Communication-efficient Distributed SGD with Sketching

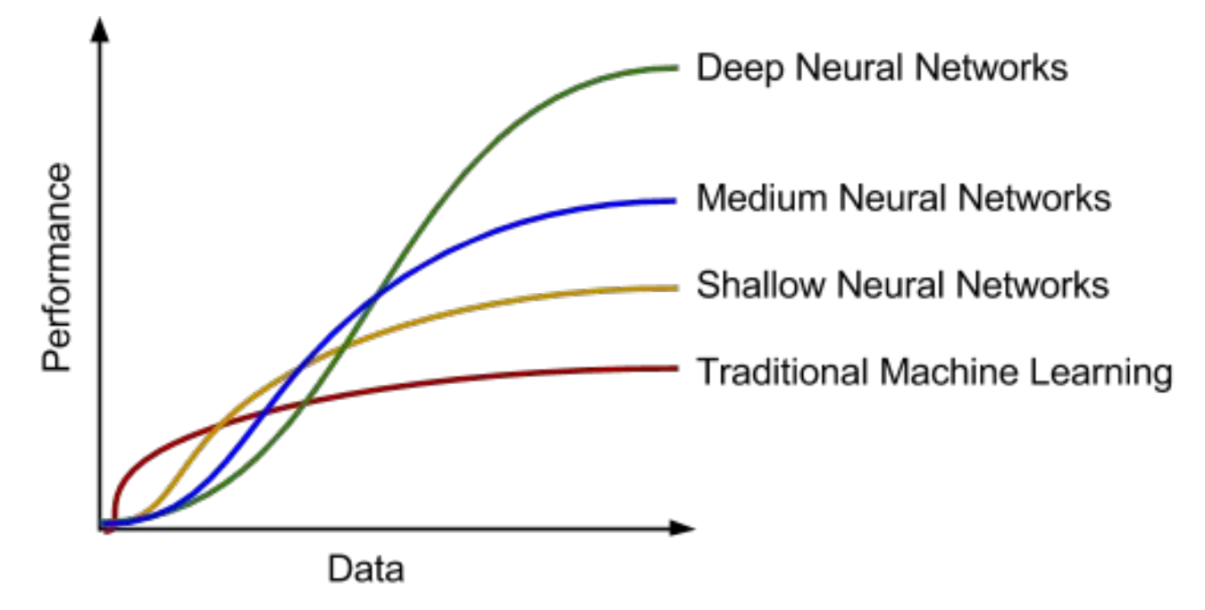
Nikita Ivkin*, Daniel Rothchild*, Enayat Ullah*, Vladimir Braverman, Ion Stoica, Raman Arora

*equal contribution

1. Introduction

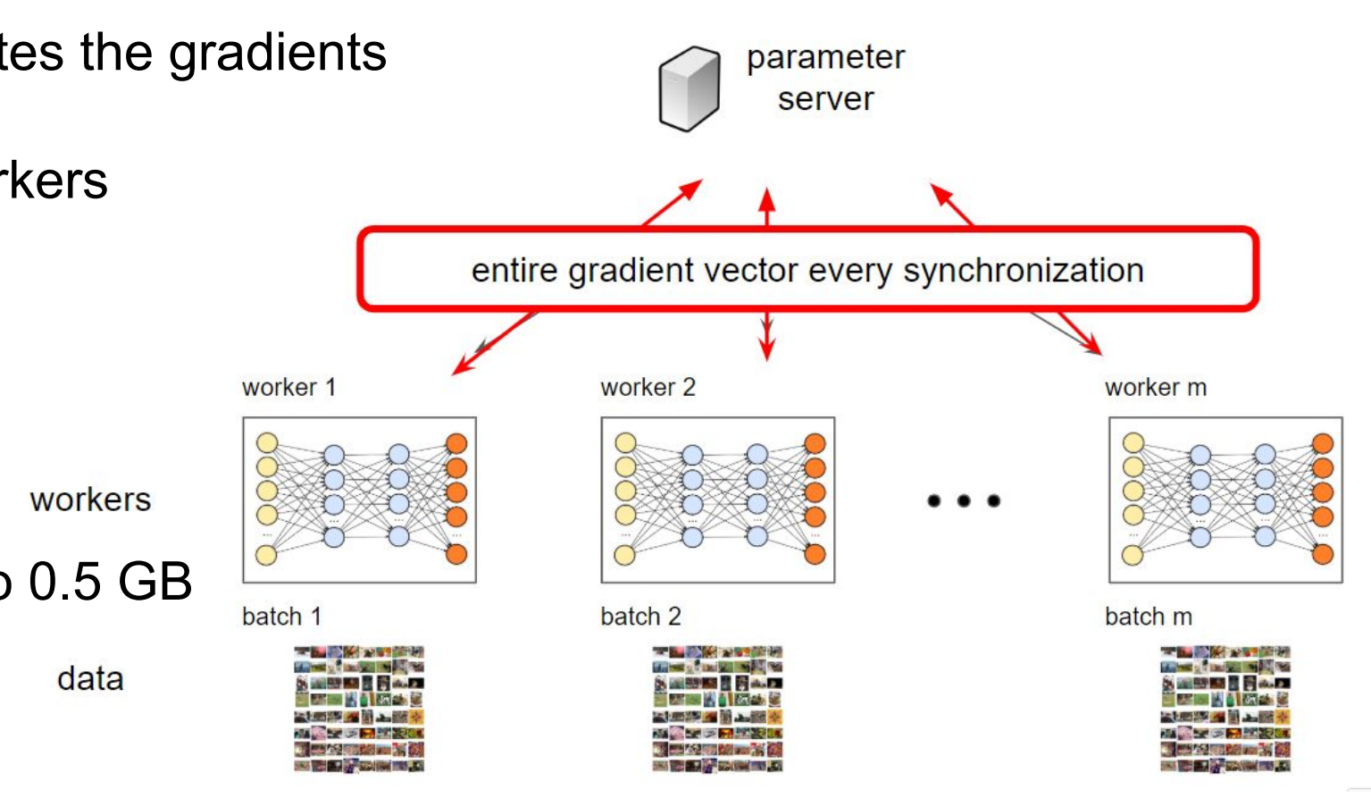
Going distributed: why?

