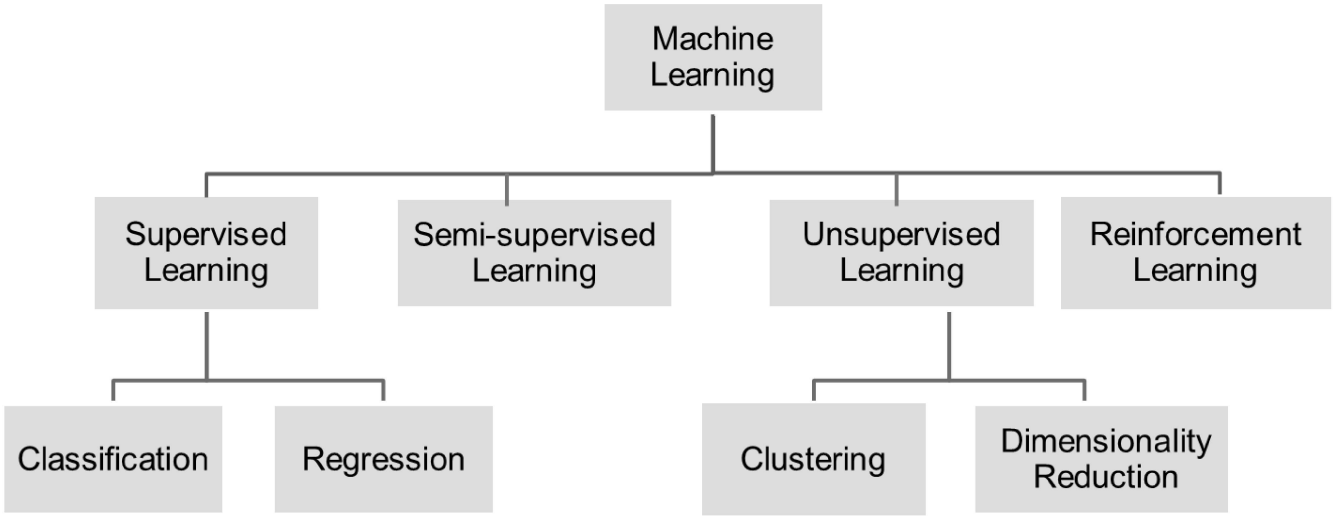
I. Learning from Data

Supervised learning: **involves building a statistical model for predicting, or estimating, an *output* based on one or more *inputs*. (e.g. linear regression alg. and SVM)**

Unsupervised learning: **there are inputs but no supervising output; nevertheless we can learn relationships and structure from such data. (e.g., *K*-means clustering alg.)**

Regression: **predicting a *continuous* or *quantitative* output value.**

Classification: **predict a non-numerical value—that is, a *categorical* or *qualitative* output.**



Parallelism in Machine Learning

Implicit Parallelization: Keep the overall algorithm structure (the sequence of operations) intact and parallelize the individual operations.

Example: parallelizing the BLAS operations in previous figure

+ Often achieves exactly the same accuracy (e.g., model parallelism in DNN training)

- Scalability can be limited if the critical path of the algorithm is long

Explicit Parallelization: Modify the algorithm to extract more parallelism, such as working on individual pieces whose results can later be combined

Examples: CA-SVM and data parallelism in DNNs

+ Significantly better scalability can be achieved

- No longer the same algorithmic properties (e.g. HogWild!).

Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent, F. Niu et al, NIPS 2011

Slides source: Aidyn Buluc

Parallelization Opportunities

Different from the data parallelism with 'owner-compute' rule (distribute output)

1. **Data parallelism**

   Distribute the input (sets of images, text, audio, etc.)

   a) **Batch parallelism**

   - Distribute a group
     processor
   - *When people mer*
     this is what they n

