Distributed Learning - Data Parallelization

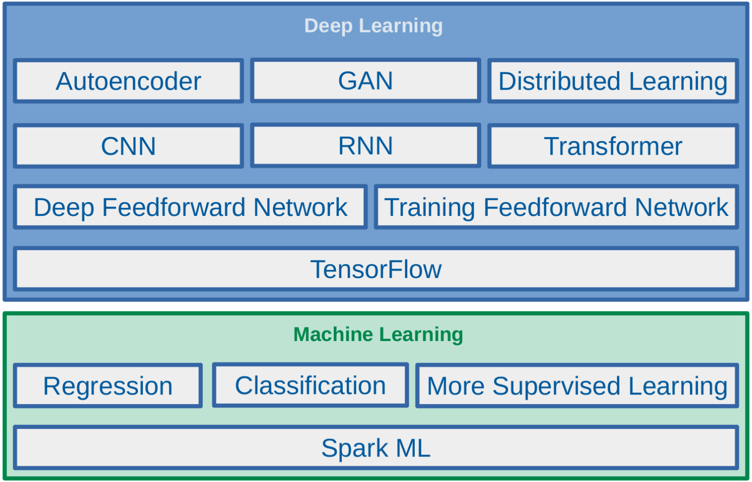
Amir H. Payberah
payberah@kth.se
2020-12-08

The Course Web Page

https://id2223kth.github.io

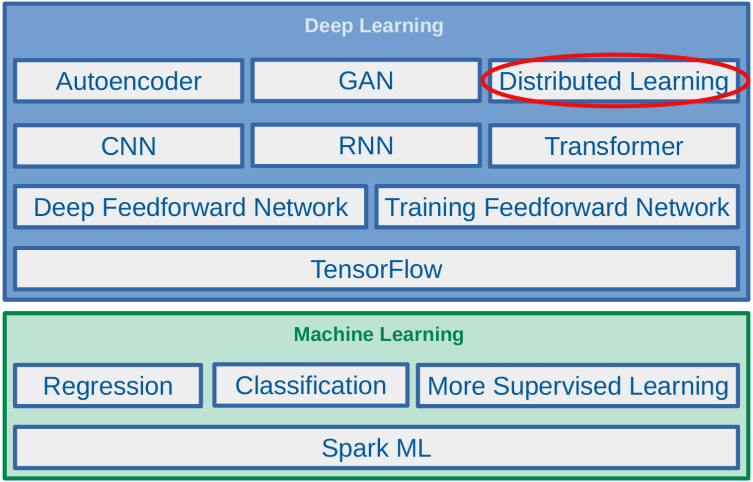
https://tinyurl.com/y6kcpmzy

Where Are We?



Deep Learning
Autoencoder
GAN
Distributed Learning
CNN
RNN
Transformer
Deep Feedforward Network
Training Feedforward Network
TensorFlow
Machine Learning
Regression
Classification
More Supervised Learning
Spark ML

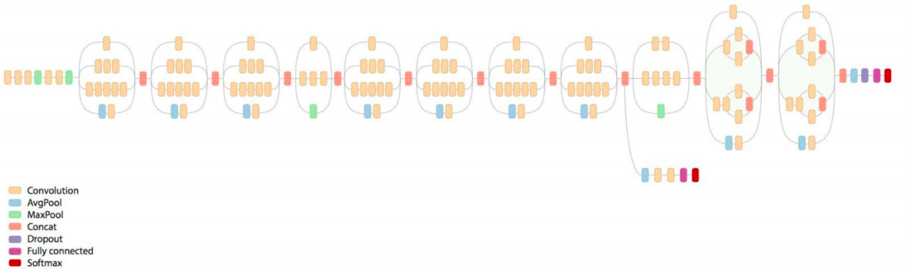
Where Are We?



