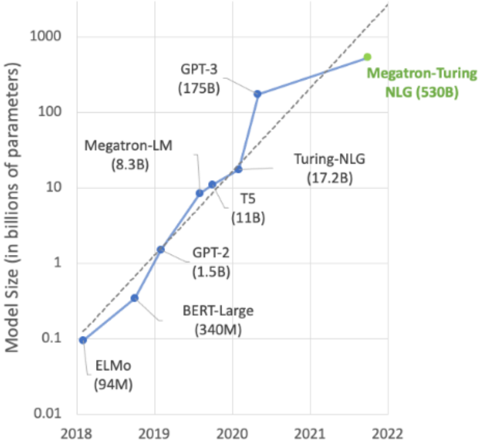
Distributed Deep Learning

Slides by Amir H. Payberah and Jim Dowling

The need for Distributed Training of DNNs

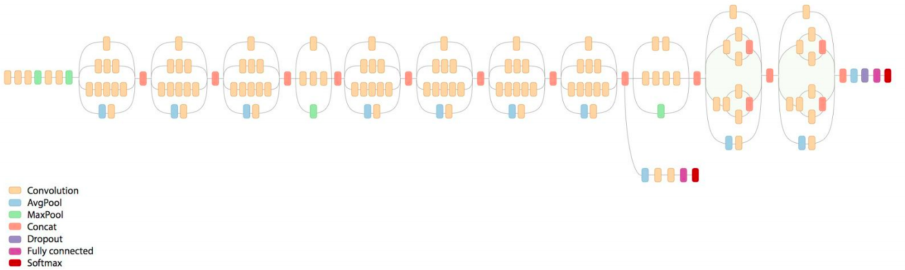
Growth in the Size of Deep Neural Networks



Nvidia Link

Training Deep Neural Networks

- Computationally intensive
- Time consuming

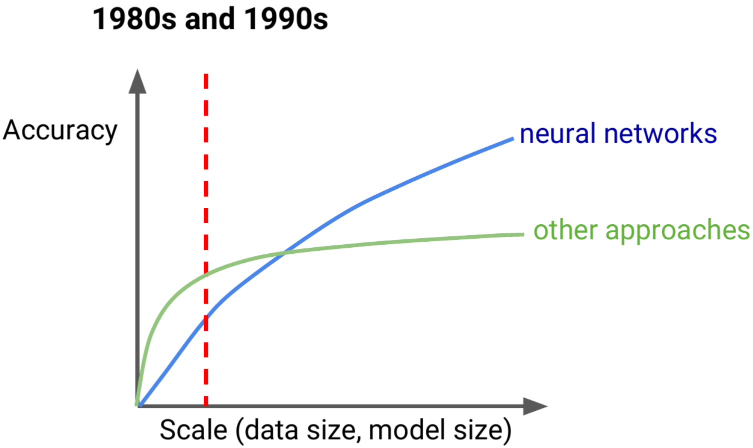


[https://cloud.google.com/tpu/docs/images/inceptionv3onc--oview.png]

Why?

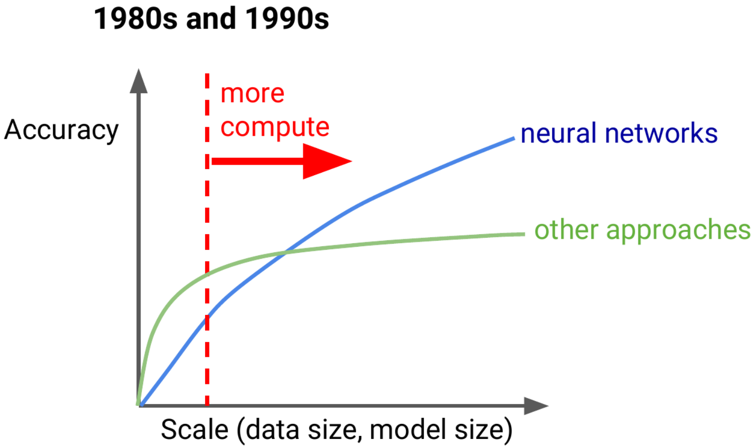
- Massive amount of training dataset
- Large number of parameters

