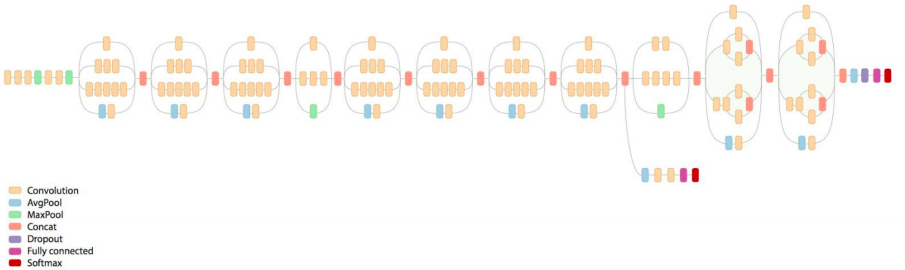
Distributed Deep Learning

Amir H. Payberah
payberah@kth.se
2020-08-20

What is the problem?

Training Deep Neural Networks

- Computationally intensive
- Time consuming

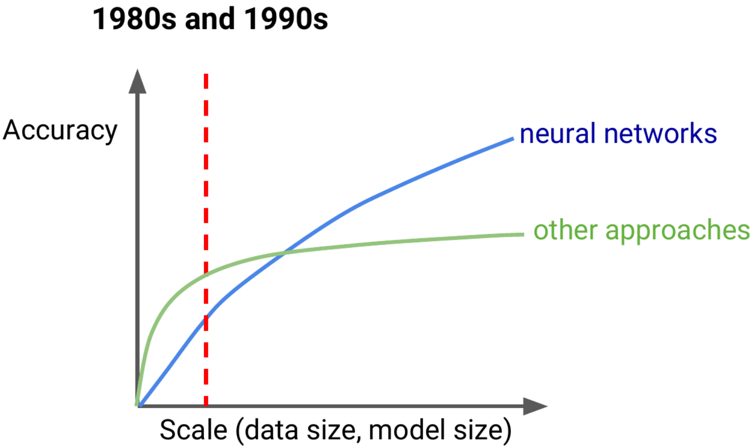


[https://cloud.google.com/tpu/docs/images/inceptionv3onc--oview.png]

Why?

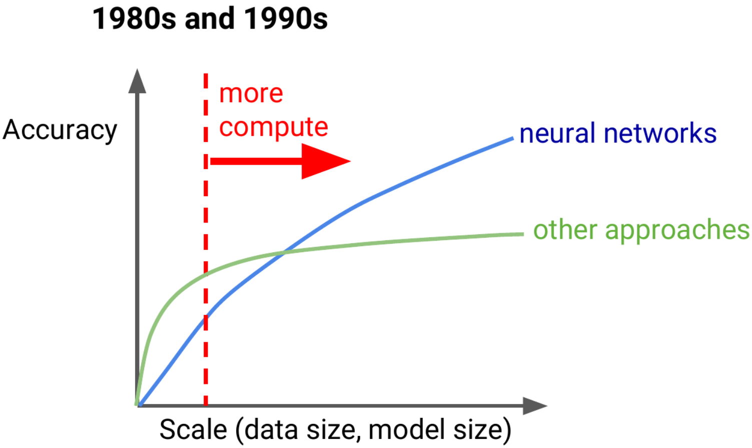
- Massive amount of training dataset
- Large number of parameters

Accuracy vs. Data/Model Size



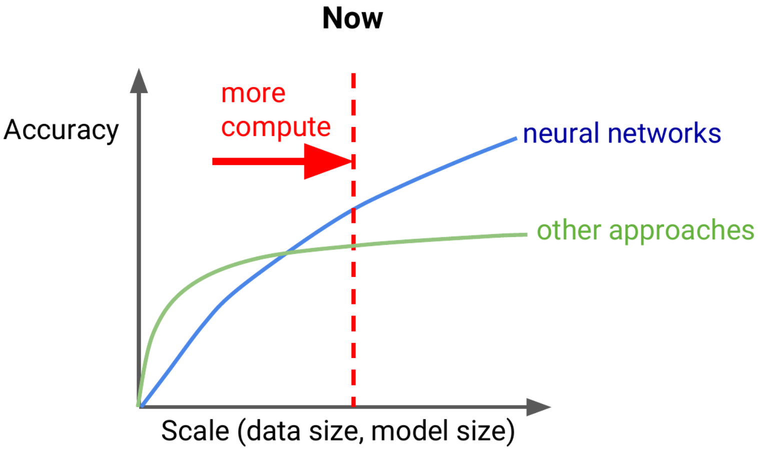
[Jeff Dean at AI Frontiers: Trends and Developments in Deep Learning Research]

Accuracy vs. Data/Model Size



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

Accuracy vs. Data/Model Size



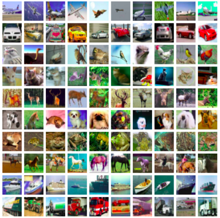
[Jeff Dean at AI Frontiers: Trends and Developments in Deep Learning Research]