   b) **Domain parallelism**

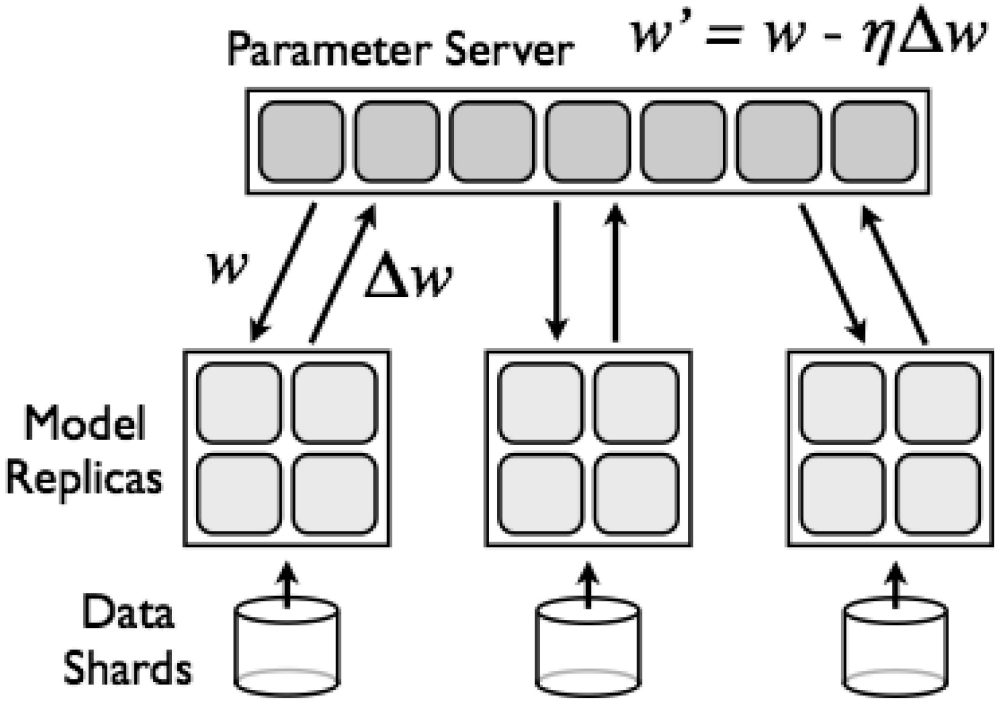
   - Data points/features of individual sample are subdivided and distribute parts to processors.

2. **Model parallelism:**

   Distribute the neural network (i.e. its weights)

Epoch: the entire training set is used once;
Mini-batch: a subset of training samples, weights are updated once the entire mini-batch is used.
SDG: update the weights for every training sample (mini-batch size=1)

Batch Parallelism #1



Parameter server is some sort of master process

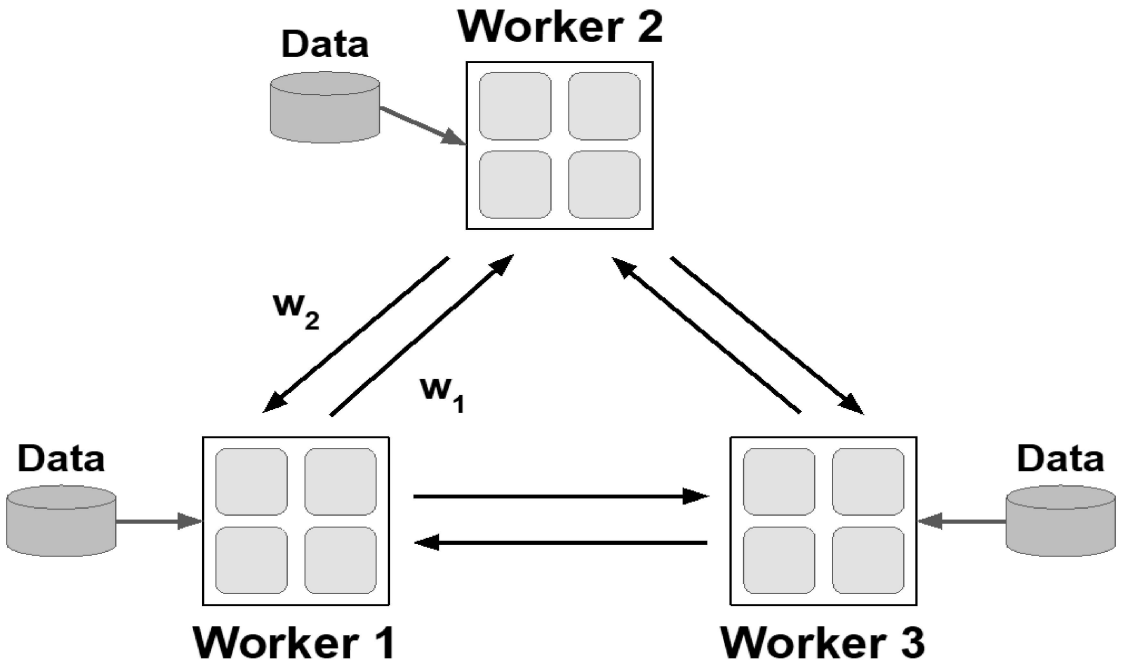
- The fetching and updating of gradients in the parameter server can be done either ***synchronously*** or ***asynchronously***.
- Both has pros and cons. Over-synchronization hurts performance where asynchrony is not-reproducible and might hurt convergence

Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.

