \times r}, Q \in \mathbb{R}^{m \times r}$)

**return** $\hat{P}Q^\top$

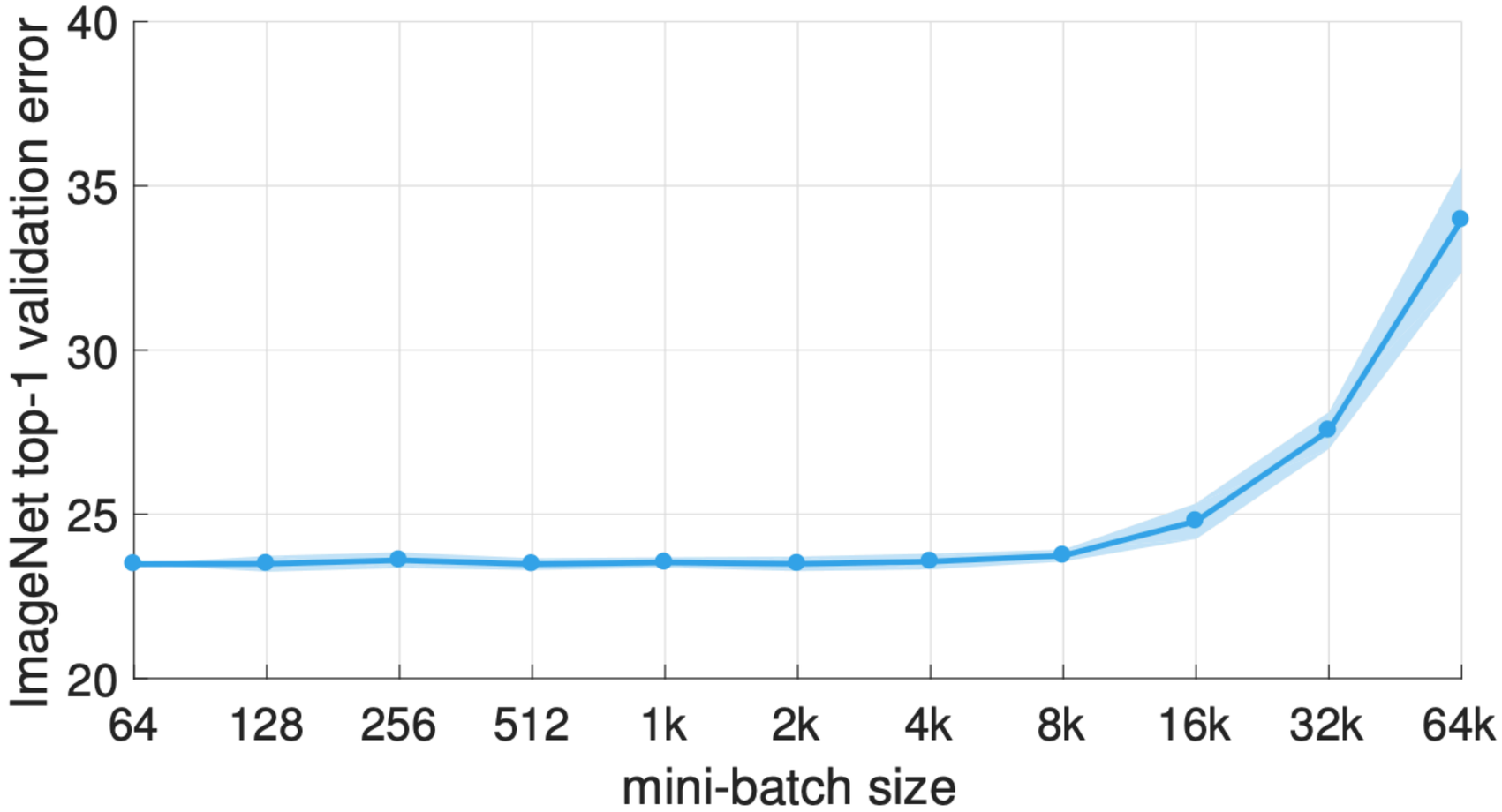
**end function**

$$b_{k+1} = \frac{Ab_k}{\|Ab_k\|}$$

*Vogels et al., "PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization" NeurIPS 2019.*

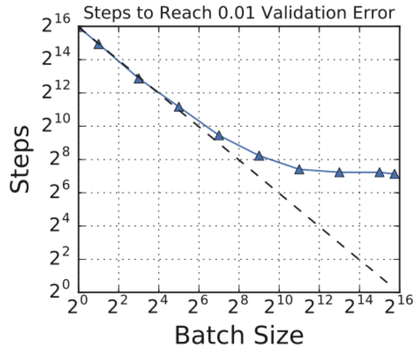
**Question.** Suppose that we can handle delay well...
Can we increase the batch size infinitely to speed up training?