Large scale machine learning is moving to the distributed setting due to growing size of datasets/models, and modern learning paradigms like Federated learning.



Synchronous SGD: how?

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes the gradients
- workers send gradients to parameter server
- parameter server sums it up and sends it back to all workers
- each worker makes a step



Bottleneck: where?

Slow communication overwhelms local computations:

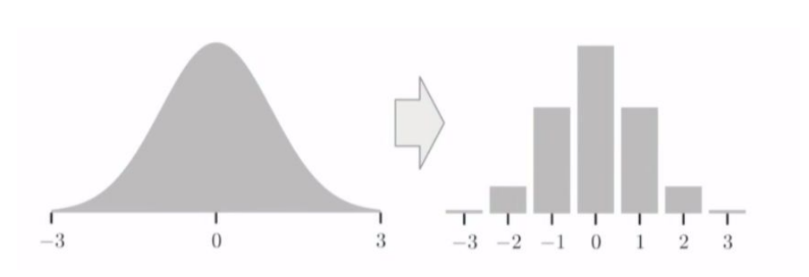
- parameter vector for large models can weight up to 0.5 GB
- synchronize every fraction of a second
- mini batch size has limit to its growth

$\downarrow$

computation resources are wasted

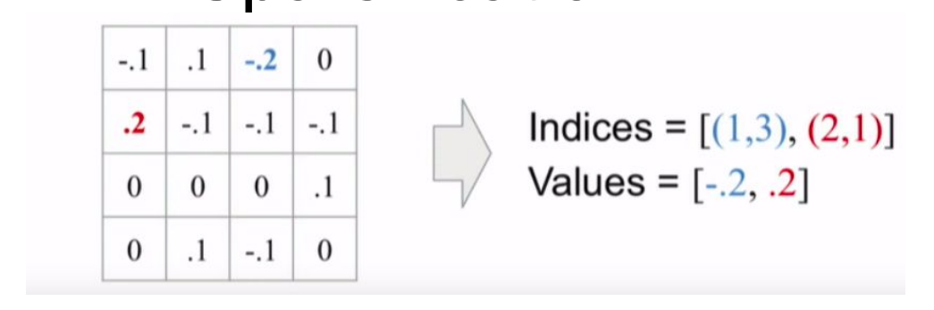
Common approaches:

Quantization



- Quantizing gradients can give a constant factor decrease in communication cost.
- Simplest quantization is 16-bit, however even 2-bit (TernGrad) and 1-bit (SignSGD) have been successful.
- Quantization techniques can in principle be combined with gradient sparsification

Sparsification



- Existing techniques either communicate Ω(Wd) in the worst case, or are heuristics; W - number of workers, d - dimension of gradient.
- Stich et al.'18 showed that SGD (on 1 machine) with top-*k* gradient updates and *error accumulation* has desirable convergence properties.
- Alistarh et al. '18 Top-k SGD (assumes that global top k is close to sum of local top k)
- Deep gradient compression (no theoretical guarantees)

2. Our Contribution

Adopted sketching based compression technique:

We introduce SKETCHED-SGD, an algorithm for carrying out distributed SGD by communicating sketches instead of full gradients

Theoretical guarantees:

Converges at *O(1/T)* rate, at par with SGD for smooth strongly convex functions.
Communicates *O(k log² d)*, size of sketch, *0< k < d*, *d*: dimension of model.

