IoT·인공지능·빅데이터 개론 및 실습

그래프 처리, 머신러닝/딥러닝

서울대학교 컴퓨터공학부
전병곤

Contents

1 그래프 처리

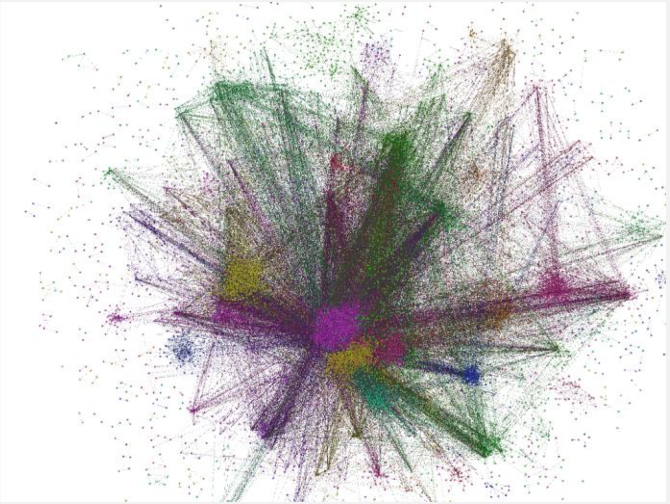
(1) 빅데이터 분석 방법

- 배치 분석
- 대화형 질의
- 스트림 처리
- 그래프 처리
- 머신 러닝/딥러닝

❶ 그래프 처리

(2) Graph Data

Web graph



[출처]:http://labs.criteo.com/2014/05/web-graph-seen-criteo/

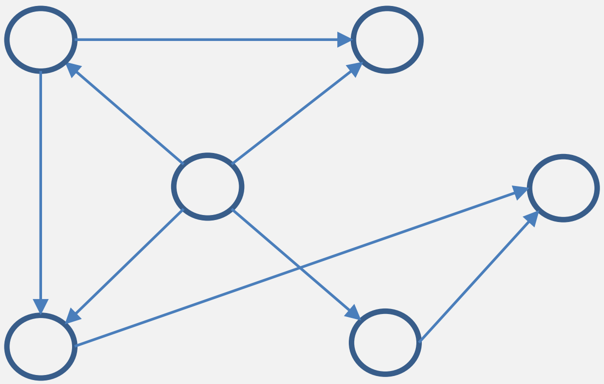
Social graph

[출처]:http://blog.shuttlecloud.com/more-than-the-social-graph/

(2) Graph Definition and Algorithms

Graph G definition

- V
  - A set of vertices
  - L(v) – value of vertex v
- E
  - A set of edges (directed or undirected edges)
  - W(u, v) – value of edge (u, v)



Variations of the above definition

(2) Graph Definition and Algorithms

Graph algorithms

- Shorted paths
- Bipartite matching
- Connected components
- Random walk
- PageRank
- ...

(2) Graph Definition and Algorithms

Graph algorithms

- Triangle counting
- Community detection
- Motif finding
- Social circles
- …

(3) Graph Processing Model

Why not MapReduce or MPI for graph processing?

- MapReduce: not efficient
- MPI: implementing specific graph processing algorithms
  -> not reusable for other graph processing algorithms

(3) Graph Processing Model

Think like a vertex

- e.g., Google Pregel, Apache Giraph
- Vertex as a computation unit

Think like a (sub)graph

- A subgraph as a computation unit

(3) Graph Processing Model

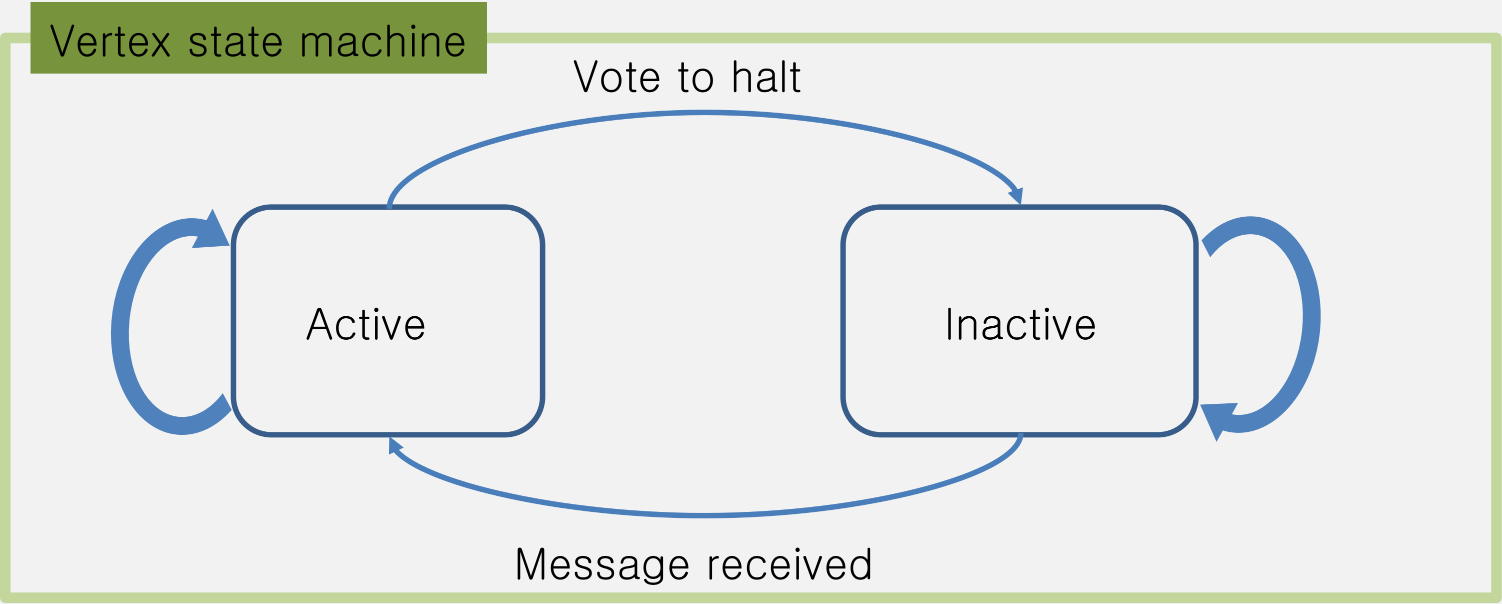
Think like a vertex

- Computations consist of a sequence of iterations called supersteps (i.e., bulk synchronous parallel)
- During a superstep, the framework invokes a user-defined function for each vertex in parallel
- A vertex can
  - read messages sent to V in superstep S-1
  - modify the state of V and its outgoing edges
  - send messages to other vertices that will be received at superstep S+1