Deep Learning
Autoencoder
GAN
Distributed Learning
CNN
RNN
Transformer
Deep Feedforward Network
Training Feedforward Network
TensorFlow
Machine Learning
Regression
Classification
More Supervised Learning
Spark ML

Training Deep Neural Networks

- Computationally intensive
- Time consuming

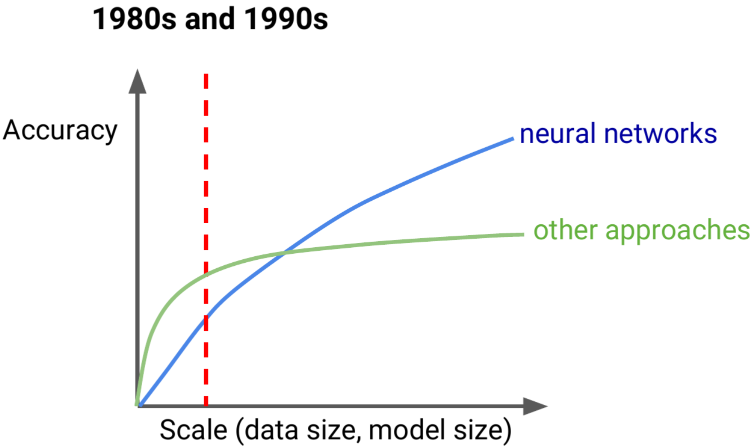


[https://cloud.google.com/tpu/docs/images/inceptionv3onc--oview.png]

Why?

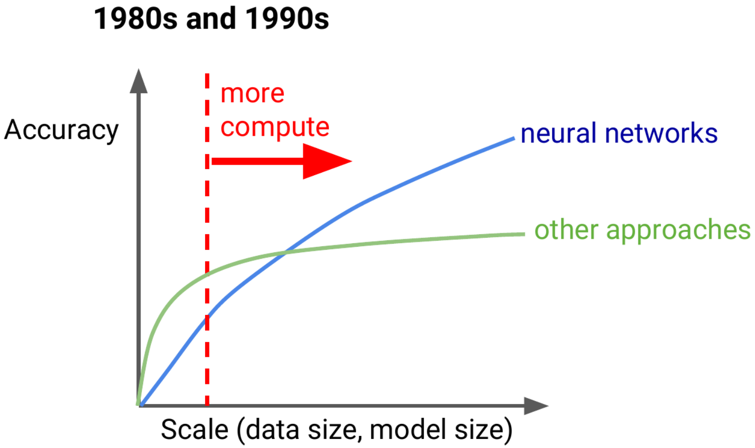
- Massive amount of training dataset
- Large number of parameters

Accuracy vs. Data/Model Size



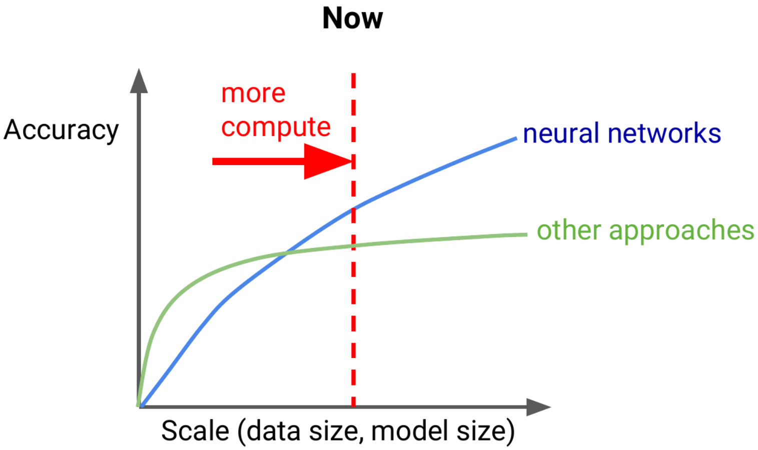
[Jeff Dean at AI Frontiers: Trends and Developments in Deep Learning Research]

Accuracy vs. Data/Model Size



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

Accuracy vs. Data/Model Size

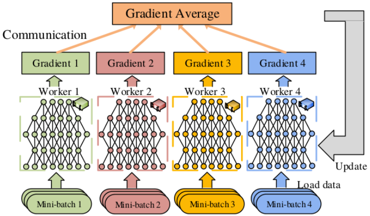


[Jeff Dean at AI Frontiers: Trends and Developments in Deep Learning Research]

Scale Matters

Data Parallelization (1/4)

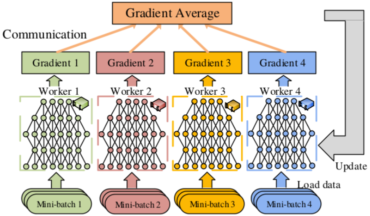
- Replicate a whole model on every device.
- Train all replicas simultaneously, using a different mini-batch for each.