**Observation.** If we do so, we somehow lose generalizability over a certain threshold.

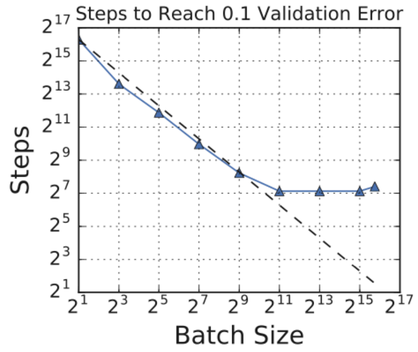



**Note.** Learning rate should grow linearly as the batch size increases (and also have some warmup phase)
**Note.** They use ring-allreduce and recursive halving

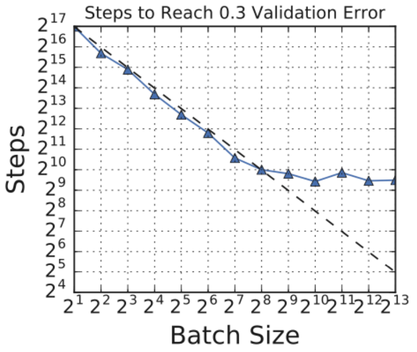
*Goyal et al., "Accurate, large mini-batch SGD: Training ImageNet in 1 hour" arXiv 2017.*