Batch Parallelism #2

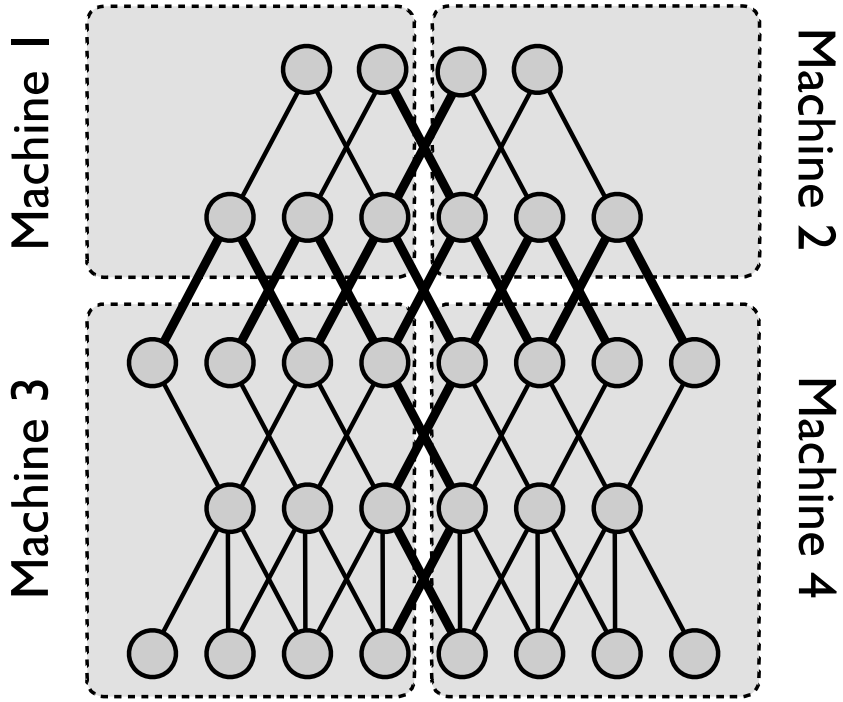
Options to avoid the parameter server bottleneck

1. **For synchronous SGD:** Perform all-reduce over the network to update gradients (good old MPI_Allreduce)
2. **For asynchronous SGD:** Peer-to-peer gossiping



Peter Jin, Forrest Iandola, Kurt Keutzer, “How to scale distributed deep learning?” NIPS ML Sys 2016

Model Parallelism



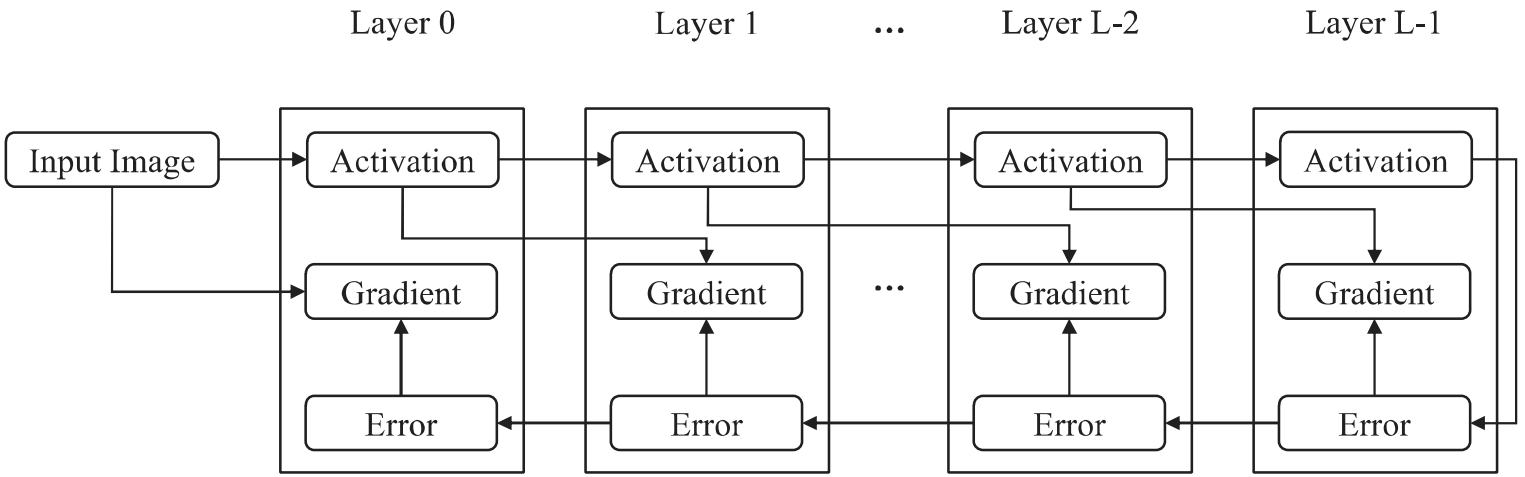
(inter layer parallelism is better called pipelining)