(3) Graph Processing Model

› Think like a vertex (e.g., Google Pregel, Apache Giraph)

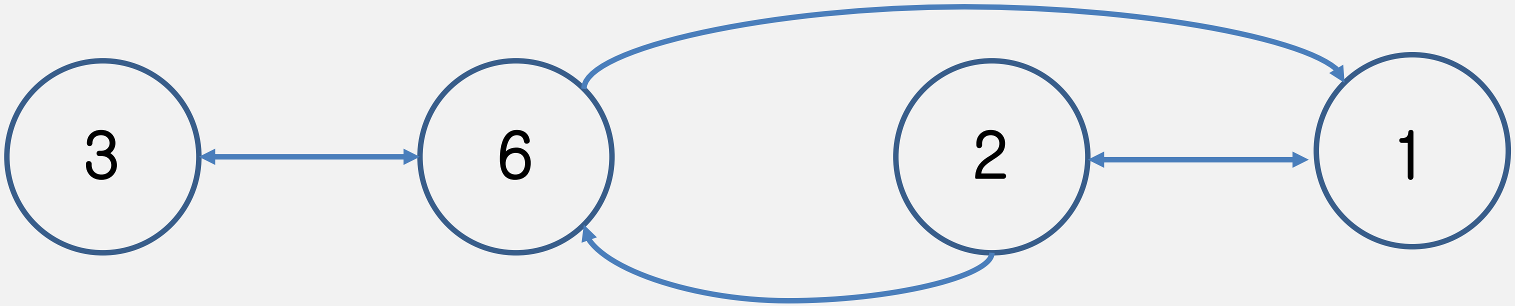
- Algorithm termination is based on every vertex voting to halt



