Parallelized Stochastic Gradient Descent

第八組 張家銘 繆穩慶 麥佩琦

Outline

- Introduction
- Problem statement
- Proposed approach
- Experiment and evaluation
- Conclusion

Introduction

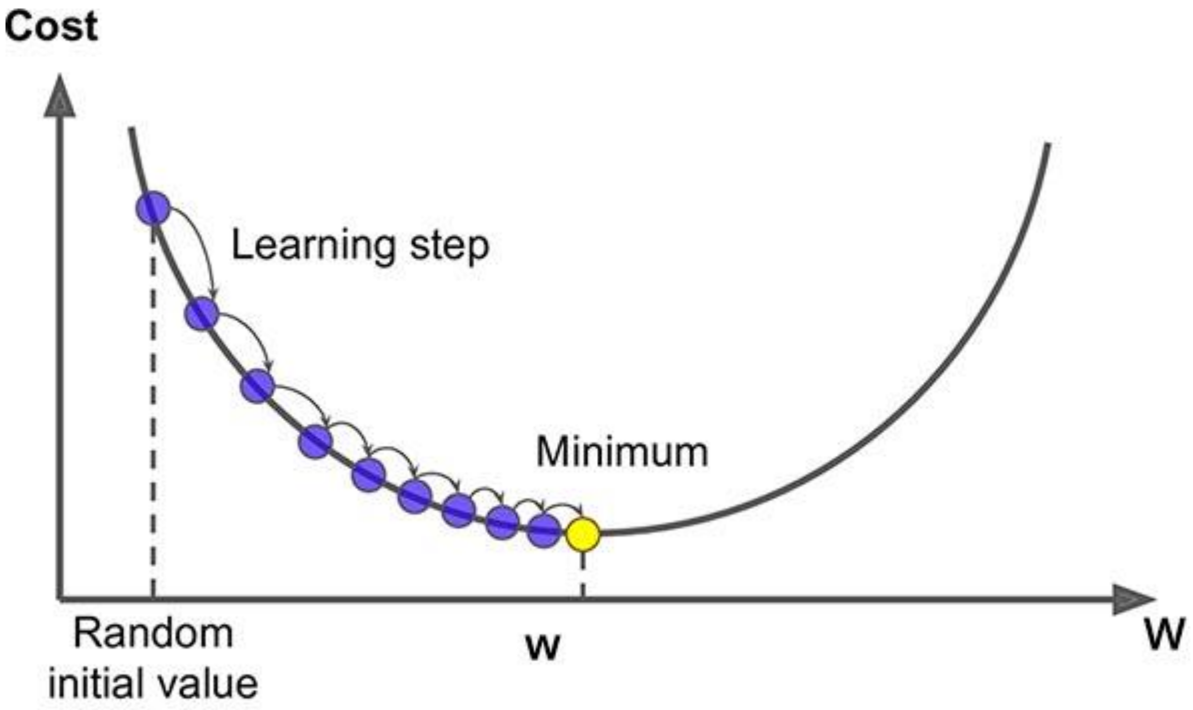
| Gradient Descent

- Iteratively moves toward parameter to find one that minimize the loss function

$$x_{t+1} = x_t - \eta g_{GD}$$

Introduction

| Gradient Descent



Introduction

| Gradient Descent

- For each iteration, compute all pair of examples in the training set to get gradient

→ **Long computation time**

Introduction

| Stochastic Gradient Descent

- At each iteration, randomly choose one sample to calculate
- Mini-batch SGD: choose a subset of training set