Accuracy vs. Data/Model Size



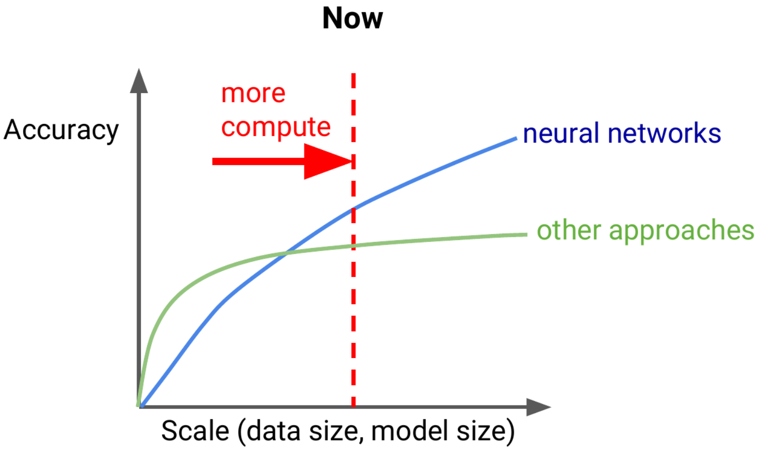
[Jeff Dean at AI Frontiers: Trends and Developments in Deep Learning Research]

Accuracy vs. Data/Model Size



[Jeff Dean at AI Frontiers: Trends and Developments in Deep Learning Research]

Accuracy vs. Data/Model Size

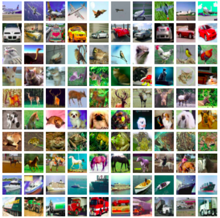


[Jeff Dean at AI Frontiers: Trends and Developments in Deep Learning Research]

Fundamentals of Machine Learning

Training Dataset

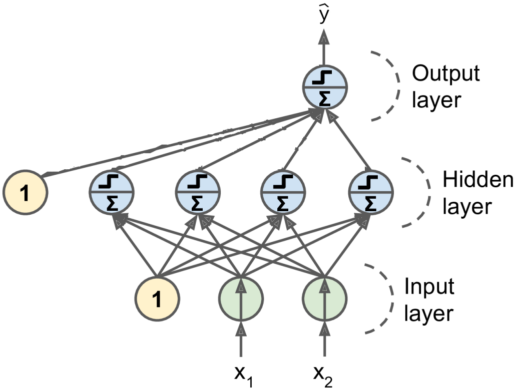
- E.g., tabular data, image, text, etc.



Date	Cost	Actions	Offsite conversions	Impressions	Clicks
2017-04-04	29.44	461	4	5655	477
2017-04-03	74.08	1331	16	18170	1340
2017-04-02	76.09	1349	12	16877	1357
2017-04-01	76.79	1382	8	19757	1378
2017-03-31	77.28	1141	21	18598	1116
2017-03-30	68.62	1065	18	14847	1046
2017-03-29	64.9	1111	25	13994	1094
2017-03-28	65.12	1137	12	15952	1145
2017-03-27	66.98	1185	7	17970	1190
2017-03-26	64.94	1118	5	14410	1116
2017-03-25	66.3	1208	6	15123	1204
2017-03-24	67.38	1143		15298	1159
2017-03-23	65.59	1147	13	14972	1143
2017-03-22	68.19	1129	4	17959	1116
2017-03-21	64.78	1081		25810	1059

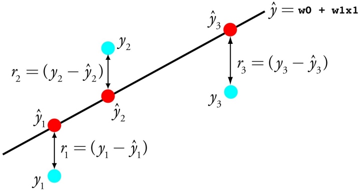
Model

- E.g., linear models, neural networks, etc.
- $\hat{y} = f_w(\mathbf{x})$



Loss function

- How good $\hat{y}$ is able to predict the expected outcome y.
- $J(\mathbf{w}) = \sum_{i=1}^{m} l(y_i, \hat{y}_i)$



- E.g., $J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$

Objective

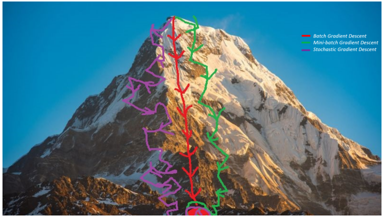
- Minimize the loss function
- $\arg\min_{\mathbf{w}} J(\mathbf{w})$
- $J(\mathbf{w}) = \sum_{i=1}^{m} l(y_i, \hat{y}_i)$

Training

- $\mathtt{J}(\mathbf{w}) = \sum_{\mathtt{i}=1}^{\mathtt{m}} \mathtt{l}(\mathtt{y_i}, \hat{\mathtt{y}}_\mathtt{i})$

- Gradient descent, i.e., $\mathbf{w} := \mathbf{w} - \eta \nabla \mathtt{J}(\mathbf{w})$

- Stochastic gradient descent, i.e., $\mathbf{w} := \mathbf{w} - \eta \tilde{\mathtt{g}} \mathtt{J}(\mathbf{w})$
  - $\tilde{\mathtt{g}}$: gradient at a randomly chosen point.