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

Data Parallelization (2/4)

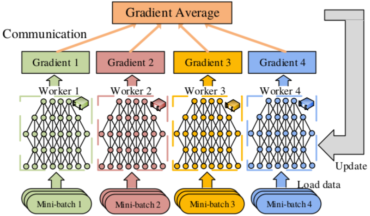
- $\mathtt{k}$ devices
- $\mathtt{J_i}(\mathbf{w}) = \frac{1}{|\beta_\mathtt{i}|}\sum_{\mathbf{x}\in\beta_\mathtt{i}} \mathtt{l}(\mathbf{x},\mathbf{w}),\ \forall \mathtt{i} = 1, 2, \cdots, \mathtt{k}$
- $\mathtt{G_i}(\mathbf{w}, \beta_\mathtt{i}) = \frac{1}{|\beta_\mathtt{i}|}\sum_{\mathbf{x}\in\beta_\mathtt{i}} \nabla\mathtt{l}(\mathbf{w},\mathbf{x})$
- $\mathtt{G_i}(\mathbf{w}, \beta_\mathtt{i})$: the local estimate of the gradient of the loss function $\nabla\mathtt{J_i}(\mathbf{w})$.



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

Data Parallelization (3/4)

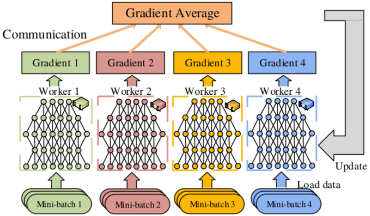
- Compute the gradients aggregation (e.g., mean of the gradients).
- $F(G_1, \cdots, G_k) = \frac{1}{k} \sum_{i=1}^{k} G_i(\mathbf{w}, \beta_i)$



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

Data Parallelization (4/4)

- Update the model.
- $\mathbf{w} := \mathbf{w} - \eta \mathtt{F}(\mathtt{G}_1, \cdots, \mathtt{G}_k)$



[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
- Parallelism of computations and communications

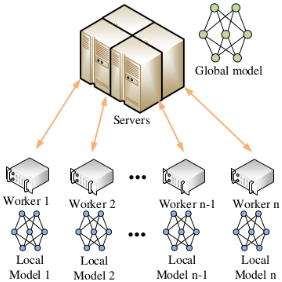
The Aggregation Algorithm

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

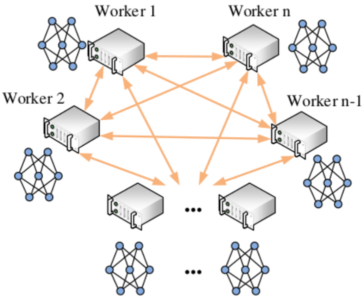
- Store the model parameters outside of the workers.
- Workers periodically report their computed parameters or parameter updates to a (set of) parameter server(s) (PSs).



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

Aggregation - Distributed - All-Reduce

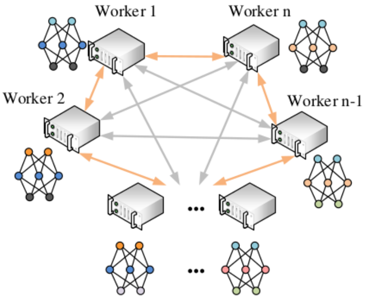
- Mirror all the model parameters across all workers (no PS).
- Workers exchange parameter updates directly via an allreduce operation.



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

Aggregation - Distributed - Gossip

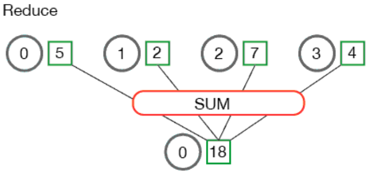
- No PS, and no global model.
- Every worker communicates updates with their neighbors.
- The consistency of parameters across all workers only at the end of the algorithm.

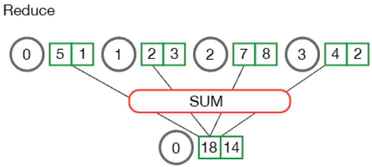


[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

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.