→ **Much faster**

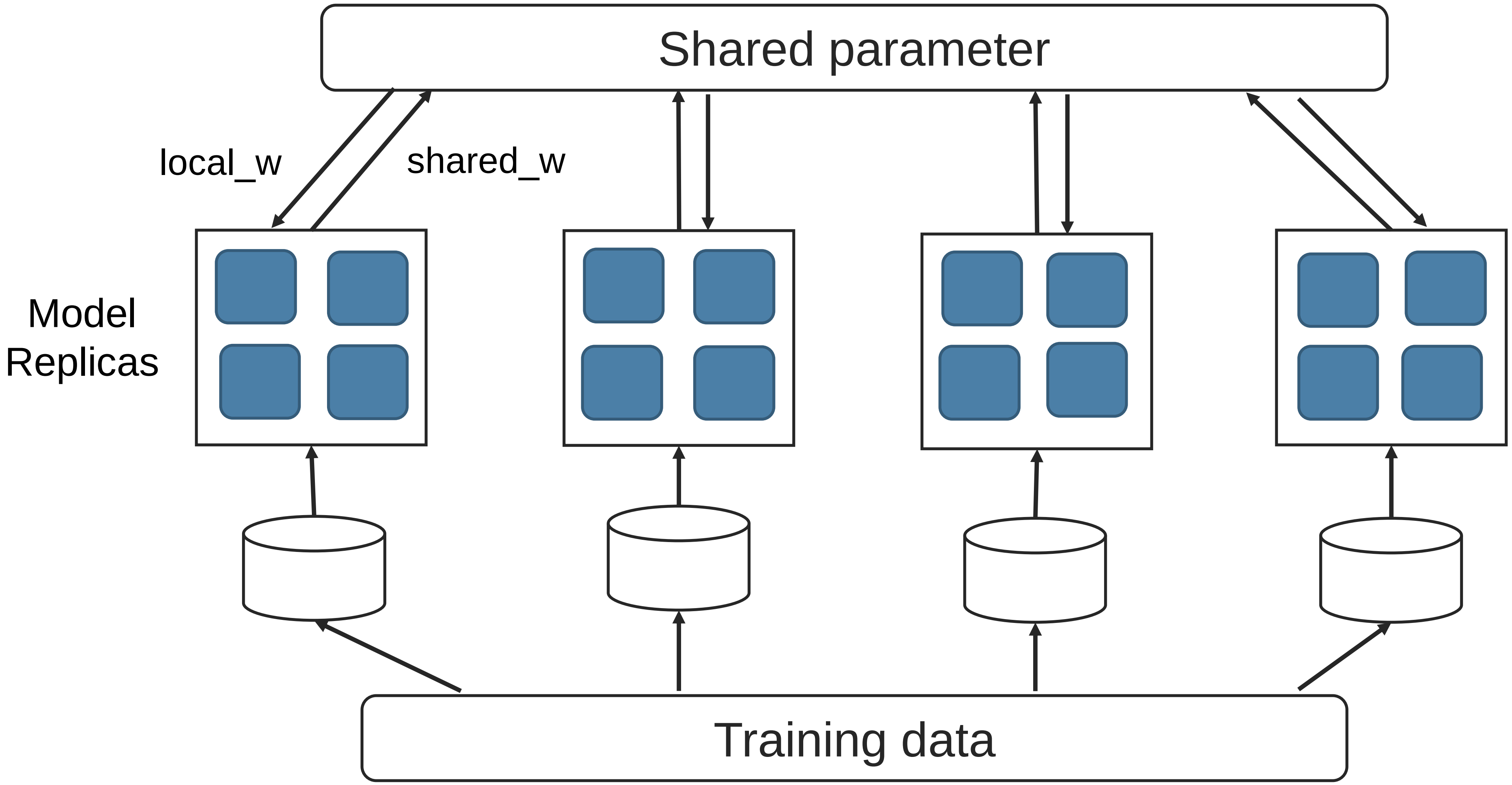
Problem statement

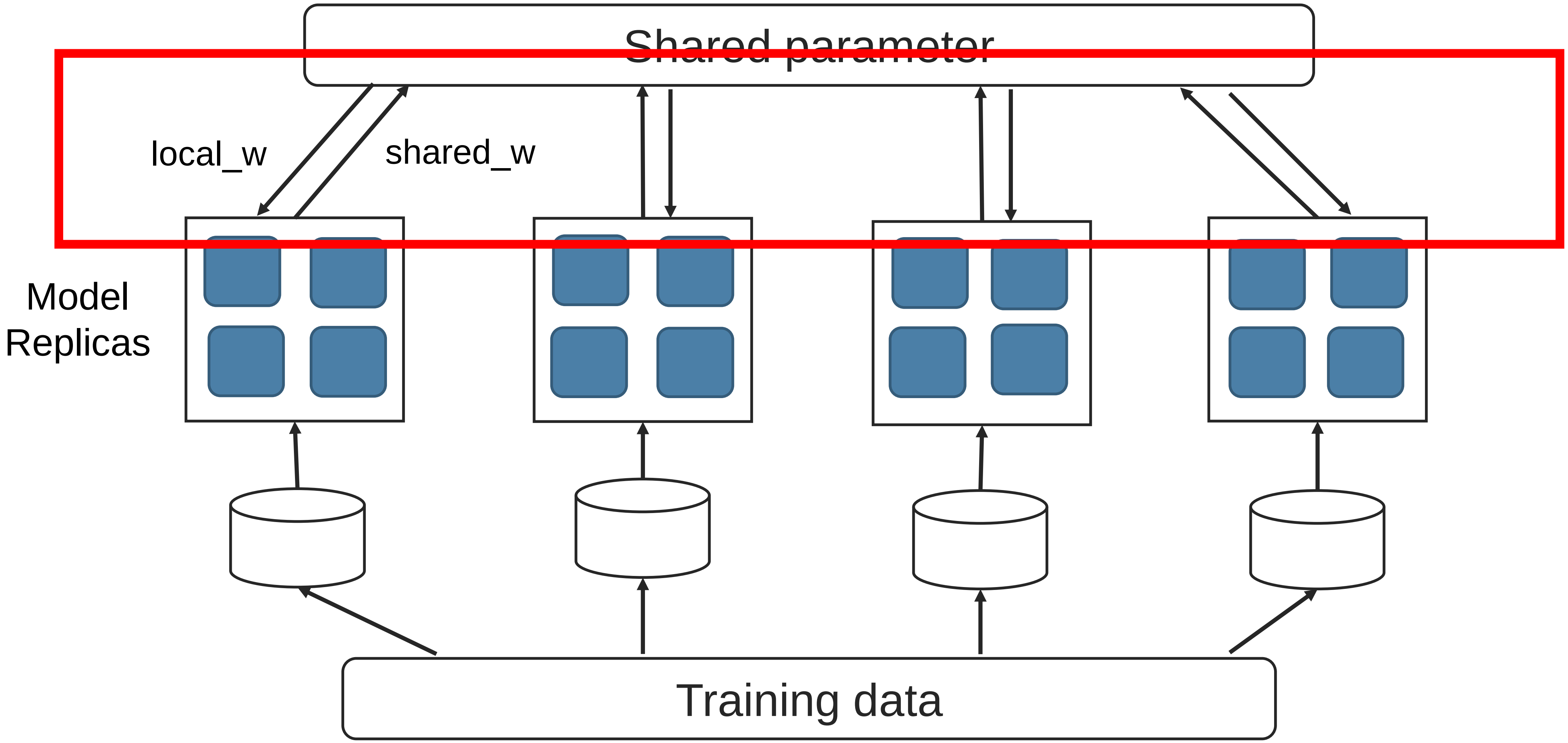
However, GD/SGD are inherently serial due to their iterative nature