Scalability:

More workers - Increasing the number of workers W doesn't affect the rate of convergence (*Federated learning*)
Bigger models - Increasing the model size d **increases** the compression ratio d/k log² d.

Experimental verification:

40x compression for the transformer network with 90M parameters
Scaling to 256 workers and beyond, drastically outperforming local topK competitor

3. Sketching

Streaming model

Stream: $p_1, p_2, \dots, p_m \in [n]$
Goal: find "frequent" items (icebergs, elephants, heavy hitters) → Approximate answer with high probability is OK
Restrictions: - access the data sequentially, make only one pass
- small space, fast updates, fast query

Frequency: $f_i = \#\{j \in [m]: p_j = i\}$
(α, ℓ_p) **heavy hitter**: $f_i \ge \alpha \|f\|_p$

Example:
1 1 2 4 5 5 6 3 7 5 2 5 5 3 4 7 5

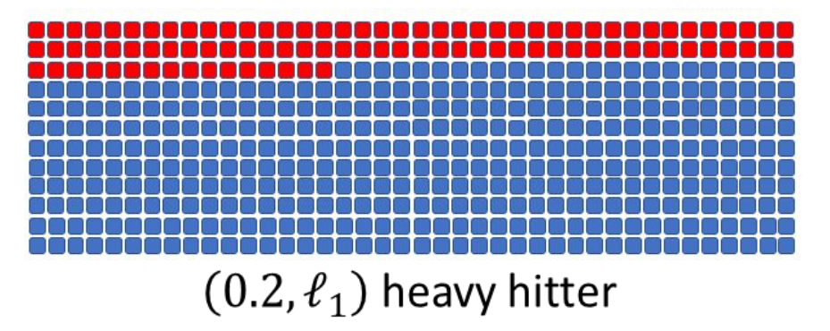
Sketches: ℓ_1 — Sampling, Count Min Sketch
ℓ_2 — Count Sketch