(3) Graph Processing Model

Example of the think-like-a-vertex model

- Compute the maximum vertex value of a graph

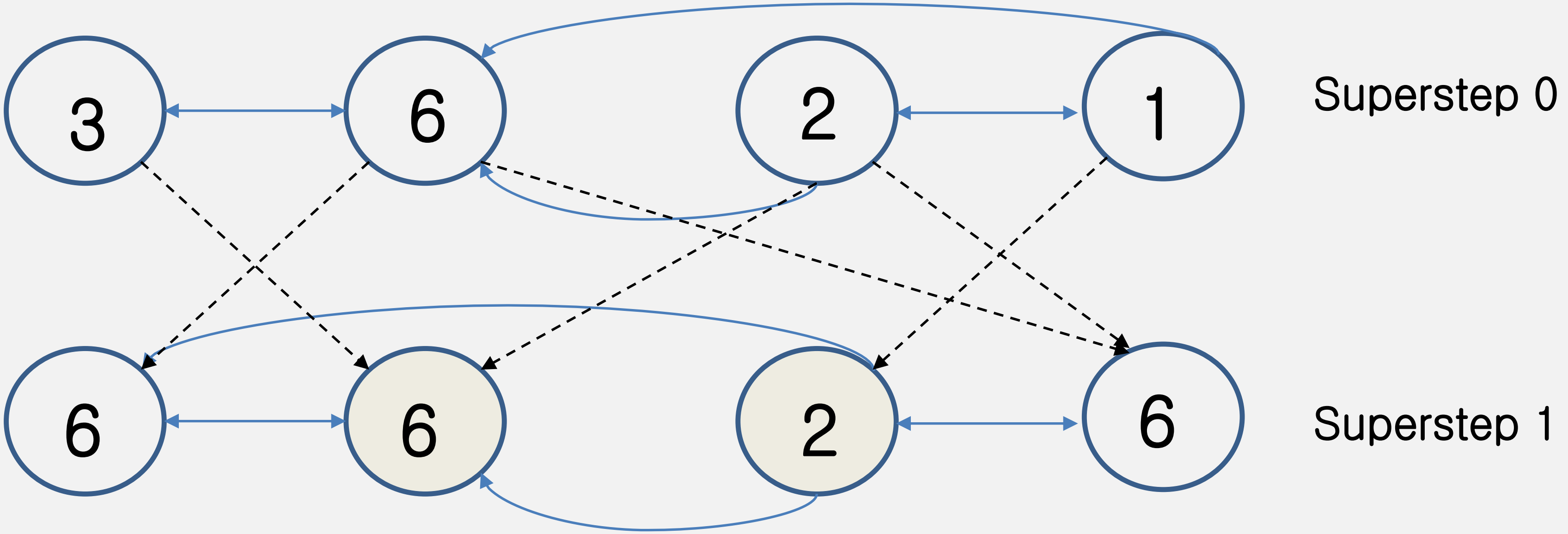


Superstep 0

(3) Graph Processing Model

Example of the think-like-a-vertex model

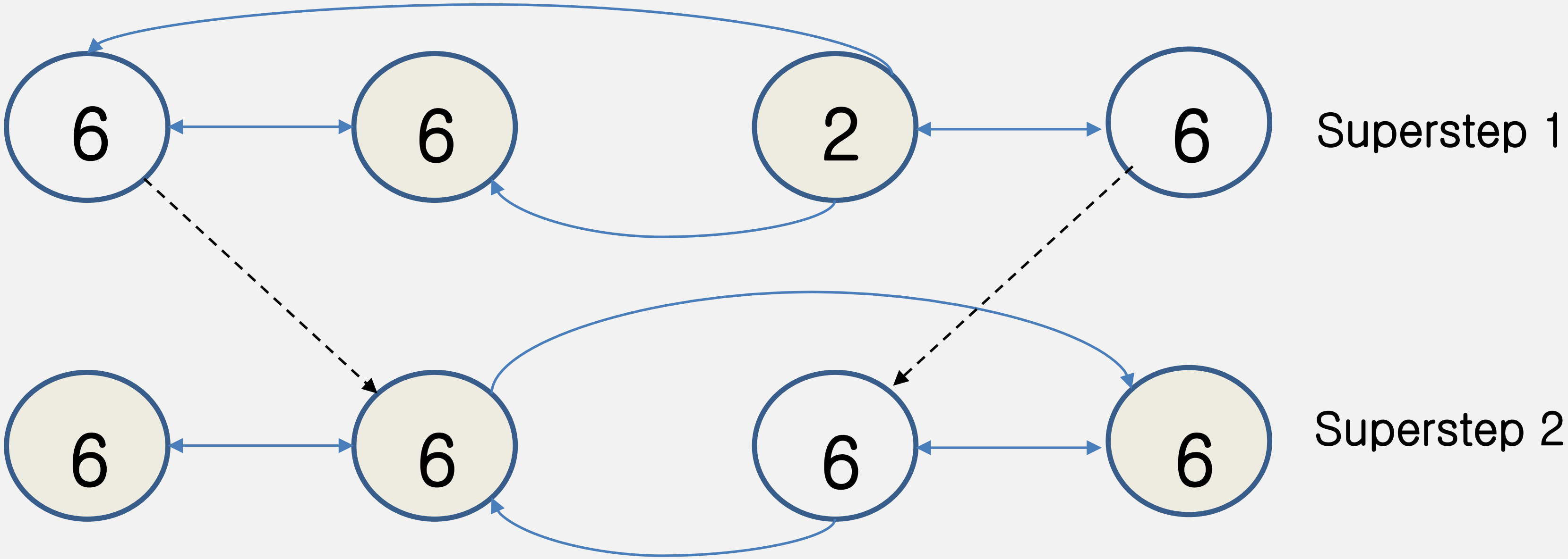
- Compute the maximum vertex value of a graph



(3) Graph Processing Model

Example of the think-like-a-vertex model

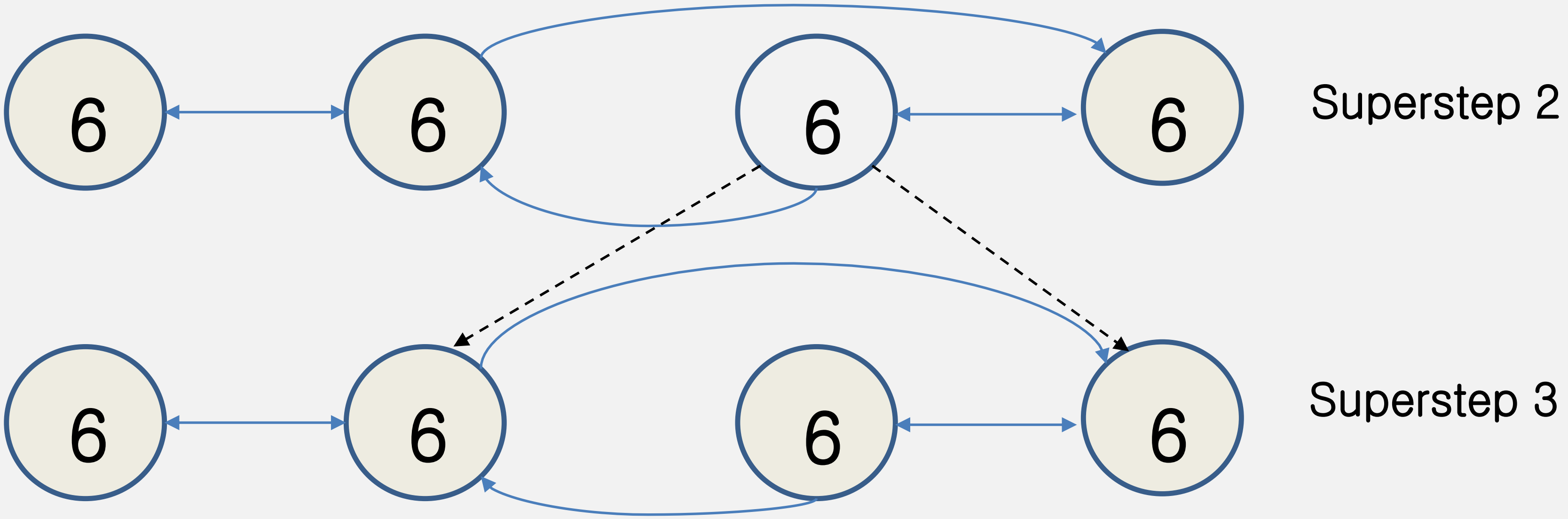
- Compute the maximum vertex value of a graph



