Neural Network Workflow Systems

CSE545 - Spring 2023

Stony Brook University

Adithya V Ganesan

with PyTorch

Big Data Analytics, The Class

Goal: Generalizations
A *model* or *summarization* of the data.

Data Workflow Frameworks

- Hadoop File System ✓
- Spark ✓
- Streaming ✓
- MapReduce ✓
- **Deep Learning Frameworks**

Analytics and Algorithms

- Similarity Search
- Hypothesis Testing
- Transformers/Self-Supervision
- Recommendation Systems
- Link Analysis

Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

However:

- Hadoop MapReduce can still be better for extreme IO, data that will not fit in memory across cluster.

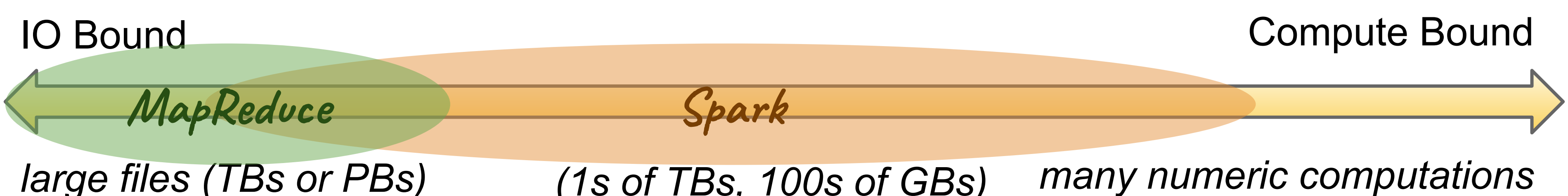
Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

However:

- Hadoop MapReduce can still be better for extreme IO, data that will not fit in memory across cluster.



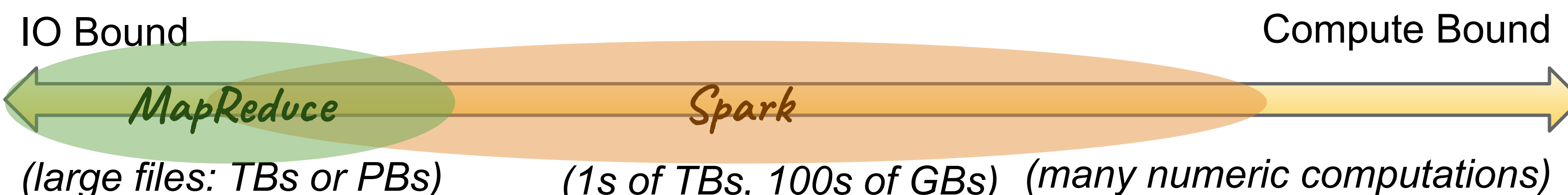
Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

However:

- Hadoop MapReduce can still be better for extreme IO, data that will not fit in memory across cluster.
- Modern machine learning (esp. Deep learning), a common big data task, requires heavy numeric computation.



* this is the subjective approximation of the instructor as of February 2020. A lot of factors at play.

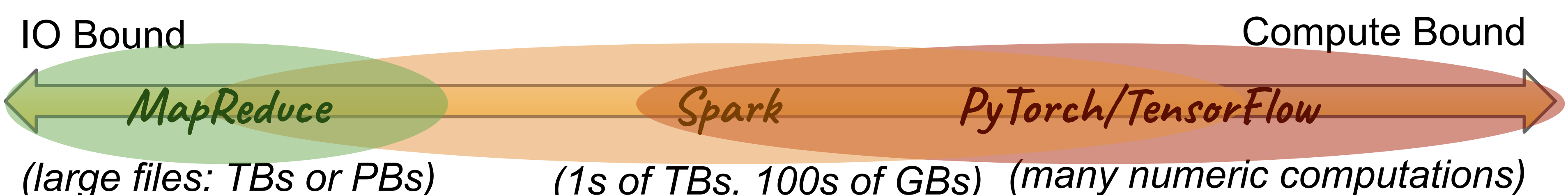
Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

However:

- Hadoop MapReduce can still be better for extreme IO, data that will not fit in memory across cluster.
- Modern machine learning (esp. Deep learning), a common big data task, requires heavy numeric computation.



* this is the subjective approximation of the instructor as of February 2020. A lot of factors at play.

Learning Objectives

- Understand a neural network as transformations on tensors.
- Understand PyTorch as a data workflow system.
  - Know the key components of PyTorch
  - Understand the key concepts around *distributed* neural network processing in PyTorch.
- Establish a foundation to distribute deep learning models

Linear Regression

Linear Regression: $\hat{y} = \beta X$

Objective: *Learn w, such that* $(y - \beta X)^2$ *is minimized*

Linear Regression

Linear Regression: $\hat{y} = \beta X$

Objective: *Learn w, such that $(y - \beta X)^2$ is minimized*

How do we solve for β?

Linear Regression

Linear Regression: $\hat{y} = \beta X$

Objective: *Learn w, such that* $(y - \beta X)^2$ *is minimized*

How do we solve for β?

1. Analytic Gradient: Differentiate the objective, solve the system of equations by equating it to 0

Linear Regression

Linear Regression: $\hat{y} = \beta X$

Objective: *Learn w, such that* $(y - \beta X)^2$ *is minimized*

How do we solve for β?

1. Analytic Gradient: Differentiate the objective, solve the system of equations by equating it to 0

$$\beta_{opt} = (X^T X)^{-1} X^T y$$

Linear Regression

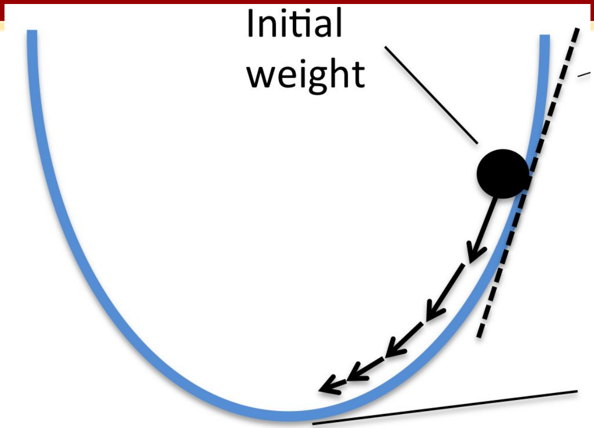
Linear Regression: $\hat{y} = \beta X$

Objective: *Learn w, such that $(y - \beta X)^2$ is minimized*

How do we solve for β?