Interpretation #1: Partition your neural network into processors
Interpretation #2: Perform your matrix operations in parallel

Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.

The overlapping opportunities

- In backpropagation, the errors are propagated from the last layer to the first layer and data dependency exists between any two consecutive layers.
- In contrast, the gradients in different layers are independent of each other.



Activations are propagated from left to right in forward stage; errors are propagated from right to left in backpropagation stage. Gradients are computed using the activations and errors. Arrows indicated data dependencies.

S. Lee, et al. "Parallel Deep Convolutional Neural Network Training by Exploiting the Overlapping of Computation and Communication.", HiPC 2017

Algorithm 1 Mini-Batch SGD CNN Training Algorithm
(M: the number of mini-batches, L: the number of layers

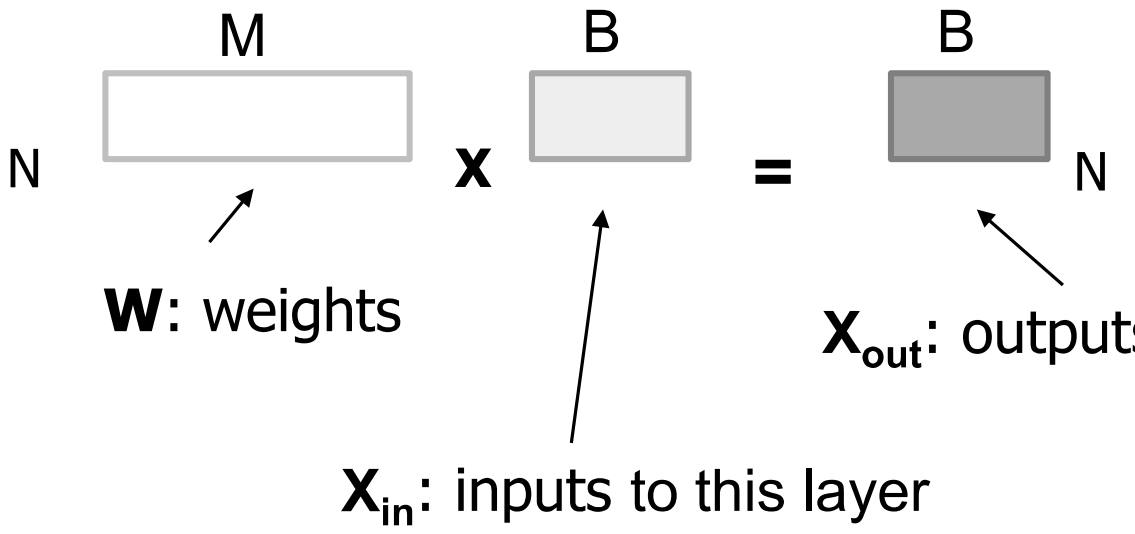
1: **for** each mini batch $m = 0, ...M-1$ **do**
2: Initialize $\Delta W = 0$
3: Get the m^{th} mini batch, D^m.
4: **for** each layer $l = 0, ...L-1$ **do**
5: Calculate activations A^l based on D^m.
6: **for** each layer $l = L-1, ...0$ **do**
7: Calculate errors E^l.
8: Calculate weight gradients ΔW^l.
9: **for** each layer $l = 0, ...L-1$ **do**
10: Update parameters, W^l and B^l.