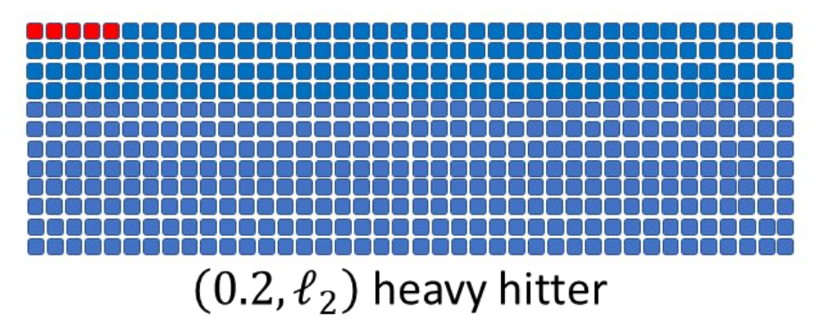
frequency vector f

i	f_i
1	2
2	2
3	2
4	2
5	6
6	1
7	2

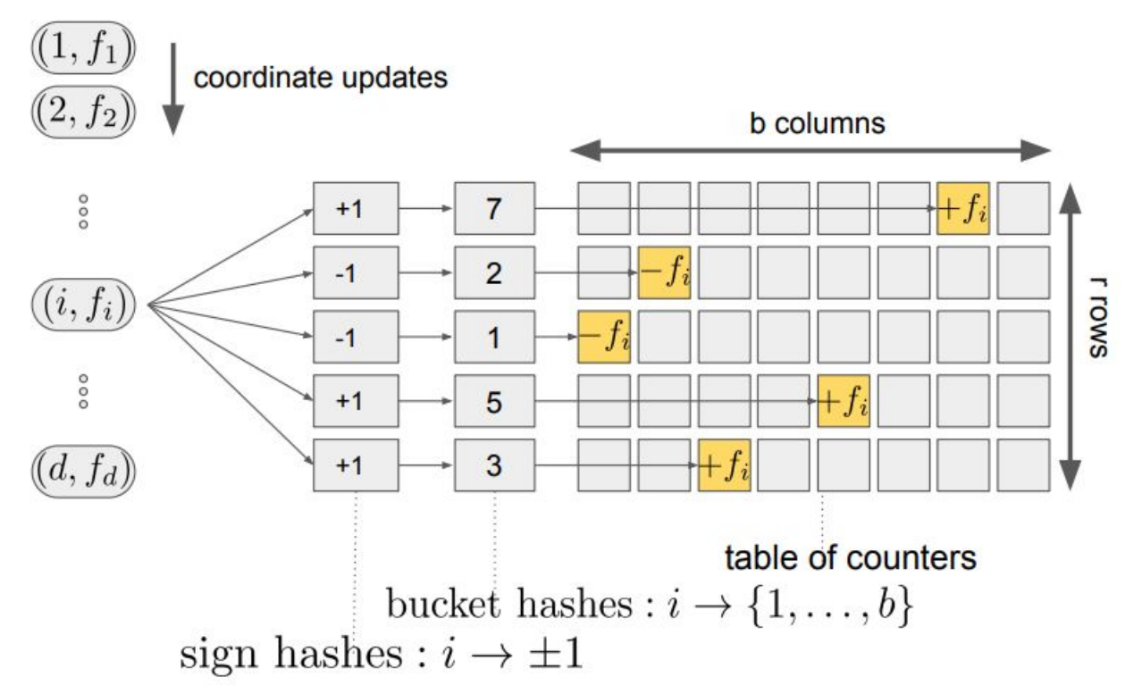
$\|f\|_1 = \sum f_i^1 = 17$
$f_5 \ge 0.3\|f\|_1$
5 is $(0.3, \ell_1)$ heavy hitter

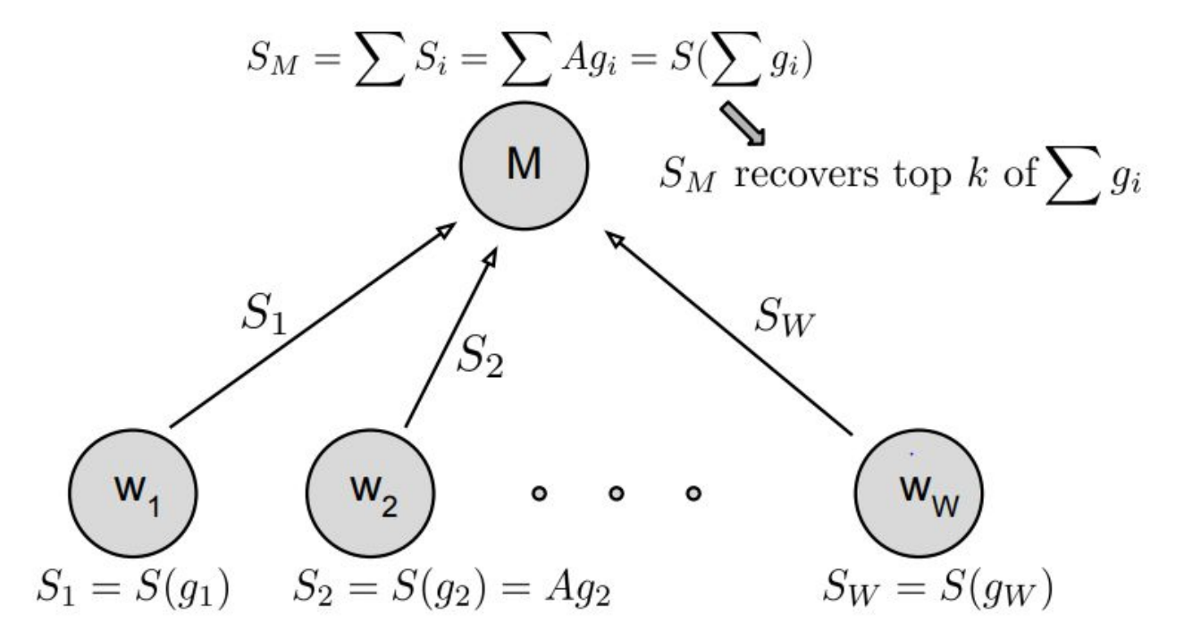
$\|f\|_2 = \left(\sum f_i^2\right)^{1/2} = 17$
$f_5 \ge 0.8\|f\|_2$
5 is $(0.8, \ell_2)$ heavy hitter