- Mini-barch gradient descent, i.e., $\mathbf{w} := \mathbf{w} - \eta \tilde{\mathtt{g}}_\mathtt{B} \mathtt{J}(\mathbf{w})$
  - $\tilde{\mathtt{g}}$: gradient with respect to a set of $\mathtt{B}$ randomly chosen points.

Let's Scale the Learning

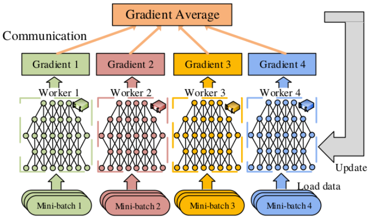
Scalable Training

- Data parallelism
- Model parallelism

Data Parallelism

Data Parallelization (1/4)

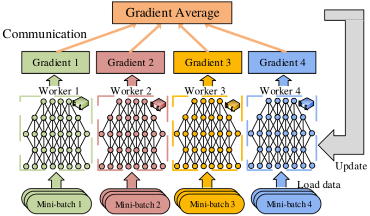
- Replicate a whole model on every device.
- Train all replicas simultaneously, using a different mini-batch for each.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (2/4)

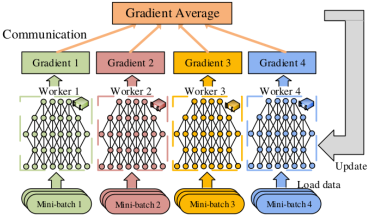
- k devices
- $J_j(\mathbf{w}) = \sum_{i=1}^{b_j} l(y_i, \hat{y}_i), \forall j = 1, 2, \cdots, k$
- $\tilde{g}_B J_j(\mathbf{w})$: gradient of $J_j(\mathbf{w})$ with respect to a set of B randomly chosen points at device j.
- Compute $\tilde{g}_B J_j(\mathbf{w})$ on each device j.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (3/4)

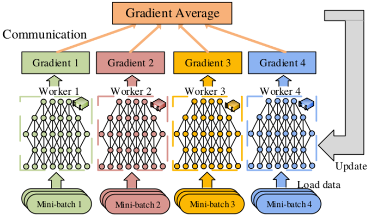
- Compute the mean of the gradients.
- $\tilde{g}_B J(\mathbf{w}) = \frac{1}{k} \sum_{j=1}^{k} \tilde{g}_B J_j(\mathbf{w})$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization (4/4)

- Update the model.
- $\mathbf{w} := \mathbf{w} - \eta \tilde{\mathrm{g}}_{\mathrm{B}} \mathrm{J}(\mathbf{w})$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Data Parallelization Design Issues

- The aggregation algorithm
- Communication synchronization and frequency
- Communication compression

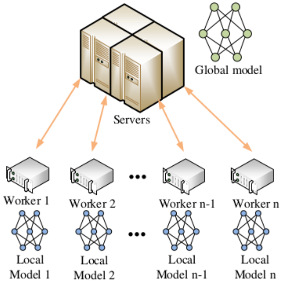
The Aggregation Algorithm

The Aggregation Algorithm

- How to aggregate gradients (compute the mean of the gradients)?
- Centralized - parameter server
- Decentralized - all-reduce
- Decentralized - gossip

Aggregation - Centralized - Parameter Server

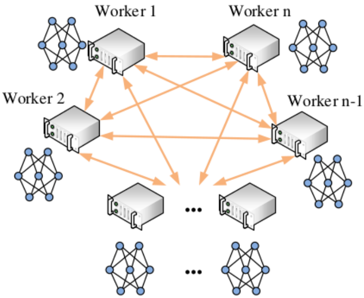
- Store the model parameters outside of the workers.
- Workers periodically report their computed parameters or parameter updates to a (set of) parameter server(s) (PSs).