- Do parallelism when mini-batches calculate gradient
- Update parameters with reduce technique

Proposed approach

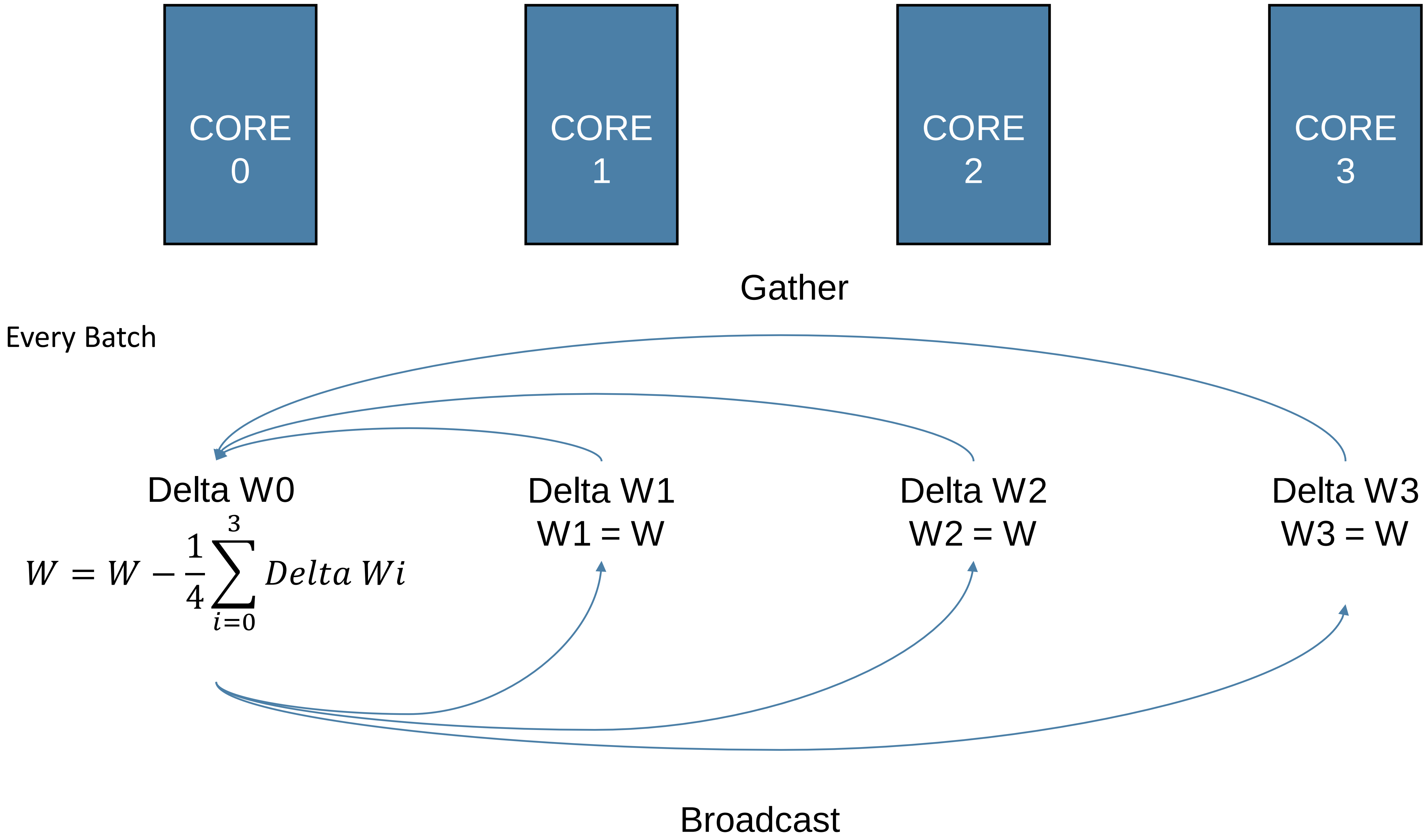
- Divide training set into several mini-batches. Each mini-batch is assigned to one processor.
- We apply two methods to update the shared parameters.


