innovate achieve lead

BITS Pilani
WILP

AIML CLZG516
ML System Optimization

Murali Parameswaran

innovate achieve lead

BITS Pilani

WILP

AIML CLZG516
ML System Optimization

Session 8

Distributed ML - Models and Platforms

- Implementation Issues
- The Parameter Server Model
- Stochastic Gradient Descent

Decision Trees

- Approach:
    - Construct a tree where each node denotes a binary decision
        - Nodes in the tree correspond to features and the order of features is chosen based on the notion of **information gain** (IG)
        - Information gain is the entropy
            - entropy of the whole set
            - minus the entropy when a particular feature is chosen

Decision Tree Construction

- Algorithm ID3 (input dataset S)
  - If all examples have the same label
    - Return a leaf with that label
  - Else if there are no features left to test
    - Return a leaf with the most common label
  - Else choose the feature F that maximizes IG of dataset S as the next node
    - Add a branch from the node for each possible value f in F
    - For each branch:
      - Calculate S_f by removing F from the set of features
      - ID3(S_f)

Parallel/Distributed Tree Construction?

- When you branch assign each branch (corresponding to one value of a feature)
  - To a different task (task parallelism)
  - At each level : number of parallel tasks = number of possible values of a feature

ML problem and Error

- Given input dataset – a vector of size n,
  - Each training example $\mathbf{x}_i$ of **d** features (or dimensions) is associated
    - with a label $\mathbf{y}_i$ and
    - model parameters (likely corresponding to the features)
  - The problem is to predict **y** corresponding to an unseen example **x**
- Training error
  - Difference between the predicted y the actual label y' for an x

Model complexity

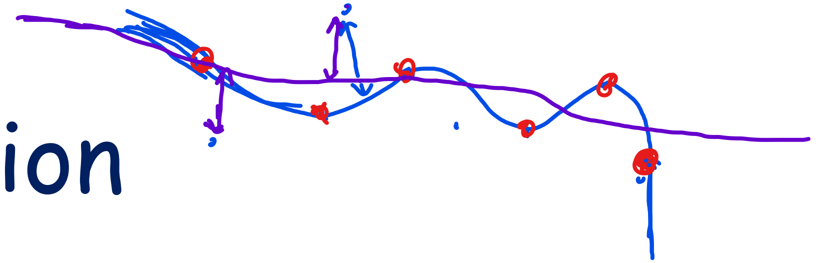
- Relation between model size (number of parameters) and data size (for training):
  - If there is too little data,
    - then a highly detailed model may overfit
  - If the model is too small,
    - then it may fail to capture relevant attributes
- Regularization addresses this issue

Nvidia

✓ (Megatron LM)

Data, Model & Pipeline

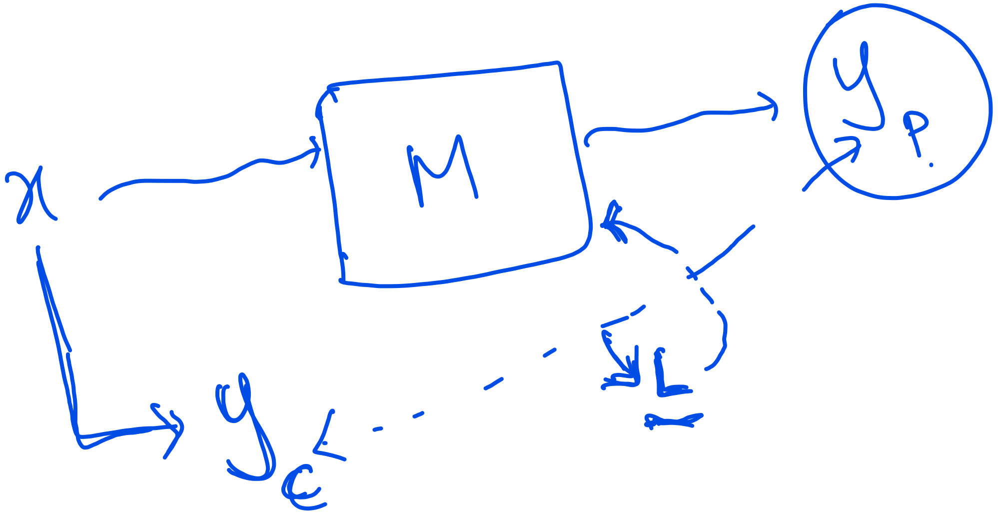
1. Data parallelism
2. Task "
3. Request " I/O →
4. Pipeline "