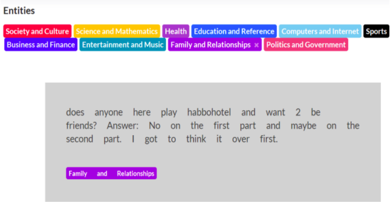
Scale Matters

Fundamentals of Machine Learning

Training Dataset

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





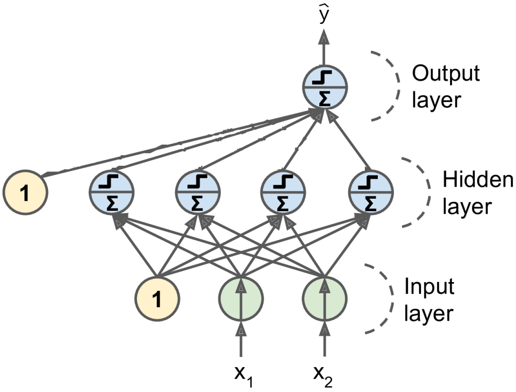
Date	Cost	Actions	Offsite conversions	Impressions	Clicks
2017-04-04	29.44	461	4	5655	477
2017-04-03	74.08	1331	16	18170	1340
2017-04-02	76.09	1349	12	16877	1357
2017-04-01	76.79	1382	8	19757	1378
2017-03-31	77.28	1141	21	18598	1116
2017-03-30	68.62	1065	18	14847	1046
2017-03-29	64.9	1111	25	13994	1094
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.

Model

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



Loss function

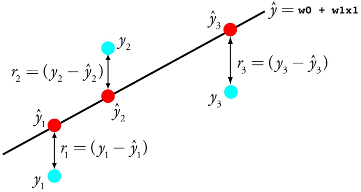
- How good $\hat{y}$ is able to predict the expected outcome y.

Loss function

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

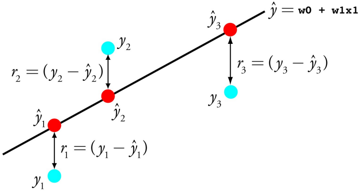
Loss function

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



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

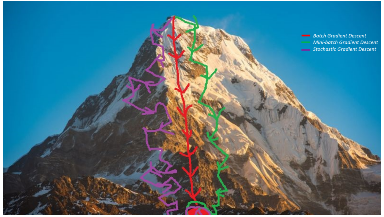
- Minimize the loss function

Objective

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

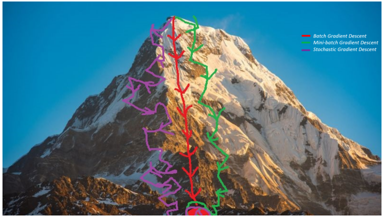
Training

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



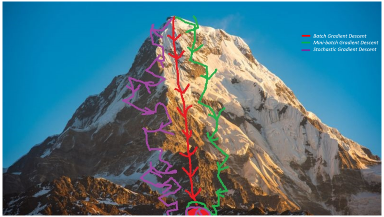
Training

- $J(\mathbf{w}) = \sum_{i=1}^{m} l(y_i, \hat{y}_i)$
- Gradient descent, i.e., $\mathbf{w} := \mathbf{w} - \eta \nabla J(\mathbf{w})$



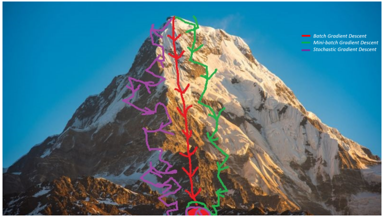
Training

- $J(\mathbf{w}) = \sum_{i=1}^{m} l(y_i, \hat{y}_i)$
- Gradient descent, i.e., $\mathbf{w} := \mathbf{w} - \eta \nabla J(\mathbf{w})$
- Stochastic gradient descent, i.e., $\mathbf{w} := \mathbf{w} - \eta \tilde{g} J(\mathbf{w})$
  - $\tilde{g}$: gradient at a randomly chosen point.



Training

- $J(\mathbf{w}) = \sum_{i=1}^{m} l(y_i, \hat{y}_i)$
- Gradient descent, i.e., $\mathbf{w} := \mathbf{w} - \eta \nabla J(\mathbf{w})$
- Stochastic gradient descent, i.e., $\mathbf{w} := \mathbf{w} - \eta \tilde{g} J(\mathbf{w})$
  - $\tilde{g}$: gradient at a randomly chosen point.
- Mini-barch gradient descent, i.e., $\mathbf{w} := \mathbf{w} - \eta \tilde{g}_B J(\mathbf{w})$
  - $\tilde{g}$: gradient with respect to a set of B randomly chosen points.



