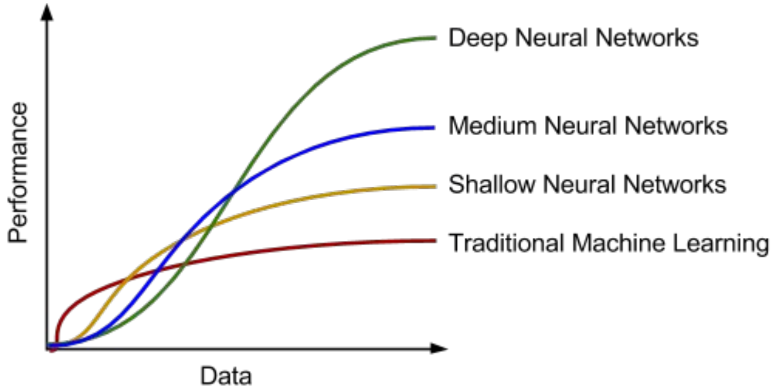
Communication-efficient Distributed SGD with Sketching

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

* equal contribution

Going distributed: why?

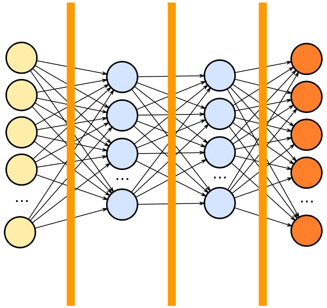
- Large scale machine learning is moving to the distributed setting due to growing size of datasets/models, and modern learning paradigms like Federated learning.



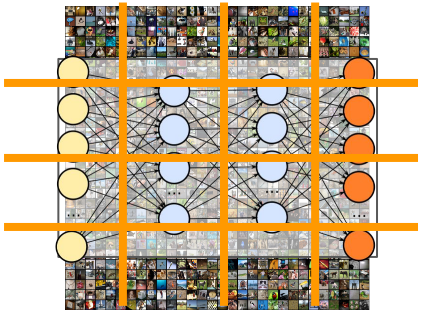
Going distributed: how?

data

most popular

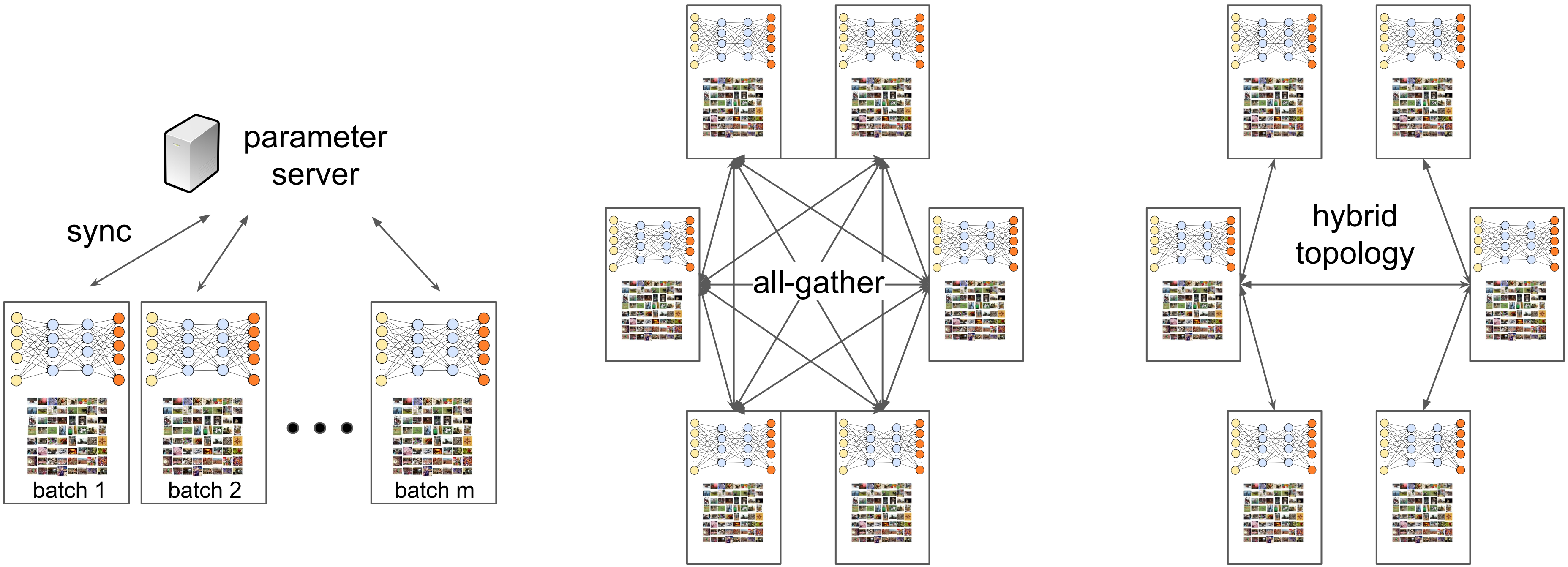


model



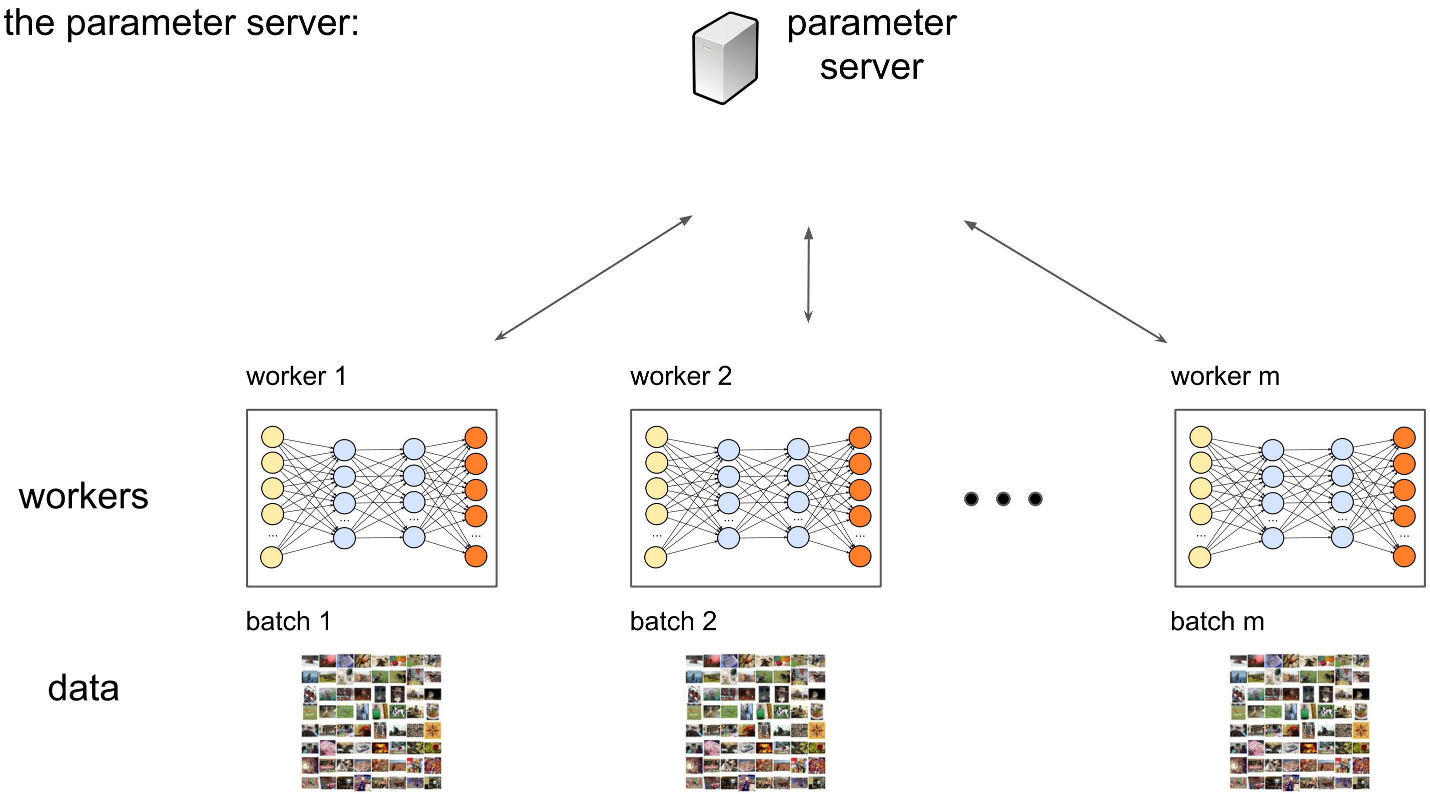
hybrid

Going distributed: how?



Going distributed: how?

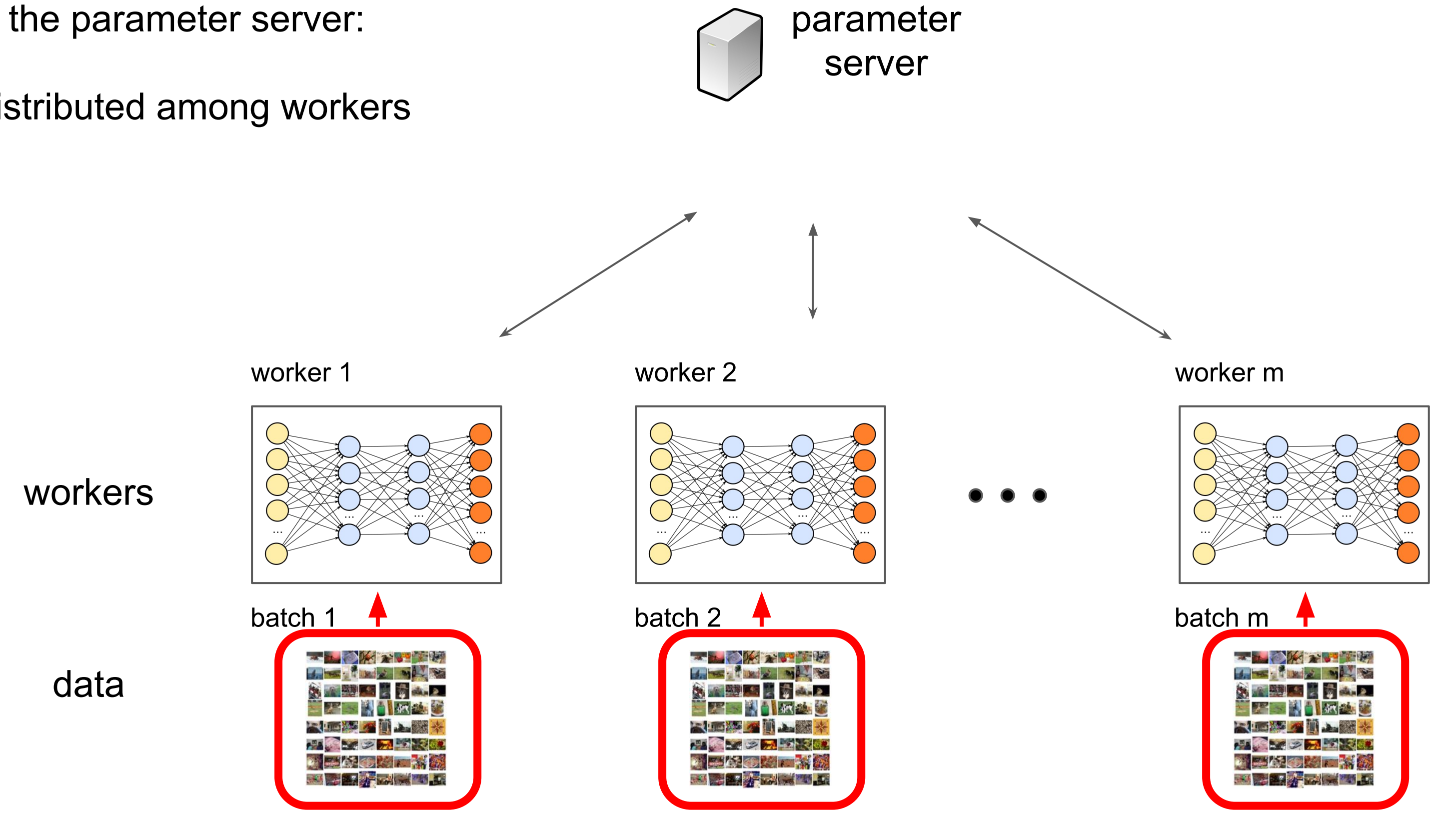
Synchronization with the parameter server:



Going distributed: how?