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Aggregation - Distributed - All-Reduce

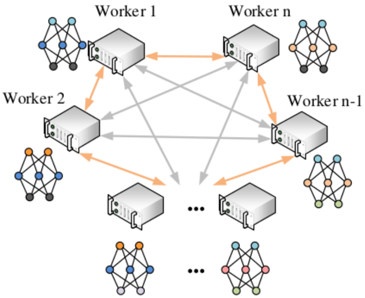
- Mirror all the model parameters across all workers (no PS).
- Workers exchange parameter updates directly via an allreduce operation.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Aggregation - Distributed - Gossip

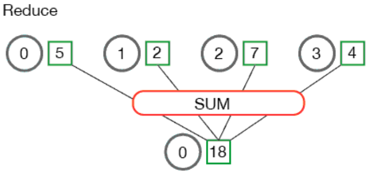
- No PS, and no global model.
- Every worker communicates updates with their neighbors.
- The consistency of parameters across all workers only at the end of the algorithm.

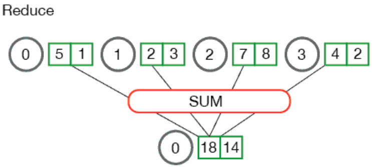


[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Reduce and AllReduce (1/2)

- Reduce: reducing a set of numbers into a smaller set of numbers via a function.
- E.g., `sum([1, 2, 3, 4, 5]) = 15`
- Reduce takes an array of input elements on each process and returns an array of output elements to the root process.

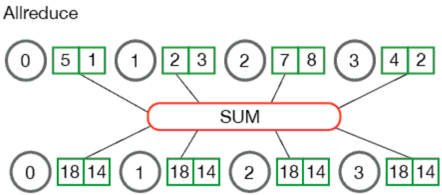




[https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce]

Reduce and AllReduce (2/2)

- AllReduce stores reduced results across all processes rather than the root process.

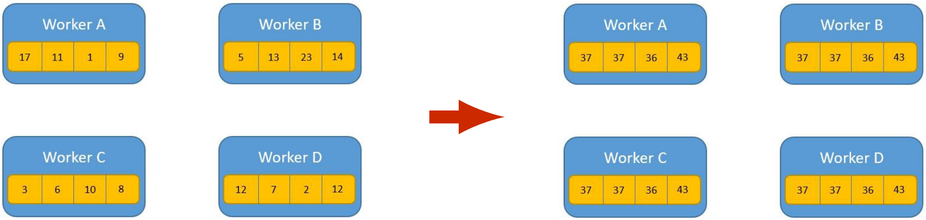


[https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce]

AllReduce Example

Initial state

After AllReduce operation



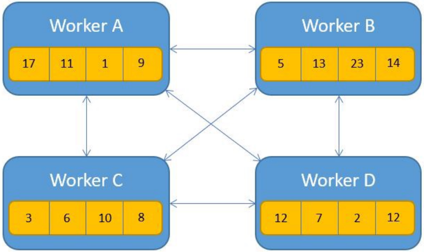
[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation

- All-to-all allreduce
- Master-worker allreduce
- Tree allreduce
- Round-robin allreduce
- Butterfly allreduce
- Ring allreduce

AllReduce Implementation - All-to-All AllReduce

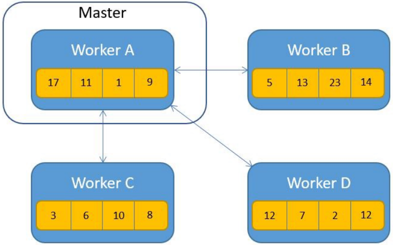
- Send the array of data to each other.
- Apply the reduction operation on each process.
- Too many unnecessary messages.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Master-Worker AllReduce

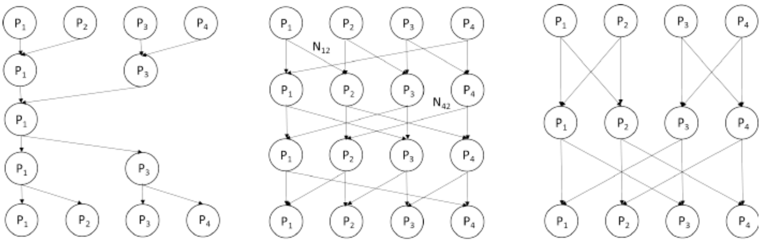
- Selecting one process as a master, gather all arrays into the master.
- Perform reduction operations locally in the master.
- Distribute the result to the other processes.
- The master becomes a bottleneck (not scalable).