Let's Scale the Learning

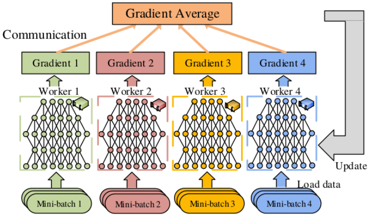
Scalable Training

- Data parallelism
- Model parallelism

Data Parallelism

Data Parallelization (1/4)

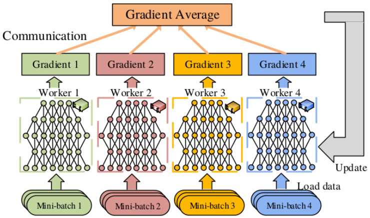
- Replicate a whole model on every device.



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

Data Parallelization (1/4)

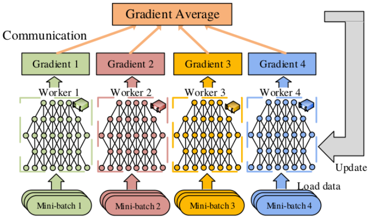
- Replicate a whole model on every device.
- Train all replicas simultaneously, using a different mini-batch for each.



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

Data Parallelization (2/4)

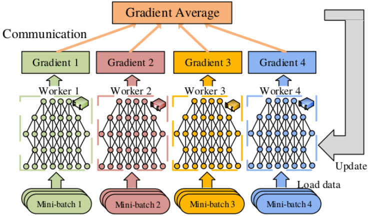
- k devices



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

Data Parallelization (2/4)

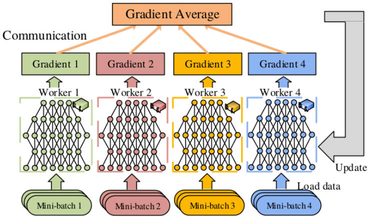
- $\mathtt{k}$ devices
- $\mathtt{J_j(\mathbf{w}) = \sum_{i=1}^{b_j} l(y_i, \hat{y}_i)}$, $\forall \mathtt{j} = 1, 2, \cdots, \mathtt{k}$



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

Data Parallelization (2/4)

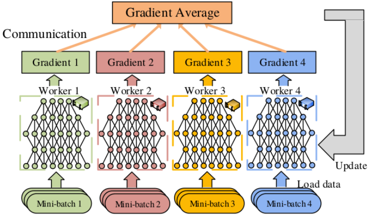
- $\mathtt{k}$ devices
- $\mathtt{J_j(\mathbf{w}) = \sum_{i=1}^{b_j} l(y_i, \hat{y}_i)}$, $\forall \mathtt{j} = 1, 2, \cdots, \mathtt{k}$
- $\tilde{\mathtt{g}}_{\mathtt{B}} \mathtt{J_j(\mathbf{w})}$: gradient of $\mathtt{J_j(\mathbf{w})}$ with respect to a set of $\mathtt{B}$ randomly chosen points at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



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

Data Parallelization (2/4)

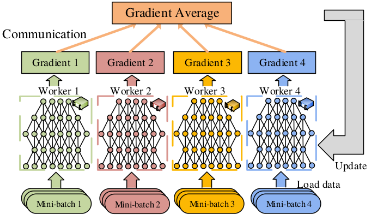
- k devices
- $J_j(\mathbf{w}) = \sum_{i=1}^{b_j} l(y_i, \hat{y}_i), \forall j = 1, 2, \cdots, k$
- $\tilde{g}_B J_j(\mathbf{w})$: gradient of $J_j(\mathbf{w})$ with respect to a set of B randomly chosen points at device j.
- Compute $\tilde{g}_B J_j(\mathbf{w})$ on each device j.



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

Data Parallelization (3/4)

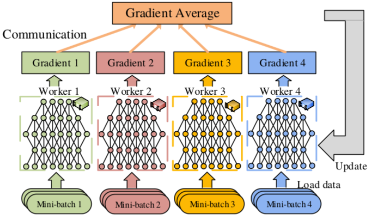
- Compute the mean of the gradients.
- $\tilde{g}_B J(\mathbf{w}) = \frac{1}{k} \sum_{j=1}^{k} \tilde{g}_B J_j(\mathbf{w})$



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

Data Parallelization (4/4)

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



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

Data Parallelization Design Issues

- Gradient aggregation: how to update the parameters

Data Parallelization Design Issues

- Gradient aggregation: how to update the parameters
- Synchronization: when to synchronize the parameters

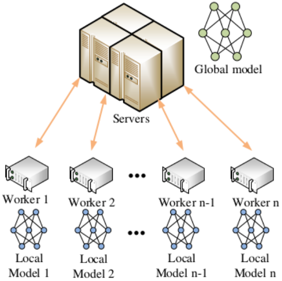
Gradient Aggregation

Gradient Aggregation

- Centralized - parameter server
- Decentralized - all-reduce

Gradient Aggregation - Centralized

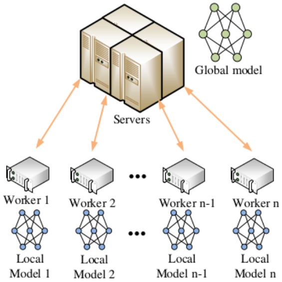
- Store the model parameters outside of the workers.