$(0.2, \ell_1)$ heavy hitter


$(0.2, \ell_2)$ heavy hitter

$\|\cdot\|_2 < \|\cdot\|_1$ => finding ℓ_2 heavy hitters is more challenging than ℓ_1




$S_M = \sum S_i = \sum A g_i = S(\sum g_i)$
M
S_M recovers top k of $\sum g_i$
S_1
S_2
S_W
W1
W2
WW
$S_1 = S(g_1)$
$S_2 = S(g_2) = Ag_2$
$S_W = S(g_W)$

Algorithm 4 Count Sketch [Charikar et al., 2002]

```
1: function init(r, c):
2:     init sign hashes $\{s_j\}_{j=1}^r$ and bucket hashes $\{h_j\}_{j=1}^r$
3:     init $r \times c$ table of counters $S$
4: function update($i$, $f_i$):
5:     for $j$ in $1 \dots r$:
6:         $S[j, h_j(i)]$ += $s_j(i) f_i$
7: function estimate($i$):
8:     init length $r$ array estimates
9:     for $j$ in $1, \dots, r$:
10:        estimates[$r$] = $s_j(i) S[j, h_j(i)]$
11:    return median(estimates)
```

4. Algorithm

Algorithm 3 EMPIRICAL TRAINING

```
Input: $k, \eta_t, m, T$
1: $\forall i : u^i, v^i \leftarrow 0$
2: for $t = 1, 2, \dots T$ do
3:     Compute stochastic gradient $g_t^i$                                          Worker$_i$
4:     Momentum: $u^i \leftarrow m u^i + g_t^i$                                     Worker$_i$
5:     Error accumulation: $v^i \leftarrow v^i + u^i$                               Worker$_i$
6:     Compute sketch $S_t^i$ of $v^i$ and send to Master                           Worker$_i$
7:     Aggregate sketches $S_t = \frac{1}{W}\sum_{i=1}^W S_t^i$                      Master
8:     Recover the top-$Pk$ coordinates from $S_t$: $\tilde{g}_t = top_{Pk}(S_t)$   Master
9:     Query all workers for exact values of nonzero elements in $\tilde{g}_t$; store the sum in $\tilde{g}_t$   Master
10:    Update $w_{t+1} = w_t - \eta_t \tilde{g}_t$, send result to Workers.          Master
11:    $u^i, v^i \leftarrow 0$, for all $i$ s.t. $\tilde{w}_t^i \neq 0$              Worker$_i$
12: end for
```

5. Results

Theory:

Definition 2 (τ-contraction [Stich et al., 2018]). *A τ-contraction operator is a possibly randomized operator comp : $\mathbb{R}^d \to \mathbb{R}^d$ that satisfies: $\forall x \in \mathbb{R}^d$, $\mathbb{E}\left[\|x - comp(x)\|^2\right] \le (1-\tau)\|x\|^2$*

Lemma 1. *SKETCHED-SGD with sketch size $\Theta(k \log(d/\delta))$ performs k/d-contraction on each synchronization with probability $\ge 1-\delta$.*

Theorem 1 (strongly convex, smooth). *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth μ-strongly convex function, and let the data be shared among W workers. Given $0 < k \le d, 0 < \alpha, \delta < 1$ SKETCHED-SGD run with sketch size $= \mathcal{O}(k \log(dT/\delta)$, step size $\eta_t = \frac{1}{t+\xi}$, with $\xi > 2 + \frac{d(1+\beta)}{k(1+\rho)}$, with $\beta > 4$ and $\rho = \frac{4\beta}{(\beta-4)(\beta+1)^2}$ after T steps outputs $\hat{w}_T$ such that the following holds,*

1. *With probability at least $1-\delta$, $\mathbb{E}[f(\hat{w}_T)] - f(w^*) \le \mathcal{O}\left(\frac{\sigma^2}{\mu T} + \frac{d^2 G^2 L}{k^2 \mu^2 T^2} + \frac{d^3 G^3}{k^3 \mu T^3}\right)$*
2. *The total communication per update is $\Theta(k \log(dT/\delta) W)$ bits.*

Practical performance:

	BLEU (transformer) 90M	BLEU (LSTM) 70M
Vanilla distributed SGD	26.29	20.87
Top-100, 000 SGD	26.65	22.2
SKETCHED-SGD, 20x compression	26.87[1]	–
SKETCHED-SGD, 40x compression	26.79[2]	20.95 [3]

BLEU scores on the test data achieved for vanilla distributed SGD, top-k SGD, and SKETCHED-SGD with 20x and 40x compression.. Larger BLEU score is better.