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Other implementations

- Some try to minimize bandwidth.
- Some try to minimize latency.



(a) Tree AllReduce (b) Round-robin AllReduce (c) Butterfly AllReduce

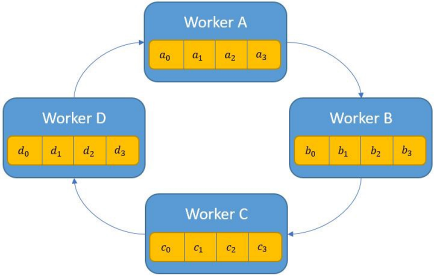
[Zhao H. et al., arXiv:1312.3020, 2013]

AllReduce Implementation - Ring-AllReduce (1/6)

- The Ring-Allreduce has two phases:
  1. First, the share-reduce phase
  2. Then, the share-only phase

AllReduce Implementation - Ring-AllReduce (2/6)

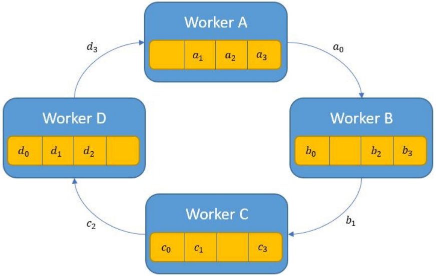
- In the share-reduce phase, each process `p` sends data to the process `(p+1)%m`
  - `m` is the number of processes, and `%` is the modulo operator.
- The array of data on each process is divided to `m` chunks (`m=4` here).
- Each one of these chunks will be indexed by `i` going forward.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (3/6)

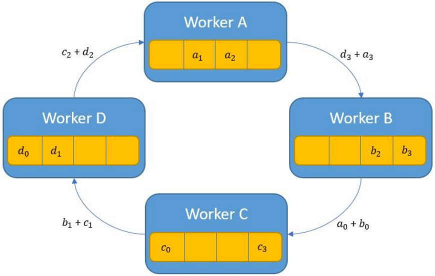
- In the first share-reduce step, process A sends a_0 to process B.
- Process B sends b_1 to process C, etc.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (4/6)

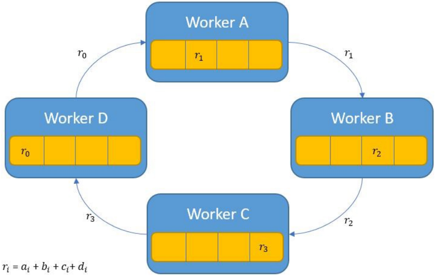
- When each process receives the data from the previous process, it applies the reduce operator (e.g., sum)
  - The reduce operator should be associative and commutative.
- It then proceeds to send it to the next process in the ring.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (5/6)

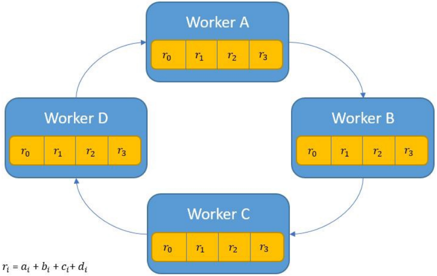
- The share-reduce phase finishes when each process holds the complete reduction of chunk `i`.
- At this point each process holds a part of the end result.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation - Ring-AllReduce (6/6)

- The share-only step is the same process of sharing the data in a ring-like fashion without applying the reduce operation.
- This consolidates the result of each chunk in every process.



[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

Master-Worker AllReduce vs. Ring-AllReduce

- $\mathtt{N}$: number of elements, $\mathtt{m}$: number of processes
- Master-Worker AllReduce
  - First each process sends $\mathtt{N}$ elements to the master: $\mathtt{N} \times (\mathtt{m} - 1)$ messages.
  - Then the master sends the results back to the process: another $\mathtt{N} \times (\mathtt{m} - 1)$ messages.
  - Total network traffic is $2(\mathtt{N} \times (\mathtt{m} - 1))$, which is proportional to $\mathtt{m}$.
- Ring-AllReduce
  - In the share-reduce step each process sends $\frac{\mathtt{N}}{\mathtt{m}}$ elements, and it does it $\mathtt{m} - 1$ times: $\frac{\mathtt{N}}{\mathtt{m}} \times (\mathtt{m} - 1)$ messages.
  - On the share-only step, each process sends the result for the chunk it calculated: another $\frac{\mathtt{N}}{\mathtt{m}} \times (\mathtt{m} - 1)$ messages.
  - Total network traffic is $2(\frac{\mathtt{N}}{\mathtt{m}} \times (\mathtt{m} - 1))$.