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

Gradient Aggregation - Centralized

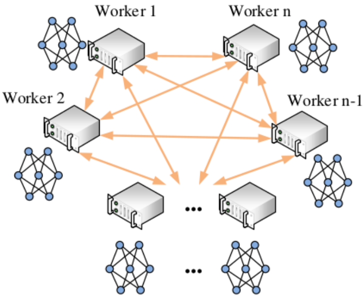
- Store the model parameters outside of the workers.
- Workers periodically report their computed parameters or parameter updates to a (set of) parameter server(s).



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

Gradient Aggregation - Decentralized

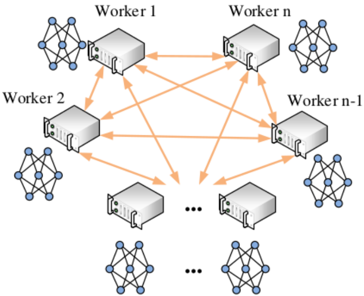
- Mirror all the model parameters across all workers (No PS).



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

Gradient Aggregation - Decentralized

- 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]

Reduce and AllReduce (1/2)

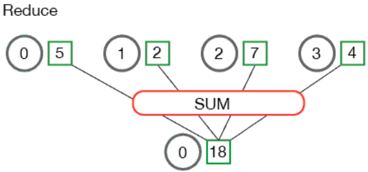
- Reduce: reducing a set of numbers into a smaller set of numbers via a function.

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 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.



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.

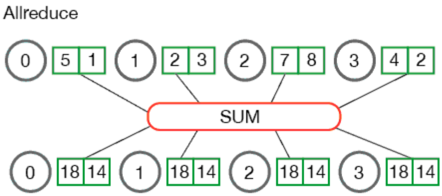
[https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce]

Reduce and AllReduce (2/2)

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

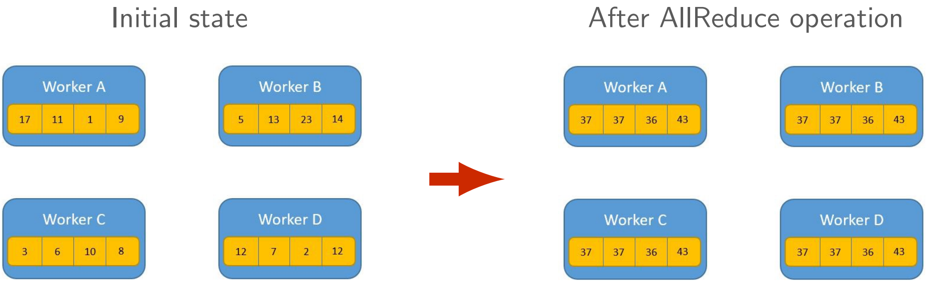
Reduce and AllReduce (2/2)

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



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

AllReduce Example

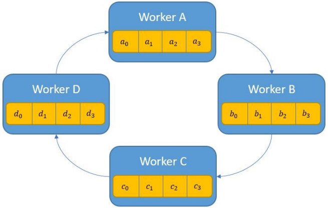


[https://towardsdatascience.com/visual-intuition-on-ring-allreduce-for-distributed-deep-learning-d1f34b4911da]

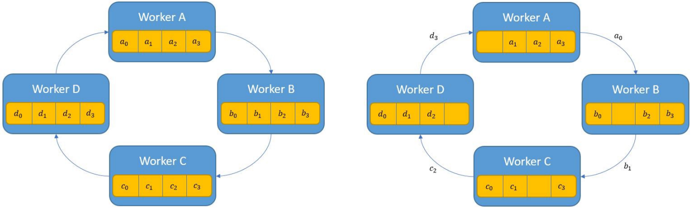
AllReduce Implementation

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

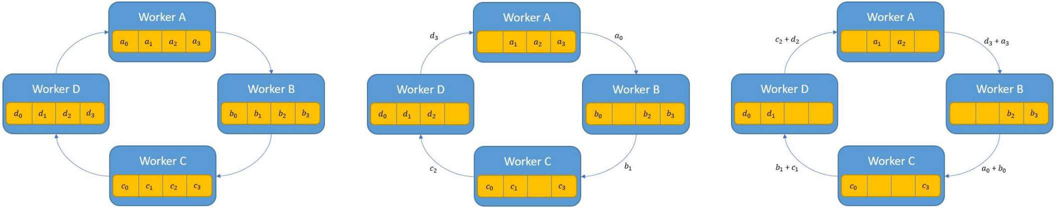
AllReduce Implementation - Ring-AllReduce