Synchronization with the parameter server:

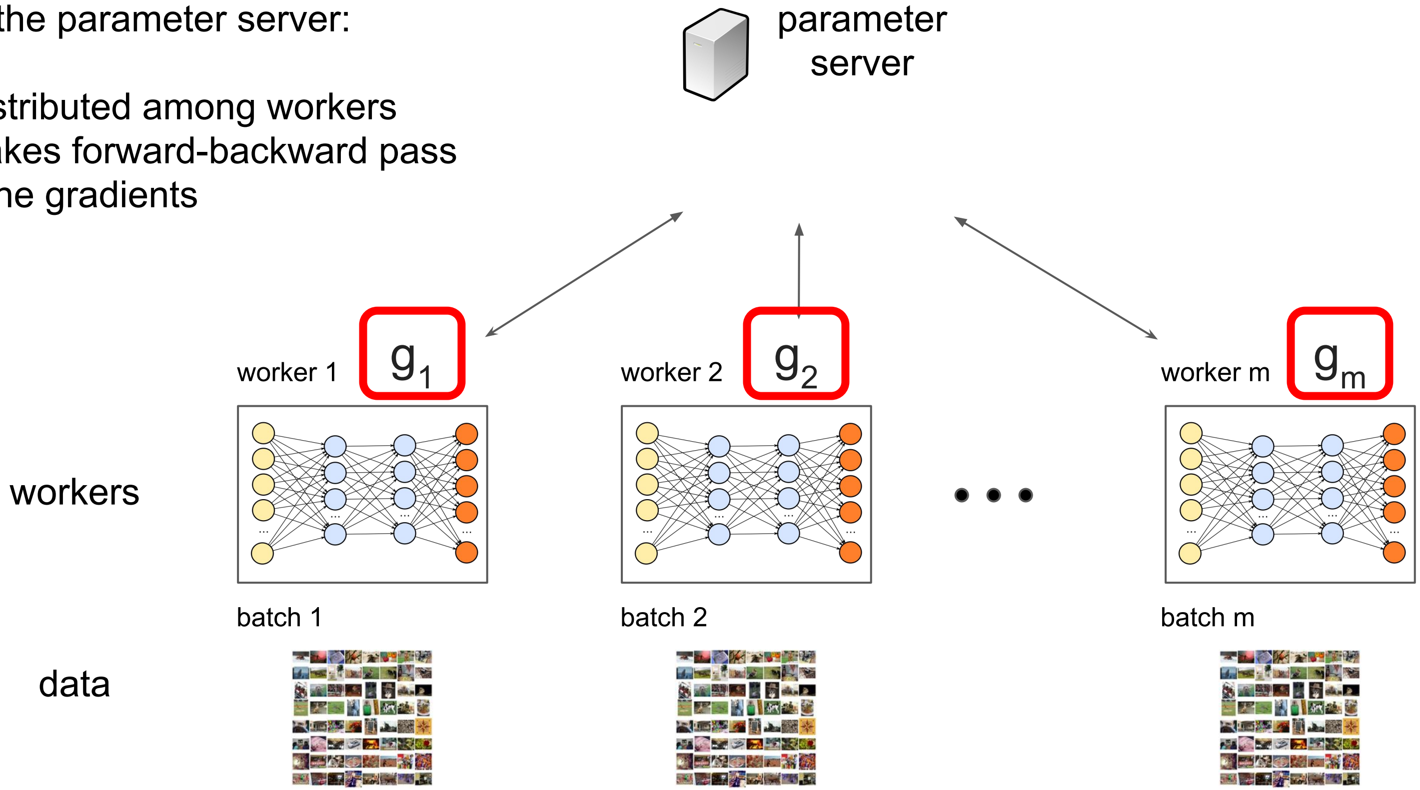
- mini-batches distributed among workers



Going distributed: how?

Synchronization with the parameter server:

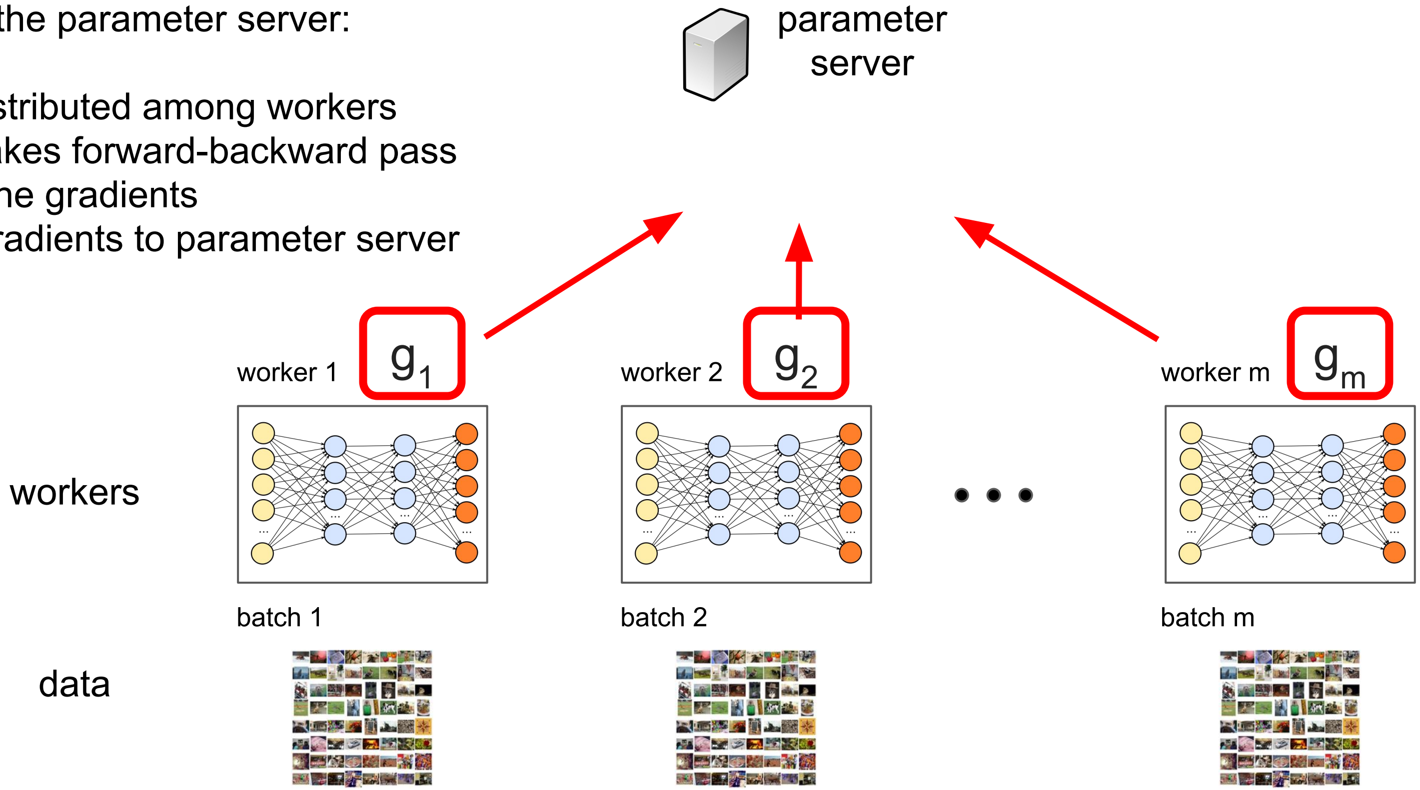
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes the gradients



Going distributed: how?

Synchronization with the parameter server:

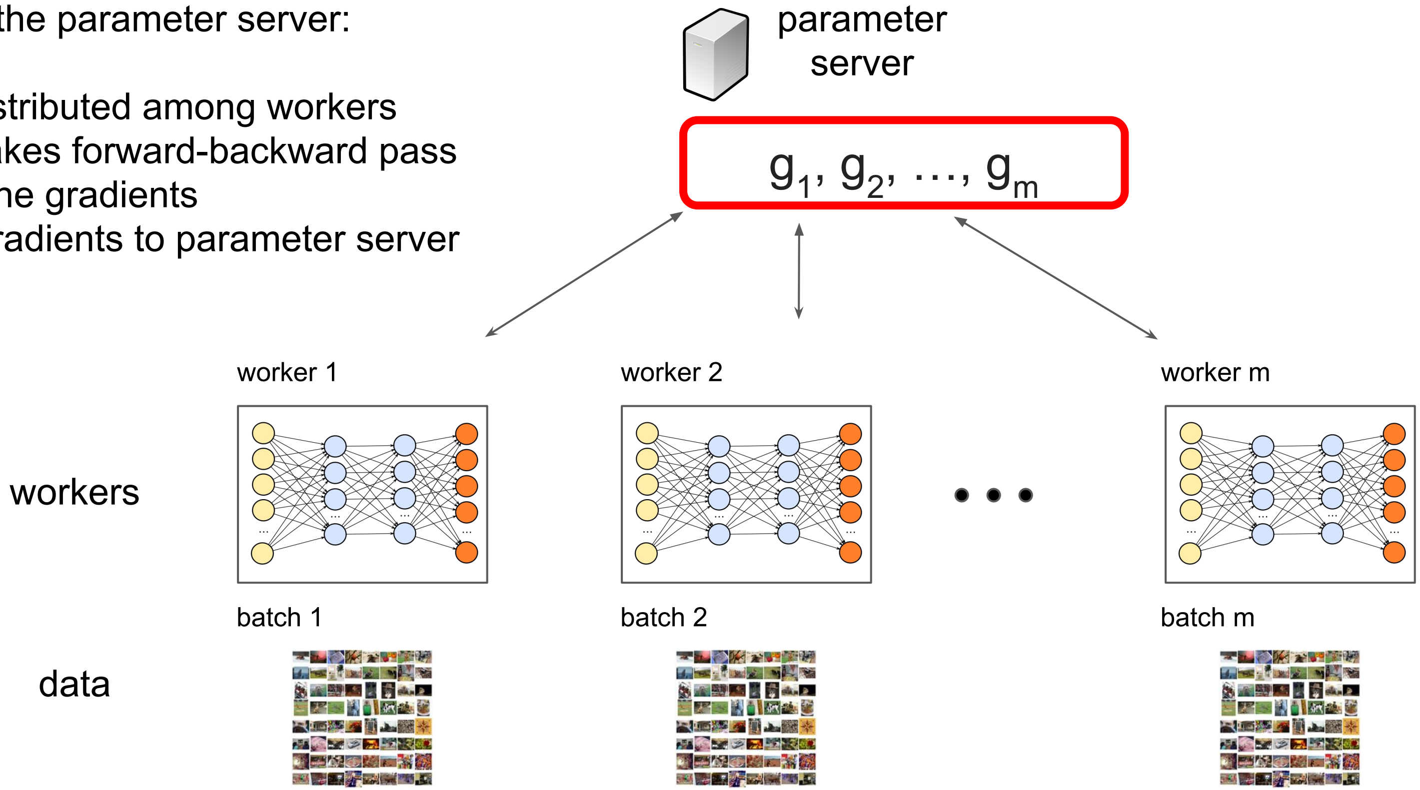
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes the gradients
- workers send gradients to parameter server



Going distributed: how?

Synchronization with the parameter server:

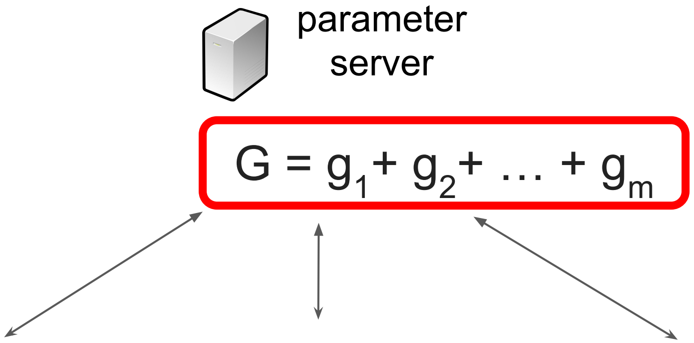
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes the gradients
- workers send gradients to parameter server

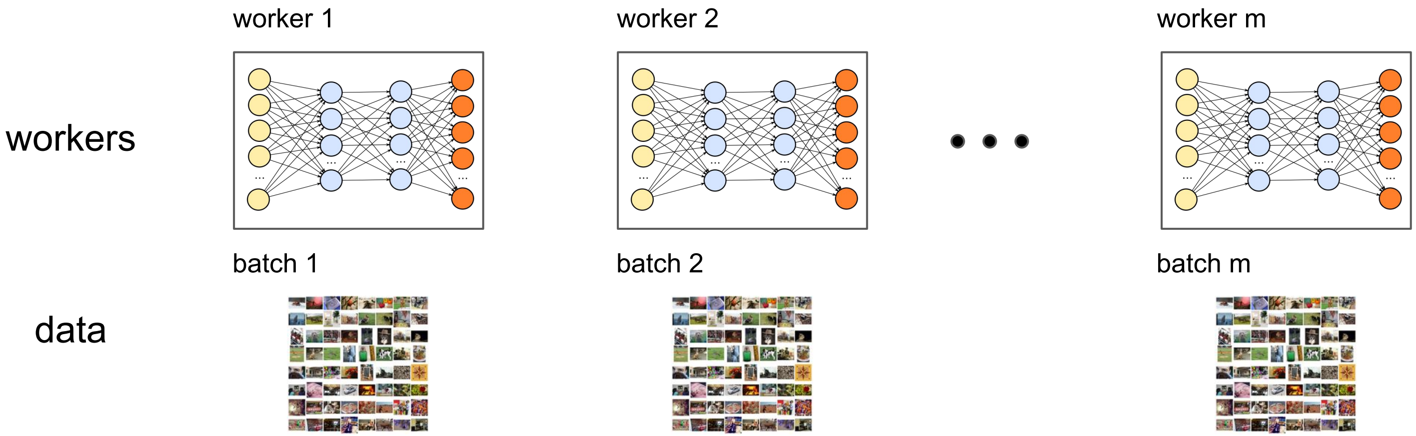


Going distributed: how?