(3) Graph Processing Model

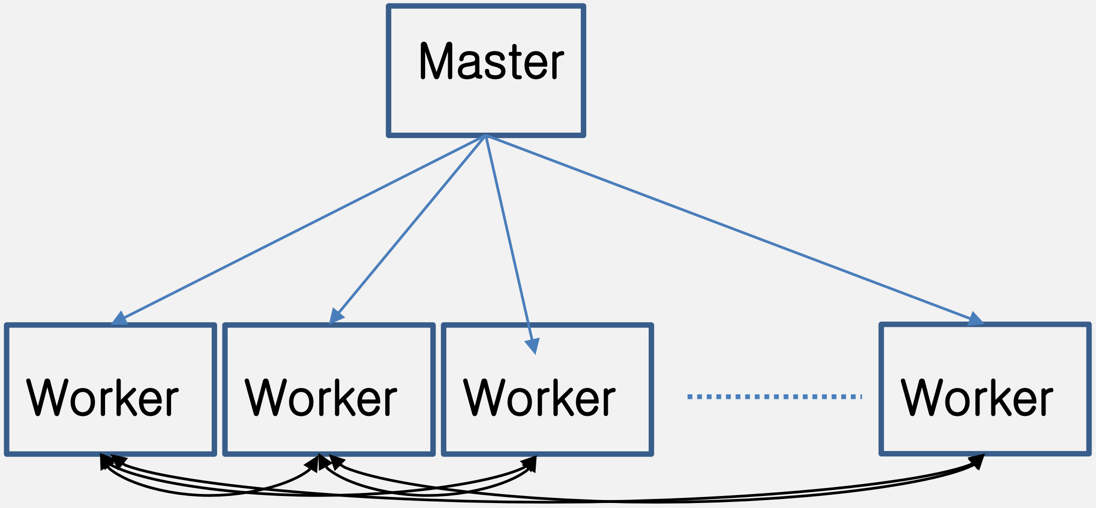
Example of the think-like-a-vertex model

- Compute the maximum vertex value of a graph



(4) Distributed Graph Processing

- Master-worker architecture
- Master divides a graph into partitions(subgraphs) to maximize parallelism



(4) Distributed Graph Processing

- Master initiates a superstep
- Each Worker runs vertex computation for the vertexes in the partition
  - Sends and receives messages across machine boundaries
- Workers notify Master their state changes
- Master checks if the current state meets its termination condition
  - If so, terminates the processing
  - If not, initiates the next superstep

Contents

(1) Classical Machine Learning vs Deep Learning

Classical machine learning

- Human-crafted features
- Great fit for data mining applications

Deep learning

- No feature extraction
- Great fit for hard vision, speech, language problems

(2) Applications

Classical machine learning(ML) applications

- Regression
- Classification
- Clustering
- Topic modeling
- Collaborative filtering
- Frequent pattern mining
- Ranking
- …

(2) Applications

Deep learning(DL) applications

- Image classification
- Speech-to-text, text-to-speech
- Video understanding
- Image, video style transfer
- Text understanding - dialog
- Machine translation
- Ranking ads, feeds, news
- Robot control, game play (e.g., GO)
- Self-driving car
- ...

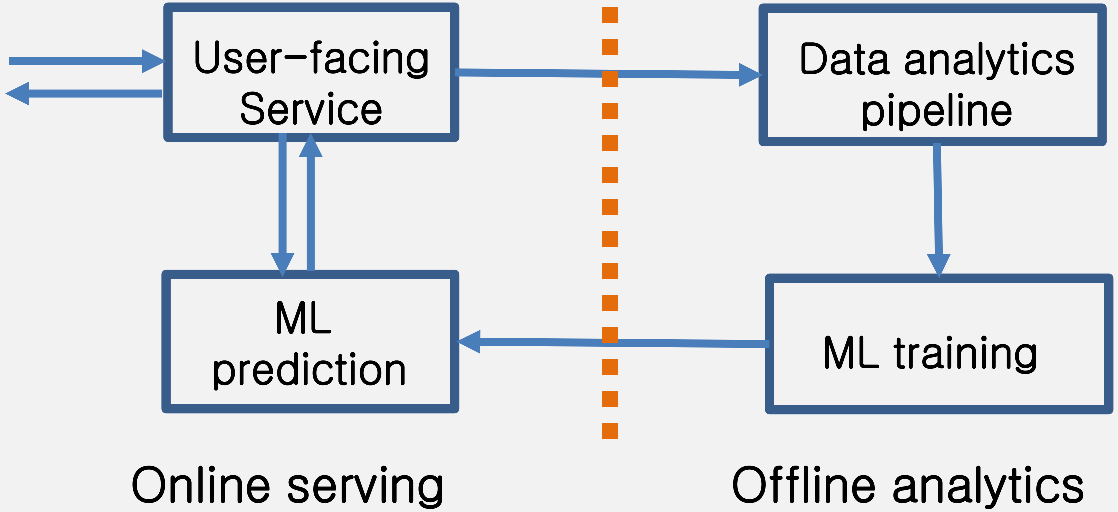
(3) ML(DL) Workflow

Two main steps of ML

- Training
  - Constructs a machine learning model by optimizing loss given training data
- Inference
  - Uses a trained model to predict for new input data