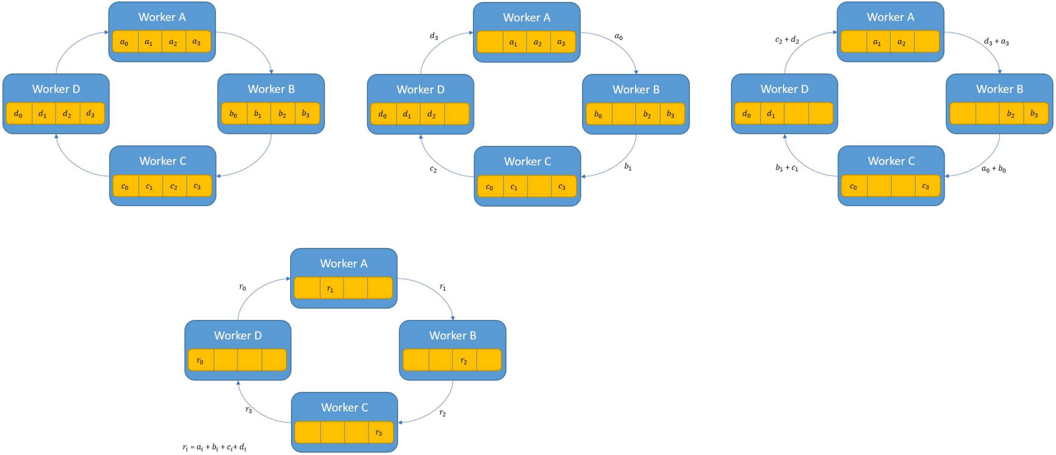
AllReduce Implementation - Ring-AllReduce



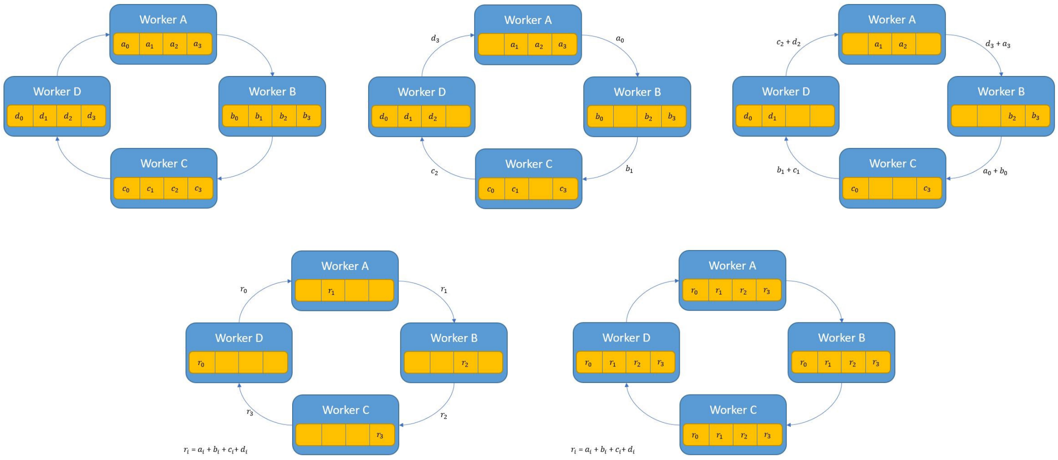
AllReduce Implementation - Ring-AllReduce



AllReduce Implementation - Ring-AllReduce



AllReduce Implementation - Ring-AllReduce



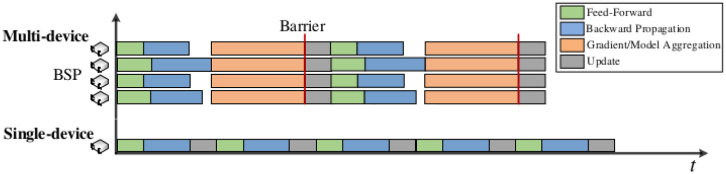
Synchronization

Synchronization

- When to synchronize the parameters among the parallel workers?
  - Synchronous
  - Asynchronous

Synchronization - Synchronous

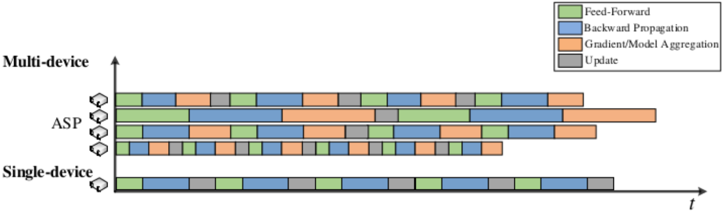
- Before the next training, every worker must wait for all workers to finish the transmission of all parameters in the current iteration.