1. Analytic Gradient: Differentiate the objective, solve the system of equations by equating it to 0
2. Numerical Gradient: Start at a random point and move in the direction of minima until optima is reached

Linear Regression



Linear Regression: $\hat{y} = \beta X$

Objective: *Learn w, such that $(y - \beta X)^2$ is minimized*

How do we solve for β?

1. Analytic Gradient: Differentiate the objective, solve the system of equations by equating it to 0
2. **Numerical Gradient: Start at a random point and move in the direction of minima until optima is reached**

Numerical Gradient Approach

Linear Regression: Trying to find “betas” that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

Numerical Gradient Approach

Linear Regression: Trying to find “betas” that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

matrix multiply

$$\hat{y}_i = X_i\beta$$

Numerical Gradient Approach

Linear Regression: Trying to find “betas” that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

matrix multiply

$$\hat{y}_i = X_i\beta$$

Thus:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i - X_i\beta)^2\}$$

Numerical Gradient Approach

Linear Regression: Trying to find “betas” that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

$\hat{y}_i = X_i\beta$ Thus: $\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i - X_i\beta)^2\}$

How to update? $\beta_{new} = \beta_{prev} - \alpha * \text{grad}$

Numerical Gradient Approach

Linear Regression: Trying to find "betas" that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

$\hat{y}_i = X_i\beta$ Thus: $$\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i - X_i\beta)^2\}$$

How to update? $\beta_{new} = \beta_{prev} - \alpha * \text{grad}$

α: Learning Rate

Numerical Gradient Approach

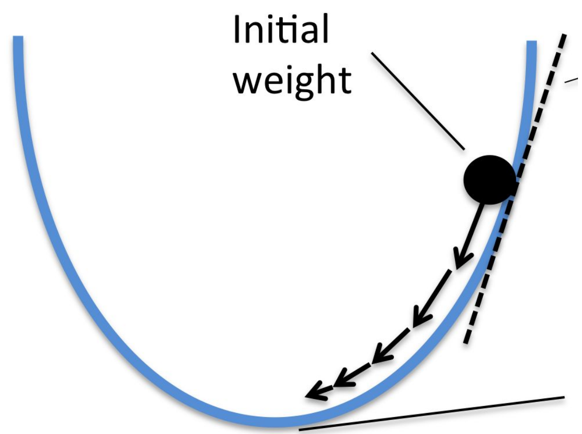
Linear Regression: Trying to find "betas" that minimize:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i}^{N}(y_i - \hat{y}_i)^2\}$$

$$\hat{y}_i = X_i\beta$$

Thus:

$$\hat{\beta} = argmin_{\beta}\{\sum_{i=0}^{N}(y_i -$$



How to update? $\beta_{new} = \beta_{prev} - \alpha * \text{grad}$

α: Learning Rate

Numerical Gradient Approach

Linear Regression: Trying to find "betas" that minimize:

Gradient Descent: $\beta_{new} = \beta_{prev} - \alpha * \text{grad}$

Numerical Gradient Approach

Linear Regression: Trying to find “betas” that minimize:

Gradient Descent: $\beta_{new} = \beta_{prev} - \alpha * \mathrm{grad}$

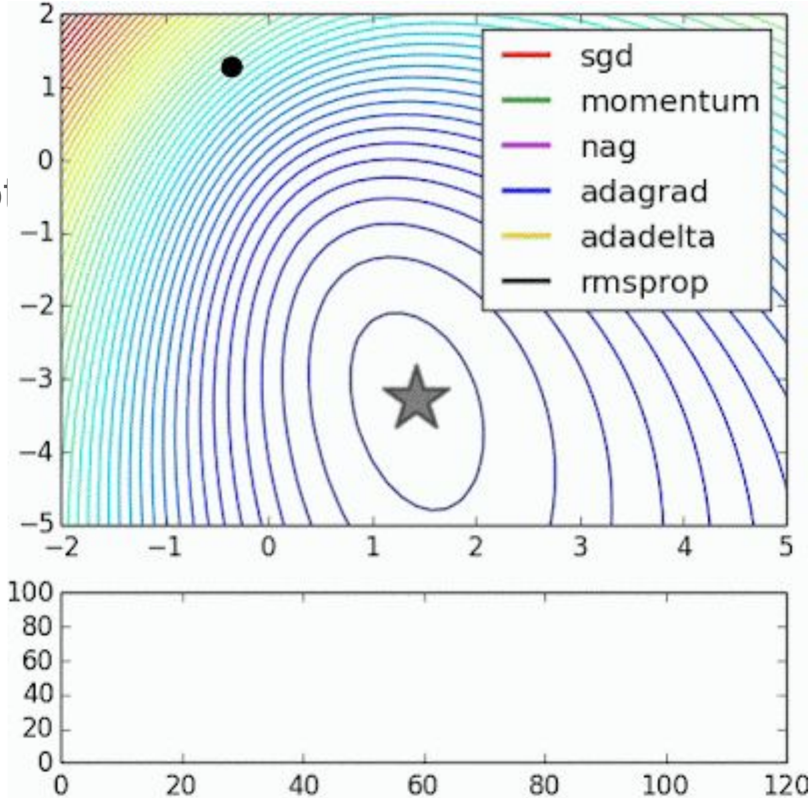
But there are other gradient descent based optimization methods which are better*

Numerical Gradient Approach

Linear Regression: Trying to find “betas” that minimize:

Gradient Descent: $\beta_{new} = \beta_{prev} - \alpha * \text{grad}$

But there are other gradient descent based opt



Animation: Alec Radford

Linear Regression as DAG

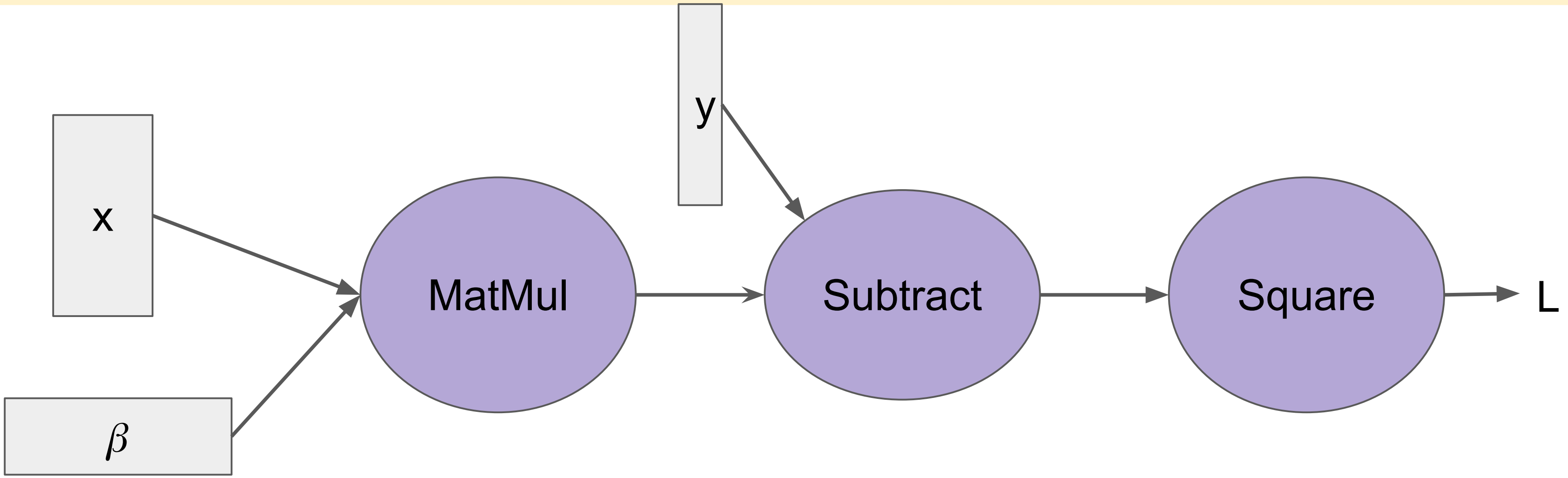
How do Machine learning/ Deep learning frameworks represent these models?

Linear Regression as DAG

How do Machine learning/ Deep learning frameworks represent these models?

Computational Graph!

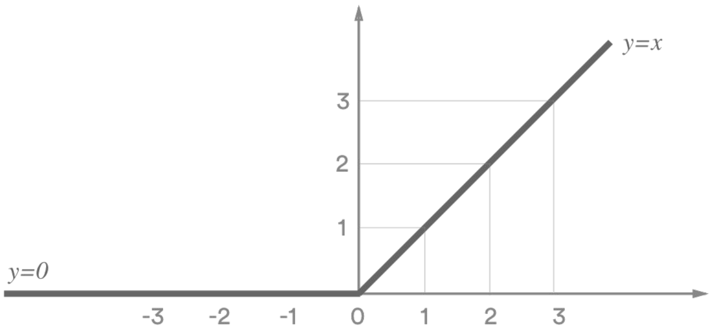
Linear Regression as DAG



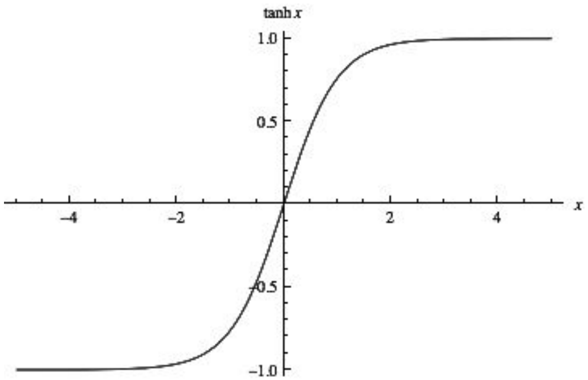
$$L = (y - \beta x)^2$$

Activations

Rectified linear unit (ReLU): $ReLU(z) = \max(0, z)$



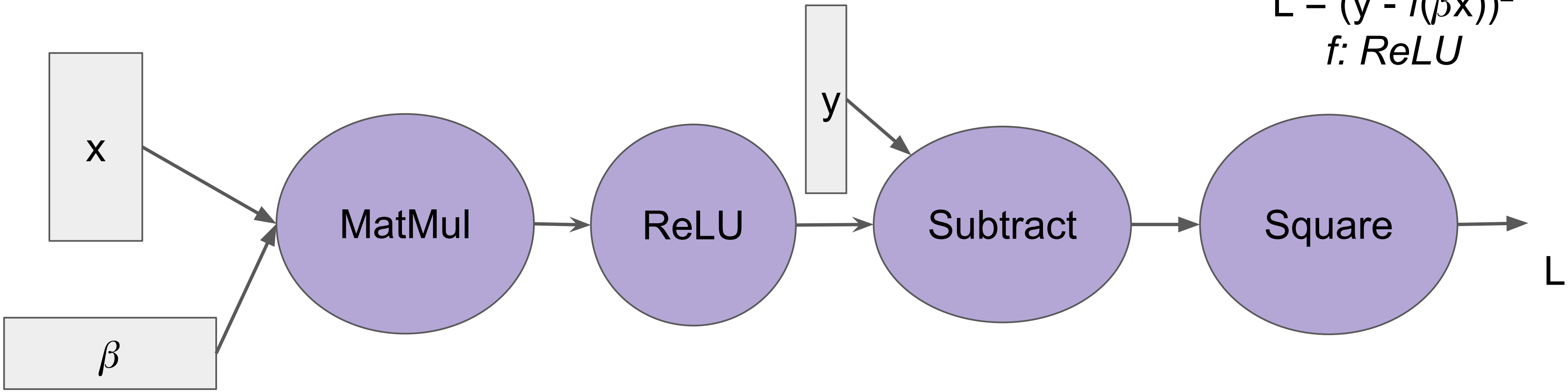
Hyperbolic tangent: $tanh(z) = (e^{2z} - 1) / (e^{2z} + 1)$



