

# Synchronous Stochastic Gradient

# Outline

- Basics
- Comparison against asynchronous SGD
- Mitigating stragglers
- Enabling larger mini-batch sizes
- Decreasing communication complexity

# Distributed SGD – Averaging Estimates

---

**Algorithm 3** SimuParallelSGD(Examples  $\{c^1, \dots, c^m\}$ , Learning Rate  $\eta$ , Machines  $k$ )

---

Define  $T = \lfloor m/k \rfloor$

Randomly partition the examples, giving  $T$  examples to each machine.

**for all**  $i \in \{1, \dots, k\}$  **parallel do**

    Randomly shuffle the data on machine  $i$ .

    Initialize  $w_{i,0} = 0$ .

**for all**  $t \in \{1, \dots, T\}$ : **do**

        Get the  $t$ th example on the  $i$ th machine (this machine),  $c^{i,t}$

$w_{i,t} \leftarrow w_{i,t-1} - \eta \partial_w c^i(w_{i,t-1})$

**end for**

**end for**

Aggregate from all computers  $v = \frac{1}{k} \sum_{i=1}^k w_{i,t}$  and **return**  $v$ .

---

**Advantage:** Needs only one round of communication. Works well for convex models.

**Disadvantage:** For non-convex models, averaging different local minima doesn't make sense.

# Distributed SGD – Averaging Gradients

## Algorithm 1: Async-SGD worker $k$

---

**Input:** Dataset  $\mathcal{X}$

**Input:**  $B$  mini-batch size

```
1 while True do
2   | Read  $\hat{\theta}_k = (\theta[0], \dots, \theta[M])$  from PS.
3   |  $G_k^{(t)} := 0$ .
4   | for  $i = 1, \dots, B$  do
5   |   | Sample datapoint  $\tilde{x}_i$  from  $\mathcal{X}$ .
6   |   |  $G_k^{(t)} \leftarrow G_k^{(t)} + \frac{1}{B} \nabla F(\tilde{x}_i; \hat{\theta}_k)$ .
7   | end
8   | Send  $G_k^{(t)}$  to parameter servers.
9 end
```

## Algorithm 2: Async-SGD Parameter Server $j$

---

**Input:**  $\gamma_0, \gamma_1, \dots$  learning rates.

**Input:**  $\alpha$  decay rate.

**Input:**  $\theta^{(0)}$  model initialization.

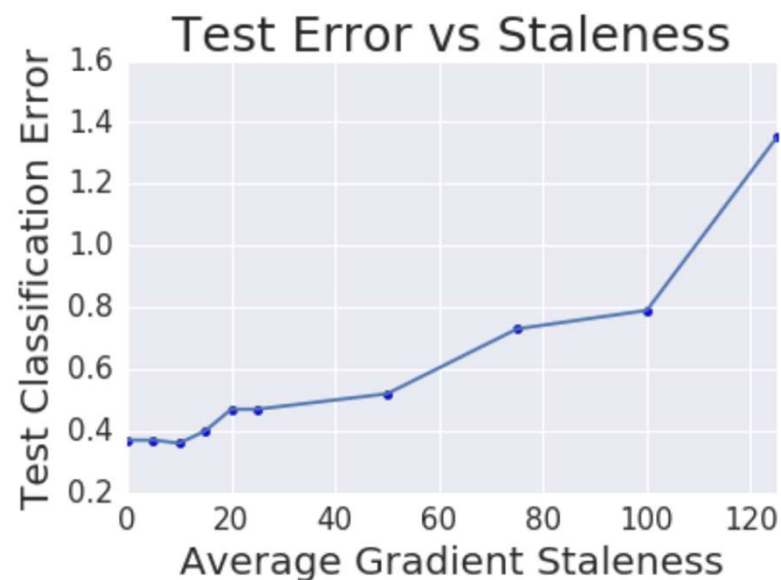
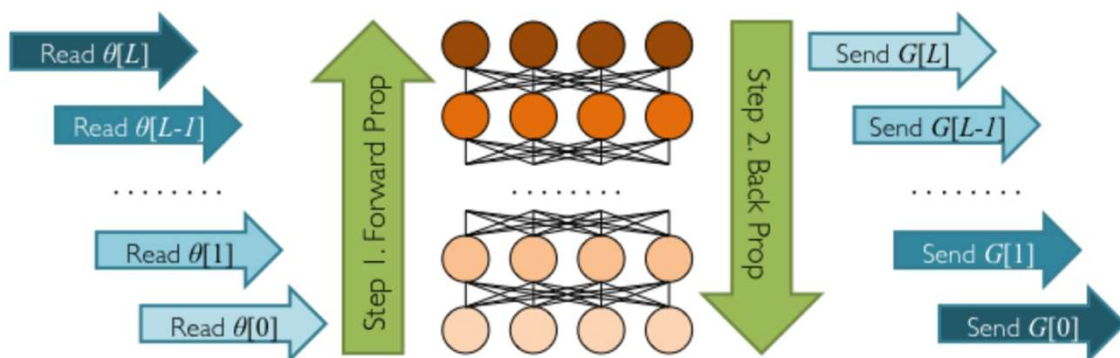
```
1 for  $t = 0, 1, \dots$  do
2   | Wait for gradient  $G$  from any worker.
3   |  $\theta^{(t+1)}[j] \leftarrow \theta^{(t)}[j] - \gamma_t G[j]$ .
4   |  $\bar{\theta}^{(t)}[j] = \alpha \bar{\theta}^{(t-1)}[j] + (1 - \alpha) \theta^{(t)}[j]$ .
5 end
```