Shared parameter
local_w
shared_w
Model
Replicas
Training data


Shared parameter
local_w
shared_w
Model
Replicas
Training data

Method 1: Synchronous

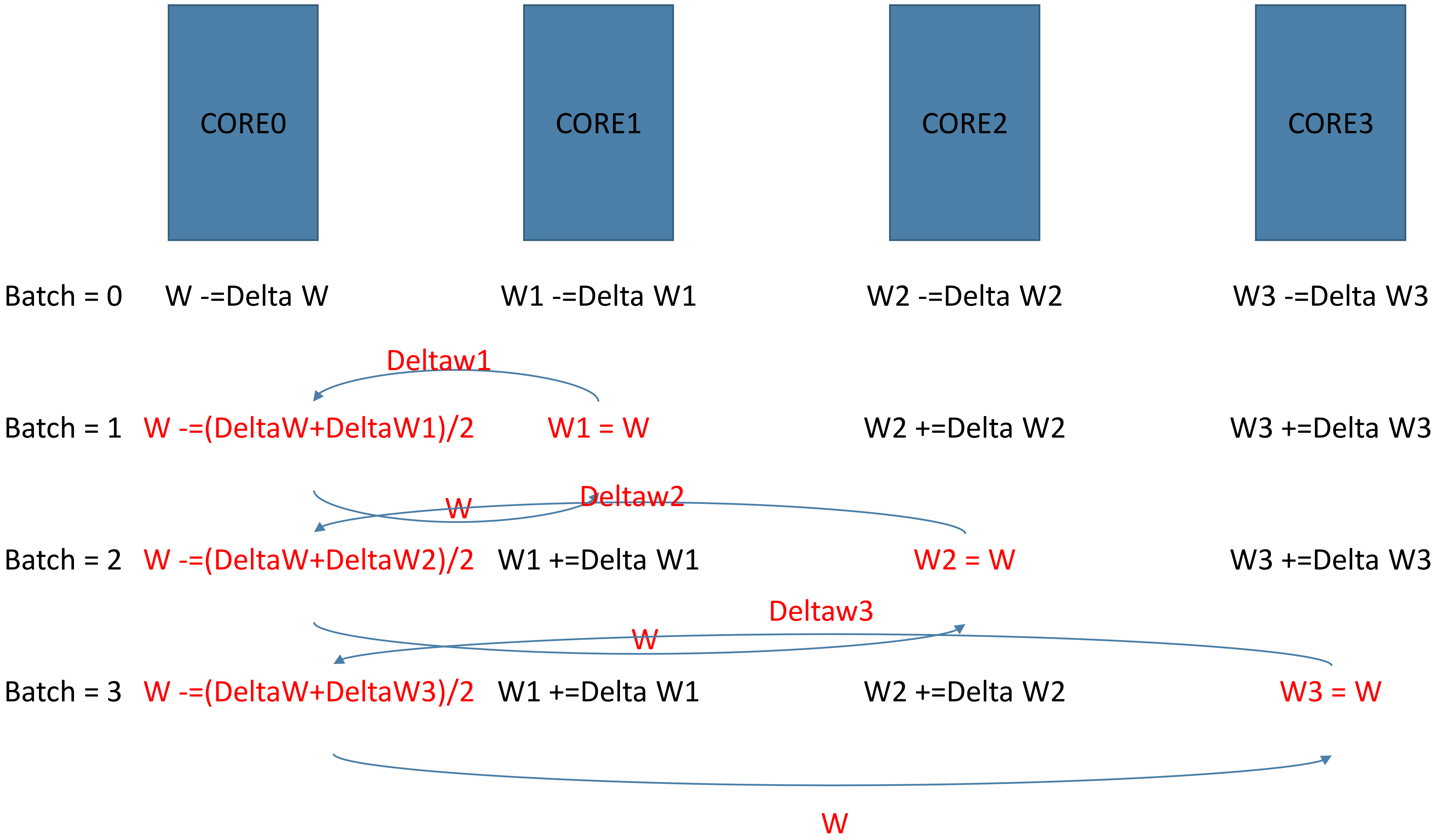
- For each processor i
  - Initialize local parameter w_i = shared parameter v
  - Calculate SGD
- With reduce technique, update share parameter $v = \frac{1}{c}\sum_{i=1}^{c} w_i$