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

Synchronization - Asynchronous

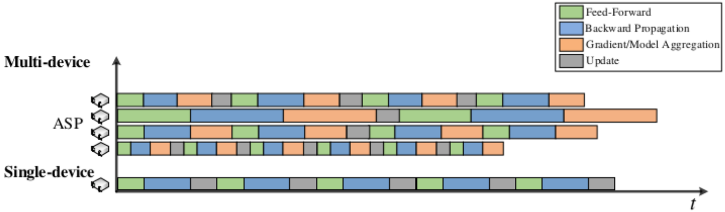
- Eliminates the synchronization.



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

Synchronization - Asynchronous

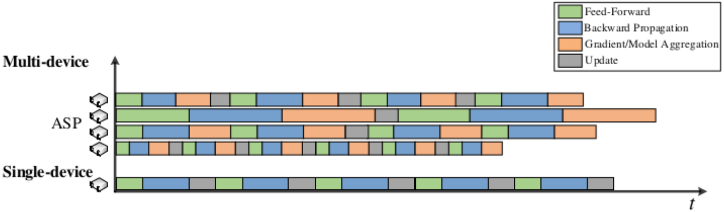
- Eliminates the synchronization.
- Each work transmits its gradients to the parameter server after it calculates the gradients.



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

Synchronization - Asynchronous

- Eliminates the synchronization.
- Each work transmits its gradients to the parameter server after it calculates the gradients.
- The parameter server updates the global model without waiting for the other workers.

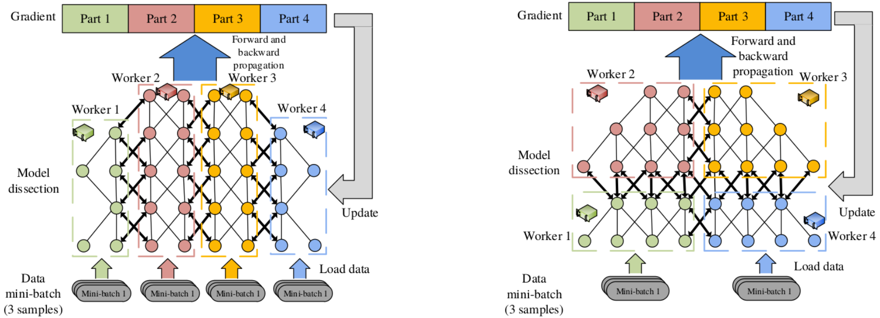


[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Model Parallelism

Model Parallelization

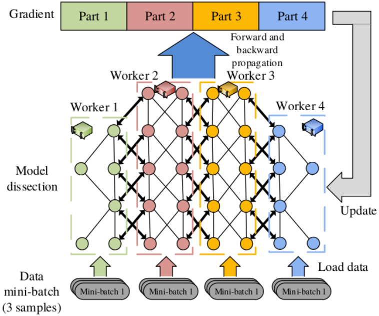
- The model is split across multiple devices.

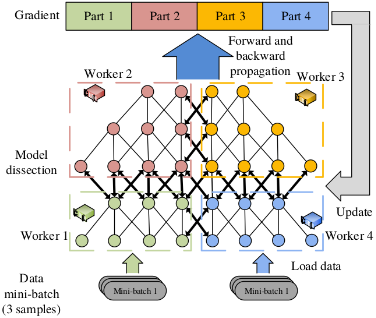


[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Model Parallelization

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

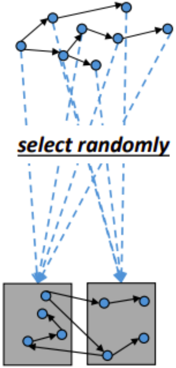




[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Model Parallelization - Hash Partitioning

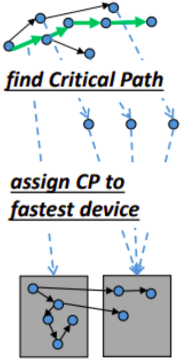
- Randomly assign vertices to devices proportionally to the capacity of the devices by using a hash function.



[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]

Model Parallelization - Critical Path

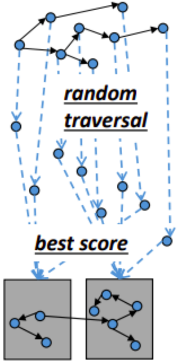
- Assigning the complete critical path to the fastest device.
- Critical path: the path with the longest computation time from source to sink vertex.