# Comparison against asynchronous SGD

- ☺ Do not need to worry about stale gradients
  - ☺ Do not need to set a smaller step-size compared to simple SGD
  - => Will lead to faster (in terms of number of epochs) convergence
- 
- ☹ Need to wait for the slowest machine (“straggler”) for each update
  - ☹ Poor robustness to machine failure

# Comparison against asynchronous SGD

- **Staleness:** number of updates that have occurred between its corresponding read and update operations.

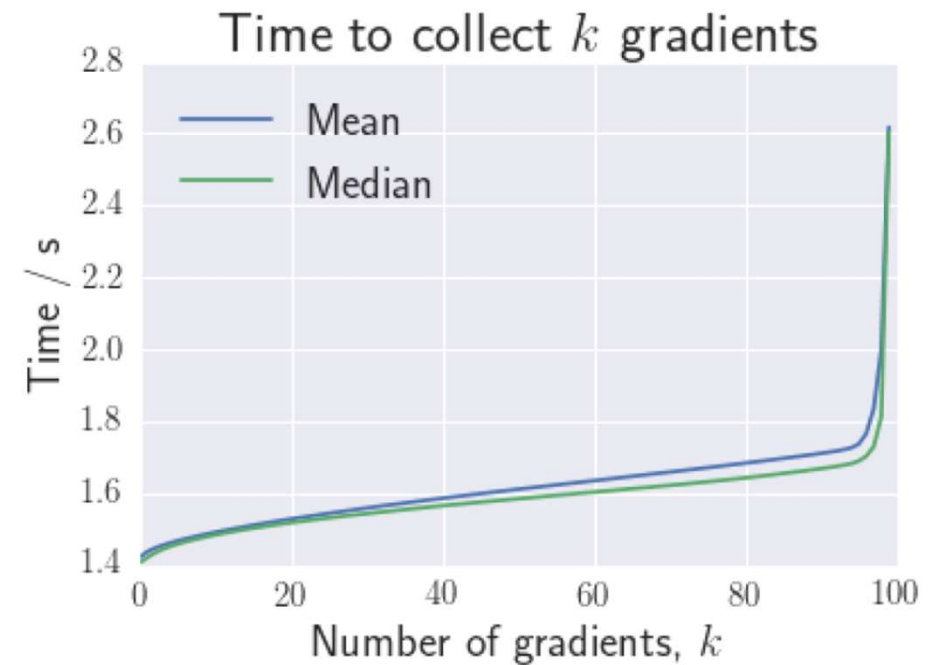
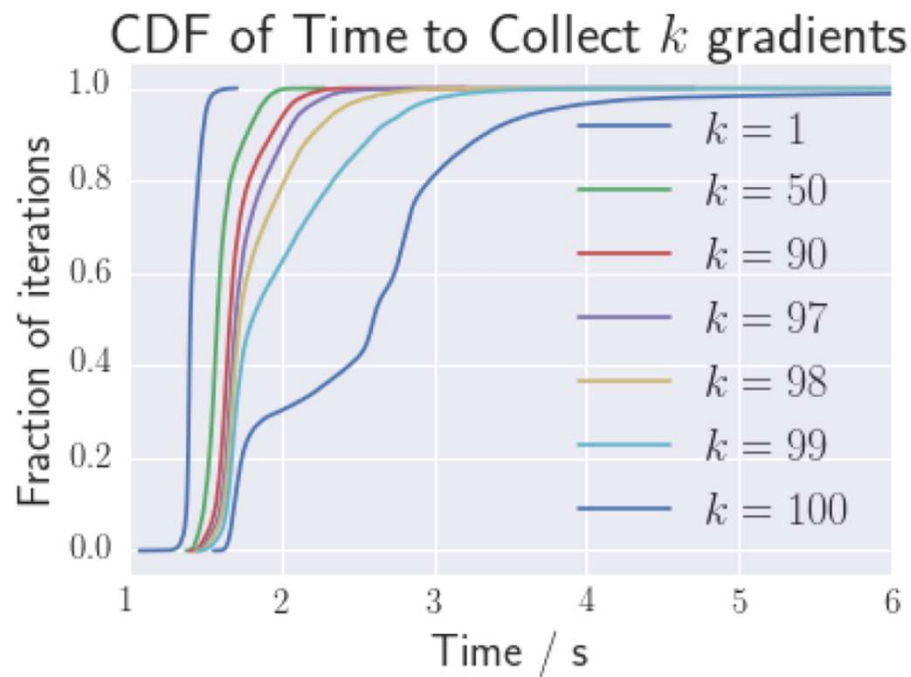