Linear Regression as DAG

$L = (y - f(\beta x))^2$

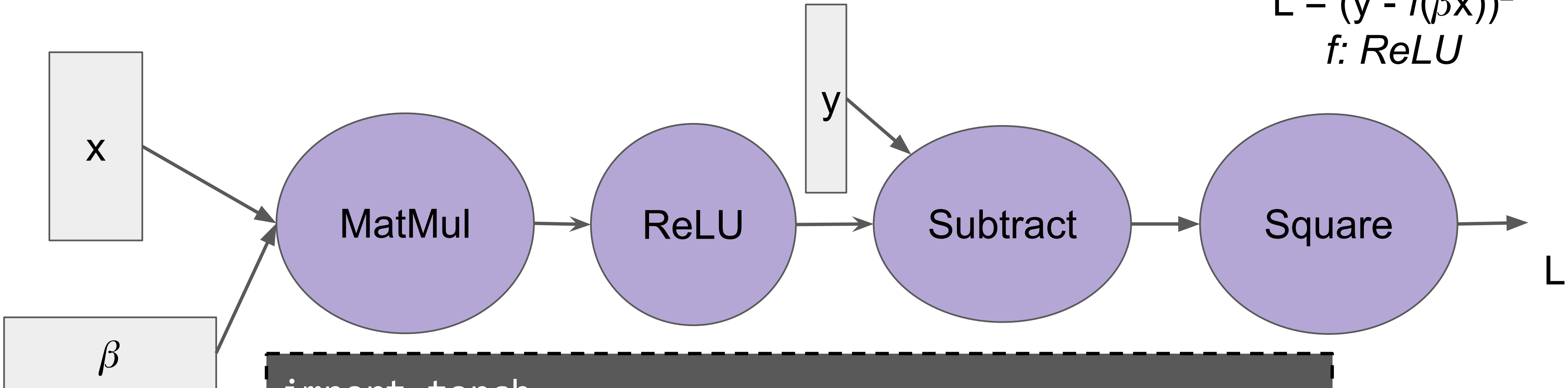
f: ReLU



Linear Regression as DAG

$$L = (y - f(\beta x))^2$$

f: ReLU




```
import torch
from torch import nn

x = torch.Tensor(input)
beta = torch.random.randn(X.shape, 1)
z = torch.matmul(x, beta)
yhat = nn.functional.relu(z)
loss = nn.MSELoss(yhat, torch.Tensor(y))
```

PyTorch Demo

Native Linear Regression Implementation (Link)

Torch.nn Linear Regression Implementation (Link)

How to train GPT3?

Time to train Bert Large (330 M) on K80, which is 530 times smaller than GPT3

# GPUs	Training Time (minutes)	Per-GPU Scaling Efficiency
1	399	1.00

How to train GPT3?

Time to train Bert Large (330 M) on K80, which is 530 times smaller than GPT3

# GPUs	Training Time (minutes)	Per-GPU Scaling Efficiency
1	399	1.00

For the same amount of data, GPT3 can be trained in 212k mins = 3533 hours = 147 days*

**GPT3 wont fit into the memory of a single K80*

How to train GPT3?

Time to train Bert Large (330 M) on K80, which is 530 times smaller than GPT3

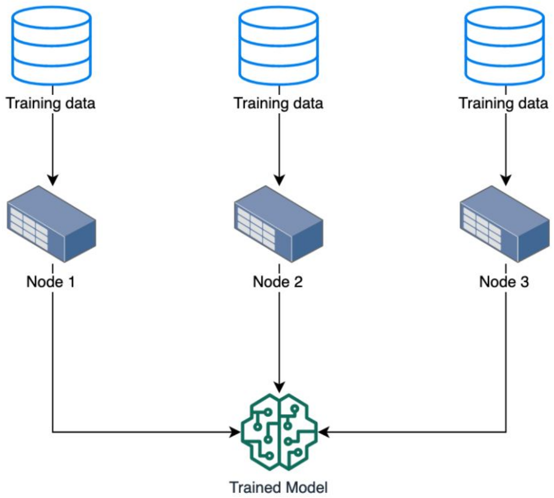
# GPUs	Training Time (minutes)	Per-GPU Scaling Efficiency
1	399	1.00
2	214	0.93
4	118	0.85
8	61	0.82

Distributed Training

- Parallelism :
  - Data Parallelism
  - Model Parallelism
  - Hybrid

Distributed PyTorch Training

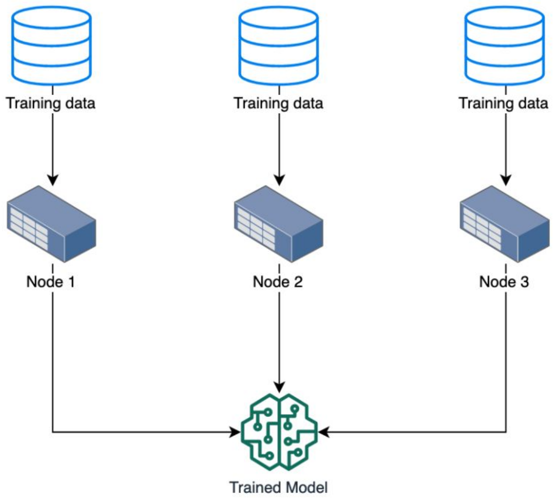
- Data Parallelism: Scatter dataset into parts across different workers to train on subsets and sync gradients



Data Parallelism

Distributed PyTorch Training

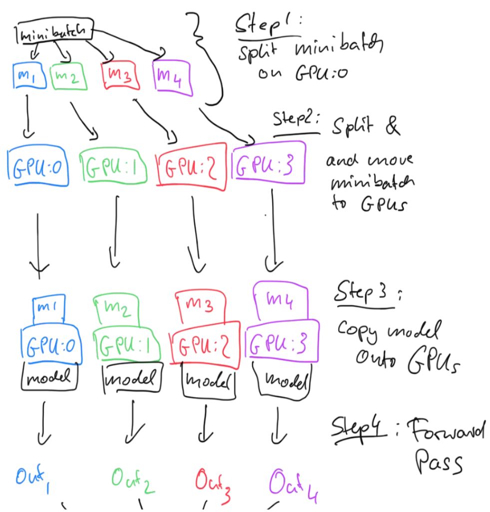
- Data Parallelism: Scatter dataset into parts across different workers to train on subsets and sync gradients
- Modes of Data Parallelism :
  - DataParallel
  - DistributedDataParallel