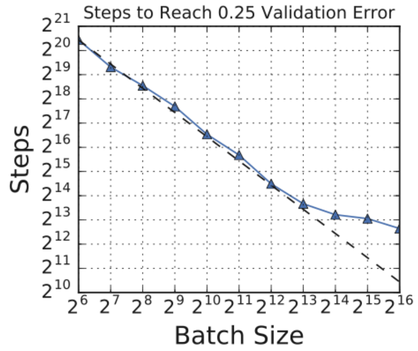

(a) Simple CNN on MNIST



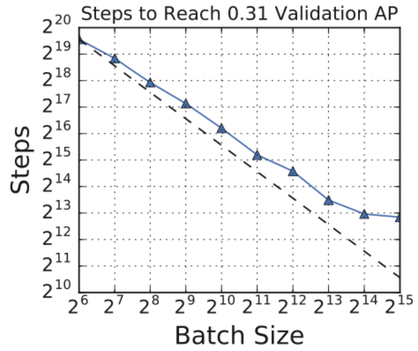

(b) Simple CNN on Fashion MNIST



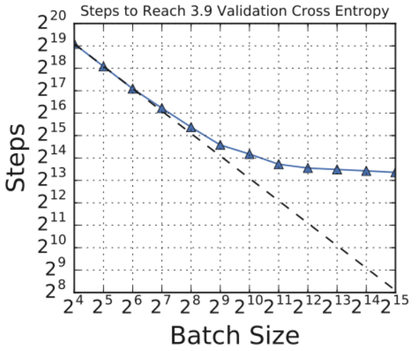

(c) ResNet-8 on CIFAR-10



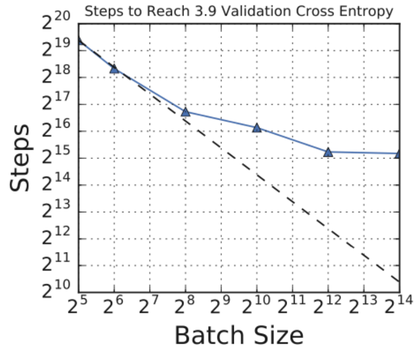

(d) ResNet-50 on ImageNet



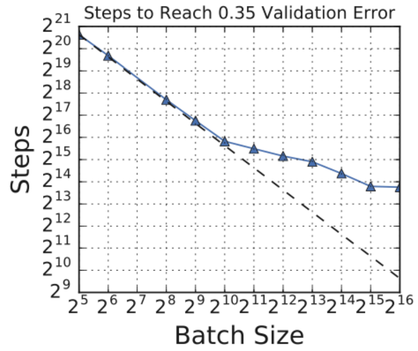

(e) ResNet-50 on Open Images



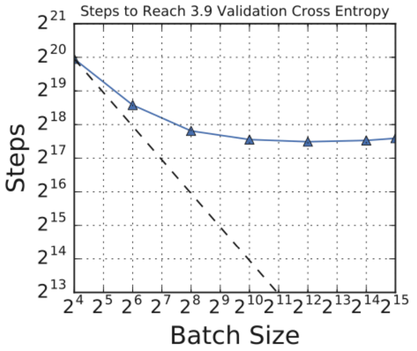

(f) Transformer on LM1B




(g) Transformer on Common Crawl



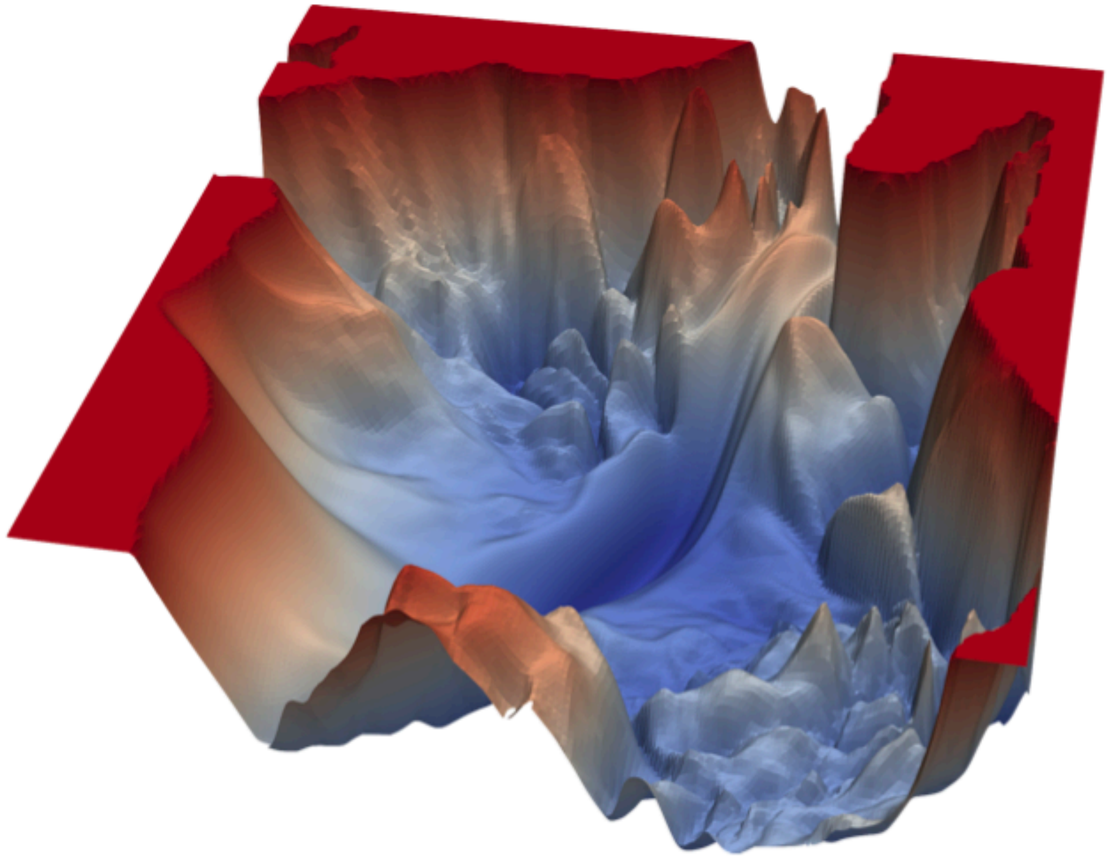

(h) VGG-11 on ImageNet




(i) LSTM on LM1B

*Shallue et al., "Measuring the effects of data parallelism on neural network training" JMLR 2019.*

**Why?** People suspect that large batch training reduces the SGD variance, which leads you to get trapped at a very narrow valley.