Tricks to make Asynchronous SGD work:

1. Slowly increase the number of workers over the first 3 epochs of training
2. Use lower initial learning rates

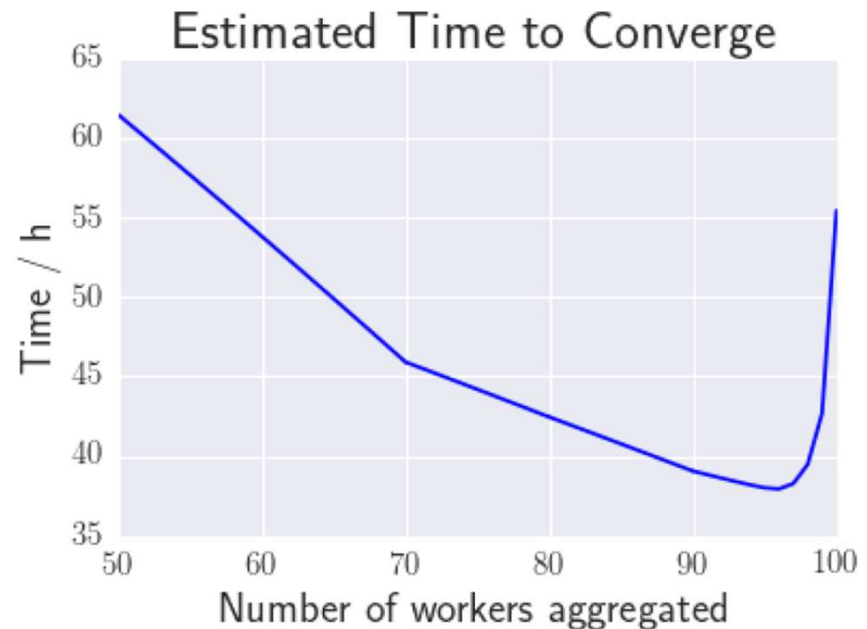
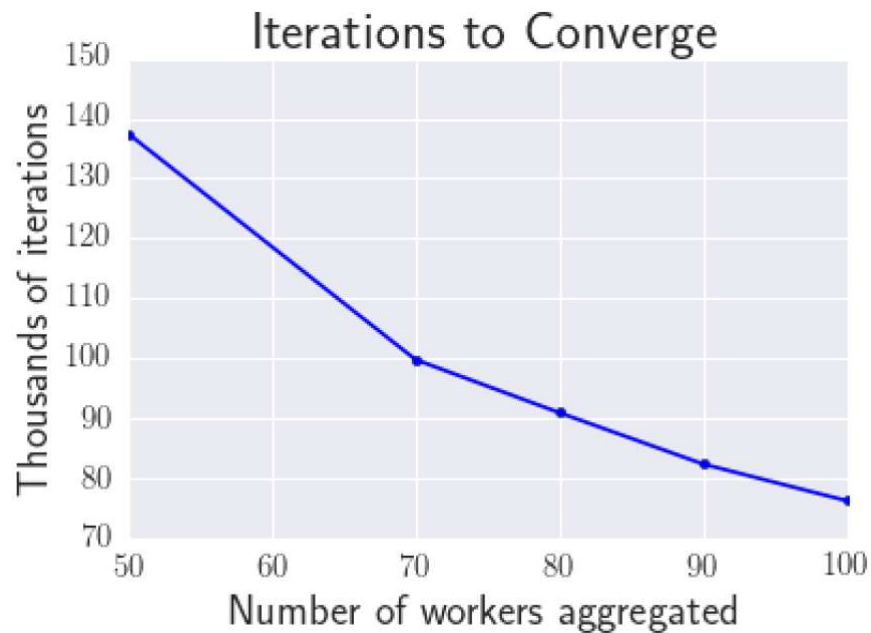
# Synchronous SGD - Problem