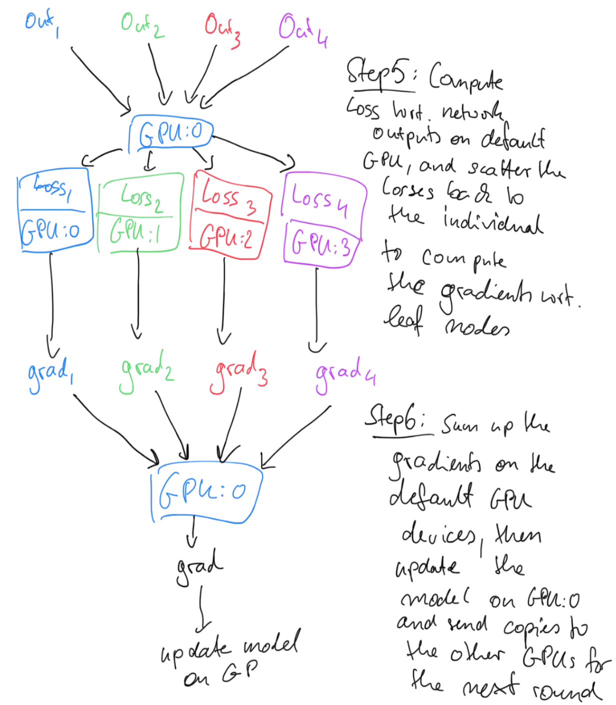


Data Parallelism

Distributed PyTorch Training

Data Parallel: How it works?





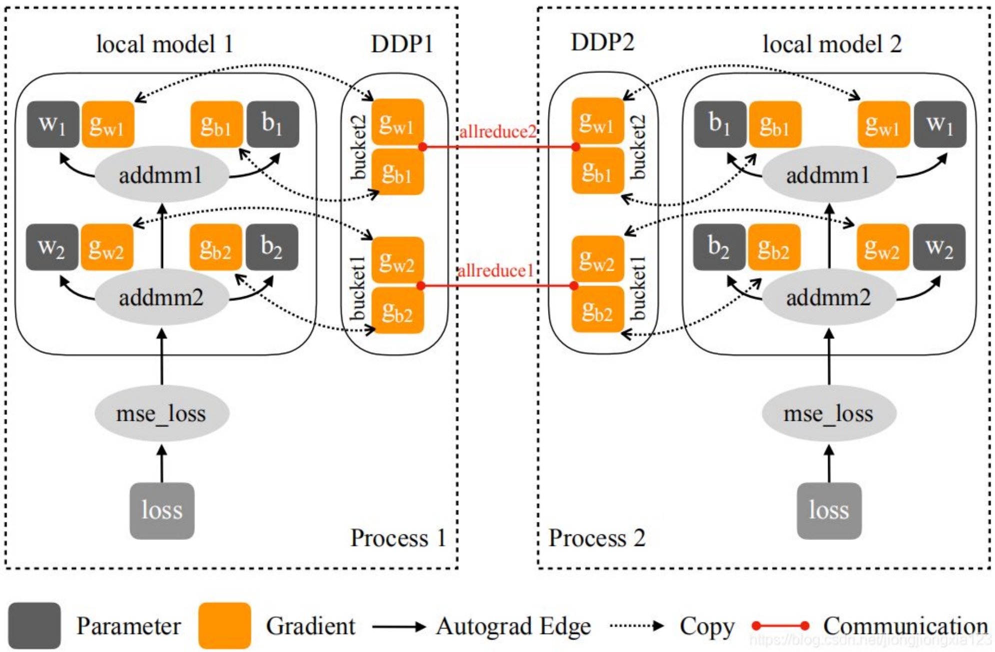
https://erickguan.me/2019/pytorch-parallel-model

Distributed PyTorch Training

- Data Parallel
  - Most simple form of parallelism with minimal code change
  - Downside: Slower form of parallelism - involves inter node communication 3x per training step

Distributed PyTorch Training

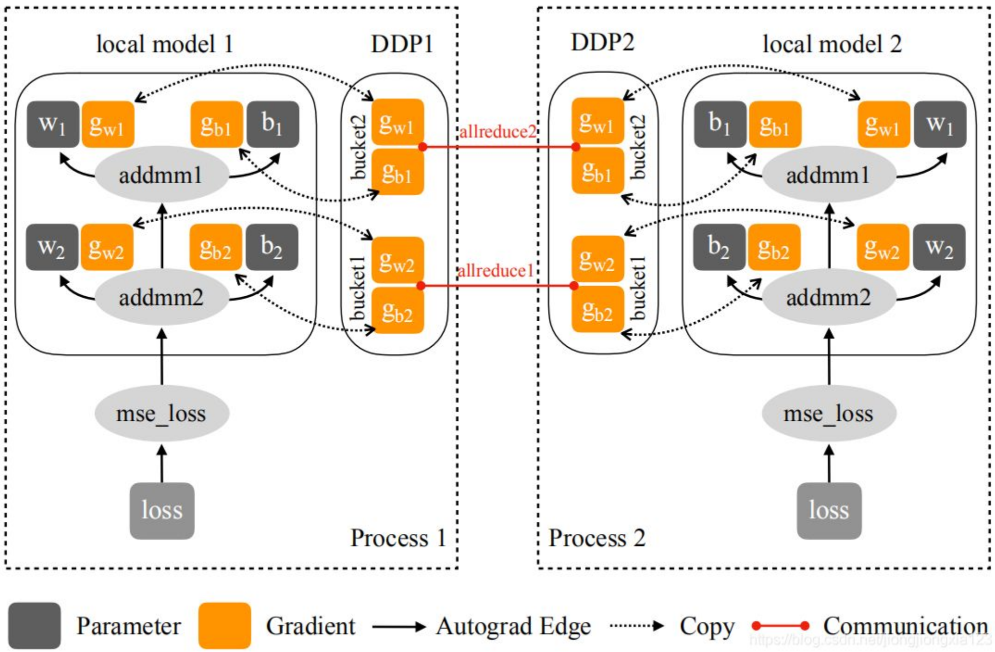
DistributedDataParallel: How it works?