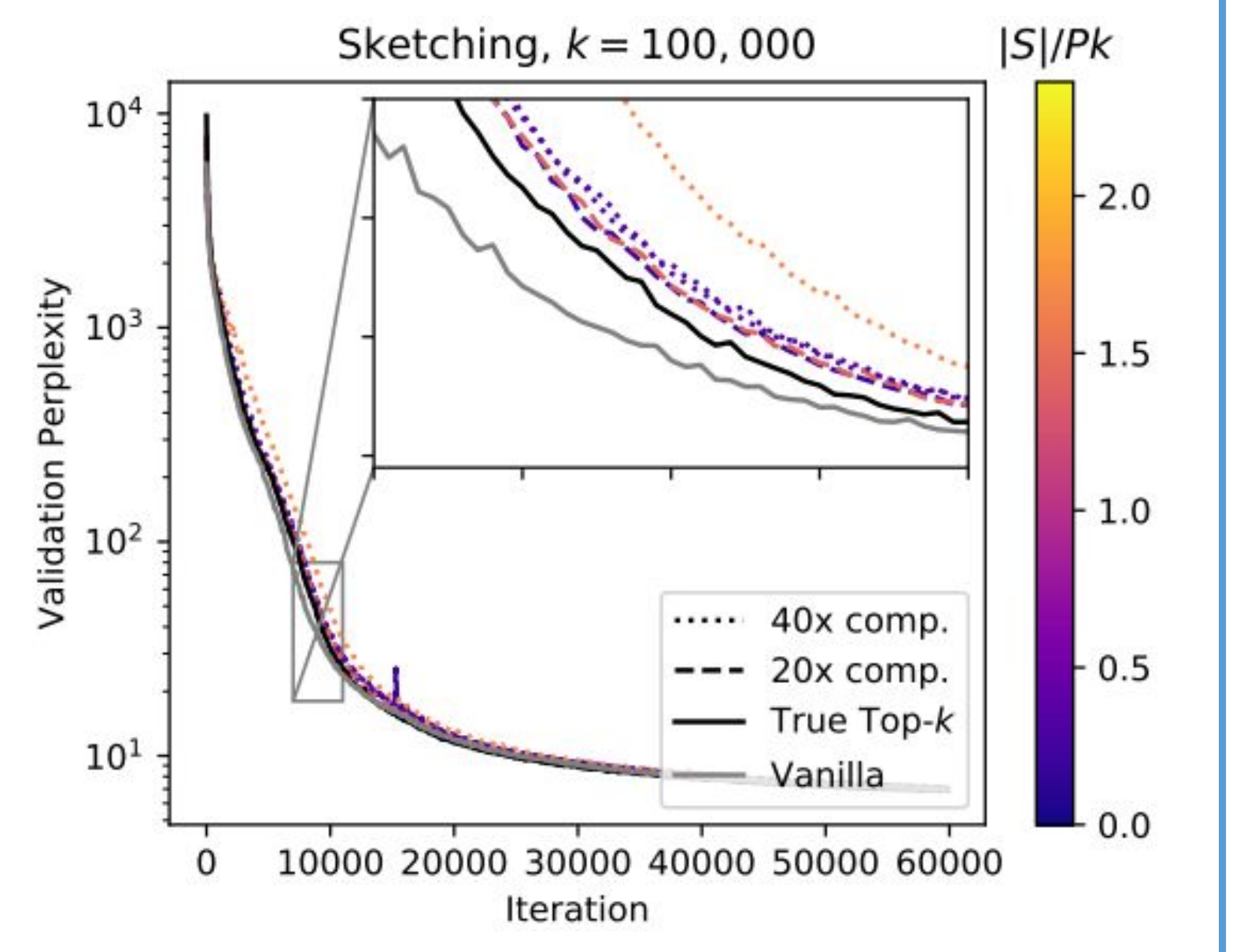
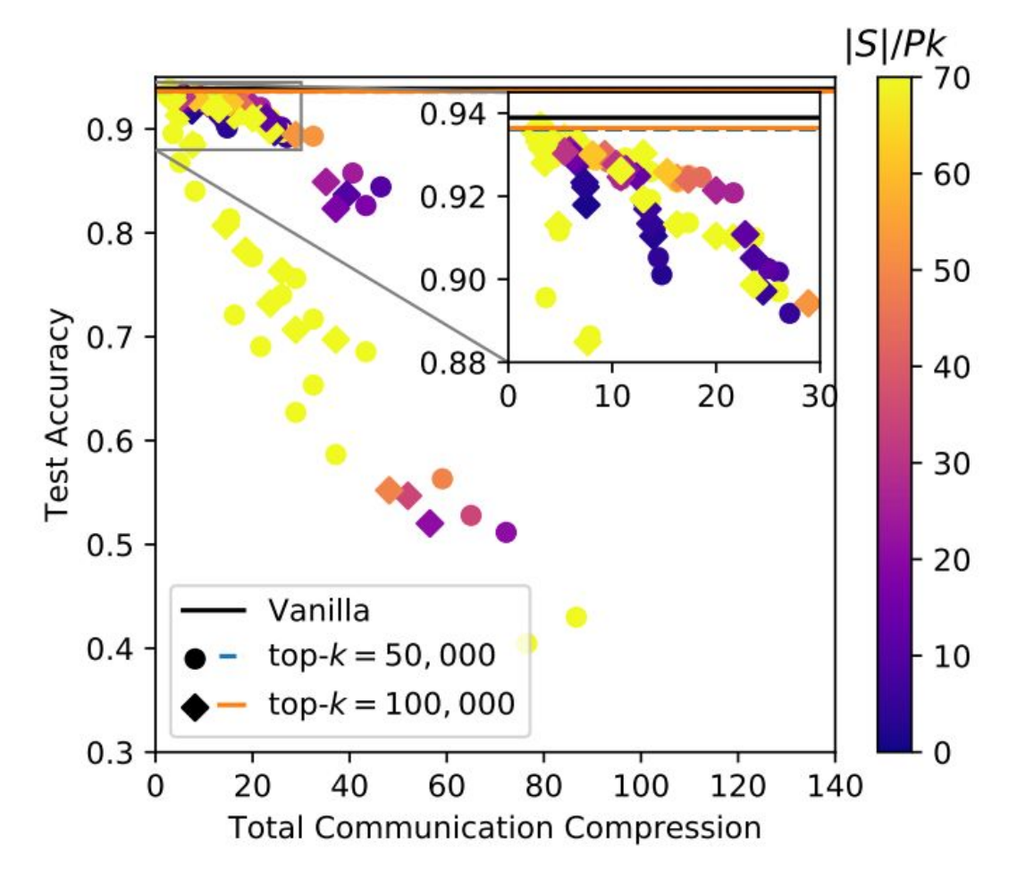