[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]

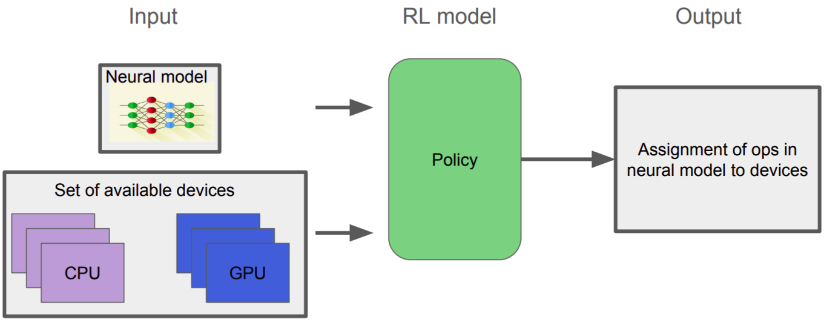
Model Parallelization - Multi-Objective Heuristics

- Different objectives, e.g., memory, importance, traffic, and execution time



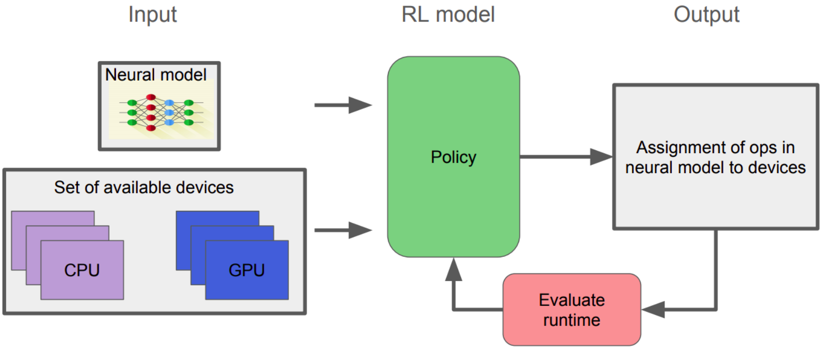
[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]

Model Parallelization - Reinforcement Learning (1/5)



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

Model Parallelization - Reinforcement Learning (2/5)



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

Model Parallelization - Reinforcement Learning (3/5)

- $\mathtt{J}(\mathtt{w}) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G},\mathtt{w})}[\mathtt{R}(\mathcal{P})|\mathcal{G}]$
- Objective: $\arg\min_{\mathtt{w}} \mathtt{J}(\mathtt{w})$

Model Parallelization - Reinforcement Learning (3/5)

- $\mathtt{J}(\mathtt{w}) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P} | \mathcal{G}, \mathtt{w})}[\mathtt{R}(\mathcal{P}) | \mathcal{G}]$
- Objective: $\arg\min_{\mathtt{w}} \mathtt{J}(\mathtt{w})$

- $\mathcal{G}$: input neural graph

Model Parallelization - Reinforcement Learning (3/5)

- $\mathtt{J}(\mathtt{w}) = \mathbb{E}_{\mathcal{P}\sim\pi(\mathcal{P}|\mathcal{G},\mathtt{w})}[\mathtt{R}(\mathcal{P})|\mathcal{G}]$
- Objective: $\arg\min_{\mathtt{w}} \mathtt{J}(\mathtt{w})$

- $\mathcal{G}$: input neural graph
- $\mathtt{R}$: runtime

Model Parallelization - Reinforcement Learning (3/5)

- $\mathtt{J}(\mathtt{w}) = \mathbb{E}_{\mathcal{P}\sim\pi(\mathcal{P}|\mathcal{G},\mathtt{w})}[\mathtt{R}(\mathcal{P})|\mathcal{G}]$
- Objective: $\arg\min_{\mathtt{w}} \mathtt{J}(\mathtt{w})$

- $\mathcal{G}$: input neural graph
- $\mathtt{R}$: runtime
- $\mathtt{J}(\mathtt{w})$: expected runtime

Model Parallelization - Reinforcement Learning (3/5)

- $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G},w)}[R(\mathcal{P})|\mathcal{G}]$
- Objective: $\arg\min_w J(w)$

- $\mathcal{G}$: input neural graph
- R: runtime
- $J(w)$: expected runtime
- w: trainable parameters of policy

Model Parallelization - Reinforcement Learning (3/5)

- $\mathtt{J}(\mathtt{w}) = \mathbb{E}_{\mathcal{P}\sim\pi(\mathcal{P}|\mathcal{G},\mathtt{w})}[\mathtt{R}(\mathcal{P})|\mathcal{G}]$
- Objective: $\arg\min_{\mathtt{w}} \mathtt{J}(\mathtt{w})$

- $\mathcal{G}$: input neural graph
- $\mathtt{R}$: runtime
- $\mathtt{J}(\mathtt{w})$: expected runtime
- $\mathtt{w}$: trainable parameters of policy
- $\pi(\mathcal{P}|\mathcal{G}, \mathtt{w})$: policy

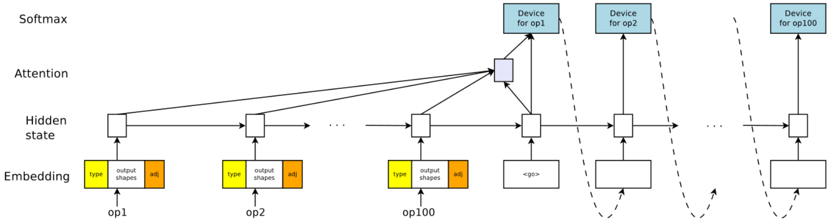
Model Parallelization - Reinforcement Learning (3/5)