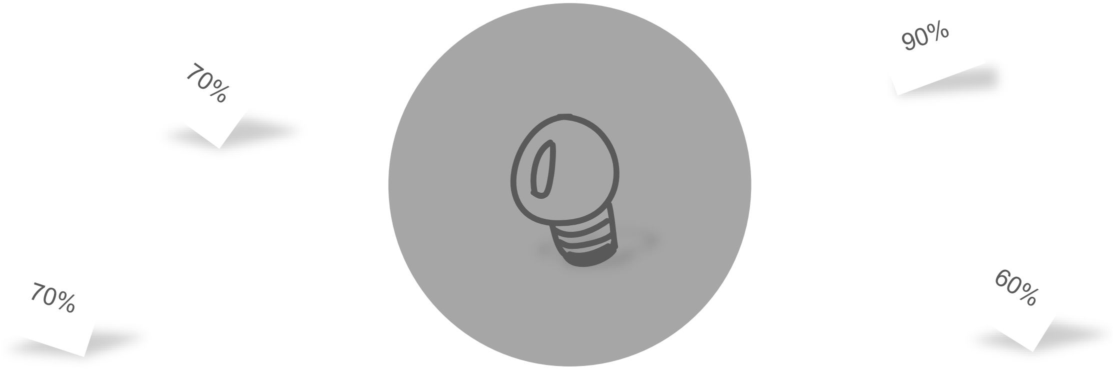

CORE 0
CORE 1
CORE 2
CORE 3
Gather
Every Batch
Delta W0
Delta W1
W1 = W
Delta W2
W2 = W
Delta W3
W3 = W
$W = W - \frac{1}{4}\sum_{i=0}^{3} Delta\ Wi$
Broadcast

Method 2: Asynchronous

Prerequisite: the speed of each processor is similar

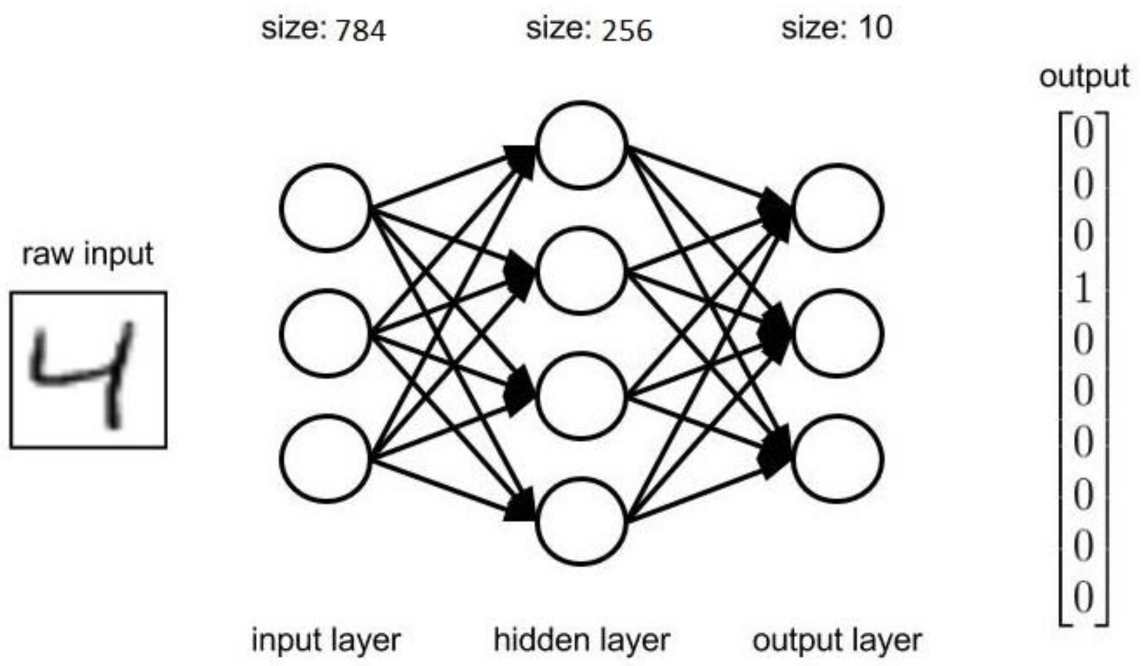
- For each processor i
  - Calculate SGD with local parameter w_i
- Send w_i to update share parameter v in turn and receive updated v as w_i


CORE0
CORE1
CORE2
CORE3
Batch = 0
W -=Delta W
W1 -=Delta W1
W2 -=Delta W2
W3 -=Delta W3
Deltaw1
Batch = 1
W -=(DeltaW+DeltaW1)/2
W1 = W
W2 +=Delta W2
W3 +=Delta W3
W
Deltaw2
Batch = 2
W -=(DeltaW+DeltaW2)/2
W1 +=Delta W1
W2 = W
W3 +=Delta W3
Deltaw3
W
Batch = 3
W -=(DeltaW+DeltaW3)/2
W1 +=Delta W1
W2 +=Delta W2
W3 = W
W