Synchronization with the parameter server:

- 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

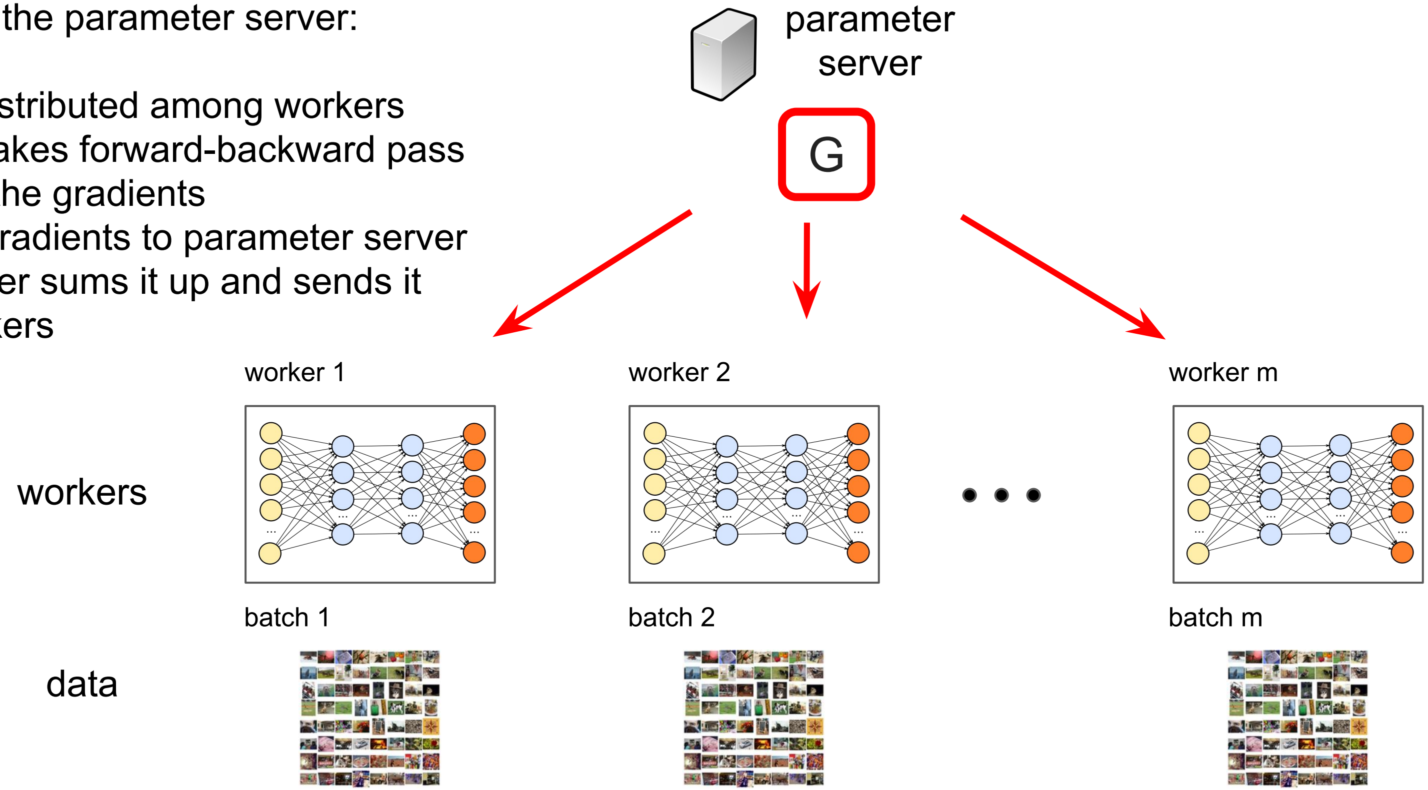




Going distributed: how?

Synchronization with the parameter server:

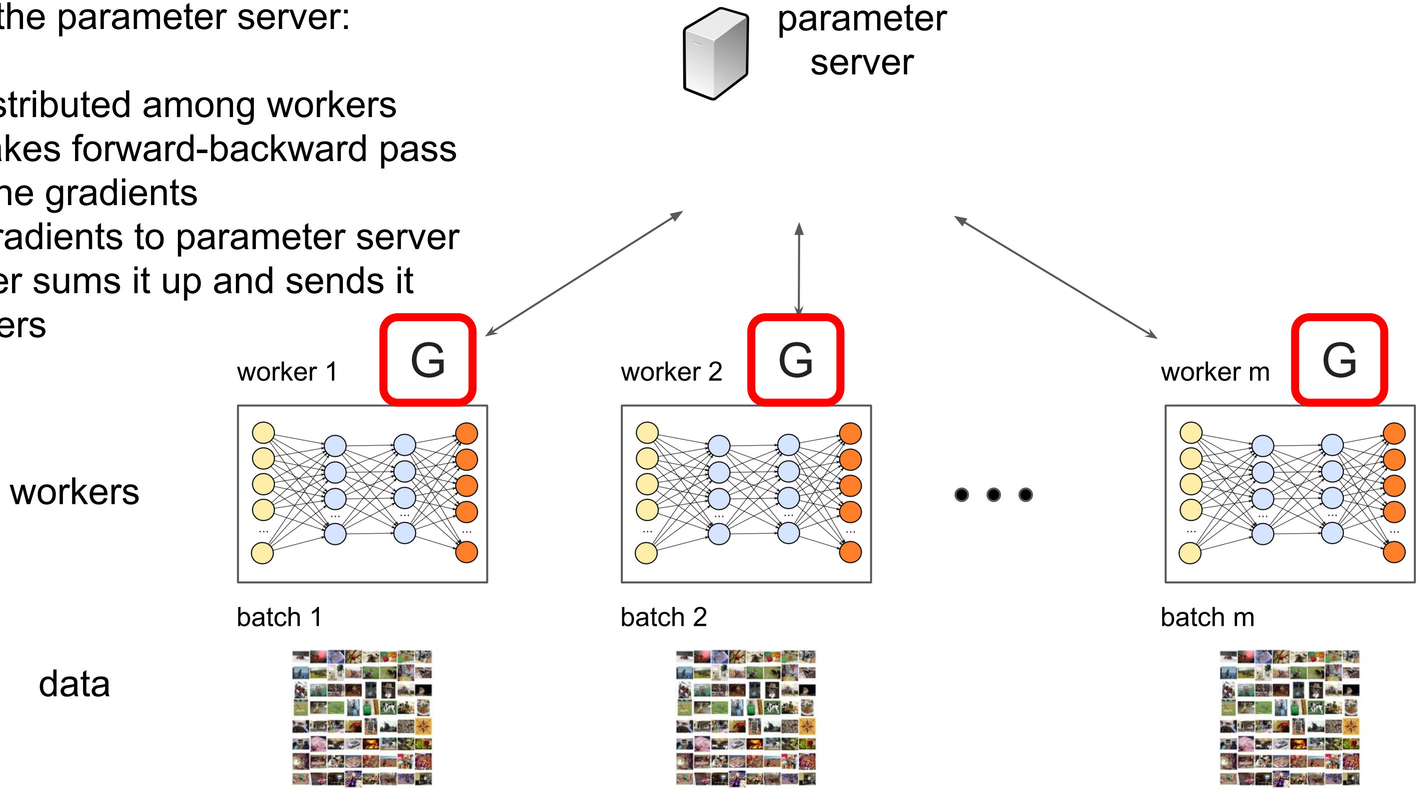
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes the gradients
- workers send gradients to parameter server
- parameter server sums it up and sends it back to all workers



Going distributed: how?

Synchronization with the parameter server:

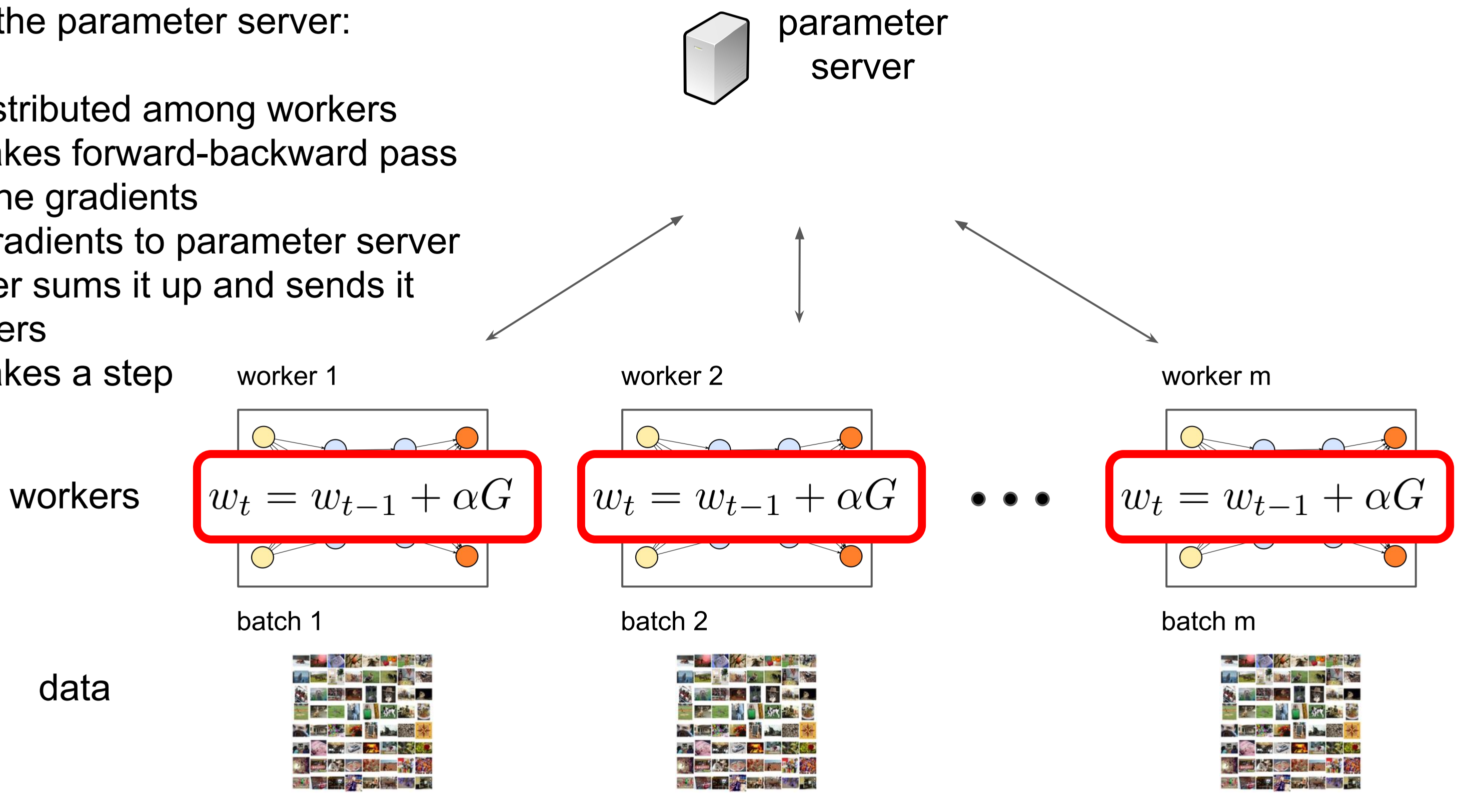
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes the gradients
- workers send gradients to parameter server
- parameter server sums it up and sends it back to all workers



Going distributed: how?