**Why?** Motivated some optimization algorithms, such as LARS/LAMB/SAM

*You et al., "Large batch optimization for deep learning: Training BERT in 76 minutes" ICLR 2020.*
*Foret et al., "Sharpness-aware minimization for effciently improving generalization" ICLR 2021.*

Empirically, this notion of “sharpness” is better correlated with the generalization performance than other complexity measures (e.g., VC-dimension, Rademacher complexity, Norm-based)

| | | batch size | dropout | learning rate | depth | optimizer | weight decay | width | overall $\tau$ | $\Psi$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Corr | vc dim 19 | 0.000 | 0.000 | 0.000 | -0.909 | 0.000 | 0.000 | -0.171 | -0.251 | -0.154 |
| | # params 20 | 0.000 | 0.000 | 0.000 | -0.909 | 0.000 | 0.000 | -0.171 | -0.175 | -0.154 |
| | $1/\gamma$ (22) | 0.312 | -0.593 | 0.234 | 0.758 | 0.223 | -0.211 | 0.125 | 0.124 | 0.121 |
| | entropy 23 | 0.346 | -0.529 | 0.251 | 0.632 | 0.220 | -0.157 | 0.104 | 0.148 | 0.124 |
| | cross-entropy 21 | 0.440 | -0.402 | 0.140 | 0.390 | 0.149 | 0.232 | 0.080 | 0.149 | 0.147 |
| | oracle 0.02 | 0.380 | 0.657 | 0.536 | 0.717 | 0.374 | 0.388 | 0.360 | 0.714 | 0.487 |
| | oracle 0.05 | 0.172 | 0.375 | 0.305 | 0.384 | 0.165 | 0.184 | 0.204 | 0.438 | 0.256 |
| | canonical ordering | 0.652 | 0.969 | 0.733 | 0.909 | -0.055 | 0.735 | 0.171 | N/A | N/A |
| | | | | | | | | | $\lvert\mathcal{S}\rvert = 2$ | $\min \forall \lvert\mathcal{S}\rvert$ |
| MI | vc dim | 0.0422 | 0.0564 | 0.0518 | 0.0039 | 0.0422 | 0.0443 | 0.0627 | 0.00 | 0.00 |
| | # param | 0.0202 | 0.0278 | 0.0259 | 0.0044 | 0.0208 | 0.0216 | 0.0379 | 0.00 | 0.00 |
| | $1/\gamma$ | 0.0108 | 0.0078 | 0.0133 | 0.0750 | 0.0105 | 0.0119 | 0.0183 | 0.0051 | 0.0051 |
| | entropy | 0.0120 | 0.0656 | 0.0113 | 0.0086 | 0.0120 | 0.0155 | 0.0125 | 0.0065 | 0.0065 |
| | cross-entropy | 0.0233 | 0.0850 | 0.0118 | 0.0075 | 0.0159 | 0.0119 | 0.0183 | 0.0040 | 0.0040 |
| | oracle 0.02 | 0.4077 | 0.3557 | 0.3929 | 0.3612 | 0.4124 | 0.4057 | 0.4154 | 0.1637 | 0.1637 |
| | oracle 0.05 | 0.1475 | 0.1167 | 0.1369 | 0.1241 | 0.1515 | 0.1469 | 0.1535 | 0.0503 | 0.0503 |
| | random | 0.0005 | 0.0002 | 0.0005 | 0.0002 | 0.0003 | 0.0006 | 0.0009 | 0.0004 | 0.0001 |

Table 1: Numerical Results for Baselines and Oracular Complexity Measures

*Jiang et al., “Fantastic generalization measures and where to find them” ICLR 2020.*