Experiment & evaluation

Experiment



Hyperparameters :

- Activation function : $sigmoid$
- Learning Rate : 1 (Reduce lr by 1% after each epoch)
- Epochs : 5
- Batch size : 100

Optimizer : mini-batch SGD

Experiment

Environment :

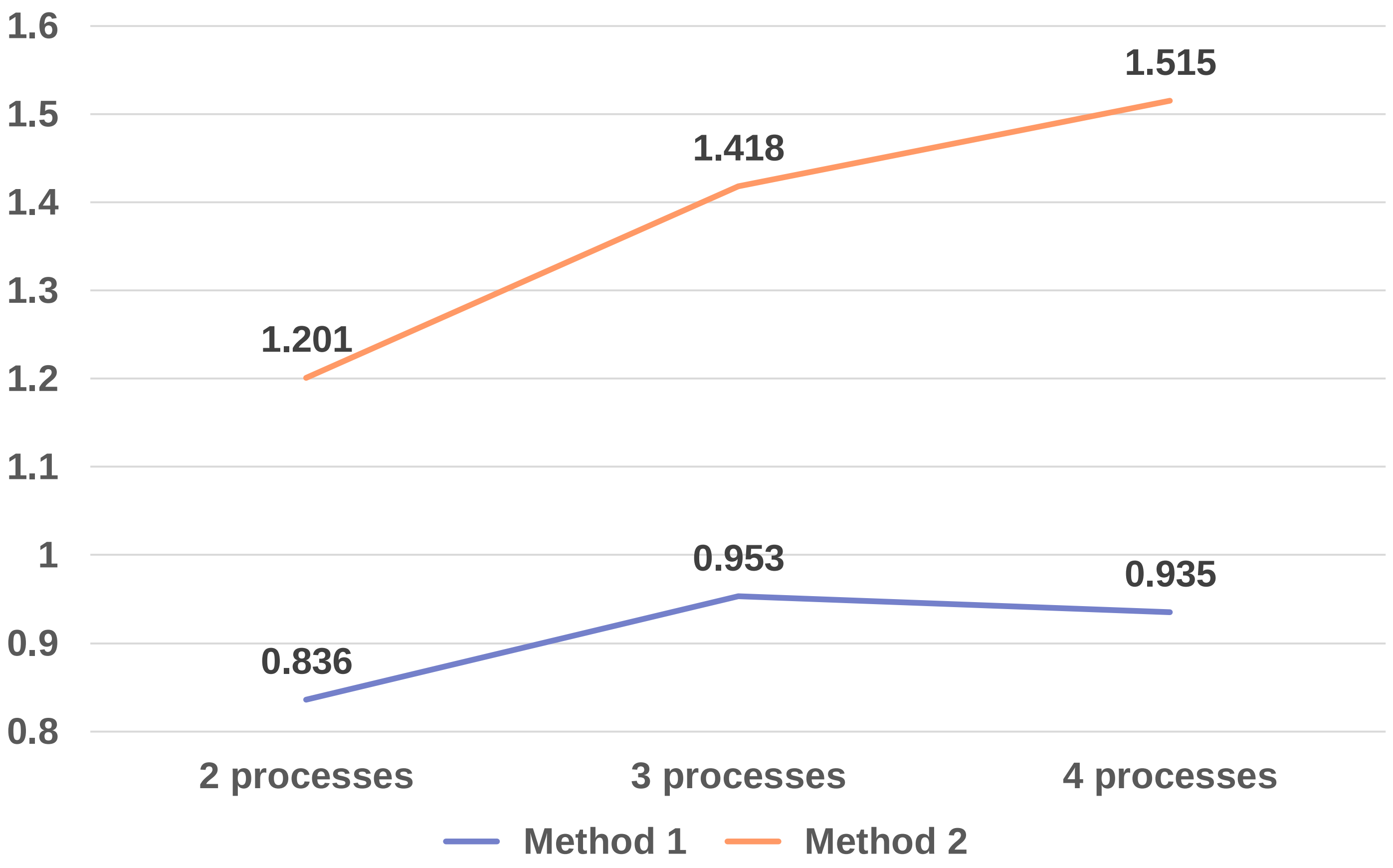
CPU: Intel i7-8700 3.20 GHz
system memory: 16 GB