- $\mathtt{J}(\mathtt{w}) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G},\mathtt{w})}[\mathtt{R}(\mathcal{P})|\mathcal{G}]$
- Objective: $\arg\min_{\mathtt{w}} \mathtt{J}(\mathtt{w})$

- $\mathcal{G}$: input neural graph
- $\mathtt{R}$: runtime
- $\mathtt{J}(\mathtt{w})$: expected runtime
- $\mathtt{w}$: trainable parameters of policy
- $\pi(\mathcal{P}|\mathcal{G}, \mathtt{w})$: policy
- $\mathcal{P}$: output placements $\in \{1, 2, ..., \mathit{num\_ops}\}^{\mathit{num\_devices}}$

Model Parallelization - Reinforcement Learning (4/5)

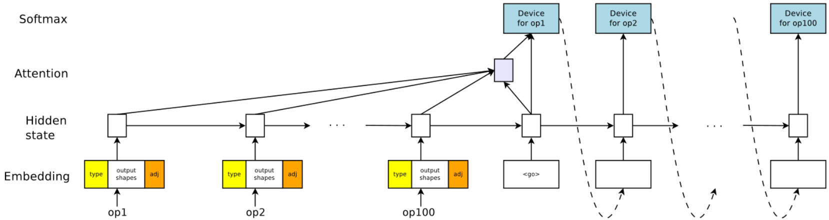
- RL reward function based on execution runtime.



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

Model Parallelization - Reinforcement Learning (4/5)

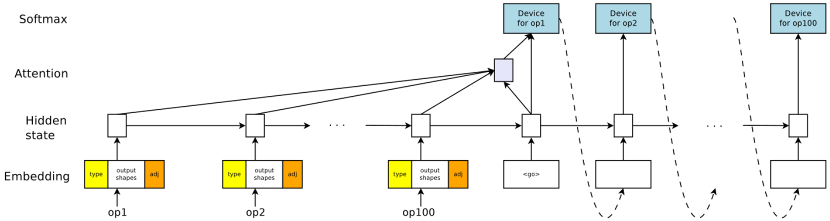
- RL reward function based on execution runtime.
- The RL policy is defined as a seq-to-seq model.



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

Model Parallelization - Reinforcement Learning (4/5)

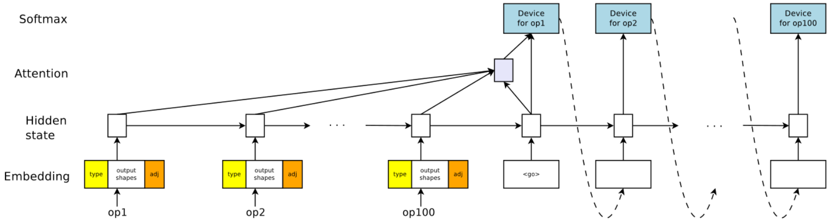
- RL reward function based on execution runtime.
- The RL policy is defined as a seq-to-seq model.
- RNN Encoder receives graph embedding for each operation.



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

Model Parallelization - Reinforcement Learning (4/5)

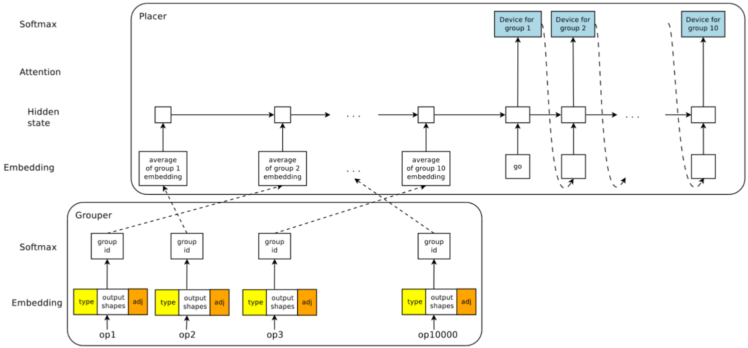
- RL reward function based on execution runtime.
- The RL policy is defined as a seq-to-seq model.
- RNN Encoder receives graph embedding for each operation.
- RNN Decoder predicts a device placement for each operation.



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

Model Parallelization - Reinforcement Learning (5/5)

- Grouping operations.
- Prediction is for group placement, not for a single operation.



[Mirhoseini et al., A Hierarchical Model for Device Placement, 2018]

Summary

Summary

- Scalability matters
- Parallelization
- Data Parallelization
  - Parameter server vs. AllReduce
  - Synchronized vs. asynchronized
- Model Parallelization
  - Random, critical path, multi-objective, RL

Thanks!