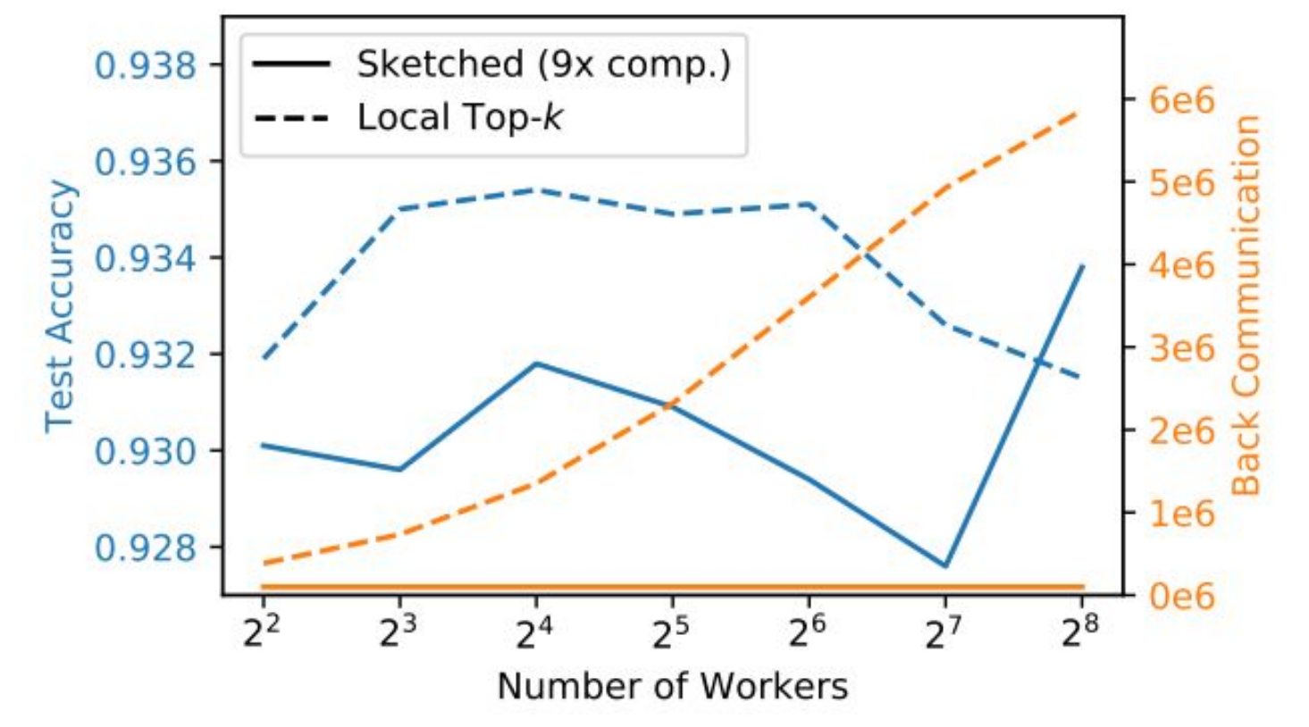