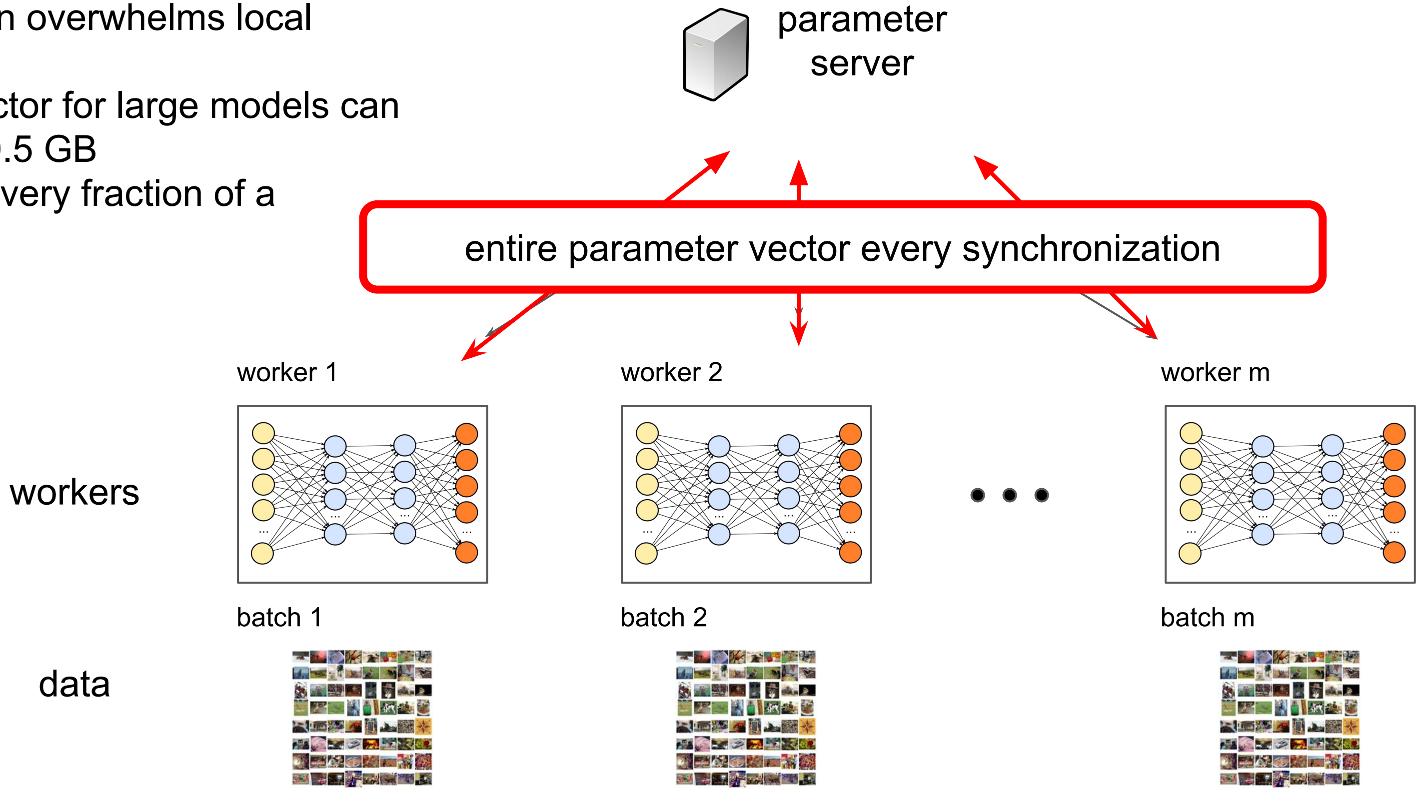
Synchronization with the parameter server:

- 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



Going distributed: what’s the problem?

- Slow communication overwhelms local computations:
  - parameter vector for large models can weight up to 0.5 GB
  - synchronize every fraction of a second



Going distributed: what’s the problem?

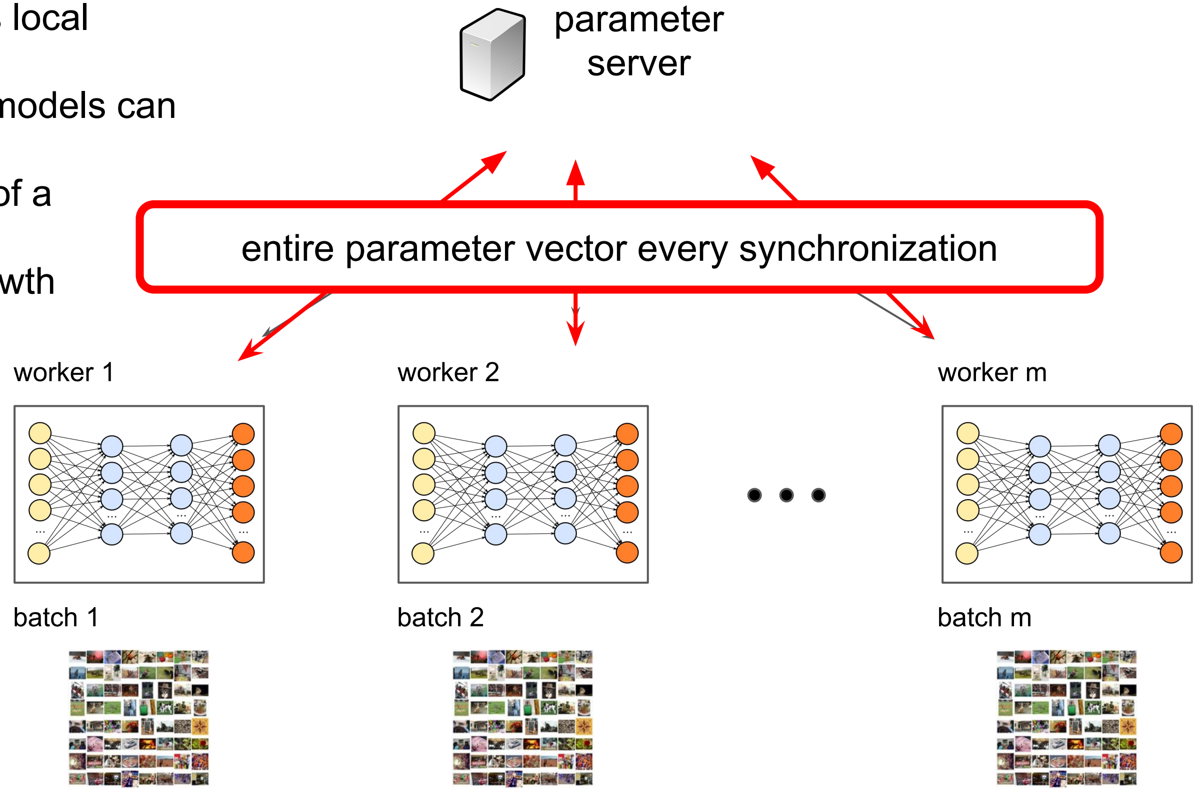
- Slow communication overwhelms local computations:
  - parameter vector for large models can weight up to 0.5 GB
  - synchronize every fraction of a second
- Mini batch size has limit to its growth

↓

computation resources are wasted

workers

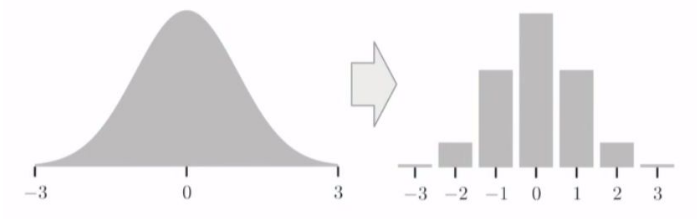
data



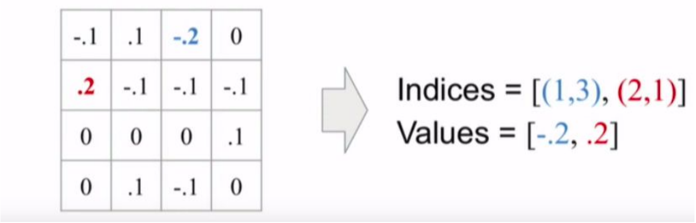
Going distributed: how others deal with it?

- Compressing the gradients:

  Quantization



  Sparsification



Quantization

- Quantizing gradients can give a constant factor decrease in communication cost.
- Simplest quantization to 16-bit, but all the way to 2-bit (TernGrad [1]) and 1-bit (SignSGD [2]) have been successful.
- Quantization techniques can in principle be combined with gradient sparsification

[1] Wen, Wei, et al. "Terngrad: Ternary gradients to reduce communication in distributed deep learning." *Advances in neural information processing systems*. 2017.
[2] Bernstein, Jeremy, et al. "signSGD: Compressed optimisation for non-convex problems." *arXiv preprint arXiv:1802.04434* (2018).
[3] Karimireddy, Sai Praneeth, et al. "Error Feedback Fixes SignSGD and other Gradient Compression Schemes." *arXiv preprint arXiv:1901.09847* (2019).

Sparsification

- Existing techniques either communicate $\Omega(Wd)$ in the worst case, or are heuristics; W - number of workers, d - dimension of gradient.
- [1] showed that SGD (on 1 machine) with top-*k* gradient updates and *error accumulation* has desirable convergence properties.
- Q. Can we extend the top-*k* to the distributed setting?
  - MEM-SGD [1] (for 1 machine, extension to distributed setting is sequential)
  - top-k SGD [2] (assumes that global top k is close to sum of local top k)
  - Deep gradient compression [3] (no theoretical guarantees).
- We resolve the above using sketches!

[1] Stich, Sebastian U., Jean-aptiste Cordonnier, and Martin Jaggi. "Sparsified sgd with memory." *Advances in Neural Information Processing Systems*. 2018.
[2] Alistarh, Dan, et al. "The convergence of sparsified gradient methods." *Advances in Neural Information Processing Systems*. 2018.
[3] Lin, Yujun, et al. "Deep gradient compression: Reducing the communication bandwidth for distributed training." *arXiv preprint arXiv:1712.01887* (2017).
APA

Streaming model

Stream: $p_1, p_2, \ldots, p_m \in [n]$

Goal: find "frequent" items (icebergs, elephants, heavy hitters)

Restrictions:
- access the data sequentially, make only one pass
- small space, fast updates, fast query

Approximate answer with high probability is OK

Frequency: $f_i = \#\{j \in [m]: p_j = i\}$

(α, ℓ_p) **heavy hitter**: $f_i \geq \alpha \|f\|_p$

Example:

1 1 2 4 5 5 6 3 7 5 2 5 5 3 4 7 5

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

frequency vector f

$\|f\|_1 = \sum f_i^1 = 17$

$f_5 \geq 0.3 \|f\|_1$

5 is $(0.3, \ell_1)$ heavy hitter

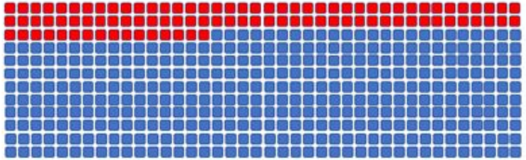
$\|f\|_2 = \left(\sum f_i^2\right)^{1/2} = 17$

$f_5 \geq 0.8 \|f\|_2$

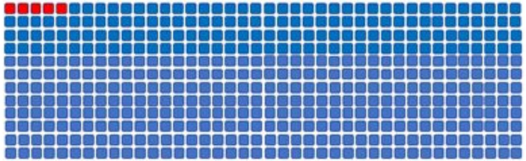
5 is $(0.8, \ell_2)$ heavy hitter

Sketches: ℓ_1 – Sampling, Count Min Sketch

ℓ_2 – Count Sketch



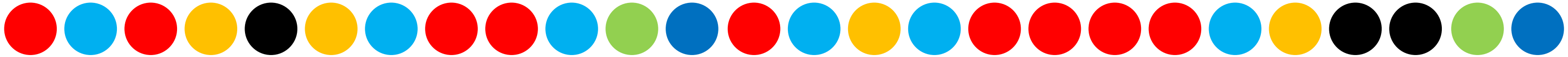
$(0.2, \ell_1)$ heavy hitter



$(0.2, \ell_2)$ heavy hitter

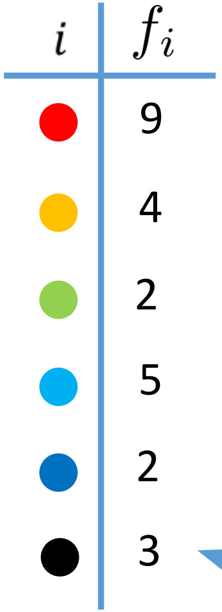
$\|\cdot\|_2 < \|\cdot\|_1$ => finding ℓ_2 heavy hitters is more challenging than ℓ_1

ℓ_2 norm estimation (AMS'96)



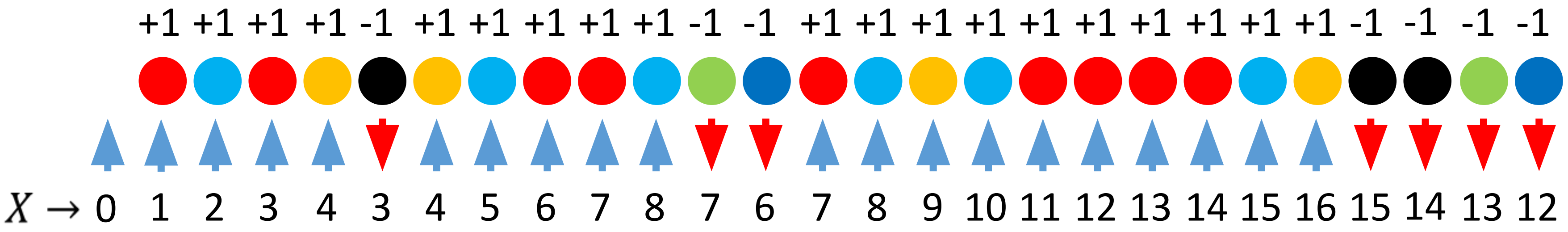
Stream: $p_1, p_2, \dots, p_m \in [n]$

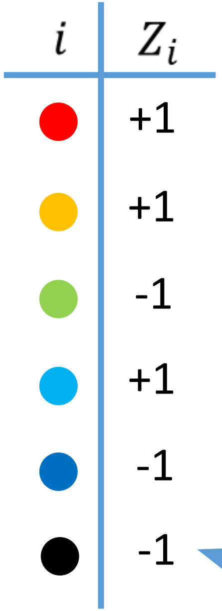
Want to find: $||f||_2$



frequencies
of balls

ℓ_2 norm estimation (AMS'96)