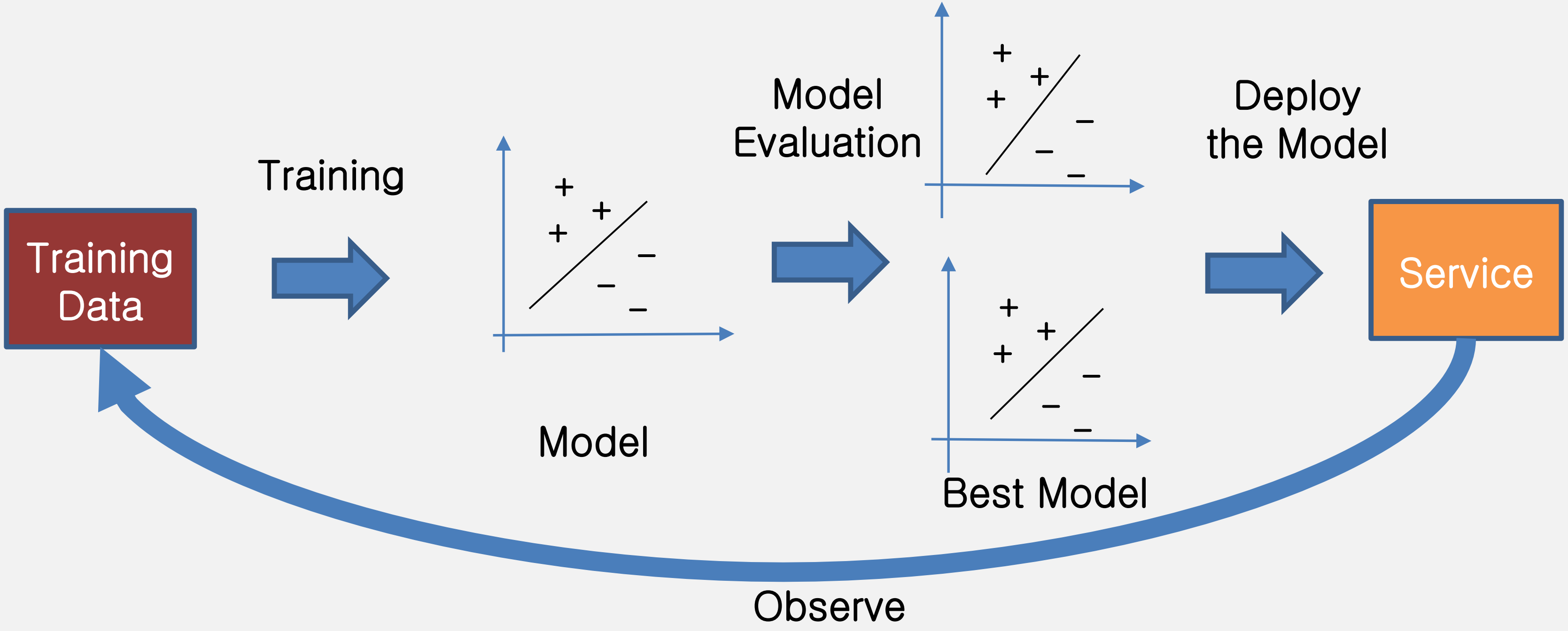
(3) ML(DL) Workflow

› Machine learning workflow



(3) ML(DL) Workflow

› Machine learning training workflow



(4) ML(DL) Framework

- Machine learning (deep learning) framework

Keras

scikit learn

Caffe

Caffe2

Chainer

CNTK

theano

mxnet

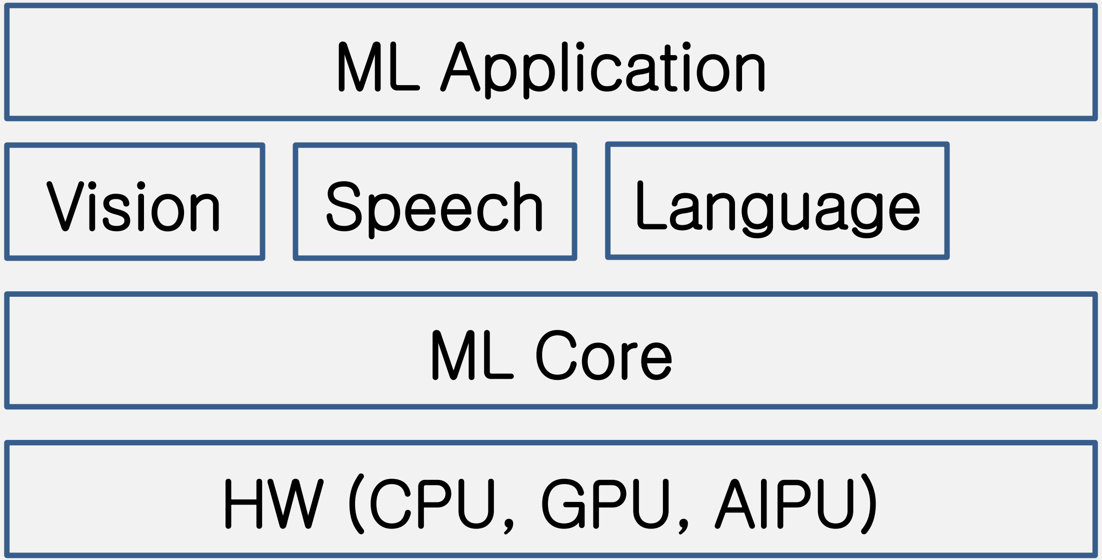
PYTORCH

torch

② 머신러닝/딥러닝

(4) ML(DL) Framework

- Machine learning software stack



(4) ML(DL) Framework

GPU (Graphics Processing Unit)

- Originally for graphics processing
- Around 2004, GPUs were programmable "enough" to do some non-graphics computations
  - Severely limited by graphics programming model (shader programming)
- In 2006, GPUs became "fully" programmable
  - NVIDIA releases "CUDA" language to write non-graphics programs that will run on GPUs
- Great fit for deep learning

(4) ML(DL) Framework

ML core high-level structure

- Python frontend: define machine learning models (mostly neural nets)

- C++ backend: execute defined models
  - CPU
  - GPU(s)
  - AIPU (e.g., TPU)
  - Distributed multi-machine

(4) ML(DL) Framework

ML core main ideas

- Describe mathematical computation
- Represent computation as a computation graph
- Execute the graph with the given data batch
- Training: iterate execution with massive dataset to optimize a goal
  - The framework supports auto-differentiation
- Inference: execute once with a data input to predict

(4) ML(DL) Framework

Framework types: symbolic vs imperative

- Symbolic: define-and-run
  - e.g., TensorFlow, Caffe2, MXNet
  - Better to optimize
  - Easier to deploy
  - Harder to program and debug

- Imperative: define-by-run
  - e.g., PyTorch, TensorFlow eager, MXNet imperative
  - Easier to program and debug
  - Slower than symbolic
  - Harder to deploy

(4) ML(DL) Framework

Google TensorFlow

- Symbolic ML framework
- Express numerical computation as a computation graph
  - Node: operation which has any number of inputs and outputs
  - Edge: tensor which flow between nodes
- Tensor: N-dimensional array
  - 1-dimension: Vector
  - 2-dimension: Matrix
  - E.g., image represented as 3-d tensor rows, cols, color
- Tensors flow through the graph => TensorFlow

(4) ML(DL) Framework

Google TensorFlow

- The graph's compiled to CPU / GPU / AIPU code

- Salient features
  - Fine-grained ops
  - Dynamic control flow: condition, loop
  - Persistent state maintenance/update

(4) ML(DL) Framework

TensorFlow programming model

- Operation: abstract computation
  (e.g., matrix multiply)

- Kernel: a particular implementation of an operation
  that can be run on a particular type of device
  (e.g., CPU or GPU)

- Variable: returns a handle to a persistent mutable
  tensor that survives executions of a graph.