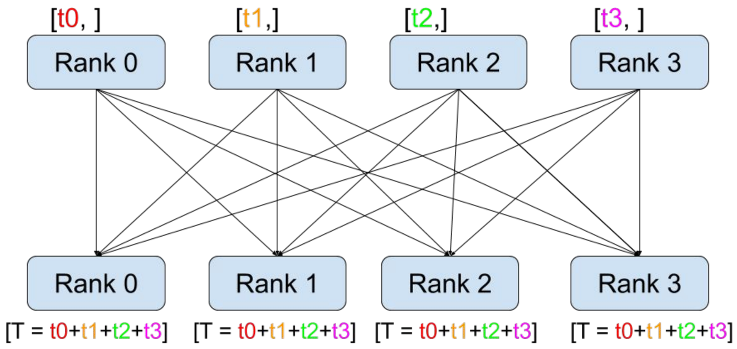


([Li et al., 2020](#))

Distributed PyTorch Training

- DistributedDataParallel





AllReduce

(Li et al., 2020)

Distributed PyTorch Training

- DistributedDataParallel (Li et al., 2020)
  - Efficient form of parallelism but involves a little extra code change*
  - Performs AllReduce on the computed gradients across all nodes and machines

** Extra code change if you are implementing using Pytorch. It has been made extremely simple by*

Distributed PyTorch Training

- DistributedDataParallel ([Li et al., 2020](#))
  - Efficient form of parallelism but involves a little extra code change*
  - Performs AllReduce on the computed gradients across all nodes and machines
  - Downside: Python pickles all objects while spawning multiple processes (which happens in DDP). Code might crash if an object is not pickle-able

** Extra code change if you are implementing using Pytorch. It has been made extremely simple by*

Distributed PyTorch Training

Model Parallelism: Distribute layer(s) of the model into different machines/GPUs to train a very large network.

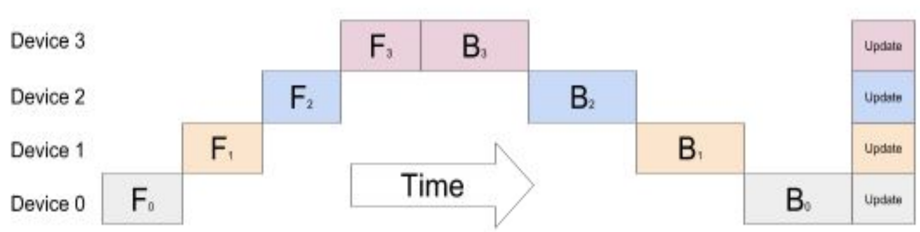
Distributed PyTorch Training

- Model Parallelism: Distribute layer(s) of the model into different machines/GPUs to train a very large network.