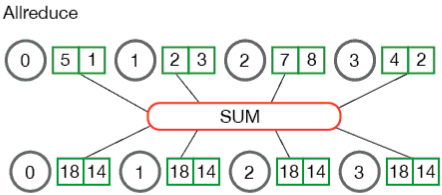




[https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce]

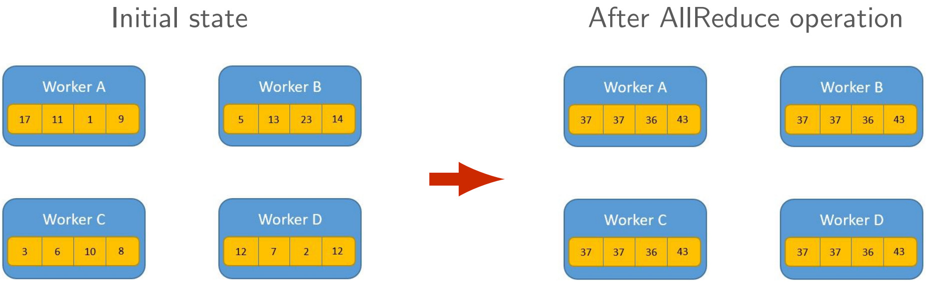
Reduce and AllReduce (2/2)

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



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

AllReduce Example



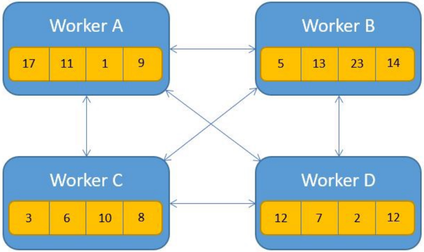
[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

AllReduce Implementation

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

AllReduce Implementation - All-to-All AllReduce

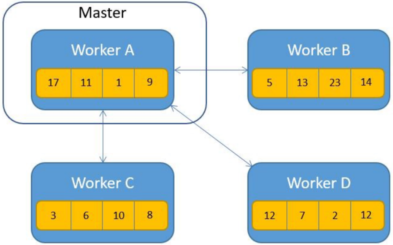
- Send the array of data to each other.
- Apply the reduction operation on each process.
- Too many unnecessary messages.



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

AllReduce Implementation - Master-Worker AllReduce

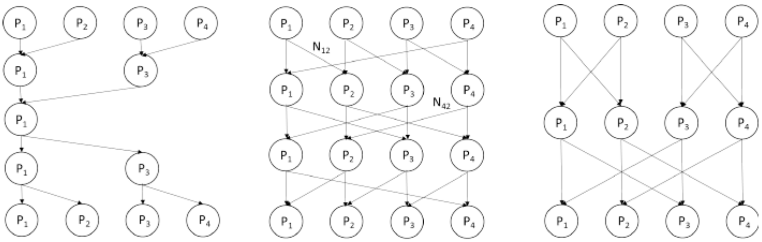
- Selecting one process as a master, gather all arrays into the master.
- Perform reduction operations locally in the master.
- Distribute the result to the other processes.
- The master becomes a bottleneck (not scalable).



[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

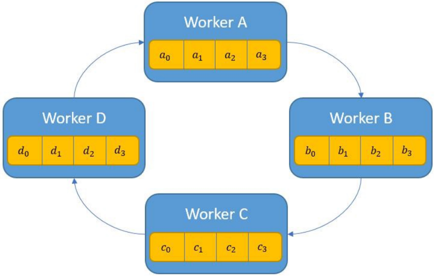
[Zhao H. et al., arXiv:1312.3020, 2013]

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)

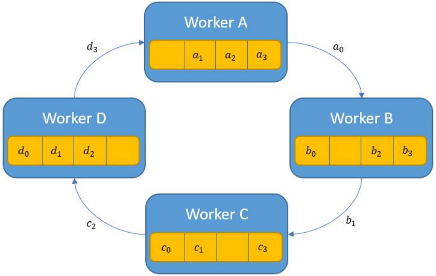
- In the share-reduce phase, each process `p` sends data to the process `(p+1)%m`
  - `m` is the number of processes, and `%` is the modulo operator.
- The array of data on each process is divided to `m` chunks (`m=4` here).
- Each one of these chunks will be indexed by `i` going forward.



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

AllReduce Implementation - Ring-AllReduce (3/6)

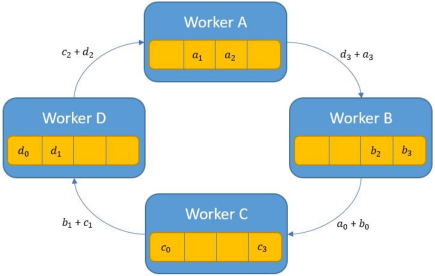
- In the first share-reduce step, process A sends a_0 to process B.
- Process B sends b_1 to process C, etc.