Few stragglers slow down the algorithm!



# Solution 1

**Basic Idea:** Drop the gradients of the slow workers





# Solution 2

**Basic Idea:** Use backup workers

---

**Algorithm 3:** Sync-SGD worker  $k$ , where  $k = 1, \dots, N + b$

---

**Input:** Dataset  $\mathcal{X}$

**Input:**  $B$  mini-batch size

```
1 for  $t = 0, 1, \dots$  do
2   | Wait to read  $\theta^{(t)} = (\theta^{(t)}[0], \dots, \theta^{(t)}[M])$ 
   | from parameter servers.
3   |  $G_k^{(t)} := 0$ 
4   | for  $i = 1, \dots, B$  do
5   |   | Sample datapoint  $\tilde{x}_{k,i}$  from  $\mathcal{X}$ .
6   |   |  $G_k^{(t)} \leftarrow G_k^{(t)} + \frac{1}{B} \nabla F(\tilde{x}_{k,i}; \theta^{(t)})$ .
7   | end
8   | Send  $(G_k^{(t)}, t)$  to parameter servers.
9 end
```

---

**Algorithm 4:** Sync-SGD Parameter Server  $j$

---

**Input:**  $\gamma_0, \gamma_1, \dots$  learning rates.

**Input:**  $\alpha$  decay rate.

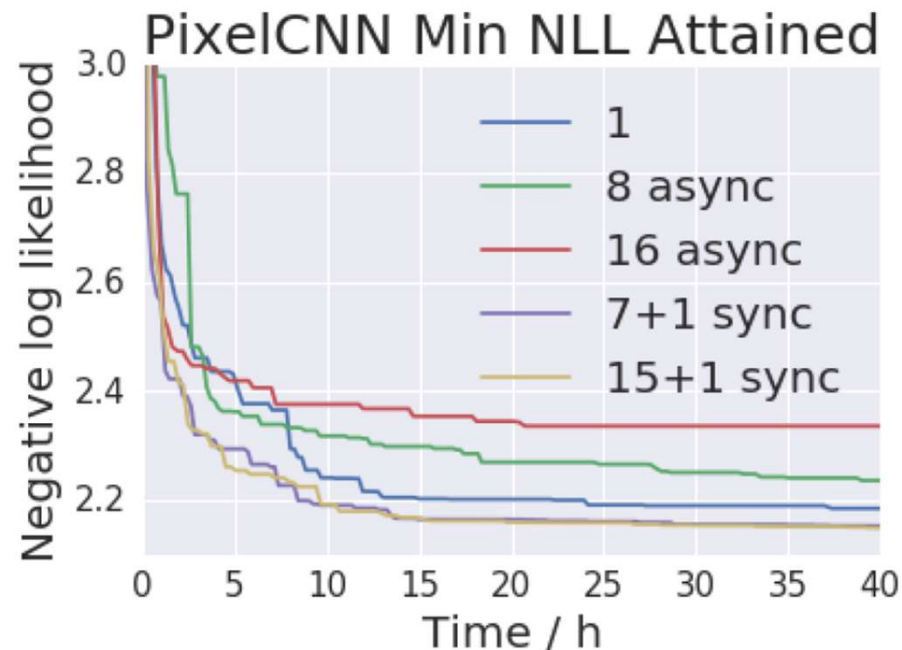
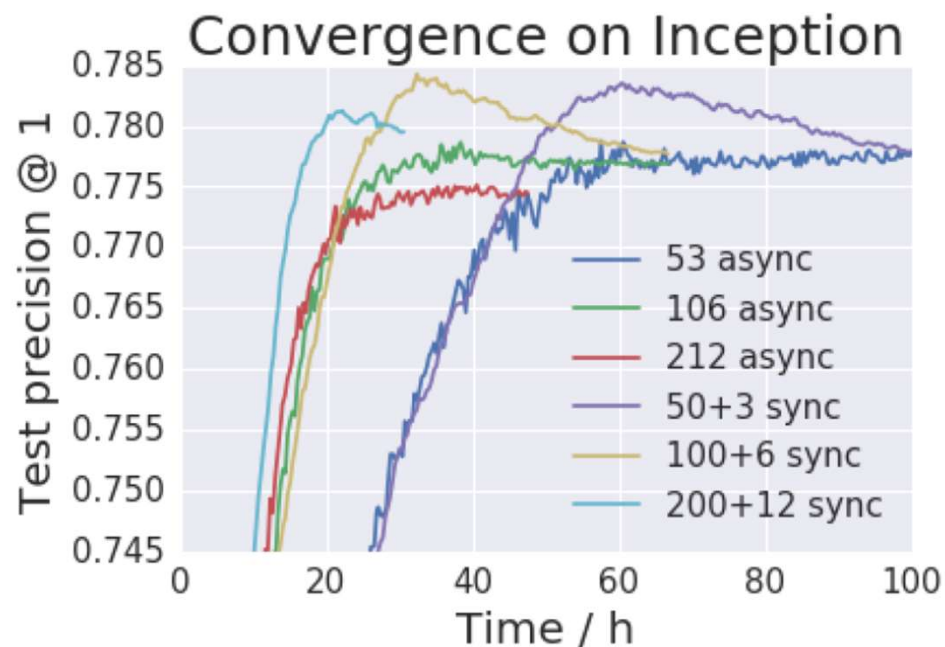
**Input:**  $N$  number of mini-batches to aggregate.

**Input:**  $\theta^{(0)}$  model initialization.

```
for  $t = 0, 1, \dots$  do
  |  $\mathcal{G} = \{\}$ 
  | while  $|\mathcal{G}| < N$  do
  |   | Wait for  $(G, t')$  from any worker.
  |   | if  $t' == t$  then  $\mathcal{G} \leftarrow \mathcal{G} \cup \{G\}$ .
  |   | else Drop gradient  $G$ .
  | end
  |  $\theta^{(t+1)}[j] \leftarrow \theta^{(t)}[j] - \frac{\gamma_t}{N} \sum_{G \in \mathcal{G}} G[j]$ .
  |  $\bar{\theta}^{(t)}[j] = \alpha \bar{\theta}^{(t-1)}[j] + (1 - \alpha) \theta^{(t)}[j]$ .
end
```