- Session: runs a graph

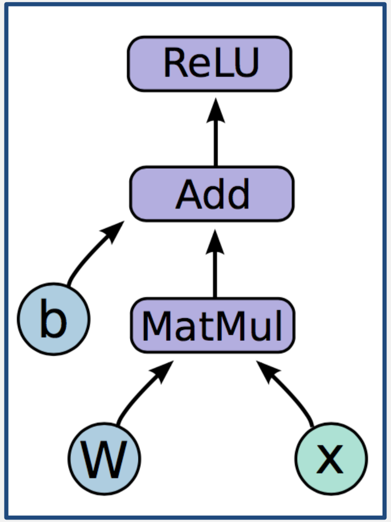
(4) ML(DL) Framework

› TensorFlow code example

```
import tensorflow as tf

  b = tf.Variable(tf.zeros((100,)))
  W = tf.Variable(tf.random_uniform((784, 100), −1, 1))

  x = tf.placeholder(tf.float32, (None, 784))
  h_i = tf.nn.relu(tf.matmul(x, W) + b)
```



For training, TensorFlow automatically creates its backpropagation graph

(4) ML(DL) Framework

FaceBook PyTorch

- Imperative ML framework
- Express numerical computation like numpy
- Programming model
  - Operation : abstract computation (e.g., matrix multiply)
  - Tensor : imperative N-dimensional array but runs either on CPU or GPU
  - Variable : wrapper of Tensor. It constructs a chain of operations between the tensors, so that the gradients can flow back

(4) ML(DL) Framework

› Interoperability between frameworks : Open Neural Network Exchange (ONNX)

- An open source format for AI models
- An extensible computation graph model and definitions of built-in operators and standard data types
- Caffe2, PyTorch, Cognitive Toolkit, MXNet support ONNX

Contents

(1) Distributed training issues

- Parallelism: data parallelism, model parallelism, hybrid parallelism
- Synchronization: synchronous, asynchronous, bounded synchronous
- Synchronization architecture: parameter server architecture, MPI

(1) Distributed training issues

Parallelism

- Data Parallelism
  - D = D1 U D2 U ⋯ U Dn
  - Each worker i processes M with Di.
- Model Parallelism
  - M = M1 U M2 U ⋯ U Mn
  - Each worker i processes Mi with D.
- Hybrid Parallelism

(1) Distributed training issues

Synchrony of training

- Synchronous training

- Asynchronous training

- Bounded-synchronous training

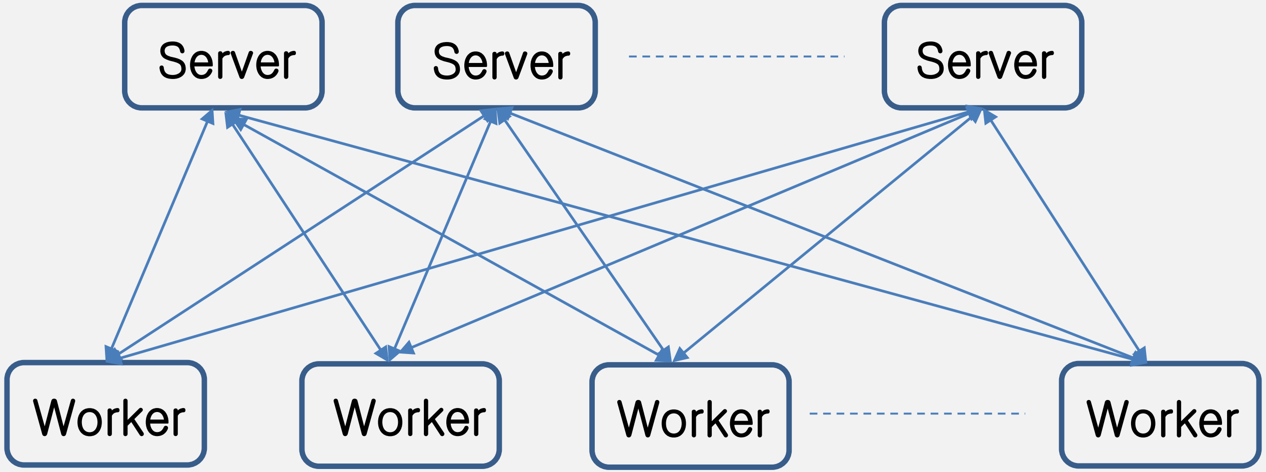
(1) Distributed training issues

Synchronization architecture

- Parameter server

- Message passing interface (MPI)