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

AllReduce Implementation - Ring-AllReduce (4/6)

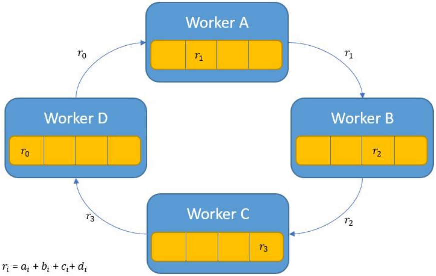
- When each process receives the data from the previous process, it applies the reduce operator (e.g., sum)
  - The reduce operator should be associative and commutative.
- It then proceeds to send it to the next process in the ring.



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

AllReduce Implementation - Ring-AllReduce (5/6)

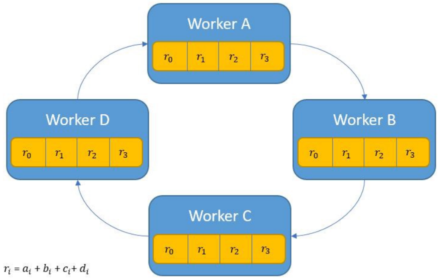
- The share-reduce phase finishes when each process holds the complete reduction of chunk `i`.
- At this point each process holds a part of the end result.



[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:

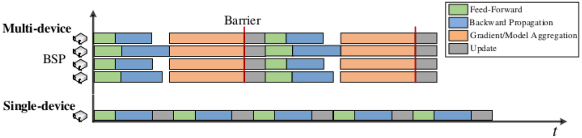
  1. To relax the synchronization among all workers.
  2. The frequency of communication can be reduced by more computation in one iteration.

Communication Synchronization Models

- Synchronous
- Stale-synchronous
- Asynchronous
- Local SGD

Communication Synchronization - Synchronous

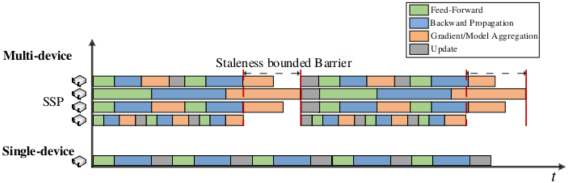
- After each iteration, the workers synchronize their parameter updates.
- Every worker must wait for all workers to finish the transmission of all parameters in the current iteration, before the next training.
- Stragglers can influence the overall system throughput.
- High communication cost that limits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



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

Communication Synchronization - Stale Synchronous (1/2)

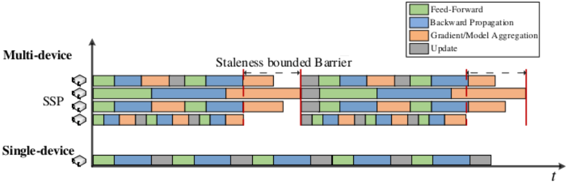
- Alleviate the straggler problem without losing synchronization.
- The faster workers to do more updates than the slower workers to reduce the waiting time of the faster workers.
- Staleness bounded barrier to limit the iteration gap between the fastest worker and the slowest worker.



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

Communication Synchronization - Stale Synchronous (2/2)

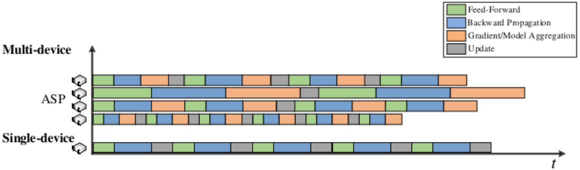
- For a maximum staleness bound $\mathtt{s}$, the update formula of worker $\mathtt{i}$ at iteration $\mathtt{t}+1$:
- $\mathbf{w}_{\mathtt{i},\mathtt{t}+1} := \mathbf{w}_0 - \eta(\sum_{\mathtt{k}=1}^{\mathtt{t}} \sum_{\mathtt{j}=1}^{\mathtt{n}} \mathtt{G}_{\mathtt{j},\mathtt{k}} + \sum_{\mathtt{k}=\mathtt{t}-\mathtt{s}}^{\mathtt{t}} \mathtt{G}_{\mathtt{i},\mathtt{k}} + \sum_{(\mathtt{j},\mathtt{k}) \in \mathtt{S}_{\mathtt{i},\mathtt{t}+1}} \mathtt{G}_{\mathtt{j},\mathtt{k}})$
- 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)

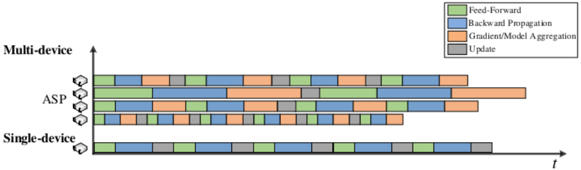
- It completely eliminates the synchronization.
- Each work transmits its gradients to the PS after it calculates the gradients.
- The PS updates the global model without waiting for the other workers.