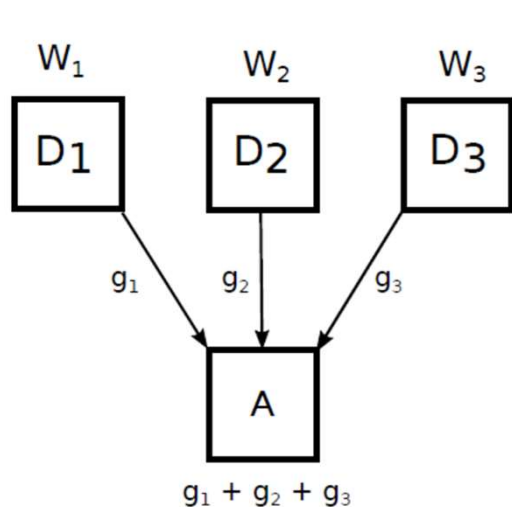
---

# Experimental Comparison

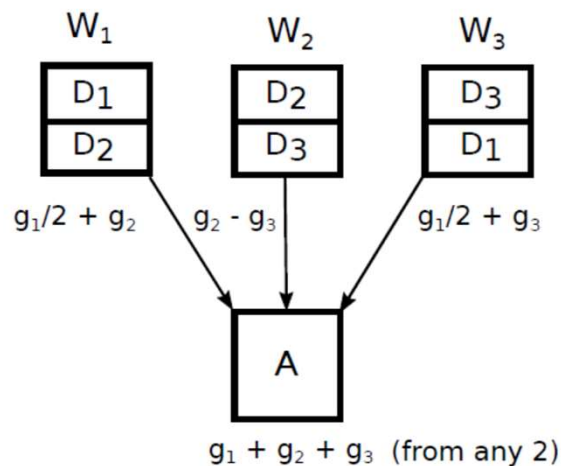


# Alternate solution

Use coding theory + data redundancy to always ensure that we get the full gradient and ensure robustness to “some” number of stragglers



(a) Naive synchronous gradient descent



(b) Gradient coding: The vector  $g_1 + g_2 + g_3$  is in the span of *any two* out of the vectors  $g_1/2 + g_2$ ,  $g_2 - g_3$  and  $g_1/2 + g_3$ .

# Improving synchronous SGD

Need larger batch-sizes to get the full benefit of parallelism

Generalization performance  
decreases with higher batch-size

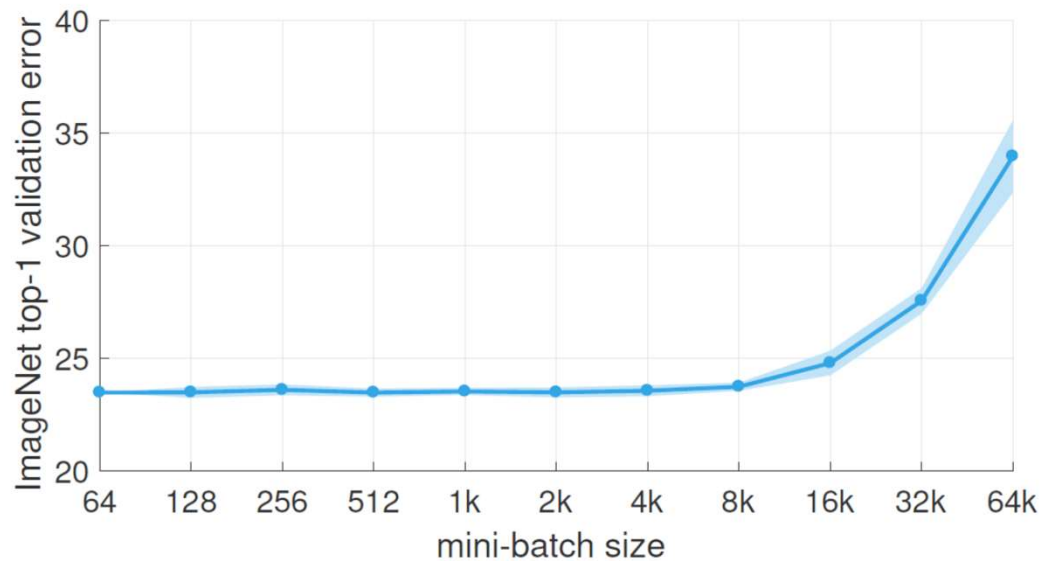


Figure 1. **ImageNet top-1 validation error vs. minibatch size.**

# Synchronous SGD with large batch-size

Competing(?) hypotheses: optimization difficulty vs poor generalization due to convergence to sharp minima

Today: Evidence for optimization difficulty and correcting for it

**Solution:**

***Linear Scaling Rule:** When the minibatch size is multiplied by  $k$ , multiply the learning rate by  $k$ .*

(Also has theoretical evidence + multiple other sources)