Forward: $a_n^l = \sigma(\sum_{i=0}^{|W|-1} w_i^l a_{n+i}^{l-1} + b_n^l)$

Backward: $e_n^l = \sum_{i=0}^{|W|-1} w_i^{l+1} e_{n-i}^{l+1},$

where a_n^l, b_n^l, and e_n^l are the n^{th} activation, bias, and error in layer l respectively, w_i^l is an weight on the i^{th} connection between layer l and layer $l-1$, $|W|$ is the number of weights in a feature map, and σ is the activation function.

SGD training of NNs as matrix operations

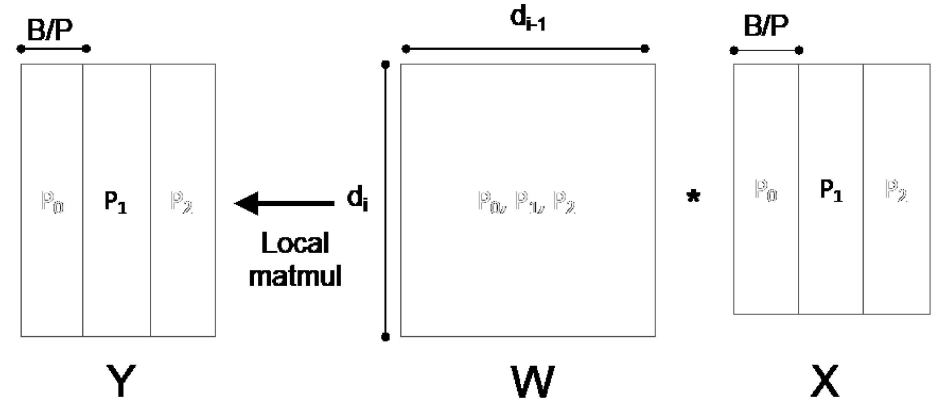


N = the number of outputs
M = the number of inputs
B = the size of the minibatch

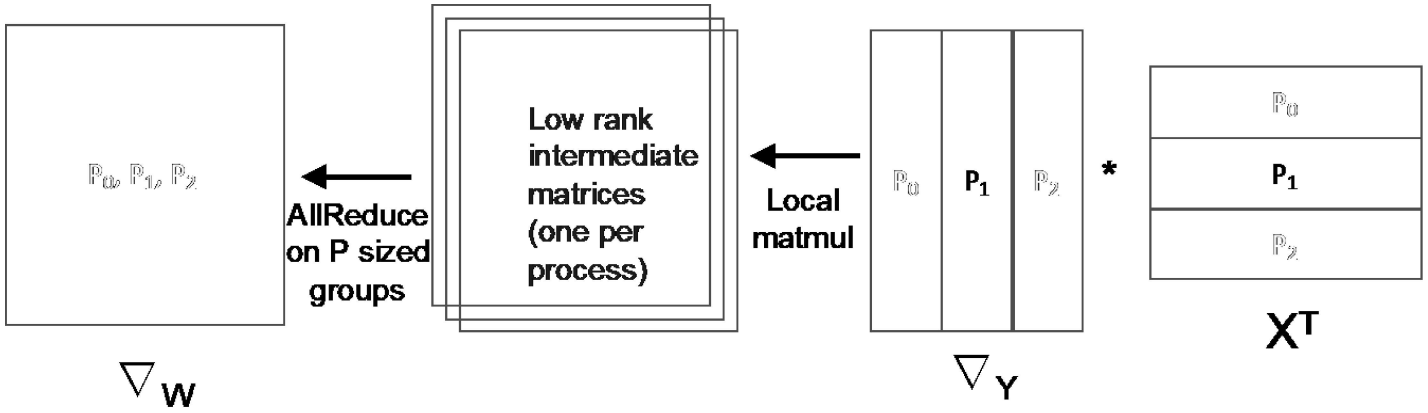
The impact to parallelism:

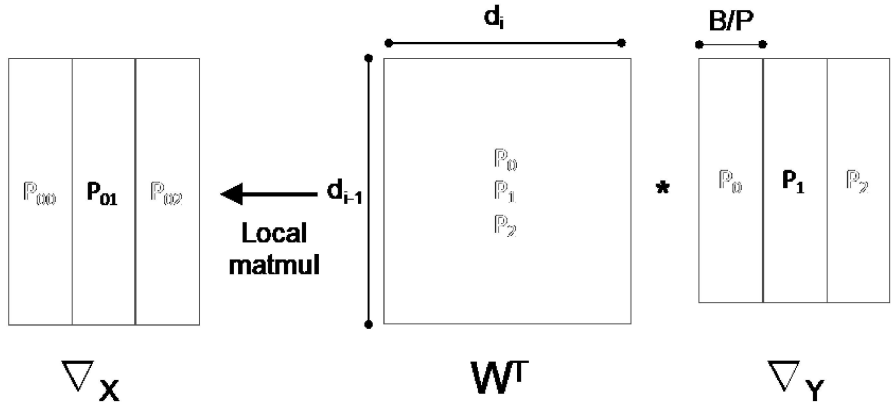
- **W** is replicated to processor, so it doesn't change
- $\mathbf{X_{in}}$ and $\mathbf{X_{out}}$ gets skinnier if we only use data parallelism, i.e. distributing **b=B/p** mini-batches per processor
- GEMM (i.e. matrix-matrix multiply) performance suffers as *matrix dimensions get smaller* and *more skewed*
- **Result:** Data parallelism can hurt single-node performance

Data Parallel SGD training of NNs as matrix operations



1. Which matrices are replicated?
2. Where is the communication?
3. Which steps can be overlapped?





$\nabla_Y = \partial L / \partial Y$ = how did the lost function change as output activations change?

$\nabla_X = \partial L / \partial X$

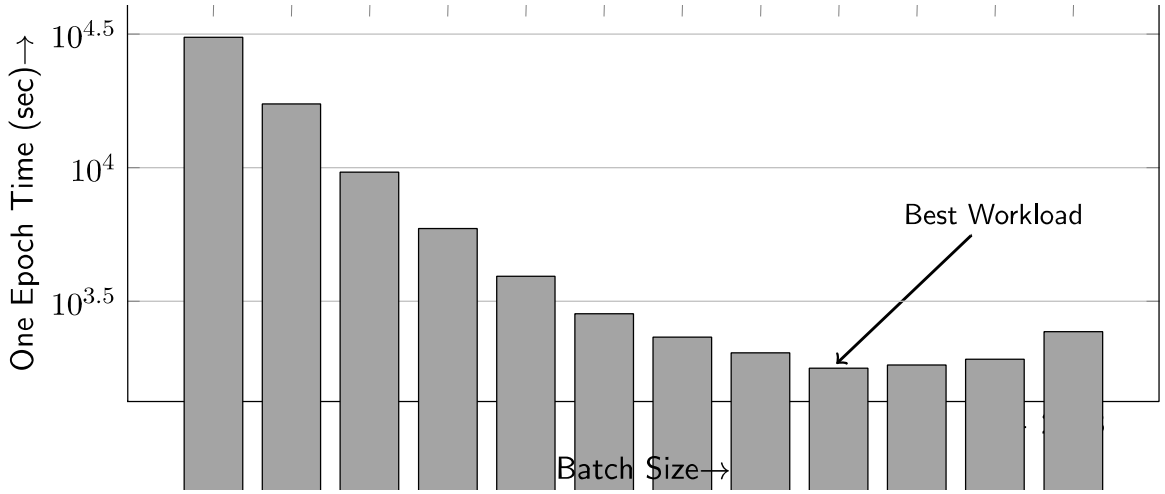
$\nabla_W = \partial L / \partial W$

Batch Parallel Strong Scaling

- **Per-iteration communication cost of batch parallelism is independent of the batch size:**
  - **larger batch → less communication per epoch (full pass over the data set)**

$$T_{comm}(batch) = 2\sum_{i=0}^{L}\left(\alpha\log(P) + \beta\frac{P-1}{P}|W_i|\right)$$