[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.

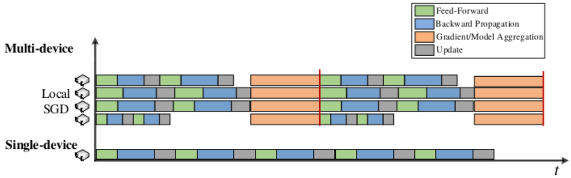


[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

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]

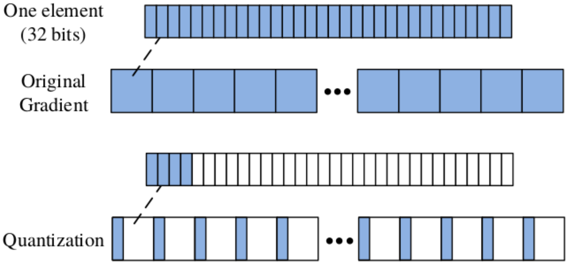
Communication Compression

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

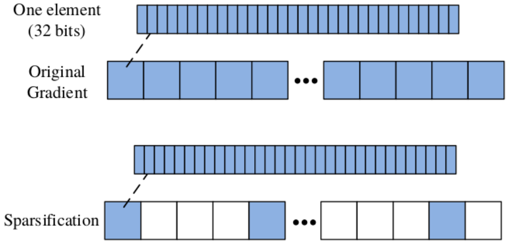
- Useing lower bits to represent the data.
- The gradients are of low precision.



[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]

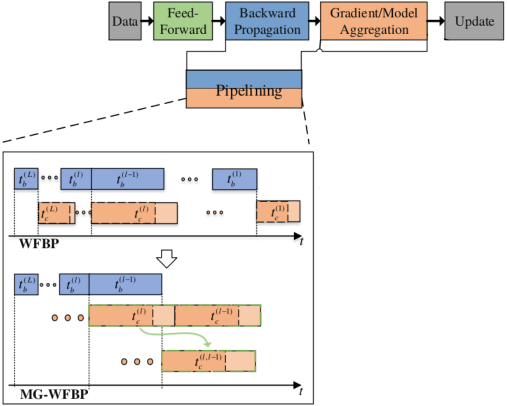
Parallelism of Computations and Communications

Parallelism of Computations and Communications (1/3)

- The layer-wise structure of deep models makes it possible to parallels the communication and computing tasks.
- Optimizing the order of computation and communication such that the communication cost can be minimized

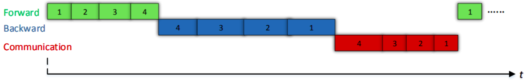
Parallelism of Computations and Communications (2/3)

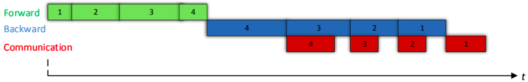
- Wait-free backward propagation (WFBP)
- Merged-gradient WFBP (MG-WFBP)



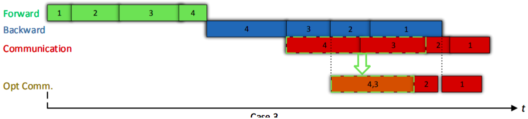
[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Parallelism of Computations and Communications (3/3)





Wait-free backward propagation (WFBP)



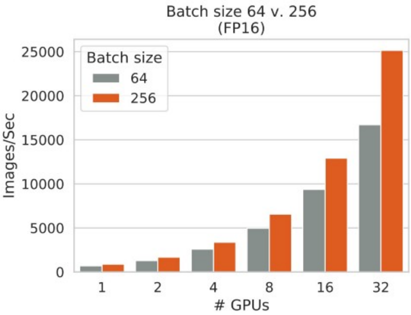
Merged-gradient WFBP (MG-WFBP)

[shi et al., MG-WFBP: Efficient Data Communication for Distributed Synchronous SGD Algorithms, 2018]