ML as regularized error minimization

- Training an ML model is minimizing the function F :
  - $F(w) = \Sigma_i L(x_i, y_i, w) + \Omega(w)$
    - where w denotes the set of parameters and
      - L is the loss function (i.e. prediction error) and
      - Ω is the regularizer that penalizes the model for complexity

Generalization



Distributed ML

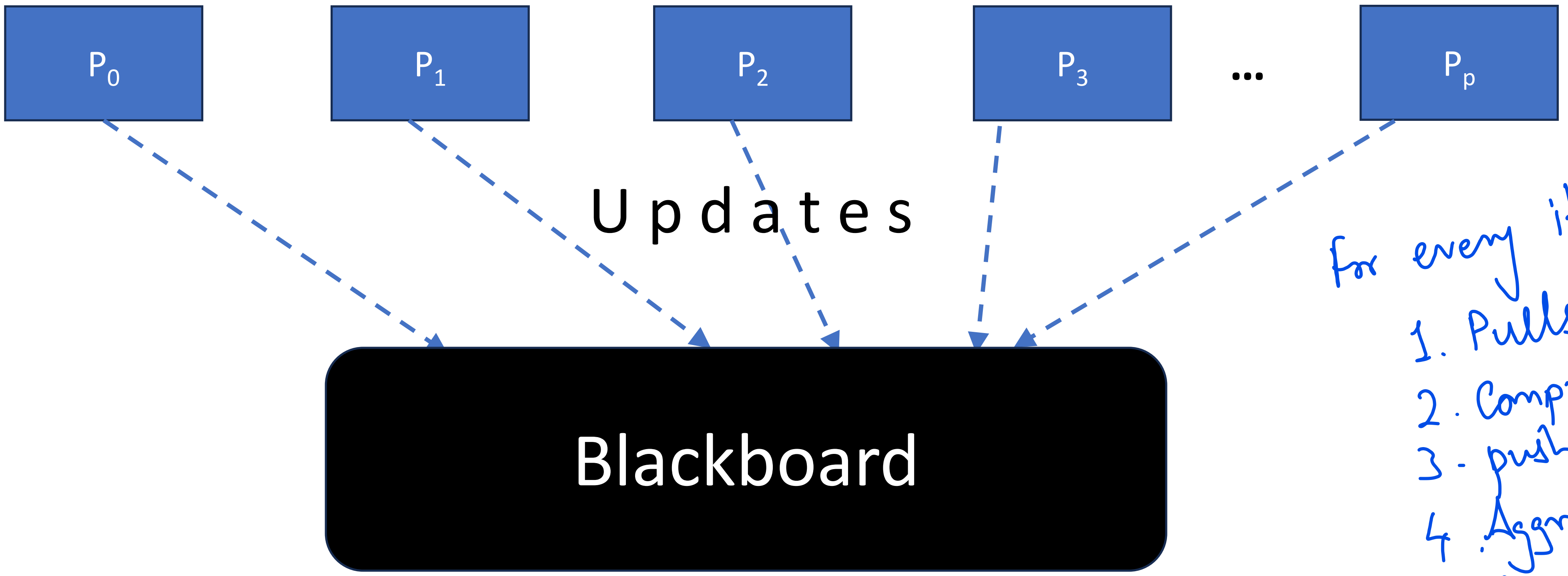
- Complexity:
  - Training Data size: from 1TB to 1PB
  - Model Size: 10^9 to 10^{12} parameters
- Examples:
  - Online Recommender System
    - millions of user profiles
  - Ad click predictor
    - each training example is a feature vector of high dimensions