Communication Synchronization and Frequency

Synchronization

- When to synchronize the parameters among the parallel workers?

Communication Synchronization (1/2)

- Synchronizing the model replicas in data-parallel training requires communication
  - between workers, in allreduce
  - between workers and parameter servers, in the centralized architecture

- The communication synchronization decides how frequently all local models are synchronized with others.

Communication Synchronization (2/2)

- It will influence:
  - The communication traffic
  - The performance
  - The convergence of model training

- There is a trade-off between the communication traffic and the convergence.

Reducing Synchronization Overhead

- Two directions for improvement:

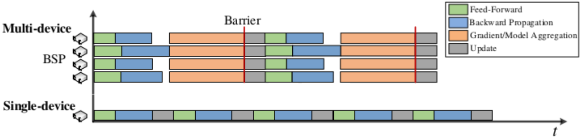
  1. To relax the synchronization among all workers.
  2. The frequency of communication can be reduced by more computation in one iteration.

Communication Synchronization Models

- Synchronous
- Stale-synchronous
- Asynchronous
- Local SGD

Communication Synchronization - Synchronous

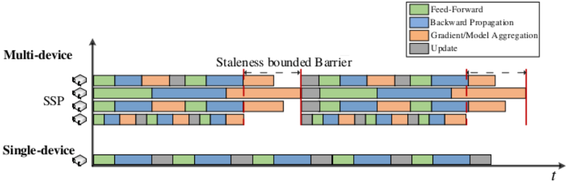
- After each iteration, the workers synchronize their parameter updates.
- Every worker must wait for all workers to finish the transmission of all parameters in the current iteration, before the next training.
- Stragglers can influence the overall system throughput.
- High communication cost that limits the system scalability.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (1/2)

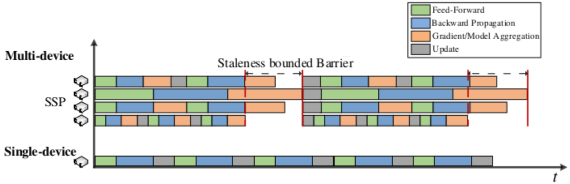
- Alleviate the straggler problem without losing synchronization.
- The faster workers to do more updates than the slower workers to reduce the waiting time of the faster workers.
- Staleness bounded barrier to limit the iteration gap between the fastest worker and the slowest worker.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Stale Synchronous (2/2)

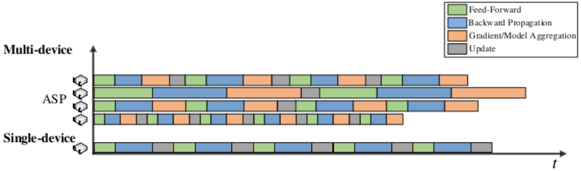
- For a maximum staleness bound $\mathtt{s}$, the update formula of worker $\mathtt{i}$ at iteration $\mathtt{t}+1$:
- $\mathbf{w}_{\mathtt{i},\mathtt{t}+1} := \mathbf{w}_0 - \eta(\sum_{\mathtt{k}=1}^{\mathtt{t}} \sum_{\mathtt{j}=1}^{\mathtt{n}} \mathtt{G}_{\mathtt{j},\mathtt{k}} + \sum_{\mathtt{k}=\mathtt{t}-\mathtt{s}}^{\mathtt{t}} \mathtt{G}_{\mathtt{i},\mathtt{k}} + \sum_{(\mathtt{j},\mathtt{k})\in \mathtt{S}_{\mathtt{i},\mathtt{t}+1}} \mathtt{G}_{\mathtt{j},\mathtt{k}})$
- The update has three parts:
  1. Guaranteed pre-window updates from clock $\mathtt{1}$ to $\mathtt{t}$ over all workers.
  2. Guaranteed read-my-writes in-window updates made by the querying worker $\mathtt{i}$.
  3. Best-effort in-window updates. $\mathtt{S}_{\mathtt{i},\mathtt{t}+1}$ is some subset of the updates from other workers during period $[\mathtt{t}-\mathtt{s}]$.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Asynchronous (1/2)

- It completely eliminates the synchronization.
- Each work transmits its gradients to the PS after it calculates the gradients.
- The PS updates the global model without waiting for the other workers.