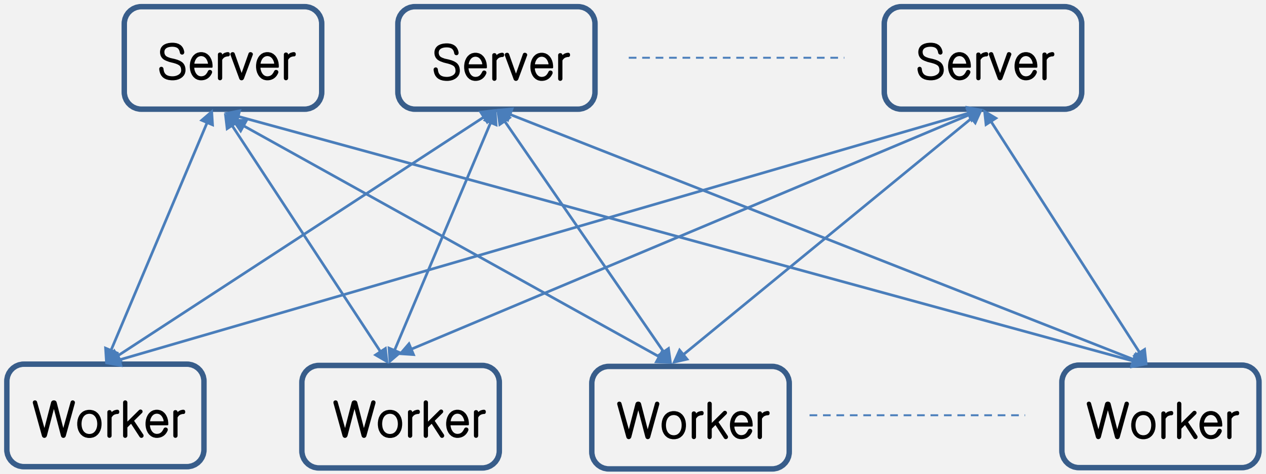
(2) Parameter Server



(2) Parameter Server

Server

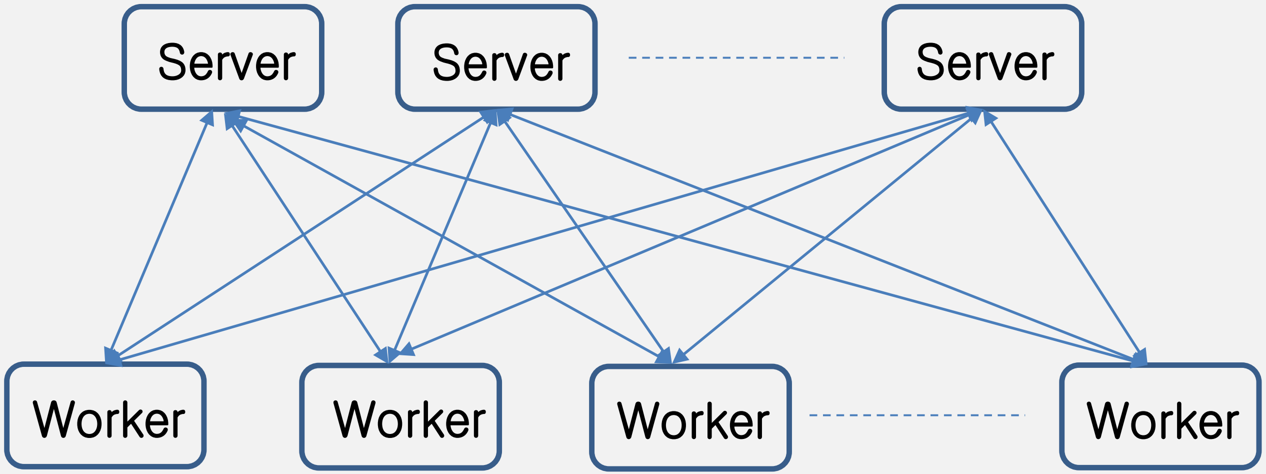
- Maintains a partition of the globally shared parameters
- Performs global aggregation steps



(2) Parameter Server

Worker

- Performs computation with (a portion of) training data communicates with servers
- Updating and retrieving the shared parameters



(2) Parameter Server

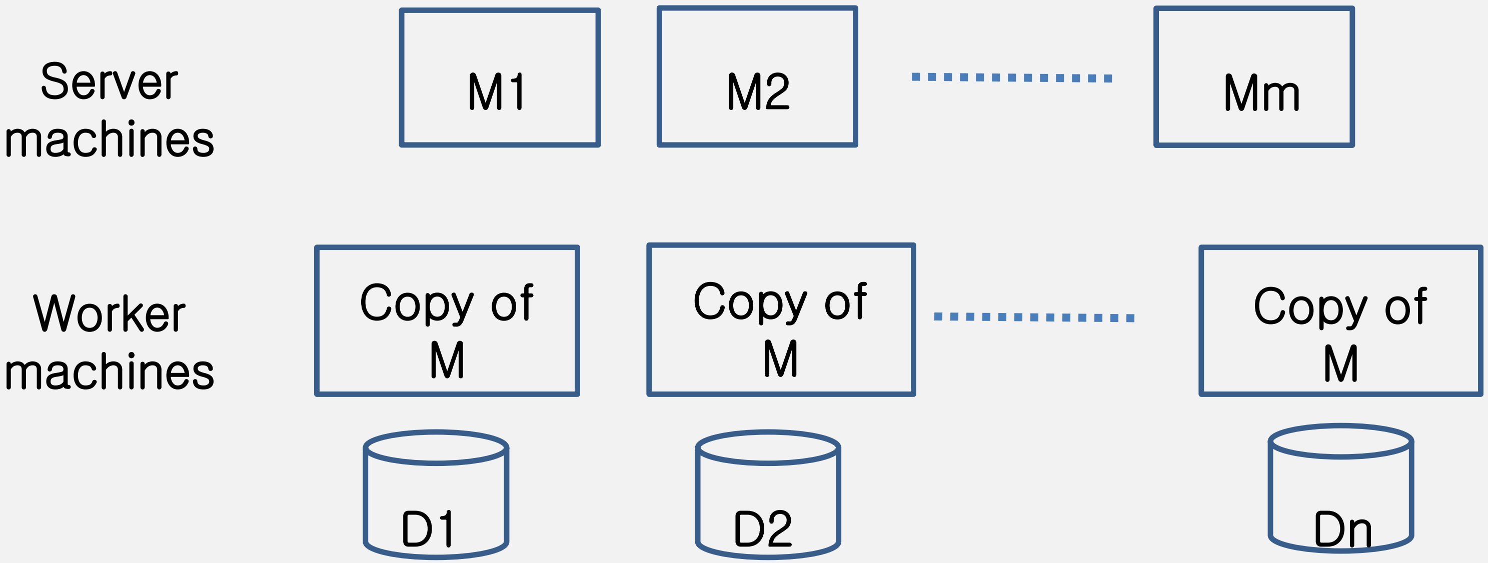
Parameter-server-based synchronous training