- Model Parallelism
    - Naive Model Parallelism
    - Pipelined Parallelism

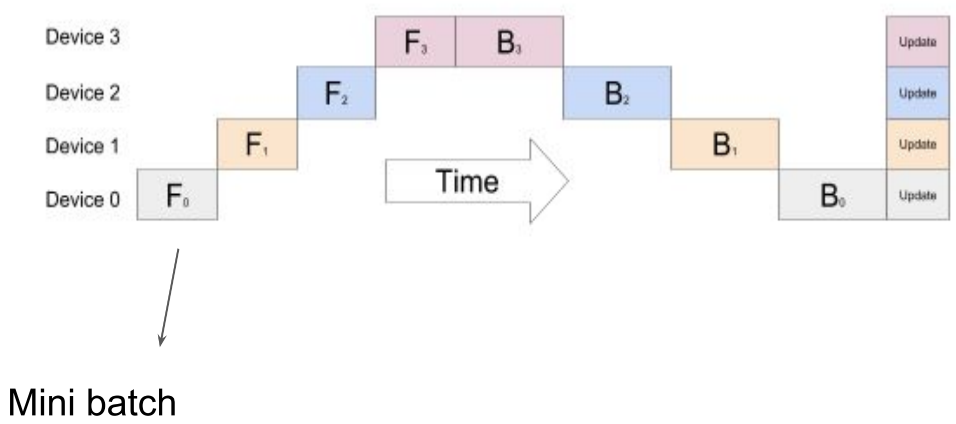
Distributed PyTorch Training

- Naive Model Parallelism



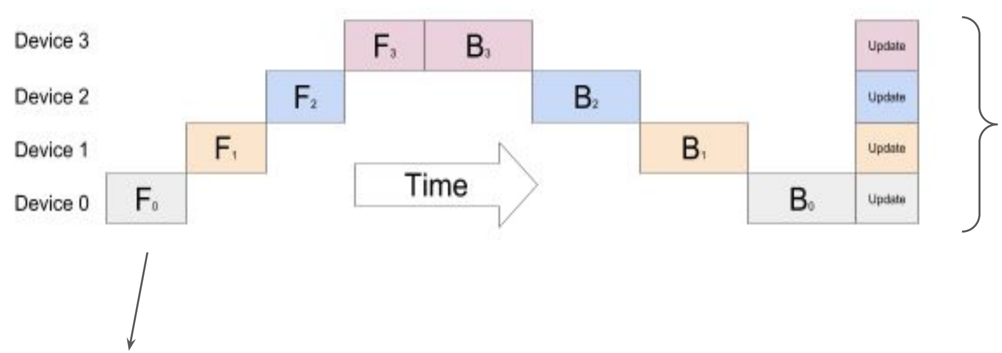
Distributed PyTorch Training

- Naive Model Parallelism



Distributed PyTorch Training

- Naive Model Parallelism



Synchronous Gradient updates at the end of a batch

Mini batch

Distributed PyTorch Training

- Naive Model Parallelism

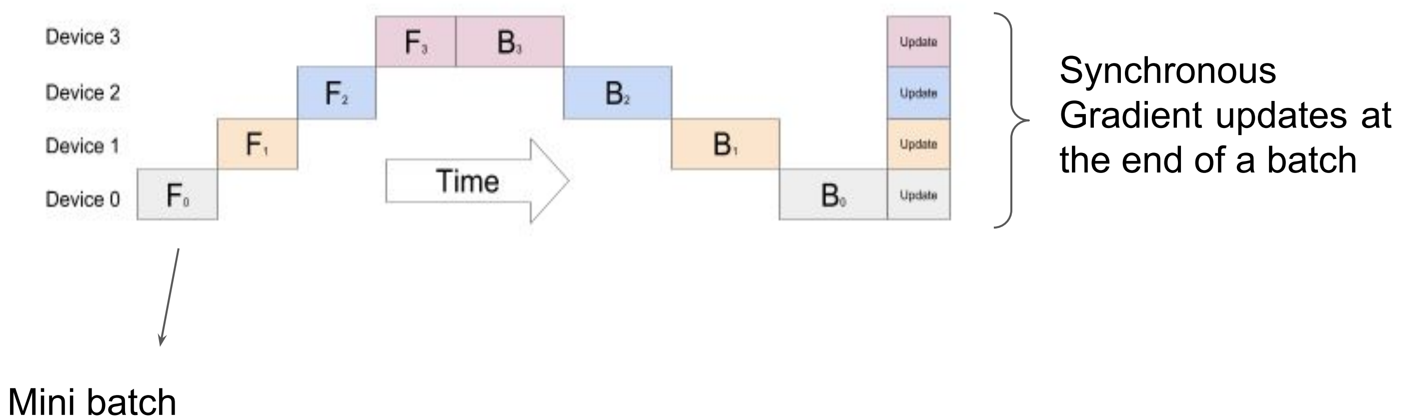

```
class ToyModel(nn.Module): #Pytorch: model_parallel_tutorial
    def __init__(self):
        super(ToyModel, self).__init__()
        self.net1 = torch.nn.Linear(10, 10).to('cuda:0')
        self.relu = torch.nn.ReLU()
        self.net2 = torch.nn.Linear(10, 5).to('cuda:1')

    def forward(self, x):
        x = self.relu(self.net1(x.to('cuda:0')))
        return self.net2(x.to('cuda:1'))
```

nous
updates at
of a batch

Distributed PyTorch Training

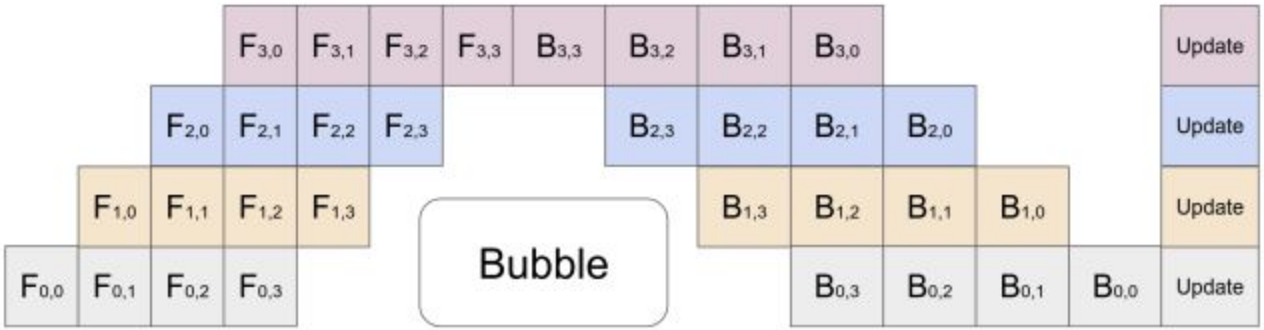
- Naive Model Parallelism



Severe under utilization of resources due to sequential dependency of the network