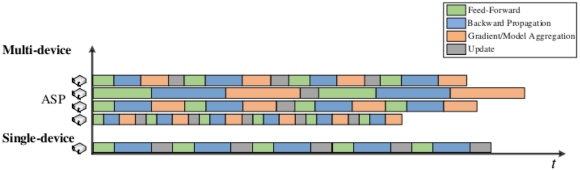


[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Asynchronous (2/2)

- $\mathbf{w}_{t+1} := \mathbf{w}_t - \eta \sum_{i=1}^{n} G_{i,t-\tau_{k,i}}$

- $\tau_{k,i}$ is the time delay between the moment when worker i calculates the gradient at the current iteration.

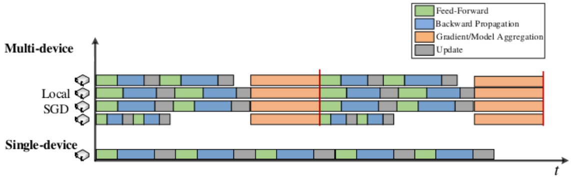


[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Synchronization - Local SGD

- All workers run several iterations, and then averages all local models into the newest global model.
- If $\mathcal{I}_T$ represents the synchronization timestamps, then:

$$\mathbf{w}_{i,t+1} = \begin{cases} \mathbf{w}_{i,t} - \eta G_{i,t} & \text{if} \quad t+1 \notin \mathcal{I}_T \\ \mathbf{w}_{i,t} - \eta \frac{1}{n} \sum_{i=1}^{n} G_{i,t} & \text{if} \quad t+1 \in \mathcal{I}_T \end{cases}$$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

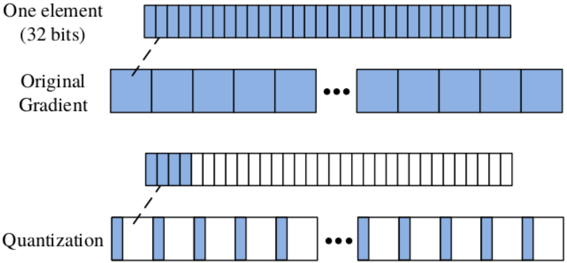
Communication Compression

Communication Compression

- Reduce the communication traffic with little impact on the model convergence.
- Compress the exchanged gradients or models before transmitting across the network.
- Quantization
- Sparsification

Communication Compression - Quantization

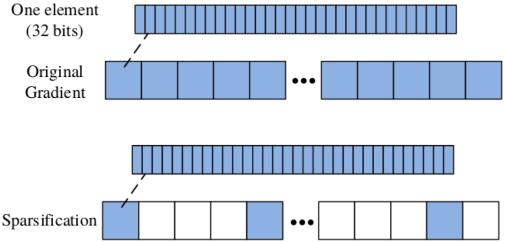
- Useing lower bits to represent the data.
- The gradients are of low precision.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Communication Compression - Sparsification

- Reducing the number of elements that are transmitted at each iteration.
- Only significant gradients are required to update the model parameter to guarantee the convergence of the training.
- E.g., the zero-valued elements are no need to transmit.

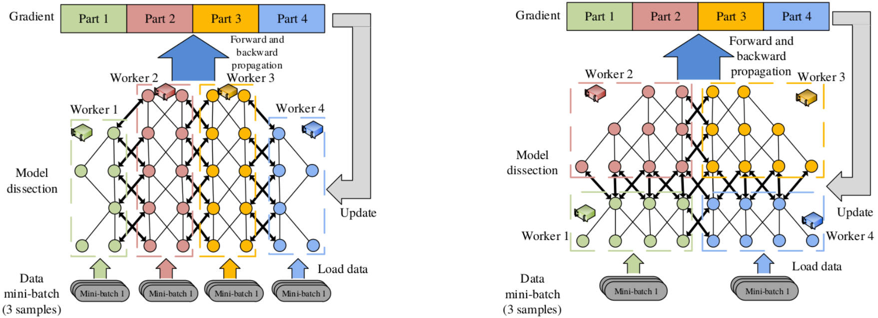


[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Model Parallelism

Model Parallelization

- The model is split across multiple devices.
- Depends on the architecture of the NN.