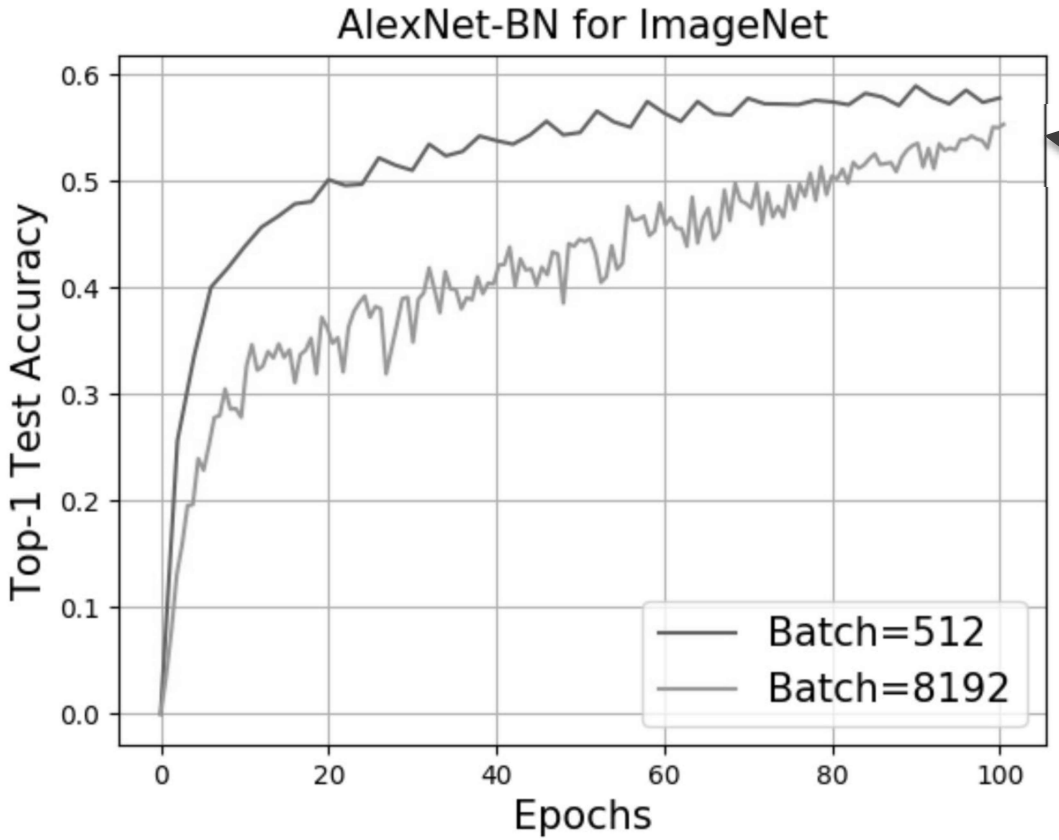
- But processor utilization goes down significantly for P>>1
  - Result: Batch parallel has poor strong scaling



One epoch training time of AlexNet computed on a single KNL

Problems with Batch Parallelism

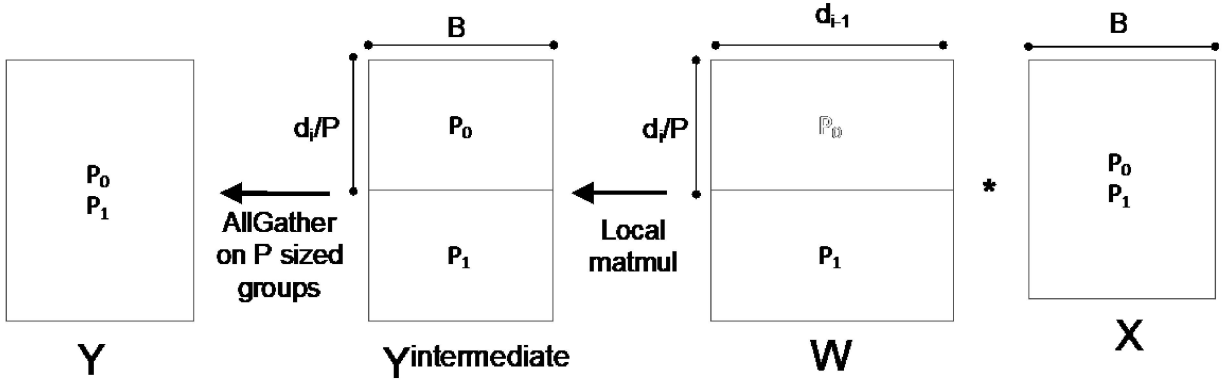
- Batch parallel scaling is limited to B
  - Larger Batch -> higher strong scaling efficiency
- But SGD does not perform well for large batch