**In practice:** Breaks down when the network is changing rapidly, which commonly occurs in early stages of training.

**Hack:** “Warmup” - Gradually ramp up the learning rate from a small to a large value across the first 5 epochs

# Synchronous SGD with large batch-size

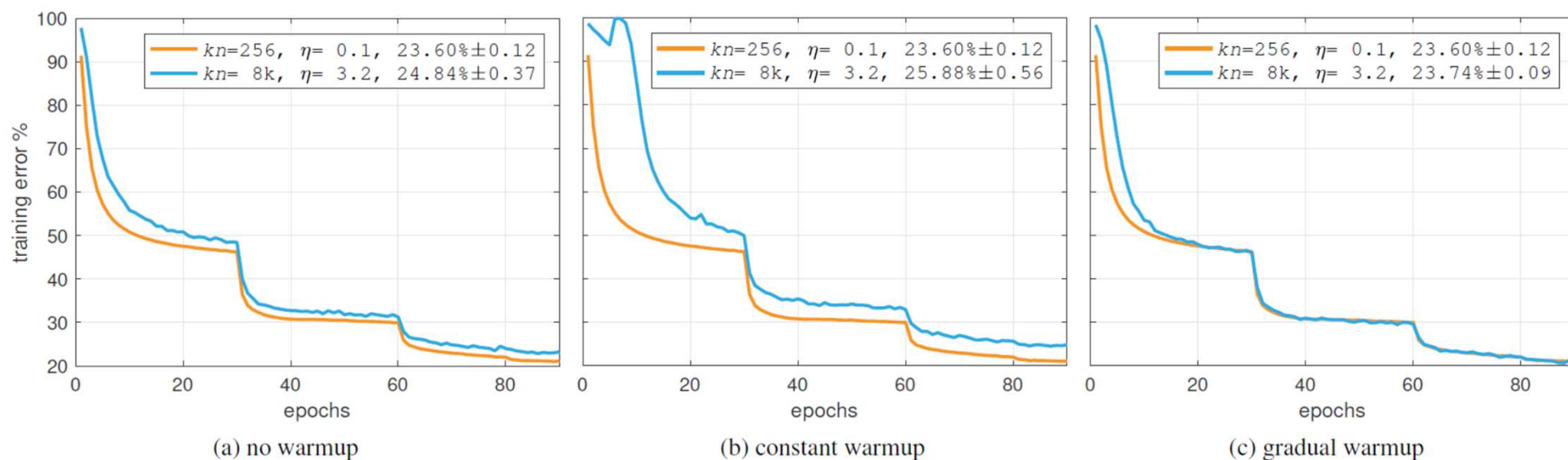
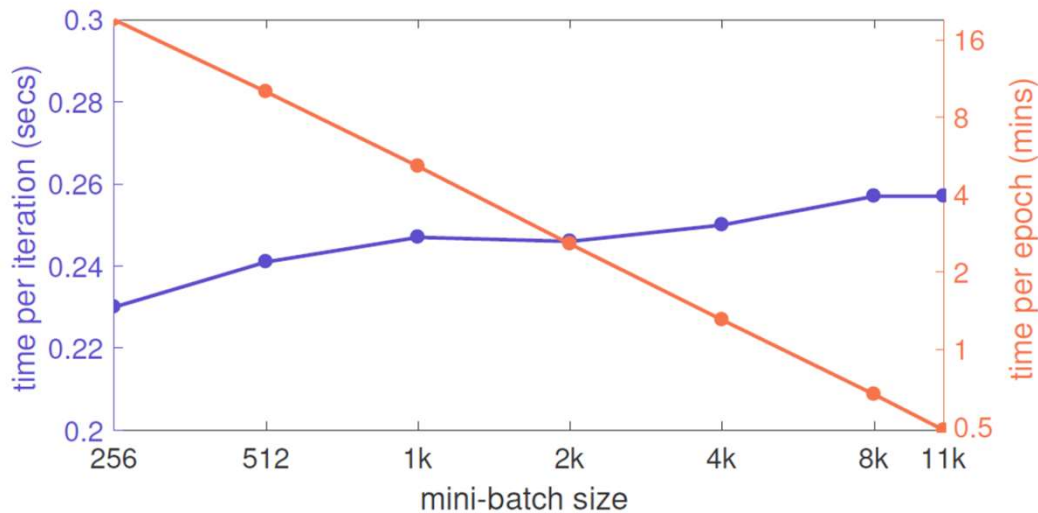


Figure 2. **Warmup.** Training error curves for minibatch size 8192 using various warmup strategies compared to minibatch size 256. Validation error (mean  $\pm$  std of 5 runs) is shown in the legend, along with minibatch size  $kn$  and reference learning rate  $\eta$ .