Distributed PyTorch Training

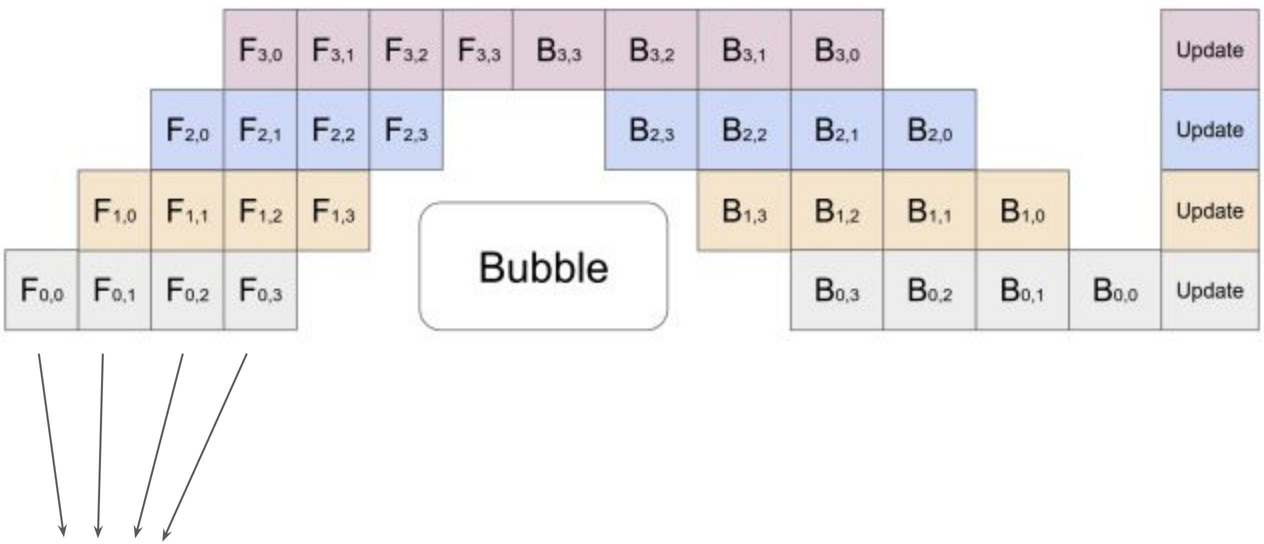
- Pipelined Parallelism



Huang et al., 2019

Distributed PyTorch Training

- Pipelined Parallelism

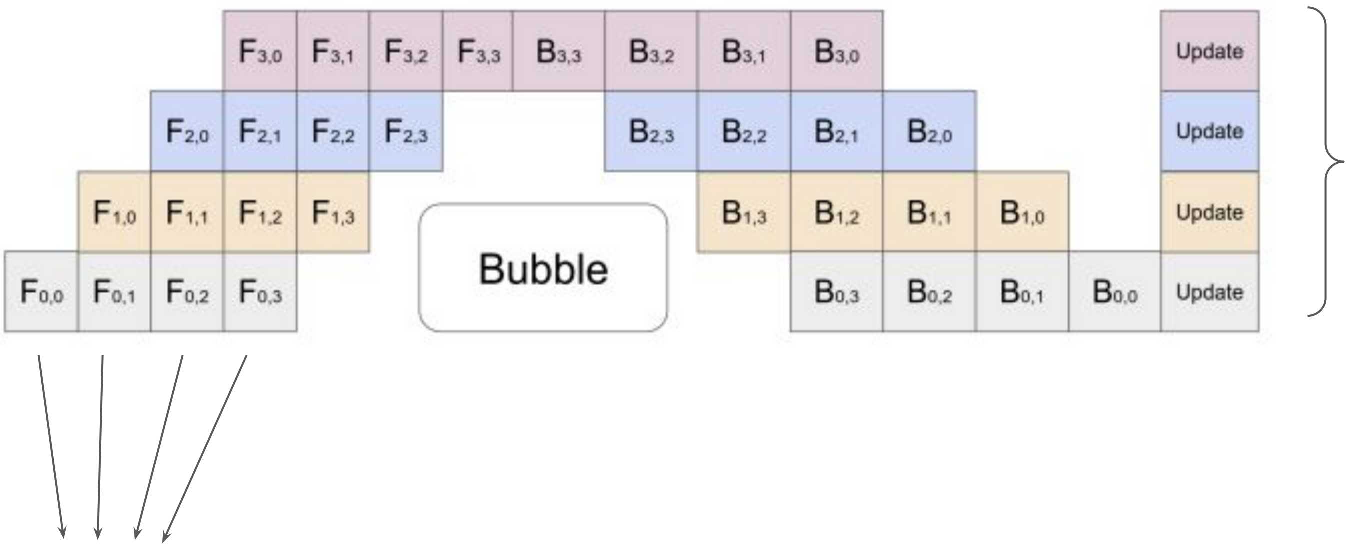


Mini batch split into
micro batches

[Huang et al., 2019](#)

Distributed PyTorch Training

- Pipelined Parallelism



Synchronous Gradient updates at the end of a batch

Mini batch split into micro batches

[Huang et al., 2019]

Distributed PyTorch Training

- Pipelined Parallelism

Synchronous Gradient updates at the end of a batch

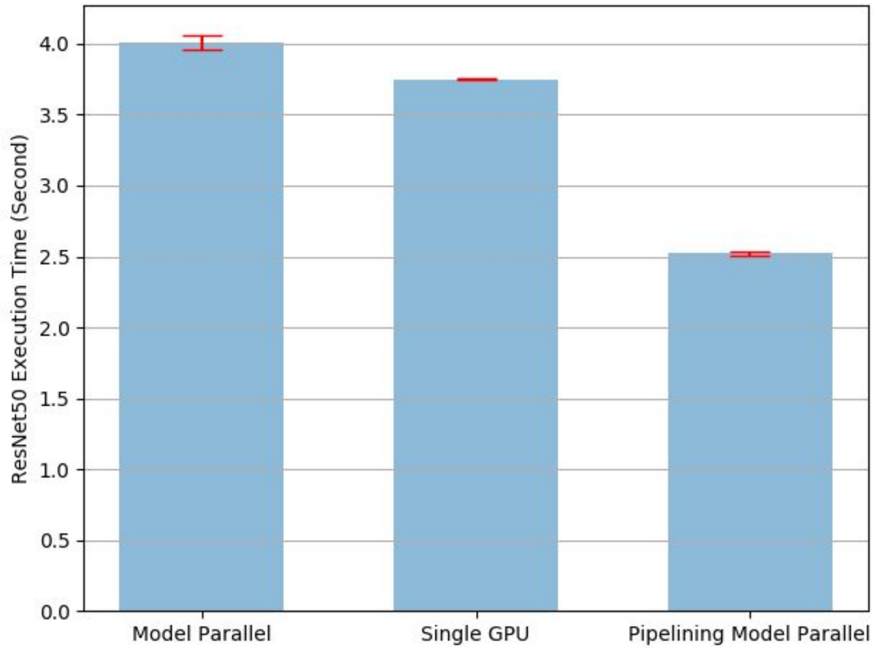
Mini batch split into micro batches

Provides high utilization of workers while ensuring reliable + stable training

[Huang et al., 2019](#)

Distributed PyTorch Training

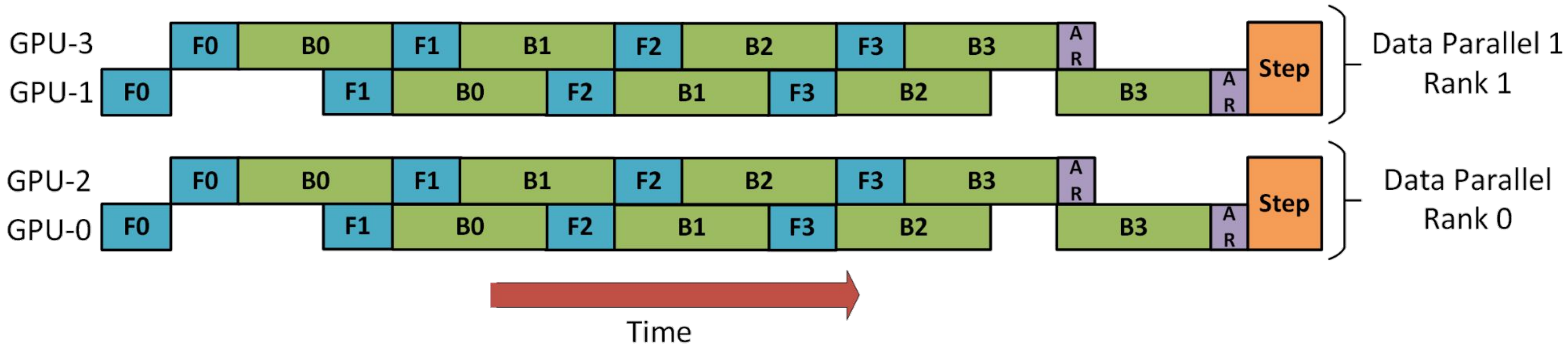
- Pipelined Parallelism



PyTorch: [Model Parallel best practices]()

Distributed PyTorch Training

- Hybrid
  - DeepSpeed (Rasley et al., 2020)



Horovod: PyTorch 🤝 PySpark

Horovod is a distributed deep learning training framework.

Horovod helps scaling single GPU (worker) into multi-GPU or even multi-host training without no code change

Horovod on spark: "provides a convenient wrapper around Horovod that makes running distributed training jobs in Spark clusters easy"