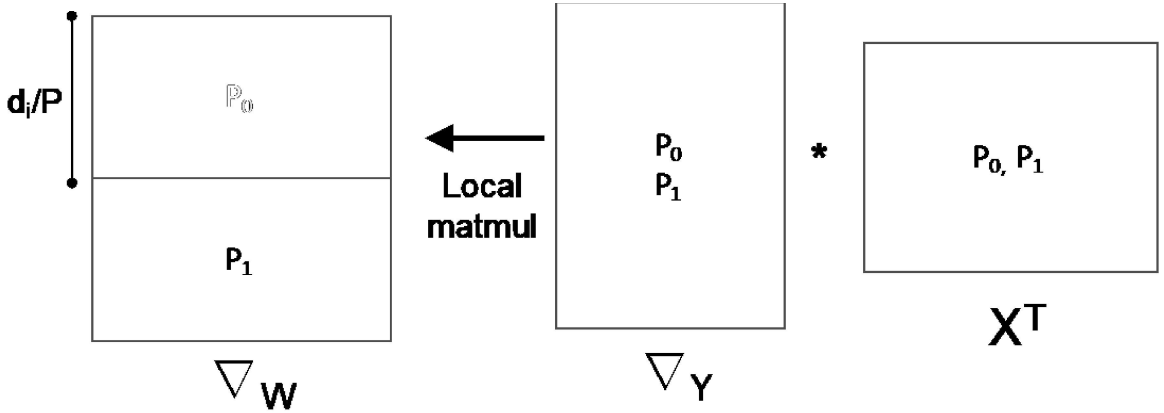


Performance Gap

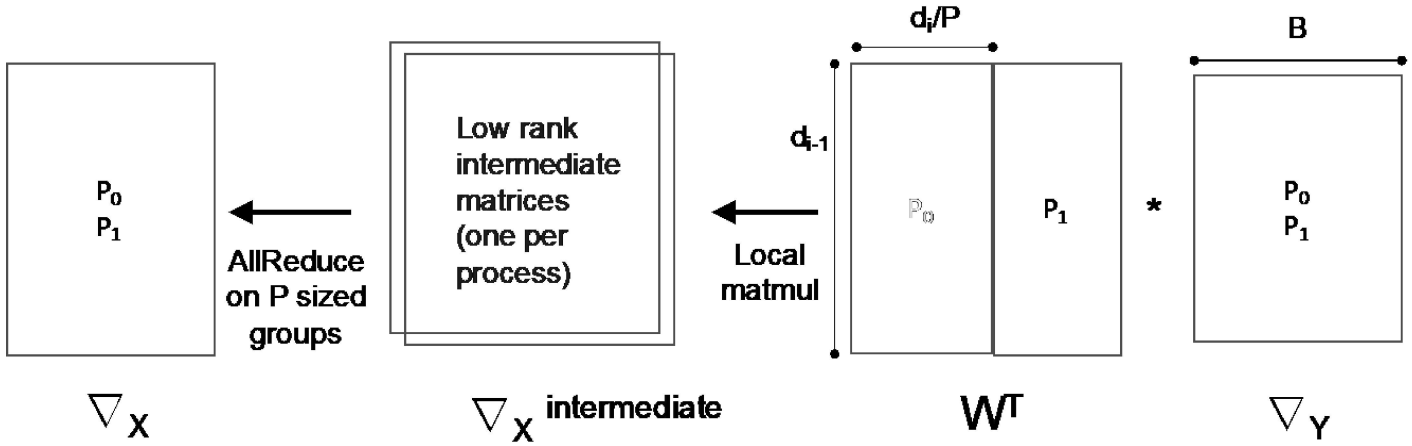
Ginsburg, Boris, Igor Gitman, and Yang You. "Large Batch Training of Convolutional Networks with Layer-wise Adaptive Rate Scaling." (2018).

Model Parallel SGD training of NNs as matrix operations





1. Which matrices are replicated?
2. Where is the communication?
3. How can matrix algebra capture both model and data parallelism?



Communication Complexity of Domain Parallel

- Additional communication for halo exchange during forward and backwards pass
  - Negligible cost for early layers for which activation size is large (i.e. convolutional)

$$T_{comm}(domain) = \sum_{i=0}^{L} \left(\alpha + \beta B X_W^i X_C^i k_h^i / 2\right)$$
$$+ \sum_{i=0}^{L} \left(\alpha + \beta B Y_W^i Y_C^i k_w^i / 2\right)$$
$$+ 2 \sum_{i=0}^{L} \left(\alpha \log(P) + \beta \frac{P-1}{P} |W_i|\right)$$