Language : python with mpi4py

Evaluation

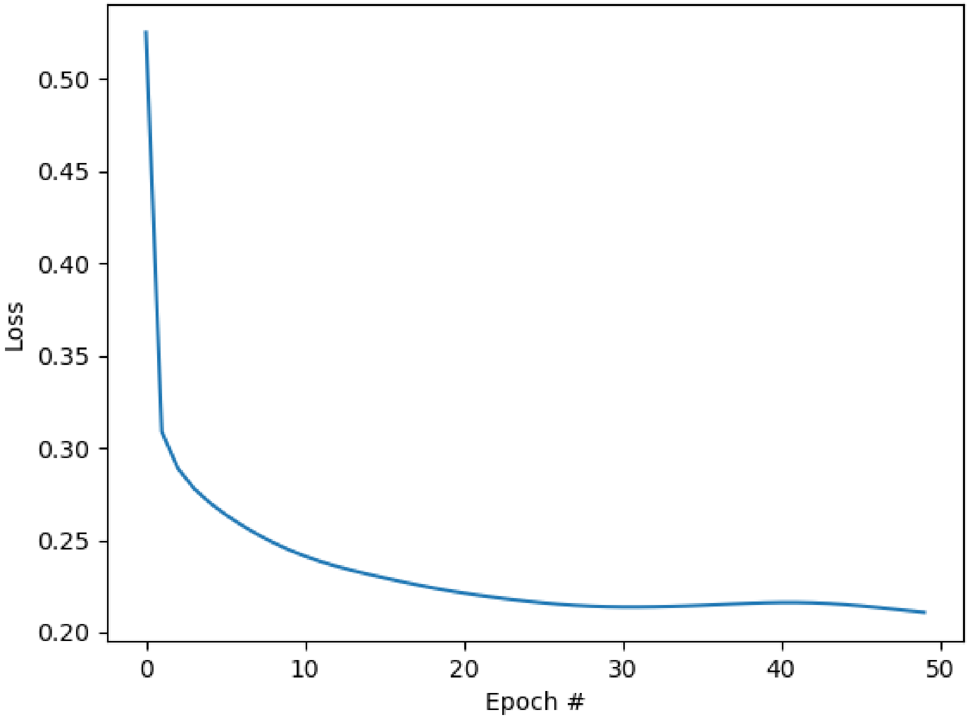
Method	# Process	Loss	Accuracy	Time (s)
Serial SGD	1	0.3726	0.8880	9.0821
Sync	2	0.2709	0.9210	10.8651
	3	0.2744	0.9213	9.5284
	4	0.2977	0.9185	9.7092
Async	2	0.3204	0.9111	7.5638
	3	0.6277	0.8426	6.4041
	4	0.4981	0.8596	5.9964

Evaluation

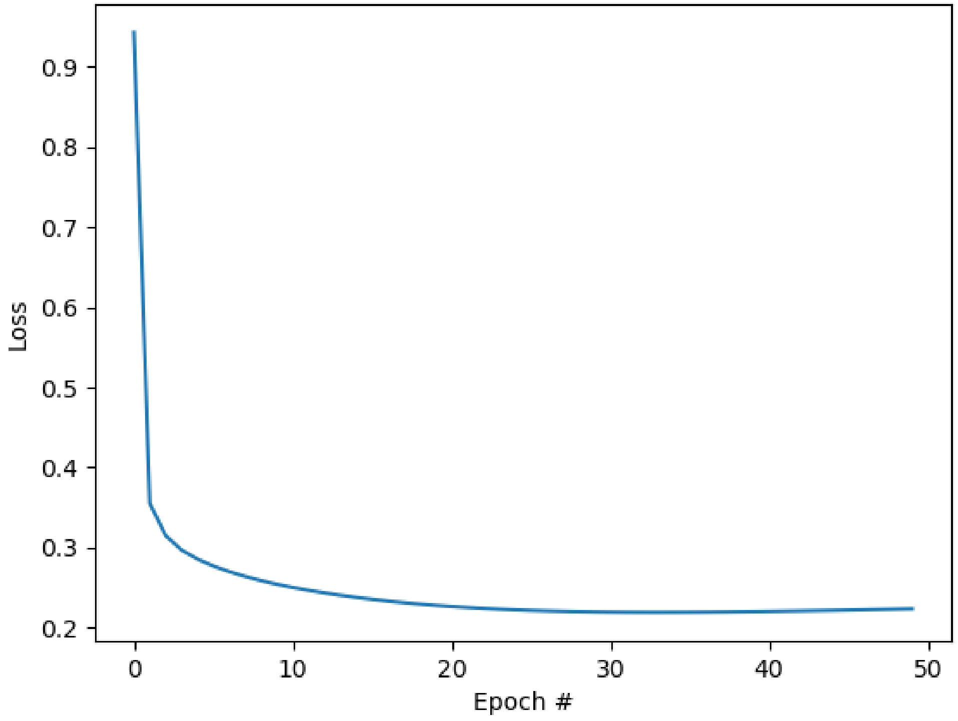

1.6
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
1.515
1.418
1.201
0.953
0.935
0.836
2 processes
3 processes
4 processes
Method 1
Method 2