Distributed ML – System Requirements

- In a distributed system,
  - the training data is partitioned among multiple nodes
    - and the nodes together learn the parameter vector w.
- The algorithm operates iteratively:
  - In each iteration,
    - every node independently uses its own training data to
      - Compute the changes to be made to **w** in order to move closer to an optimal value
    - Each node computes changes to **w** based only on its local data,
      - <u>a central place is needed to aggregate these changes</u>

Distributed Systems – Blackboard architecture

- Blackboard architecture is a pattern for distributed computation
  - where multiple nodes have to combine results
    - <u>*computed locally, in parallel –*</u> *see processes P_i below*



For every iteration,
1. Pulls (worker)
2. Compute (Worker)
3. push (worker)
4. Aggregate (server)
5. Update (server)

$\boxed{W^{(x)}_{t+1} = W^{(x)}_t - \eta \Delta W^{(x)}_t}$ Mini-batch

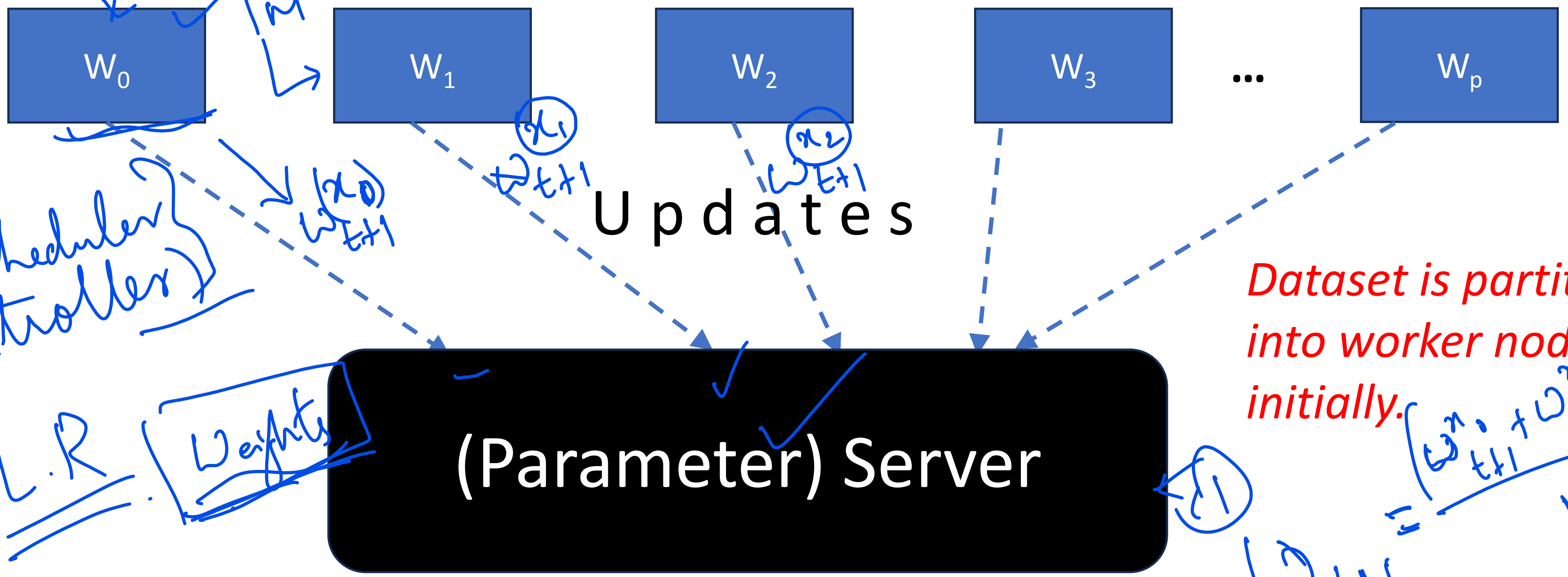
1M (Mini batch) billing (X)

x_i

Regularized Error Minimization : A Distributed Architecture

In each iteration:

- Each worker node W_i pulls (current) parameters w from the server, computes F(w), and posts it back.
- Server updates and minimizes F(w)



1M

1M

$W^{(x_0)}_{t+1}$

$W^{(x_1)}_{t+1}$

$W^{(x_2)}_{t+1}$

Task Scheduler (Controller)

L.R

Weights

Dataset is partitioned into worker nodes initially.

$\omega_{t+1} = \frac{(\omega^{n}_{t+1} + \omega^{n_1}_{t+1} \ldots)}{n}$

Distributed Systems and Failures

1000

1. 100th

2. 200th

201 ~~2~~

300

3. 300th

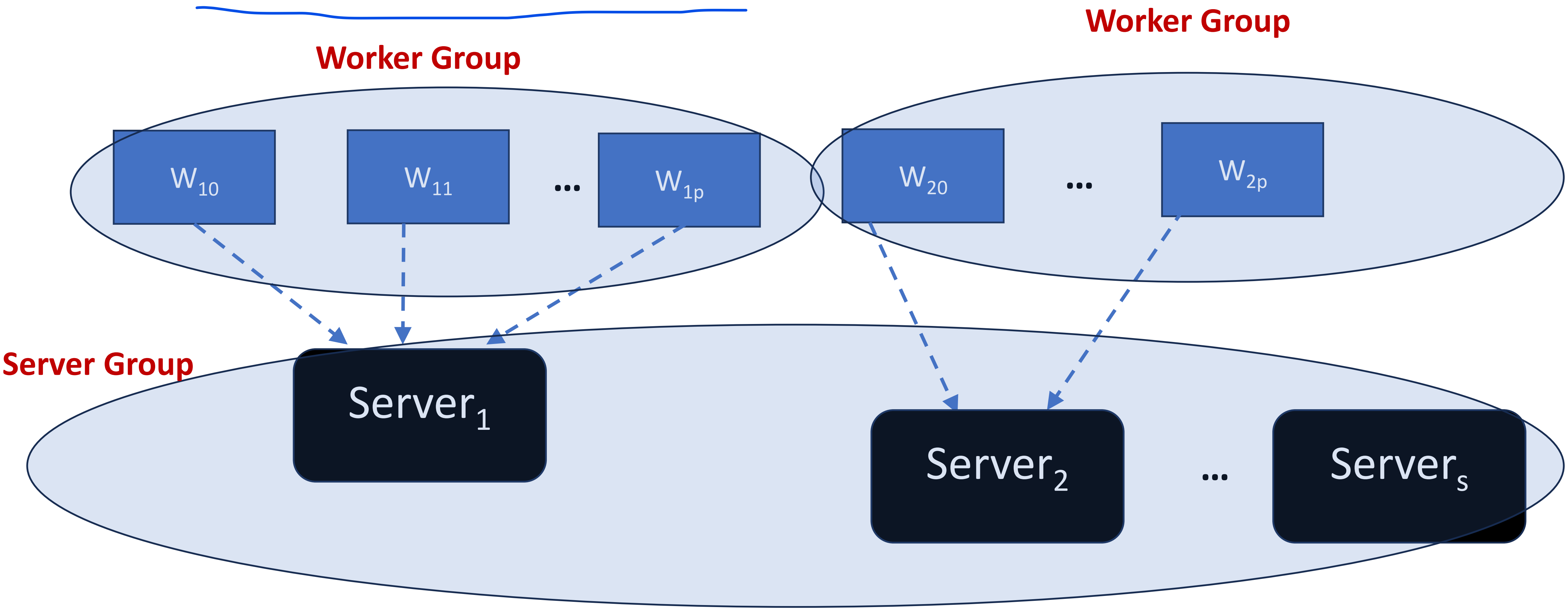
- Individual Nodes may fail frequently in distributed systems:
  - This is particularly so in commodity clusters
- Rate of node failure increases with the size of the system (*i.e., more nodes and more processes*) – (2 - 5%)
  - **Cloud data centers** are made out of commodity clusters!
- Distributed Systems have to function (i.e. be available) despite node failures
  - This is referred to as fault tolerance and is achieved via
    - redundancy and failure recovery