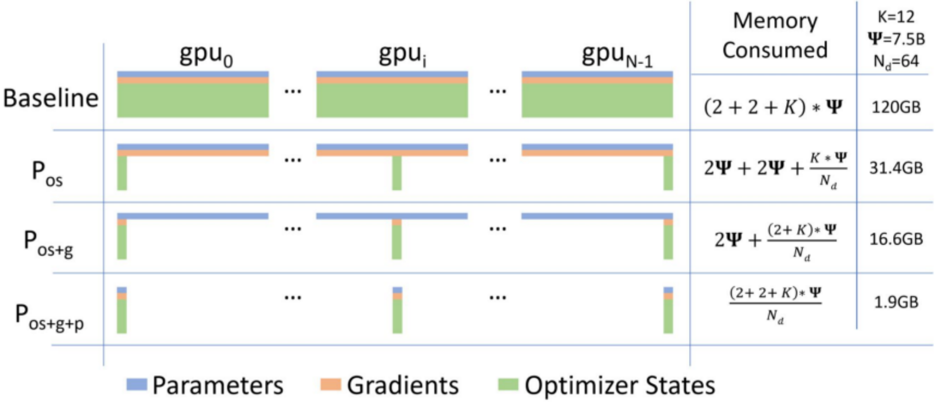
[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

DeepSpeed Zero

- Memory requirement from model states (Ψ := # parameters)
  - Parameters (fp16): 2Ψ
  - Gradients (fp16): 2Ψ
  - Optimizer states: e.g. Adam 12Ψ
    1. Parameters (fp32): 4Ψ
    2. Momentum (fp32): 4Ψ
    3. Variance (fp32): 4Ψ

- Approach: partition each of them to all DP processes

DeepSpeed Zero



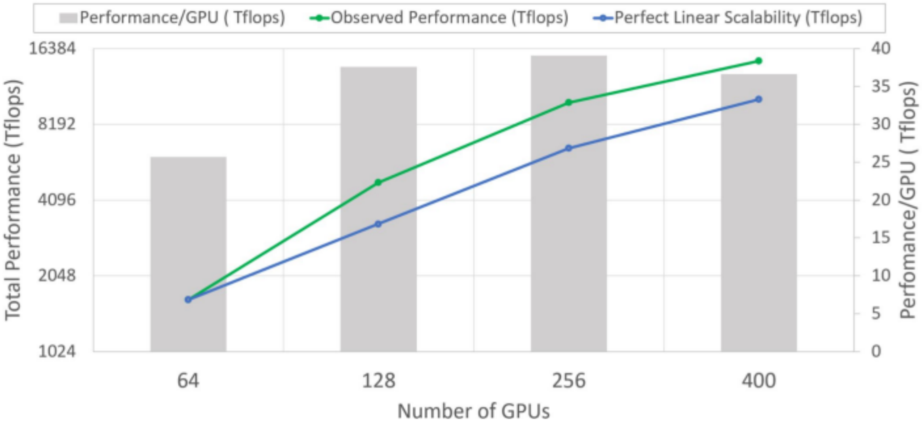
DeepSpeed Zero

- Communication analysis (Ψ := # parameters)
  - Baseline DP: one all-reduce, 2Ψ
  - P_{os+g}: 2Ψ
    i. Scatter-reduce on gradients: Ψ
    ii. All-gather on updated parameters: Ψ

DeepSpeed Zero

- Communication analysis (Ψ := # parameters)
  - Baseline DP: one all-reduce, 2Ψ
  - P_{os+g}: 2Ψ
    1. Scatter-reduce on gradients: Ψ
    2. All-gather on updated parameters: Ψ
  - P_{os+g+p}: 3Ψ (1.5x communication)
    1. All-gather on parameters for forward: Ψ
    2. All-gather on parameters for backward: Ψ
    3. Scatter-reduce on gradients: Ψ

DeepSpeed Zero



Summary

Summary

- Scalability matters
- Parallelization
- Data Parallelization
  - Parameter server vs. AllReduce
  - Synchronized vs. asynchronized
- Model Parallelization
  - DeepSpeed-Zero

Thanks!