**Note:** Fails beyond batch-size of 8K.

# Synchronous SGD with large batch-size



“With 352 GPUs (44 servers) our implementation completes one pass over all 1.28 million ImageNet training images in about 30 seconds”

“No generalization issues when transferring across datasets and across tasks using models trained with large minibatches.”

# Decreasing communication complexity

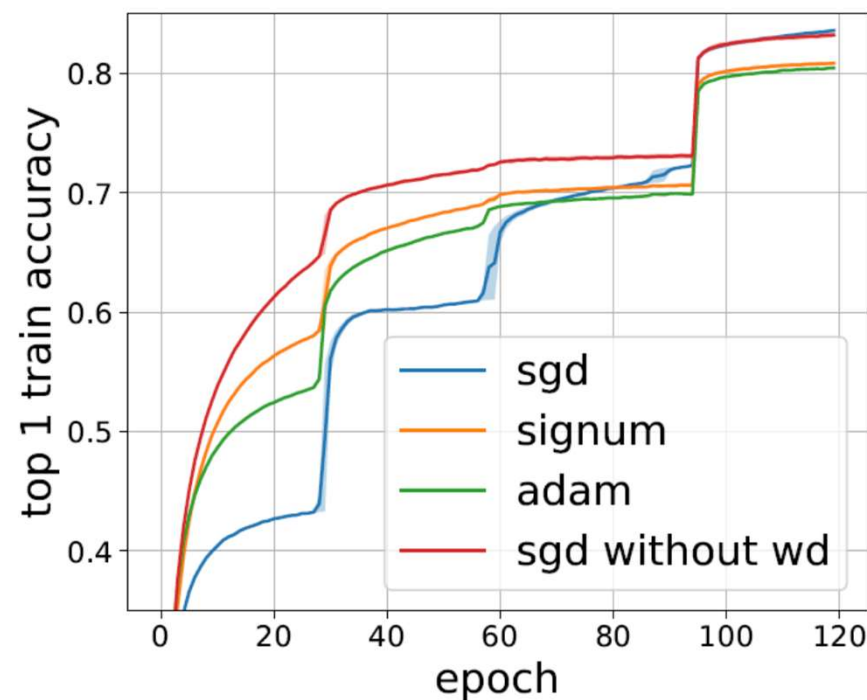
**Basic Idea:** Transfer just the signs of gradients

## Algorithm 1: SIGNSGD

**Input:** learning rate  $\delta$ , current point  $x_k$   
 $\tilde{g}_k \leftarrow \text{stochasticGradient}(x_k)$   
 $x_{k+1} \leftarrow x_k - \delta \text{sign}(\tilde{g}_k)$

## Algorithm 2: SIGNSGD with majority vote

**Input:** learning rate  $\delta$ , current point  $x_k$ , # workers  $M$  each with an independent gradient estimate  $\tilde{g}_m(x_k)$   
**on server**  
    **pull**  $\text{sign}(\tilde{g}_m)$  **from** each worker  
    **push**  $\text{sign}\left[\sum_{m=1}^M \text{sign}(\tilde{g}_m)\right]$  **to** each worker  
**on each worker**  
     $x_{k+1} \leftarrow x_k - \delta \text{sign}\left[\sum_{m=1}^M \text{sign}(\tilde{g}_m)\right]$





# Conclusion

- Synchronous SGD is simple and typically works better (both in terms of time and performance) than asynchronous SGD.
- There are some ways to mitigate the effect of stragglers.
- To utilize the full power of the hardware, we need to enable training with large mini-batch sizes. Increasing the learning rate with large batches leads to fast convergence without any loss in performance.
- We can reduce the communication complexity by compressing the gradients or using just their signs.

# References

- Zinkevich, Martin, et al. "Parallelized stochastic gradient descent." *Advances in neural information processing systems*. 2010.
- Chen, Jianmin, et al. "Revisiting distributed synchronous SGD." *arXiv preprint arXiv:1604.00981* (2016).
- Tandon, Rashish, et al. "Gradient coding: Avoiding stragglers in distributed learning." *International Conference on Machine Learning*. 2017.
- Goyal, Priya, et al. "Accurate, large minibatch SGD: training imagenet in 1 hour." *arXiv preprint arXiv:1706.02677* (2017).
- Bernstein, Jeremy, et al. "Compression by the signs: distributed learning is a two-way street." (2018).