| | | batch size | dropout | learning rate | depth | optimizer | weight decay | width | overall $\tau$ | $\Psi$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Corr | Frob distance 40 | -0.317 | -0.833 | -0.718 | 0.526 | -0.214 | -0.669 | -0.166 | -0.263 | -0.341 |
| | Spectral orig 26 | -0.262 | -0.762 | -0.665 | -0.908 | -0.131 | -0.073 | -0.240 | -0.537 | -0.434 |
| | Parameter norm 42 | 0.236 | -0.516 | 0.174 | 0.330 | **0.187** | 0.124 | -0.170 | 0.073 | 0.052 |
| | Path norm 44 | 0.252 | **0.270** | 0.049 | **0.934** | 0.153 | 0.338 | **0.178** | **0.373** | **0.311** |
| | Fisher-Rao 45 | **0.396** | 0.147 | **0.240** | -0.553 | 0.120 | **0.551** | 0.177 | 0.078 | 0.154 |
| | oracle 0.02 | 0.380 | 0.657 | 0.536 | 0.717 | 0.374 | 0.388 | 0.360 | 0.714 | 0.487 |
| | | | | | | | | | $\lvert\mathcal{S}\rvert = 2$ | $\min \forall \lvert\mathcal{S}\rvert$ |
| MI | Frob distance | 0.0462 | 0.0530 | 0.0196 | **0.1559** | 0.0502 | 0.0379 | 0.0506 | 0.0128 | 0.0128 |
| | Spectral orig | **0.2197** | **0.2815** | **0.2045** | 0.0808 | **0.2180** | **0.2285** | **0.2181** | **0.0359** | **0.0359** |
| | Parameter norm | 0.0039 | 0.0197 | 0.0066 | 0.0115 | 0.0064 | 0.0049 | 0.0167 | 0.0047 | 0.0038 |
| | Path norm | 0.1027 | 0.1230 | 0.1308 | 0.0315 | 0.1056 | 0.1028 | 0.1160 | 0.0240 | 0.0240 |
| | Fisher Rao | 0.0060 | 0.0072 | 0.0020 | 0.0713 | 0.0057 | 0.0014 | 0.0071 | 0.0018 | 0.0013 |
| | oracle 0.05 | 0.1475 | 0.1167 | 0.1369 | 0.1241 | 0.1515 | 0.1469 | 0.1535 | 0.0503 | 0.0503 |

Table 2: Numerical Results for Selected (Norm & Margin)-Based Complexity Measures

*Jiang et al., "Fantastic generalization measures and where to find them" ICLR 2020.*

| | | batch size | dropout | learning rate | depth | optimizer | weight decay | width | overall $\tau$ | $\Psi$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Corr | sharpness-orig 52 | 0.542 | -0.359 | 0.716 | 0.816 | 0.297 | 0.591 | 0.185 | 0.400 | 0.398 |
| | pacbayes-orig 49 | 0.526 | -0.076 | 0.705 | 0.546 | **0.341** | 0.564 | -0.086 | 0.293 | 0.360 |
| | $1/\alpha'$ sharpness mag 62 | **0.570** | **0.148** | **0.762** | 0.824 | 0.297 | **0.741** | **0.269** | **0.484** | **0.516** |
| | $1/\sigma'$ pacbayes mag 61 | 0.490 | -0.215 | 0.505 | **0.896** | 0.186 | 0.147 | 0.195 | 0.365 | 0.315 |
| | oracle 0.02 | 0.380 | 0.657 | 0.536 | 0.717 | 0.374 | 0.388 | 0.360 | 0.714 | 0.487 |
| | | | | | | | | | $\lvert\mathcal{S}\rvert = 2$ | $\min \forall \lvert\mathcal{S}\rvert$ |
| MI | sharpness-orig | 0.1117 | 0.2353 | 0.0809 | 0.0658 | 0.1223 | 0.1071 | 0.1254 | 0.0224 | 0.0224 |
| | pacbayes-orig | 0.0620 | 0.1071 | 0.0392 | 0.0597 | 0.0645 | 0.0550 | 0.0977 | 0.0225 | 0.0225 |
| | $1/\alpha'$ sharpness mag | **0.1640** | **0.2572** | **0.1228** | **0.1424** | **0.1779** | **0.1562** | **0.1786** | **0.0544** | **0.0544** |
| | $1/\sigma'$ pacbayes mag | 0.0884 | 0.1514 | 0.0813 | 0.0399 | 0.1004 | 0.1025 | 0.0986 | 0.0241 | 0.0241 |
| | oracle 0.05 | 0.1475 | 0.1167 | 0.1369 | 0.1241 | 0.1515 | 0.1469 | 0.1535 | 0.0503 | 0.0503 |

Table 3: Numerical results for selected Sharpness-Based Measures; all the measure use the origin as the reference and `mag` refers to magnitude-aware version of the measure.

Motivated some optimization algorithms, such as LAMB/SAM

*You et al., "Large batch optimization for deep learning: Training BERT in 76 minutes" ICLR 2020.*
*Foret et al., "Sharpness-aware minimization for effciently improving generalization" ICLR 2021.*