- Step 1 : Workers compute gradients with training data and push them to servers
- Step 2 : Each Server receives gradients from Workers, aggregates them, and applies the sums to update the model parameters
- Step 3 : Workers pull the new model parameters
- The above steps iterate until training converges

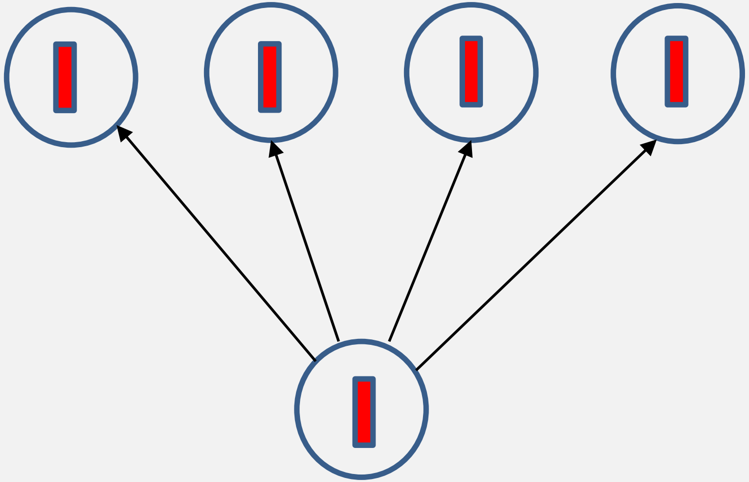
(2) Parameter Server

Parallelism

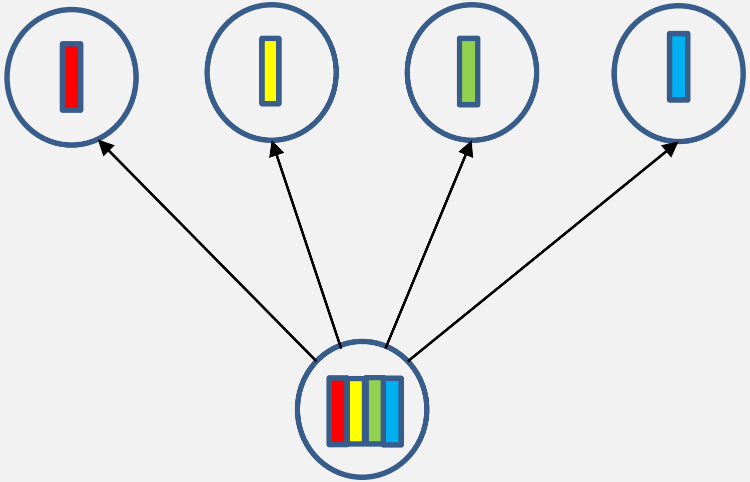
- Data Parallelism



(3) MPI Collective Communication

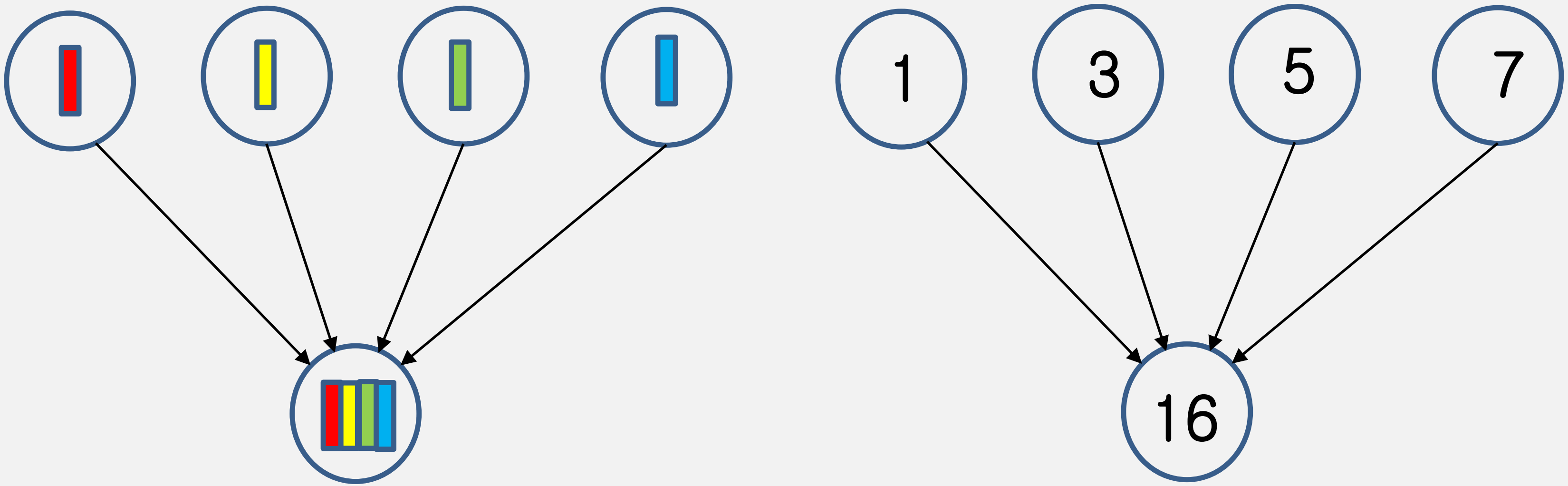


Broadcast



Scatter

(3) MPI Collective Communication



Gather

AllGather – Gather + Broadcast

Reduce

AllReduce – Reduce + Broadcast

(3) MPI Collective Communication

MPI-based synchronous training

- Step 1 : Workers compute gradients with training data
- Step 2 : Workers run AllReduce (or AllGather) to aggregate them and apply the sums to update the model parameters
- The above steps iterate until training converges

요약

- 그래프 처리
- 머신러닝/딥러닝
- 머신러닝/딥러닝 분산학습