Evaluation

- Serial SGD

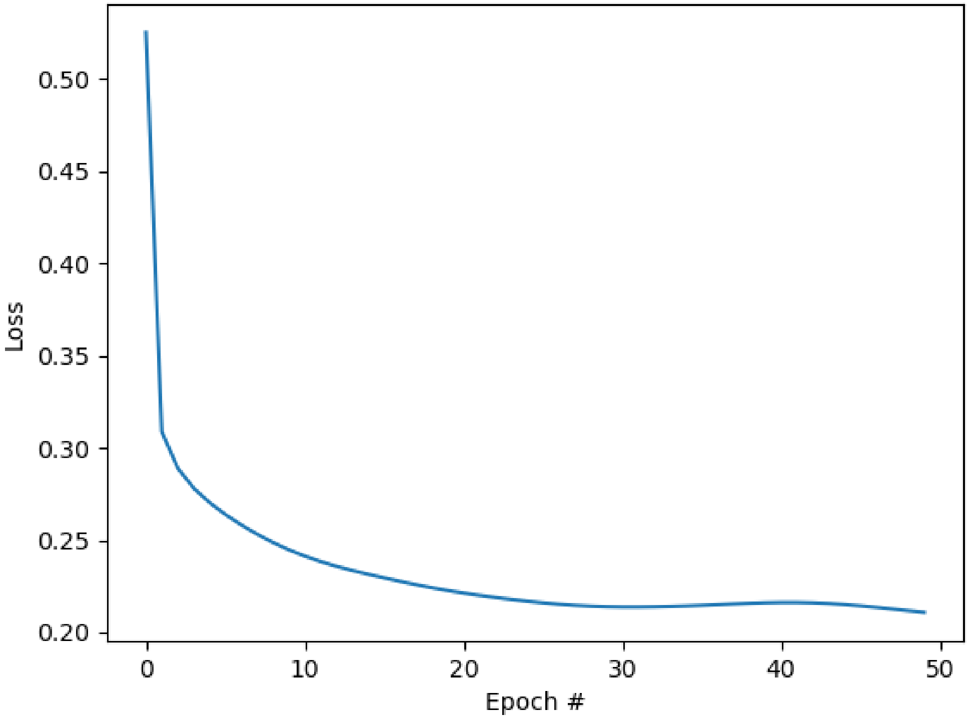


- Synchronous method

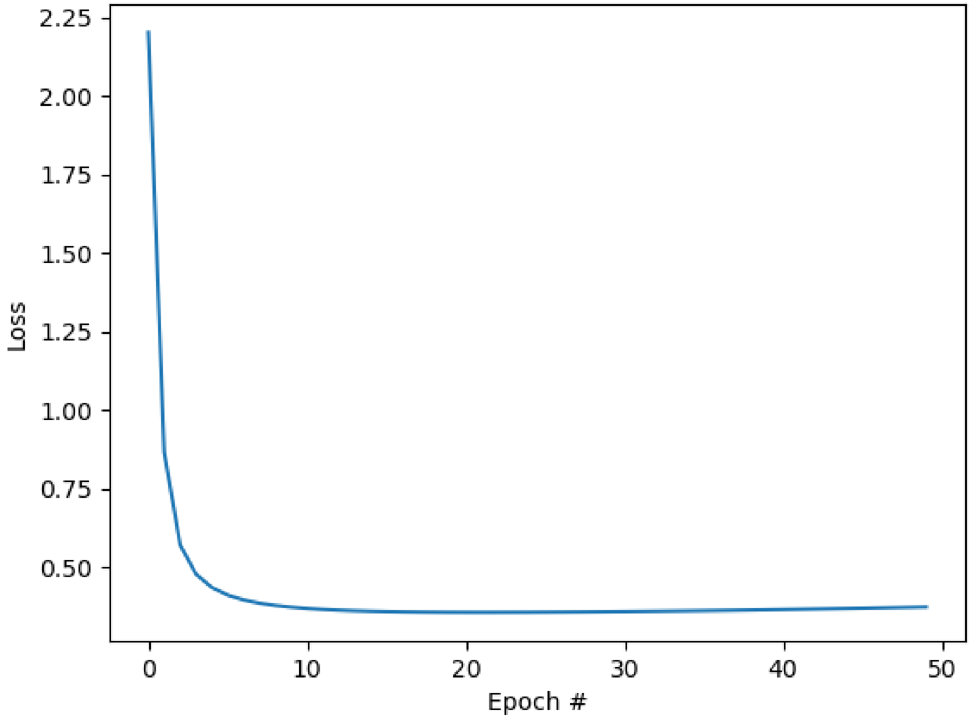


Evaluation

- Serial SGD



- Asynchronous method



Evaluation

Method	# Process	CPU usage	Memory usage
Serial SGD	1	35~40%	12%
Sync	2	40~45%	28.7%
	3	55~57%	36.5%
	4	65~70%	43.3%
Async	2	52~55%	28%
	3	70~75%	35%
	4	80~85%	42.9%

Conclusion

- Provide two methods to parallelize SGD
- The performance of first method is worse than serial program, we think this is because of that communication overhead is too high.
- The second method may cause the decrease of accuracy, this is because of the share weights are not update in some process in each iteration.

Q & A

Team 8