Distributed SGD and Batch Size

Batch Size vs. Number of GPUs

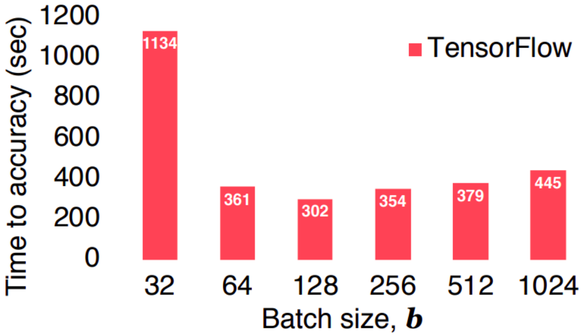
- $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{|\beta|} \sum_{\mathbf{x} \in \beta} \nabla \mathtt{l}(\mathbf{x}, \mathbf{w})$
- The more samples processed during each batch, the faster a training job will complete.
- E.g., ImageNet + ResNet-50



[https://medium.com/@emwatz/lessons-for-improving-training-performance-part-1-b5efd0f0dcea]

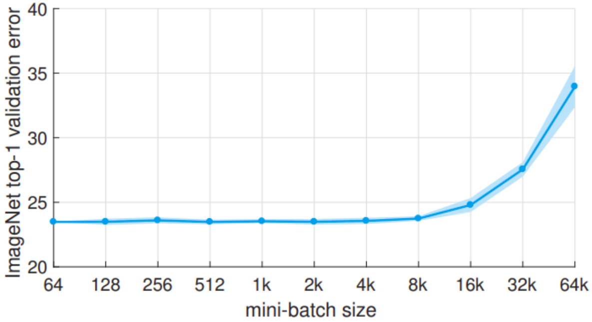
Batch Size vs. Time to Accuracy

- ResNet-32 on Titan X GPU



[Peter Pietzuch - Imperial College London]

Batch Size vs. Validation Error



[Goyal et al., Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, 2018]

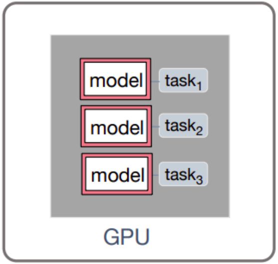
CROSSBOW: Scaling Deep Learning with Small Batch Sizes on Multi-GPU Servers

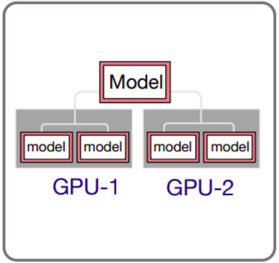
- How to design a deep learning system that scales training with multiple GPUs, even when the preferred batch size is small?

Crossbow

(1) How to increase efficiency with small batches?

(2) How to synchronise model replicas?

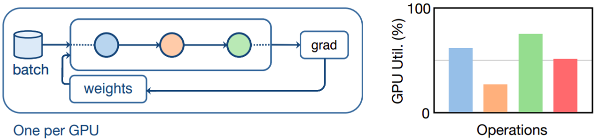




[Peter Pietzuch - Imperial College London]

Problem: Small Batches

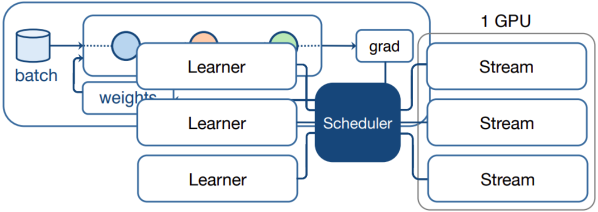
- Small batch sizes underutilise GPUs.
- One batch per GPU: not enough data and instruction parallelism for every operator.



[Peter Pietzuch - Imperial College London]

Idea: Multiple Replicas Per GPU

- Train multiple model replicas per GPU.
- A learner is an entity that trains a single model replica independently with a given batch size.

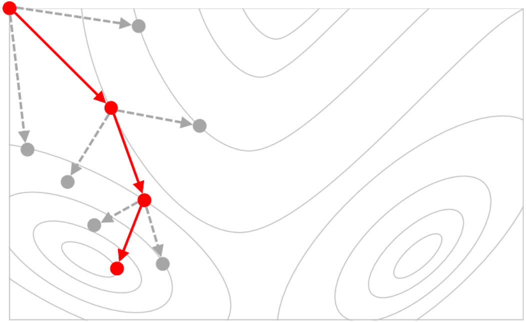


[Peter Pietzuch - Imperial College London]

- But, now we must synchronise a large number of model replicas.