+/-1
equiprobably,
independent

Algorithm 1 AMS sketch

$X = 0$
function UPDATE(i)
$\quad X += Z_i$
return X^2

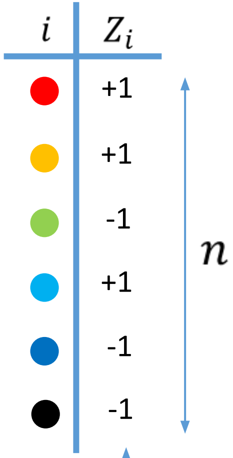
Stream: $p_1, p_2, \ldots, p_m \in [n]$

$X = \sum_{j=1}^{m} Z_{p_j}$

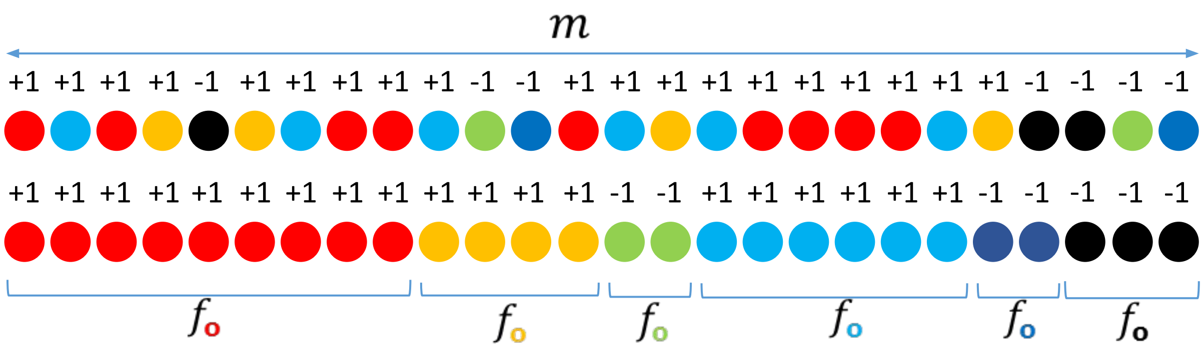
$X^2 = 144$

$\|f\|_2^2 = 150$

ℓ_2 norm estimation (AMS'96)



+/-1 equiprobably, independent



Stream: $p_1, p_2, \ldots, p_m \in [n]$

$$X = \sum_{j=1}^{m} Z_{p_j} = \sum_{i=1}^{n} Z_i f_i$$

$$E(X^2) = E\left(\sum_{i=1}^{n} Z_i^2 f_i^2\right) + E\left(\sum_{i \neq j} f_i f_j Z_i Z_j\right) = \sum_{i=1}^{n} f_i^2 = \|f\|_2^2$$