Figure 1: Learning curves for a transformer model trained on the WMT 2014 English to German translation task. All models included here achieve comparable BLEU scores after 60,000 iterations (see Table 1). Each run used 4 workers.



Tradeoff between compression and model accuracy for a residual network trained on CIFAR-10 for k = 50000 and 100000. The (nearly overlapping) solid orange and dashed blue lines show the accuracy achieved by top-k SGD for the two values of k, and the black line shows the accuracy achieved by vanilla distributed SGD.

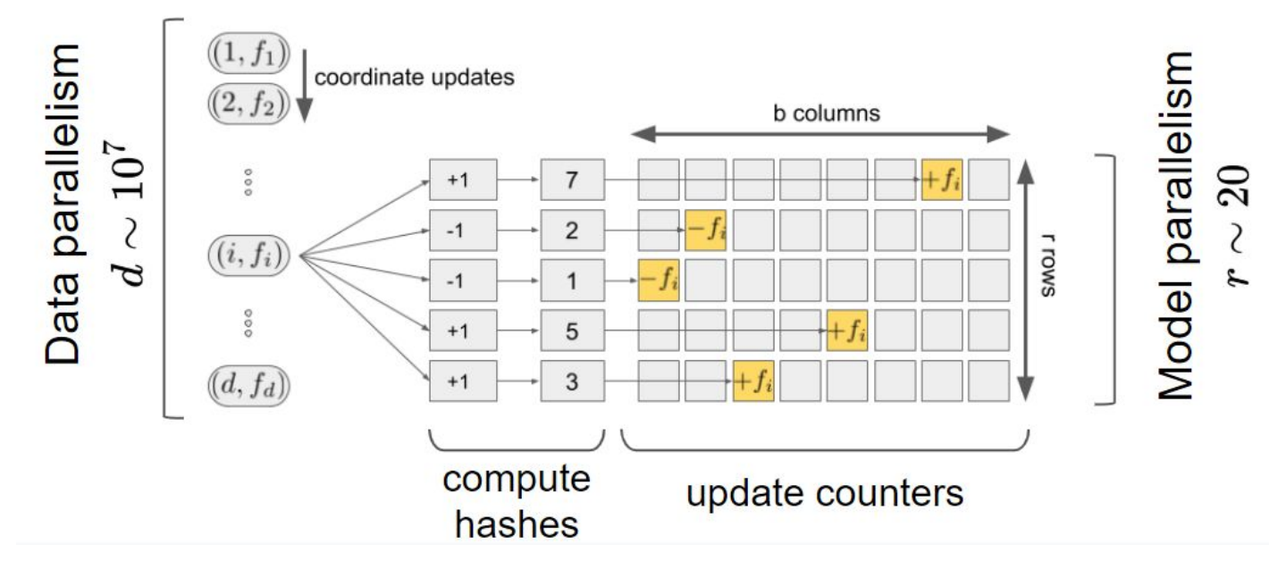


Comparison between SKETCHED-SGD and local top-k SGD on CIFAR10. The best overall compression that local top-k can achieve for many workers is 2x, this happens due to local topKs being disjoint, therefore parameter server after aggregation sends back almost entire gradient vector.

Computational overhead:

Simple to parallelize the sketching part:

```
for each coordinate:
  for each row:
    compute hashes (bucket + sign)
    update corresponding counter
```



100x acceleration on modern GPU
Specifics of distributed SGD application:
- gradient vector is already on GPU
- for reasonable d, all hashes can be precomputed
- one-liner to parallelize using pytorch framework (20x speed up)

```
table[row,:] += torch.bincount(bucketsHashes[row,:], signsHashes[row]*vec)
```