Problem: Similiar Starting Point

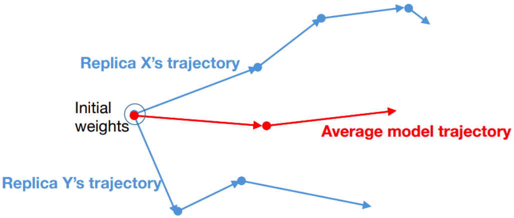
- All learners always start from the same point.
- Limited exploration of parameter space.



[Peter Pietzuch - Imperial College London]

Idea: Independent Replicas

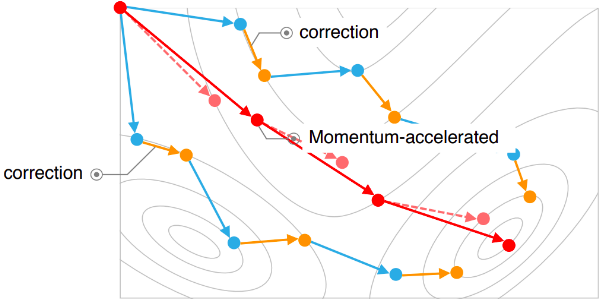
- Maintain independent model replicas.
- Increased exploration of space through parallelism.
- Each model replica uses small batch size.



[Peter Pietzuch - Imperial College London]

Crossbow: Synchronous Model Averaging

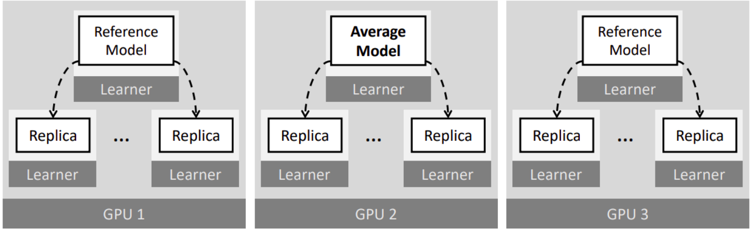
- Allow learners to diverge, but correct trajectories based on average model.
- Accelerate average model trajectory with momentum to find minima faster.



[Peter Pietzuch - Imperial College London]

GPUs with Synchronous Model Averaging

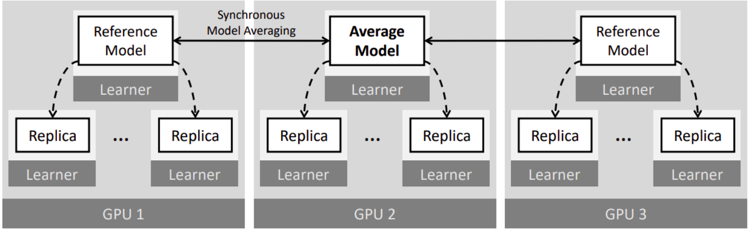
- Synchronously apply corrections to model replicas.



[Peter Pietzuch - Imperial College London]

GPUs with Synchronous Model Averaging

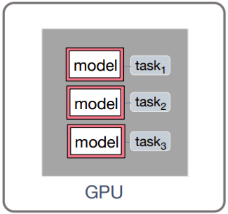
- Ensures consistent view of average model.
- Takes GPU bandwidth into account during synchronisation.



[Peter Pietzuch - Imperial College London]

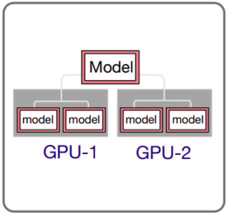
Crossbow

(1) How to increase efficiency with small batches?



Train multiple model replicas per GPU

(2) How to synchronise model replicas?



Use synchronous model averaging

[Peter Pietzuch - Imperial College London]

Summary

Summary

- Data-parallel
- The aggregation algorithm
- Communication synchronization
- Communication compression
- Parallelism of computations and communications
- Batch Size

Reference

- Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020
- P. Goyal et al., Accurate, large minibatch sgd: Training imagenet in 1 hour, 2017
- C. Shallue et al., Measuring the effects of data parallelism on neural network training, 2018
- A. Koliousis et al. CROSSBOW: scaling deep learning with small batch sizes on multi-gpu servers, 2019

Questions?