$Z_i^2 = 1$

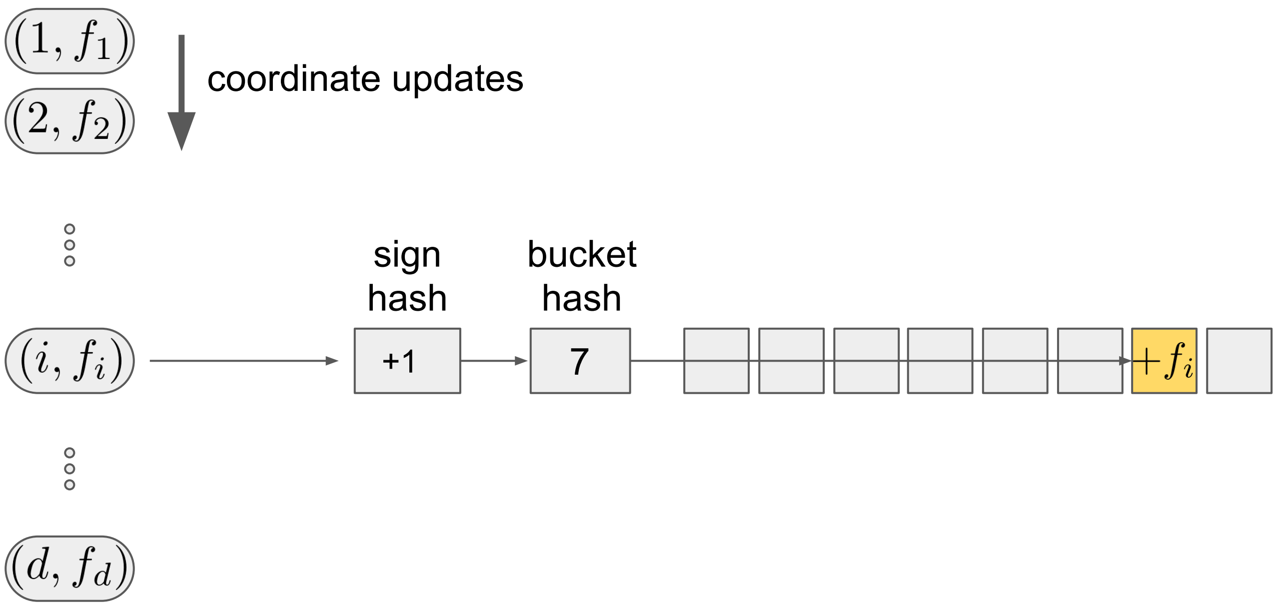
$\sum_{i=1}^{n} f_i^2$

Z_i indep

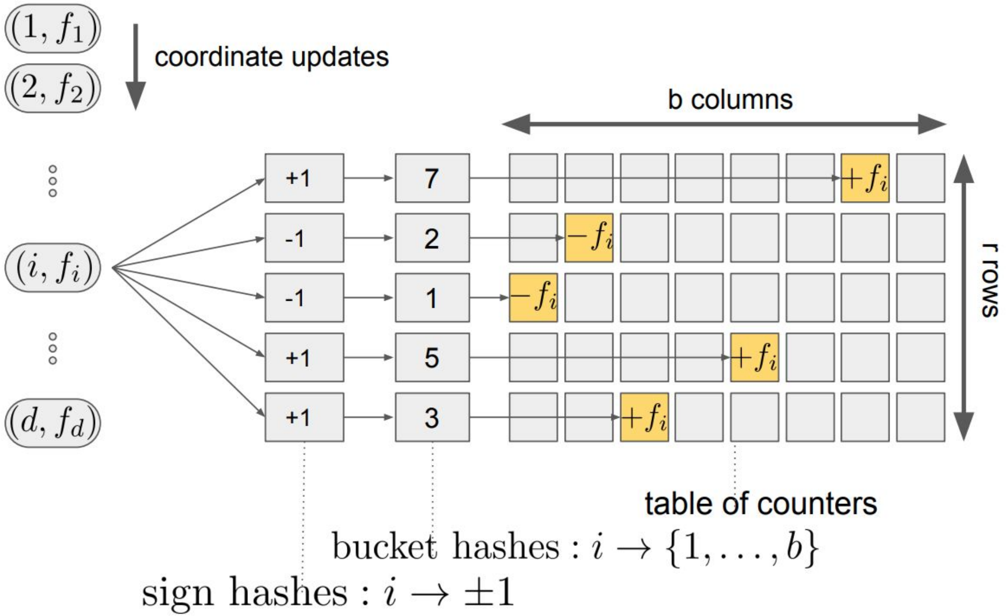
$E(Z_i) = 0$

$\sum_{i=1}^{n} f_i f_j E(Z_i) E(Z_j) = 0$

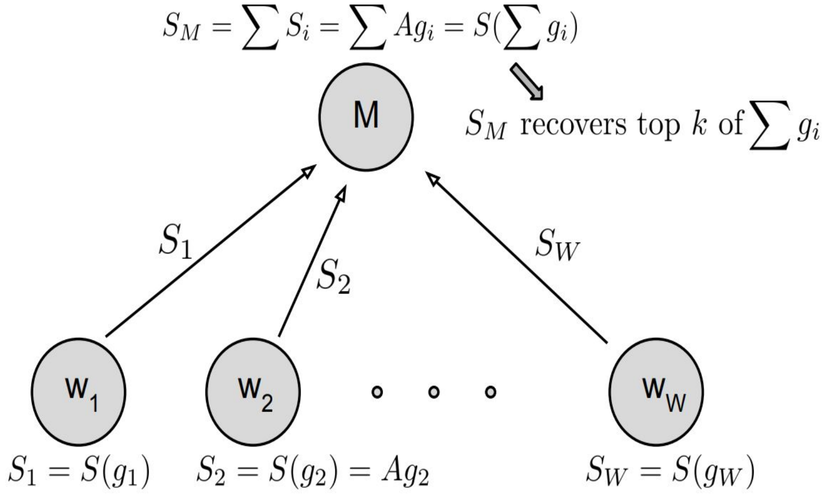
Count Sketch



Count Sketch

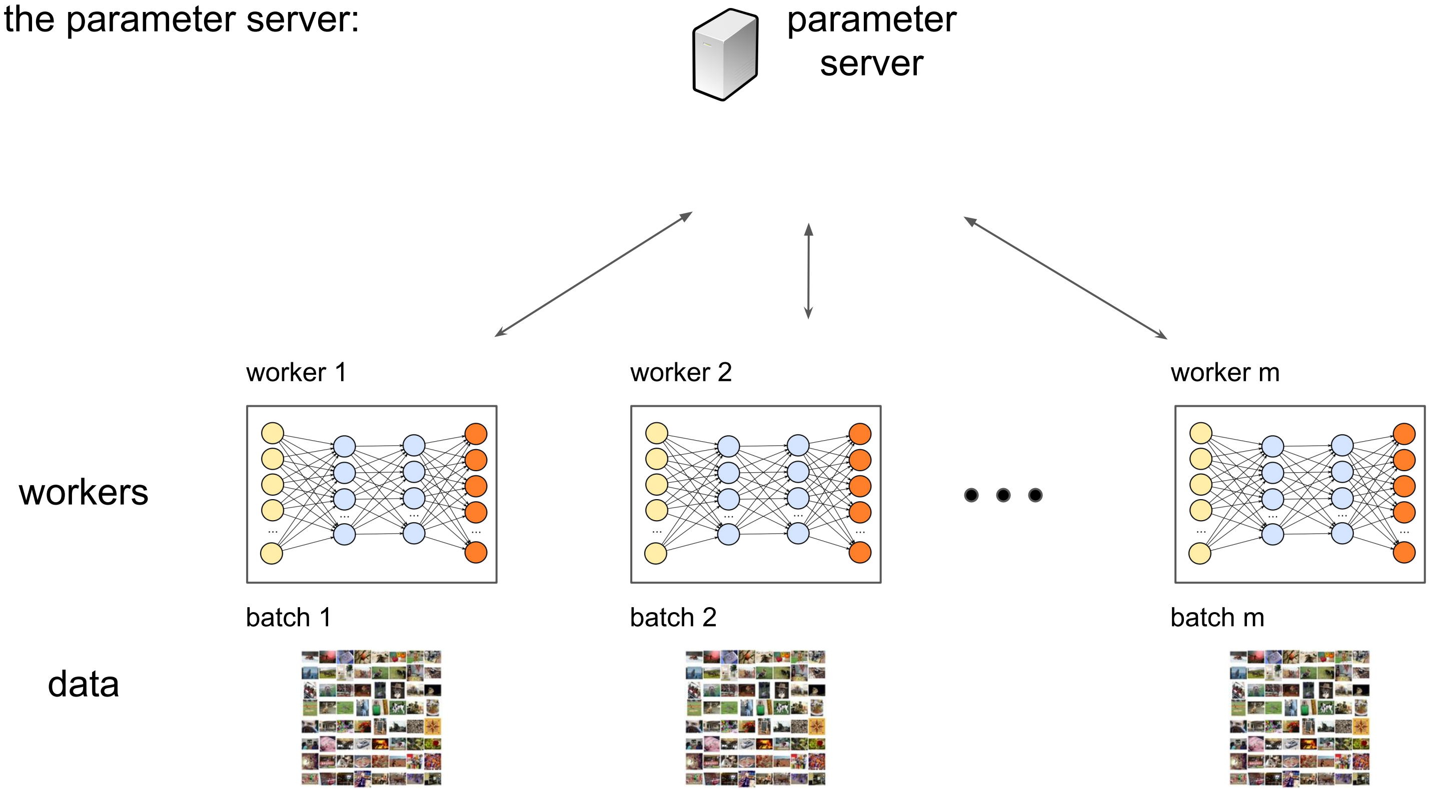


Mergebility



Compression scheme

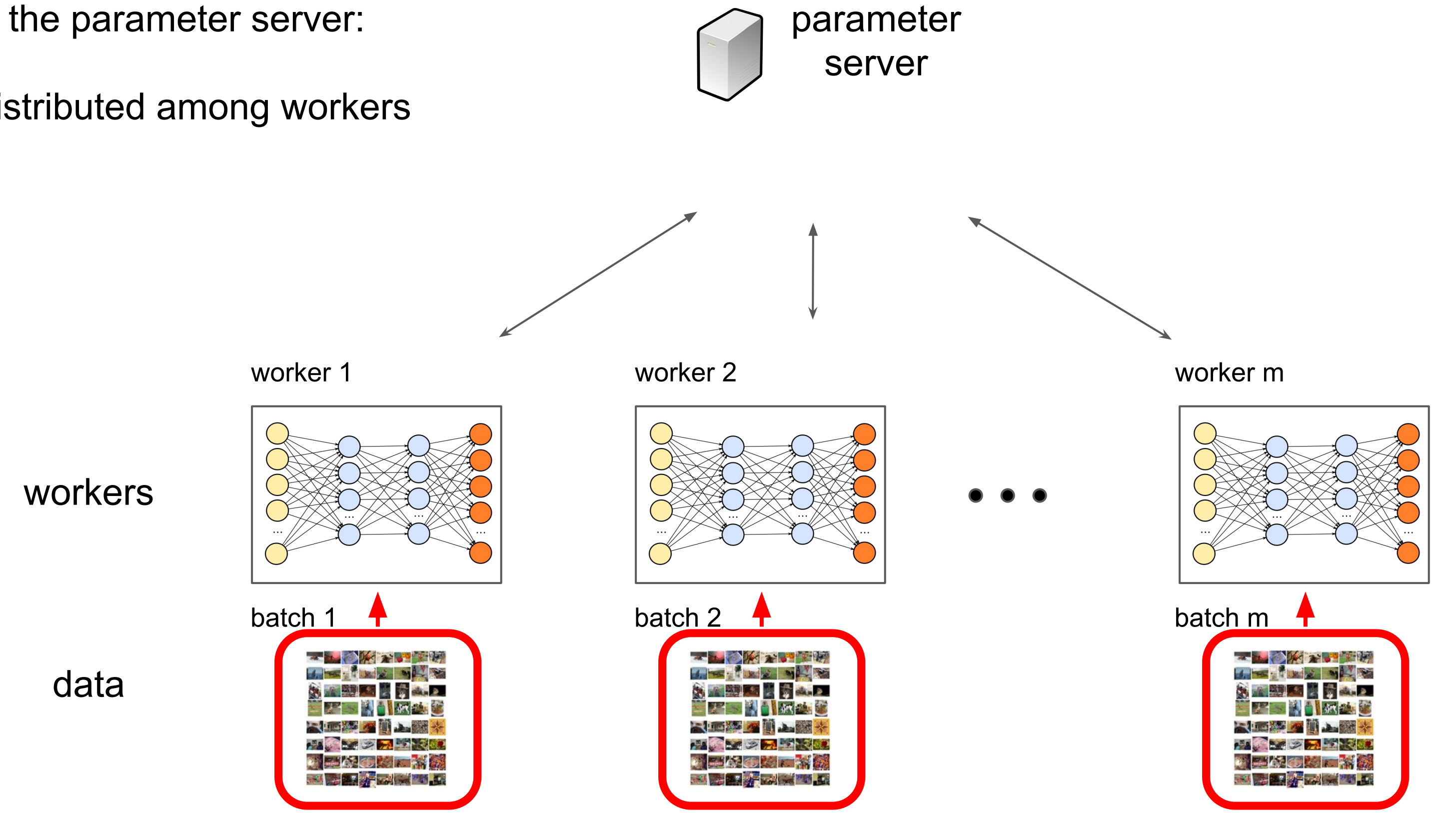
Synchronization with the parameter server:



Compression scheme

Synchronization with the parameter server:

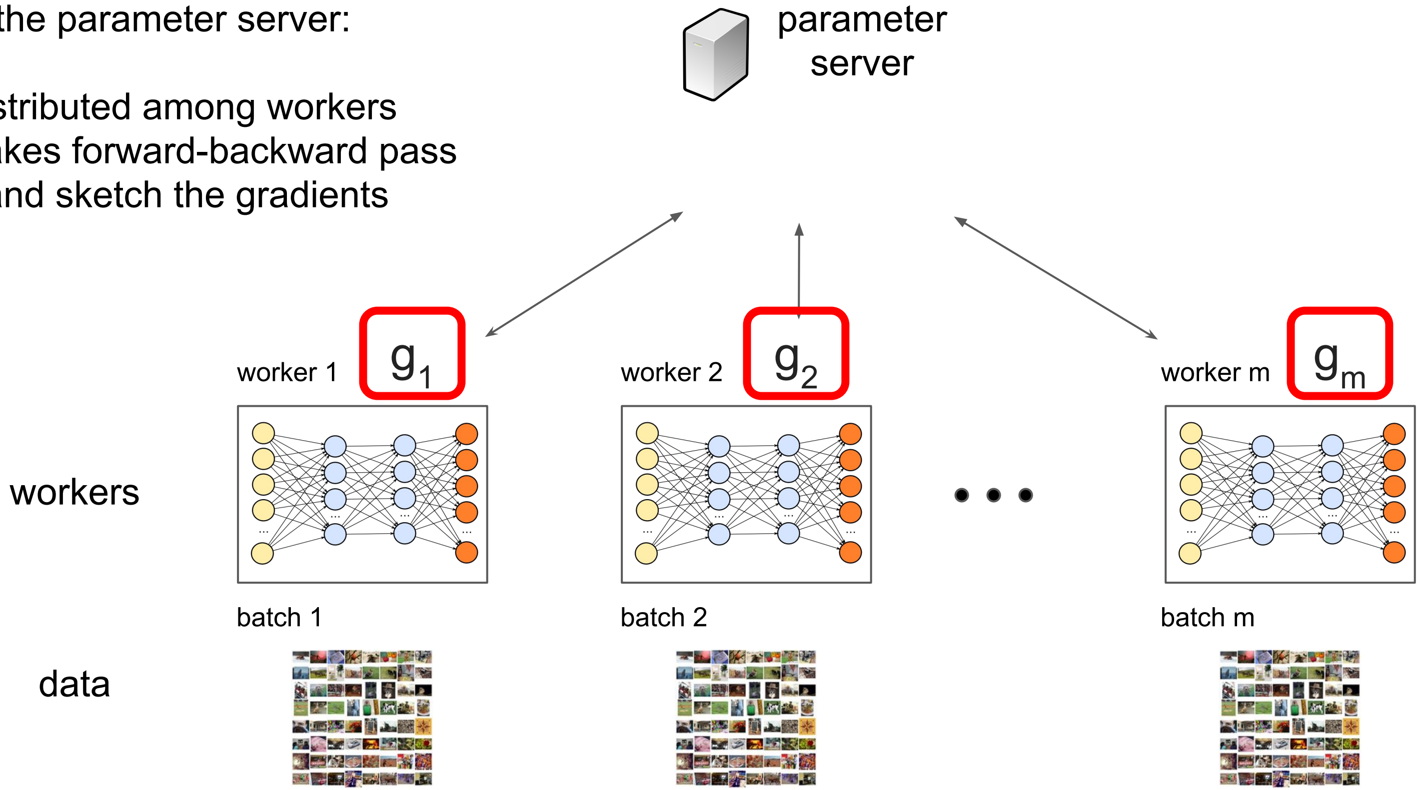
- mini-batches distributed among workers



Compression scheme

Synchronization with the parameter server:

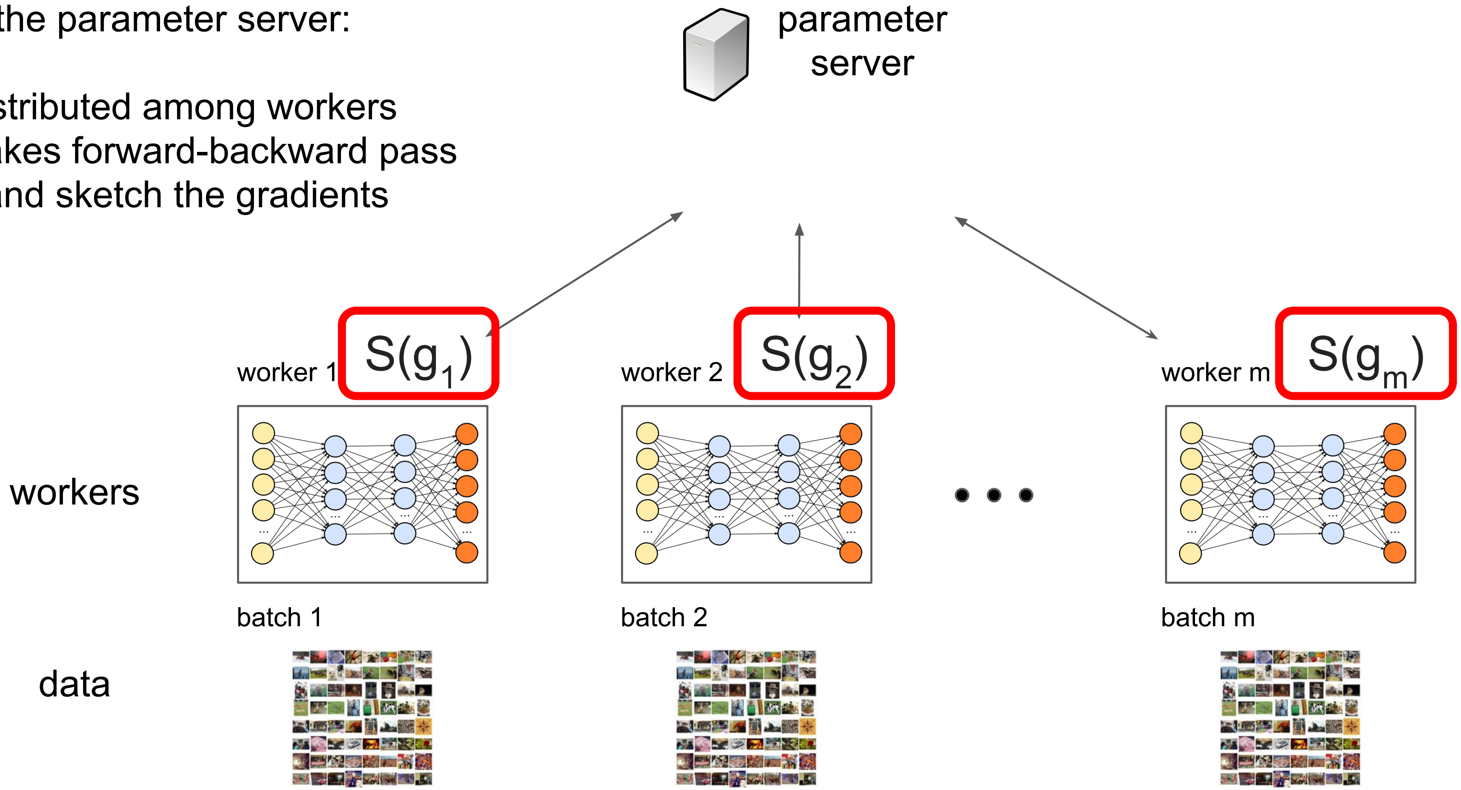
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients



Compression scheme

Synchronization with the parameter server:

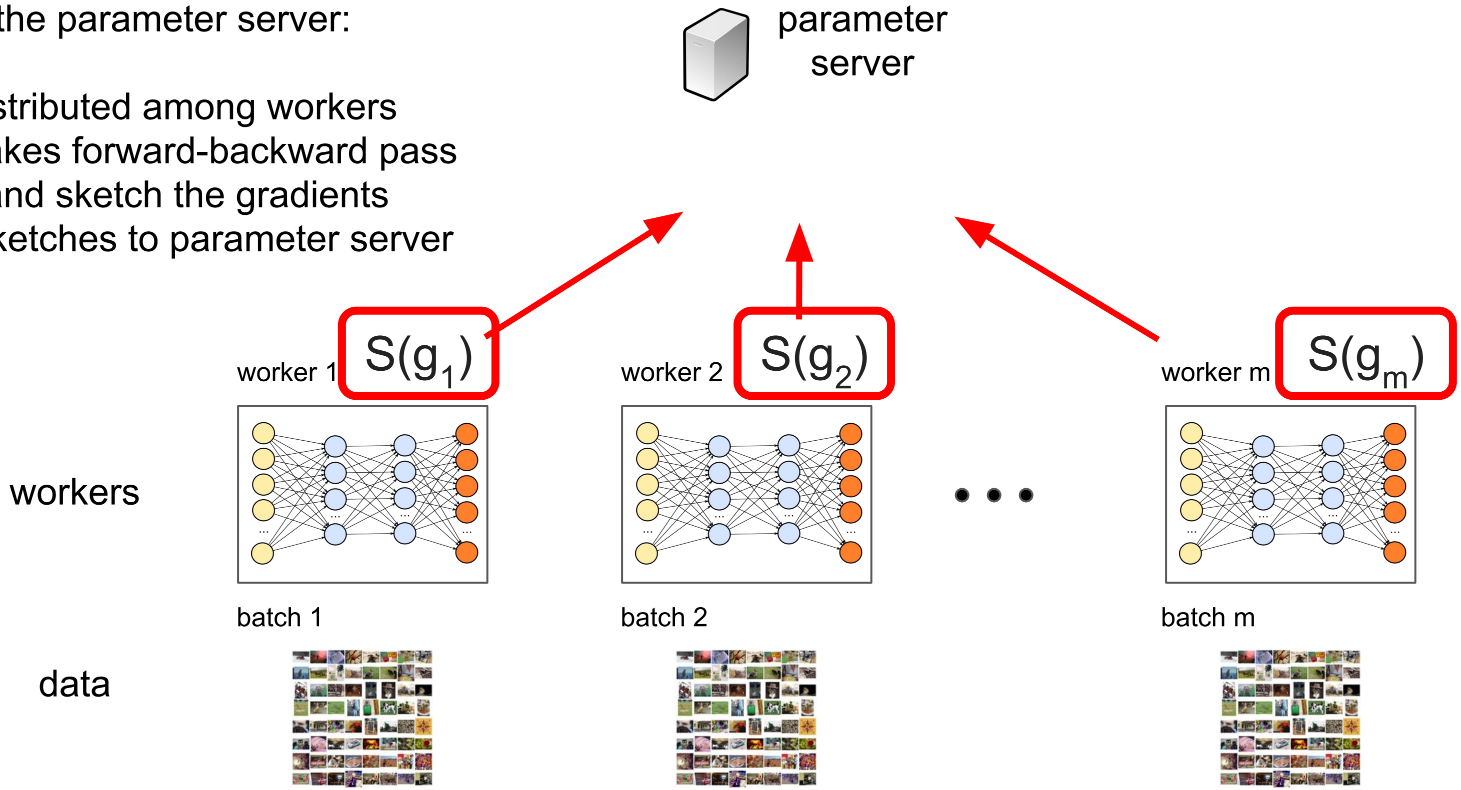
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients



Compression scheme

Synchronization with the parameter server:

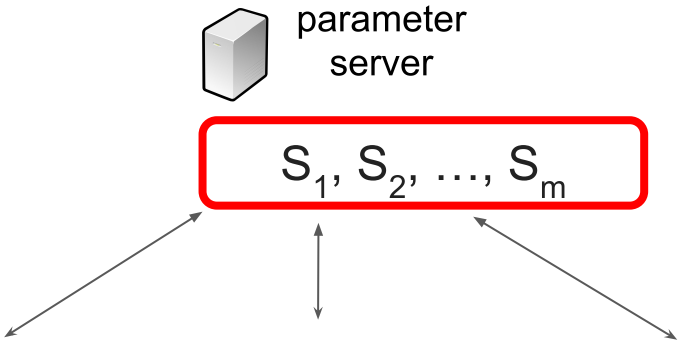
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients
- workers send sketches to parameter server

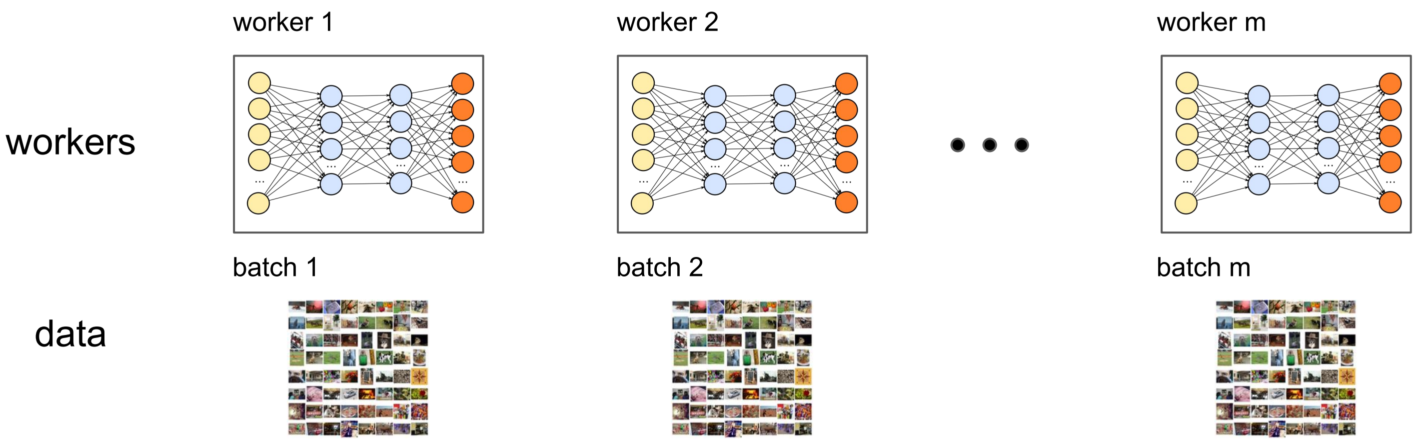


Compression scheme

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients
- workers send sketches to parameter server

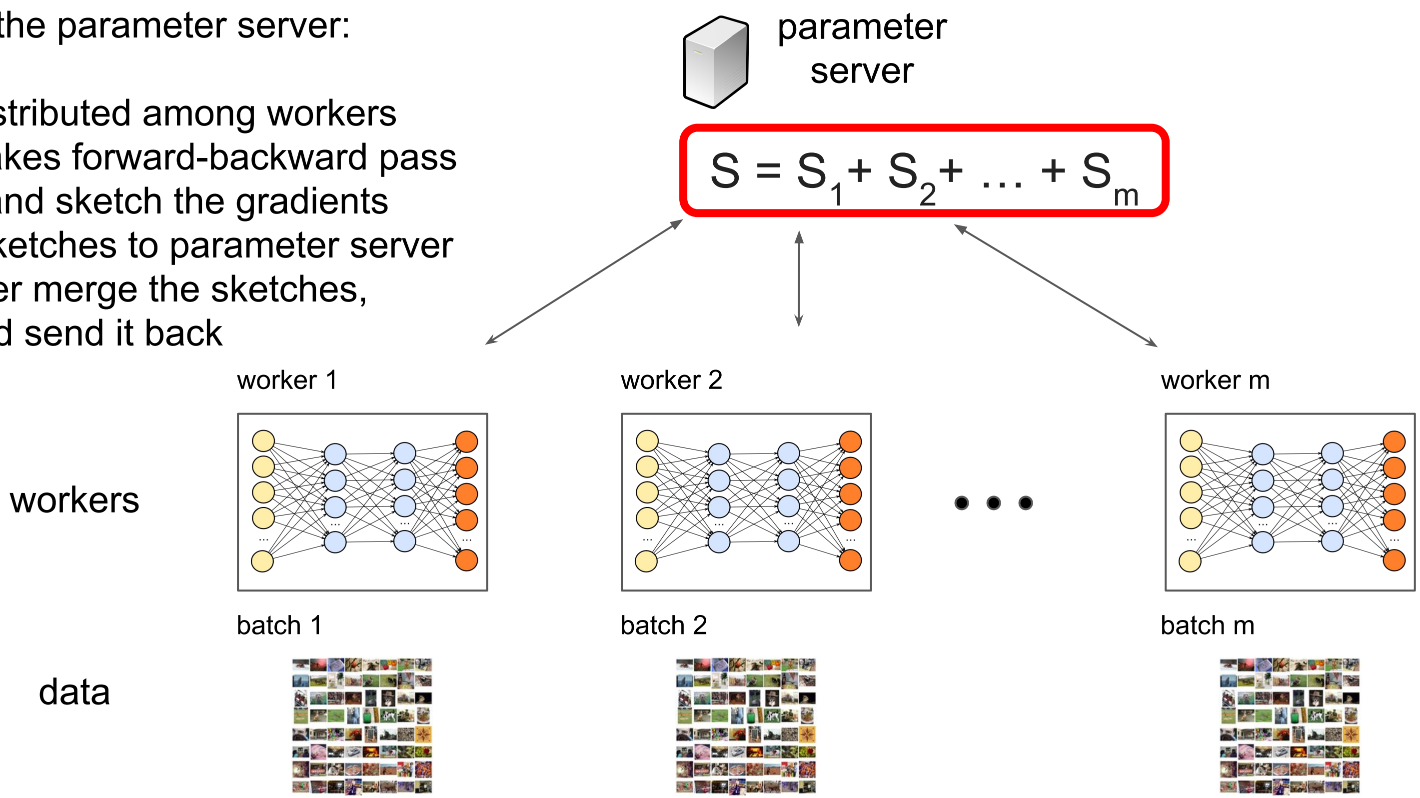




Compression scheme

Synchronization with the parameter server:

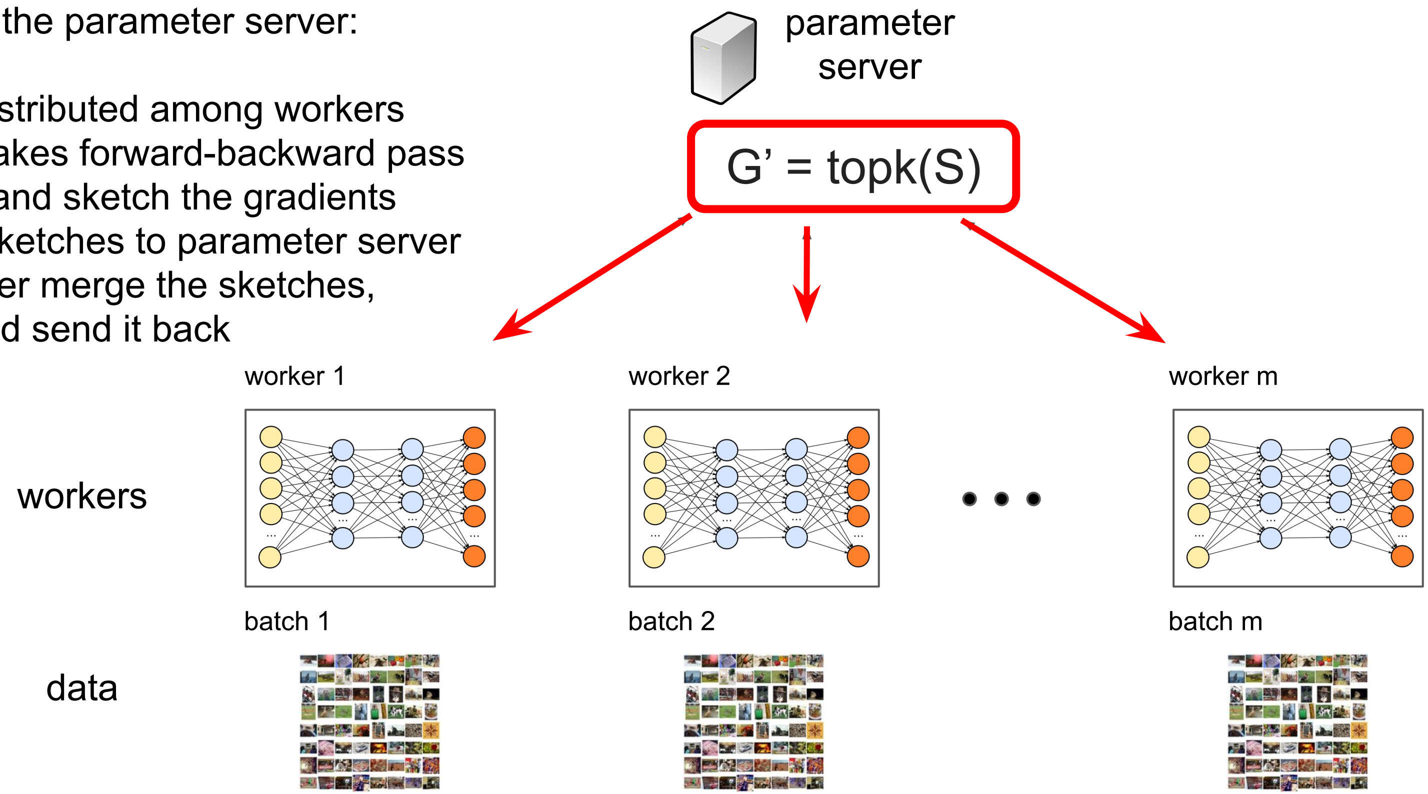
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients
- workers send sketches to parameter server
- parameter server merge the sketches, extract top k and send it back



Compression scheme

Synchronization with the parameter server:

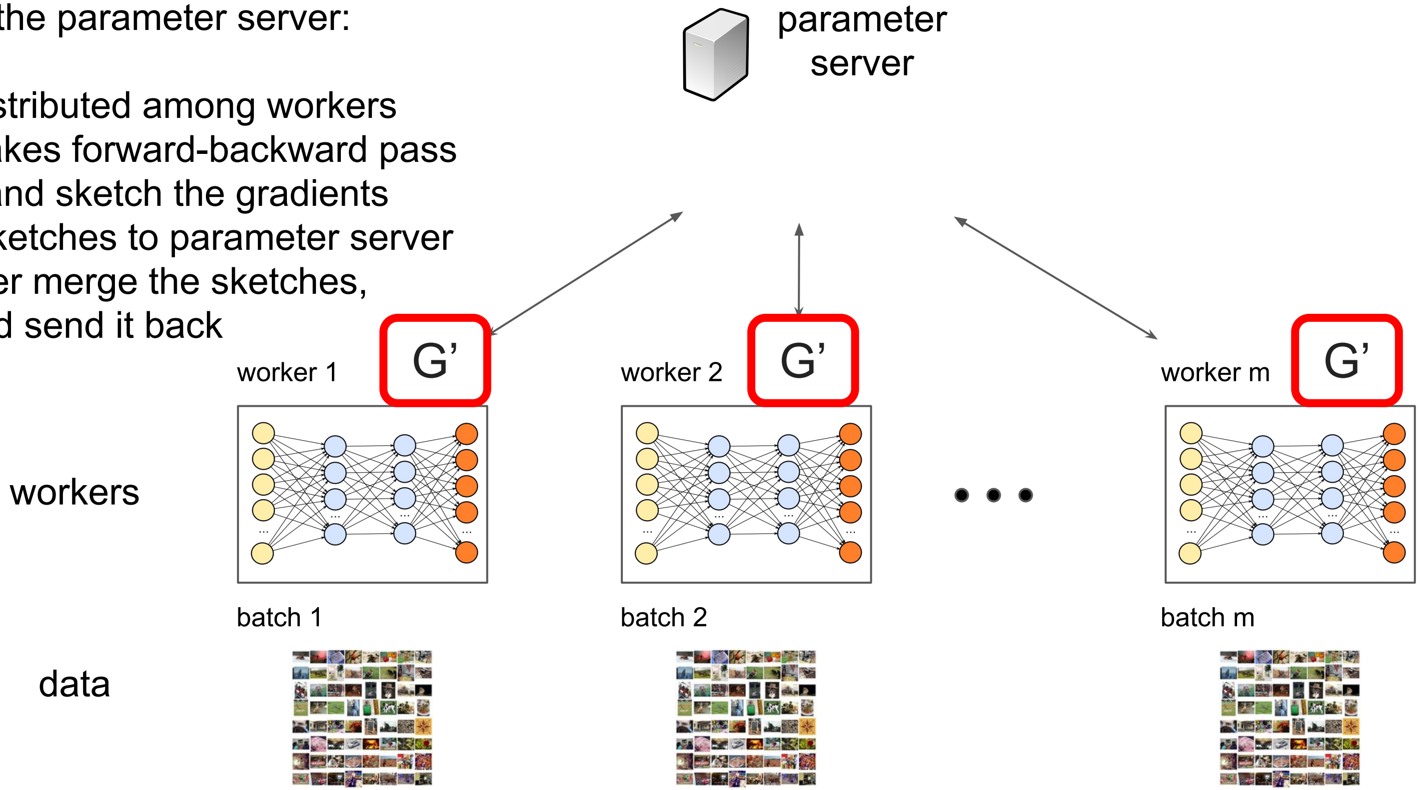
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients
- workers send sketches to parameter server
- parameter server merge the sketches, extract top k and send it back



Compression scheme

Synchronization with the parameter server:

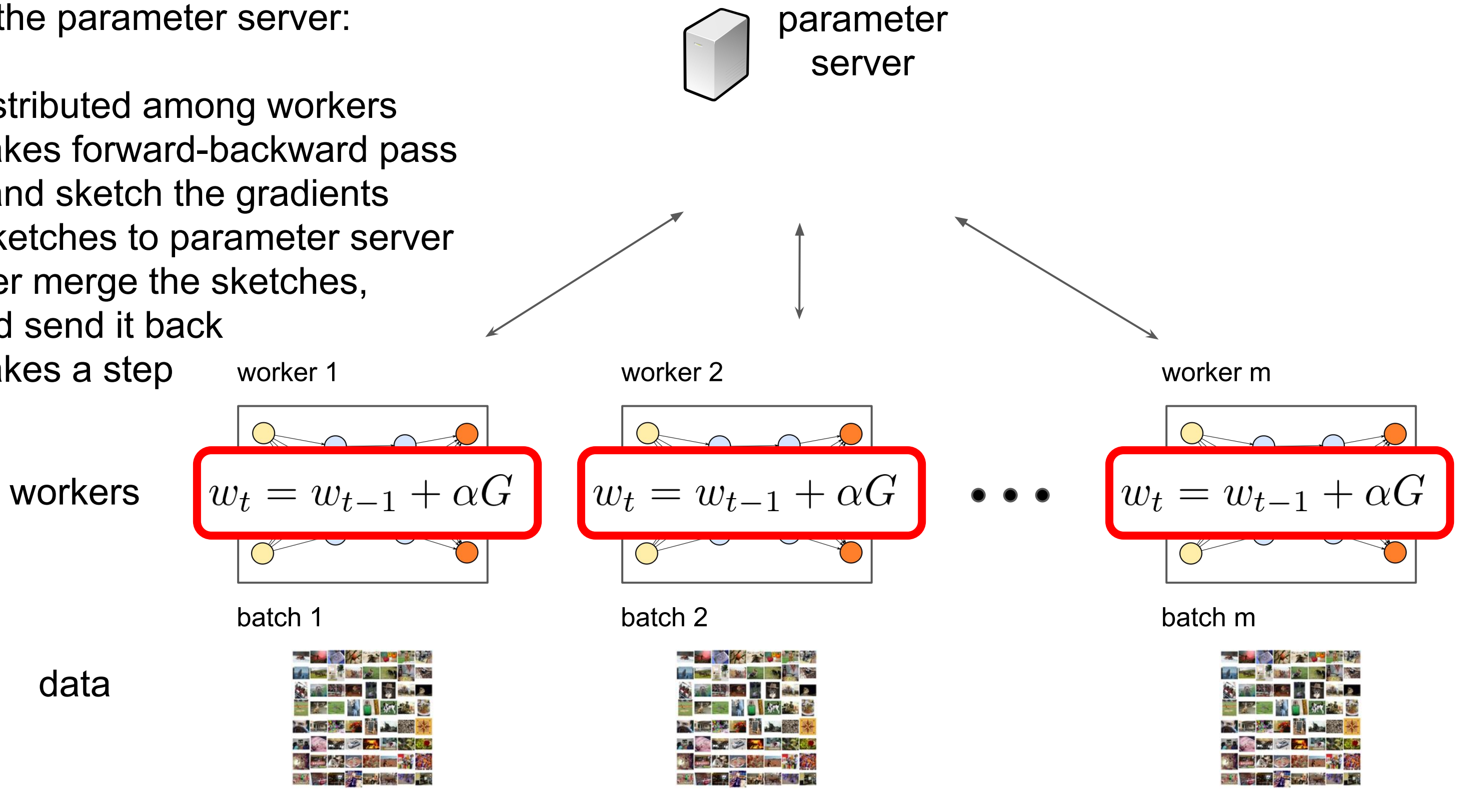
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients
- workers send sketches to parameter server
- parameter server merge the sketches, extract top k and send it back



Compression scheme

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients
- workers send sketches to parameter server
- parameter server merge the sketches, extract top k and send it back
- each worker makes a step



Algorithm and theory

Algorithm 2 SKETCHED-D-SGD

Input: $k, \epsilon, \xi, \delta, W, P$

$\eta_t \leftarrow \frac{1}{t+\xi}, q_t \leftarrow (\xi + t)^2, Q_T = \sum_{t=1}^{T} q_t, \mathrm{a}_0 = 0$
while $t = 1, 2, \cdots T$ **do**
  Compute stochastic gradient g_t^i (Worker$_i$)
  Error correction: $\bar{\mathrm{g}}_t^i = \eta_t \mathrm{g}_t^i + \mathrm{a}_{t-1}^i$ (Worker$_i$)
  Compute sketches S_t^i of $\bar{\mathrm{g}}_t^i$ (Worker$_i$)
  Communicate sketches S_t^i to master (Worker$_i$)
  Aggregate sketches $\mathrm{S}_t = \frac{1}{W} \sum_{i=1}^{W} \mathrm{S}_t^i$ (Master)
  Unsketch: Get top Pk elements from S_t (Master)
  Communicate co-ordinates of Pk elements to all workers and get exact values of top k as $\tilde{\mathrm{g}}_t$ (Master)
  Communicate $\tilde{\mathrm{g}}_t$ to all workers (Master)
  Update $\mathrm{w}_{t+1} = \mathrm{w}_t - \tilde{\mathrm{g}}_t$ (Master)
  Communicate the k updated model parameters of w_{t+1} to all workers (Master)
  Error accumulation: $\mathrm{a}_t^i = \bar{\mathrm{g}}_t^i - \tilde{\mathrm{g}}_t$ (Worker$_i$)
end while

Output: $\hat{\mathrm{w}}_T = \frac{1}{Q_T} \sum_{t=1}^{T} q_t \mathrm{w}_t$

❑ **Theoretical guarantees**

- Converges at *O(1/WT)* rate, at par with SGD for smooth strongly convex functions, where *W* is the number of workers.
- Communicates *O(k log^2 d)*, size of sketch, *0< k < d*, *d*: dimension of model.

❑ **Scalability**

- More workers - Increasing the number of workers W **increases** the rate of convergence (suitable for *Federated learning*)
- Bigger models - Increasing the model size d **increases** the compression ratio d/k log^2 d.

Empirical Results

90M

70M

	BLEU (transformer)	BLEU (LSTM)
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.

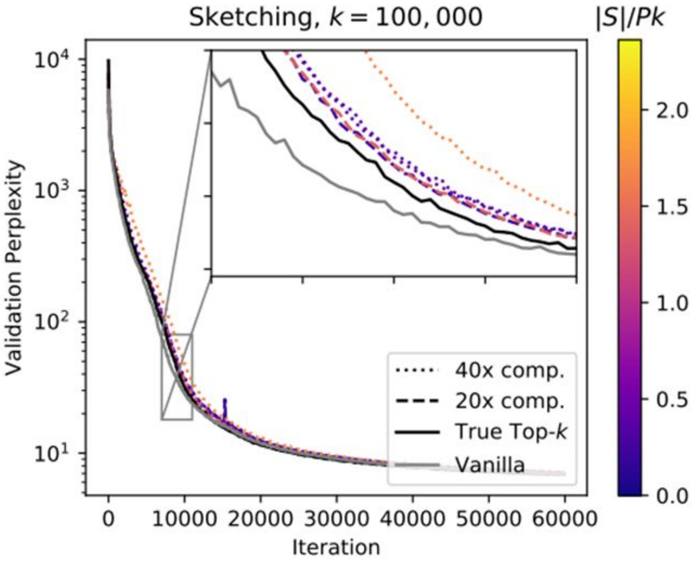
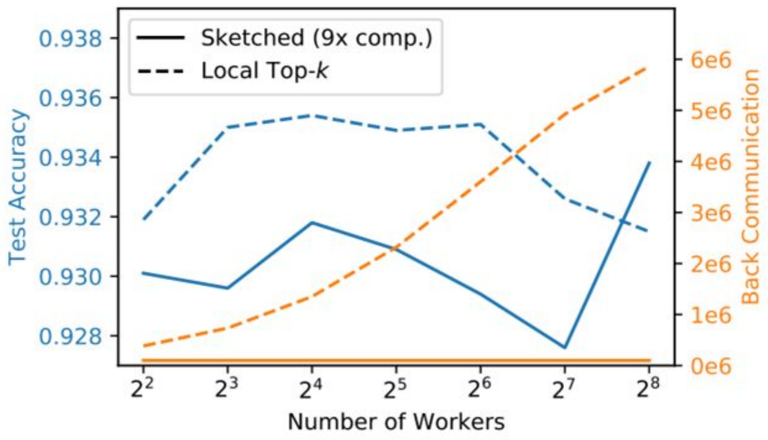


Figure 1: Learning curves for a transformer model trained on the WMT 2014 English to German translation task. All models included here achieve comparable BLEU scores after 60,000 iterations (see Table 1). Each run used 4 workers.

Empirical Results

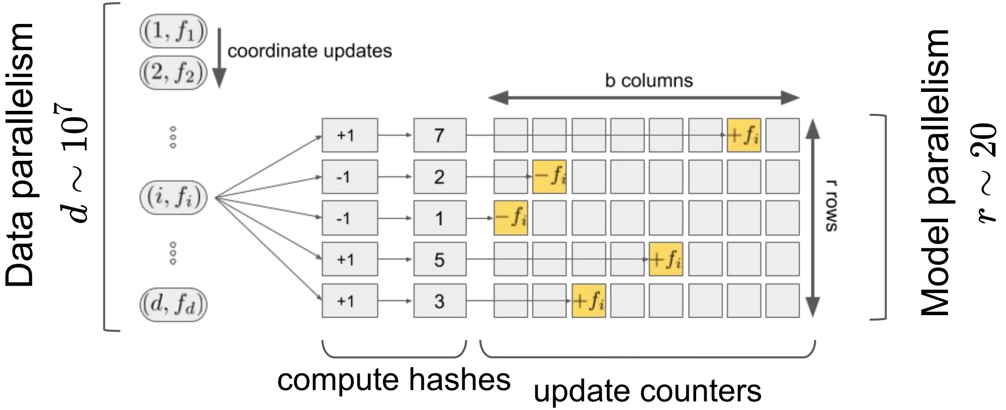


Comparison between SKETCHED-SGD and local top-k SGD on CIFAR10.
The best overall compression that local top-k can achieve for many workers is 2x.

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)
```

Thanks a lot!