1. checkpoints
2. Replica (Server)
3. Elastic training

Scalable and Dependable (*reliable and available*) architecture for ML

Each Worker Group includes a task scheduler.

Server Group must be fault tolerant!



Scalable and Dependable (*reliable and available*) architecture for ML

- This architecture is referred to as the ***parameter server*** model:
  - Different servers may handle different problems
  - Multiple servers may work on the same problem to improve performance
    - This will require additional combination/minimization processes and servers.
  - Multiple servers may work on the same problem for redundancy.

Parameter Server Model

- This model was popular for a few years
  - Google built an internal platform named DistBelief based on this model
    - DistBelief was optimized primarily for large clusters of multi-core nodes
    - GPU clusters were enabled later
  - Later, Google's TensorFlow provided programming flexibilities not available in DistBelief:
    1. Adding new layers
    2. Adding new ML training workflows
    3. Optimizing or tuning ML algorithms

4. Data Abstraction.

Kera API

TensorFlow

- GPU acceleration has become a common tool for ML algorithms.
  - Building and testing on GPU workstations before scaling it to a GPU cluster is a common scenario as well.
- TensorFlow provides a unified programming interface and a common runtime on all these hardware platforms
  - while also supporting heterogeneous accelerators.
- e.g. Google's TPUs are special purpose accelerators for ML
  - that enable increased performance-per-watt compared to other state-of-the-art hardware.
- TensorFlow supports a common device abstraction for heterogeneous accelerators.

Gradient Descent

- One approach to error minimization is known as Gradient Descent:
  - Use the slope (i.e., gradient) of the loss function to update the parameters.
- This is particularly useful in Neural Networks in the back-propagation phase

Gradient Descent

- The gradient descent approach to minimize the error requires the following update to the parameters
  - $w = w - \eta * g(L, D, w)$
    - where g is the gradient function, L the loss function, and D, the dataset.
    - η denotes the learning rate – controls the amount by which the parameters are updated.
- The updates are done iteratively.

Gradient Descent

```
for i = 1 to num_iter :
    1.  grad = eval_gradient ( loss_function , D , w )
    2.  w = w - learning_rate * grad
```

- This is Batch GD!
  - i.e., update is done after all the points in the dataset are considered.
- Batch Gradient Descent is slow to converge, particularly if same data (or similar) data is repeated within the dataset.:

Stochastic Gradient Descent (SGD)

1. for i = 1 to num_iter :

    1.1 shuffle (data)

    1.2 for example in data :

        1.2.1 grad = eval_gradient (loss_function , example , w)

        1.2.2. w = w - learning_rate * grad

This may converge faster but is completely sequential i.e., not easy to parallelize!

Mini-batch SGD

for i = 1 to num_iter:

1. shuffle (data)
2. for batch in get_batches (data , batch_size): → Parallelize / distribute
    1. grad = eval_gradient (loss_function , batch , w)
    2. w = w - learning_rate * grad

pull
Compute
push

Aggregate
Update
Server

- Batches in the inner loop can be executed independently (locally)!
    - If necessary, batches can be obtained and stored locally at the start.
- Update has to be done in the parameter server