Distributed Deep Learning

CS5242

Lee Hwee Kuan & **Wang Wei**

Teaching assistant:

Connie Kou Khor Li, Ji Xin, Ouyang Kun

cs5242@comp.nus.edu.sg

Administrative

- Assignment 3
  - GAN
  - Vanilla RNN (forward and backward)
  - Due on 24 Nov. 2017, 23:59.
- Projects
  - Due on 10 Nov. 2017, 23:59.
  - Report and code due on 16 Nov. 2017, 16:00
  - Submission
    - Report
      - <=4 pages, task description, changes to existing models, findings, performance.
      - Evaluation based on clarity, completeness and findings. NOT based on number of pages.
    - Code
    - See the next slide for the submission folder structure
  - **NOTE: Additional training data is not allowed**

Administrative

- Choose 2 submissions for private leader board ranking on kaggle
  - Final evaluation is based on the private leader board
- Rename your kaggle group name as Group ID (e.g. Group 08).
- Group ID (e.g. group08)
  - report.pdf (including the group member information)
  - workspace
    - train.py or train.sh
    - predict.py or predict.sh
    - readme.txt (for any other things to be noted for evaluation, e.g. the running environment, GPU, CUDA, Cudnn versions, memory requirement)
    - requirements.txt (third party library name and version)
    - test.csv (to be generated by predict.py or predict.sh for submission)

    - data (for project 1)
      - train_images
      - test_images
      - train.csv
      - sampleSubmission.csv
    - data (for project 2)
      - train.json
      - test.json
- **NOTE**:
  - You can have other code files or folders in workspace
  - Before you zip the submission folder, remove the data folder and other intermediate data (including the checkpoint of the models and word vectors).
  - Do not use absolute path like “/home/wangwei/cs5242/project1/workspace/data”. Use relative path like “./data/”

Intended learning outcome

know the approaches to improve the efficiency of deep learning training

Understand the two parallelisms for distributed deep learning

(Optional) use distributed training provided by existing frameworks

Challenges of deep learning training

- Long Time
  - Total training time to achieve a certain loss
  - AlexNet
    - 5-6 days using two Nvidia GTX 580 (3GB)
- Big Memory
  - The maximum memory consumption during training
  - VGG
    - 4 NVIDIA Titan Black GPU (6GB), 2-3 weeks

Cost analysis

- Time
  - Number of SGD iterations
    - Learning rate, momentum
  - Time per SGD iteration
    - Back-propagation time
      - Convolution layer, pooling layer, fully-connected layer, etc.
      - Batchsize, input sample size, kernel size, number of kernels, output feature size
    - SGD update time
      - Element-wise operations, fast.
      - Number of parameters
- How to speed up?

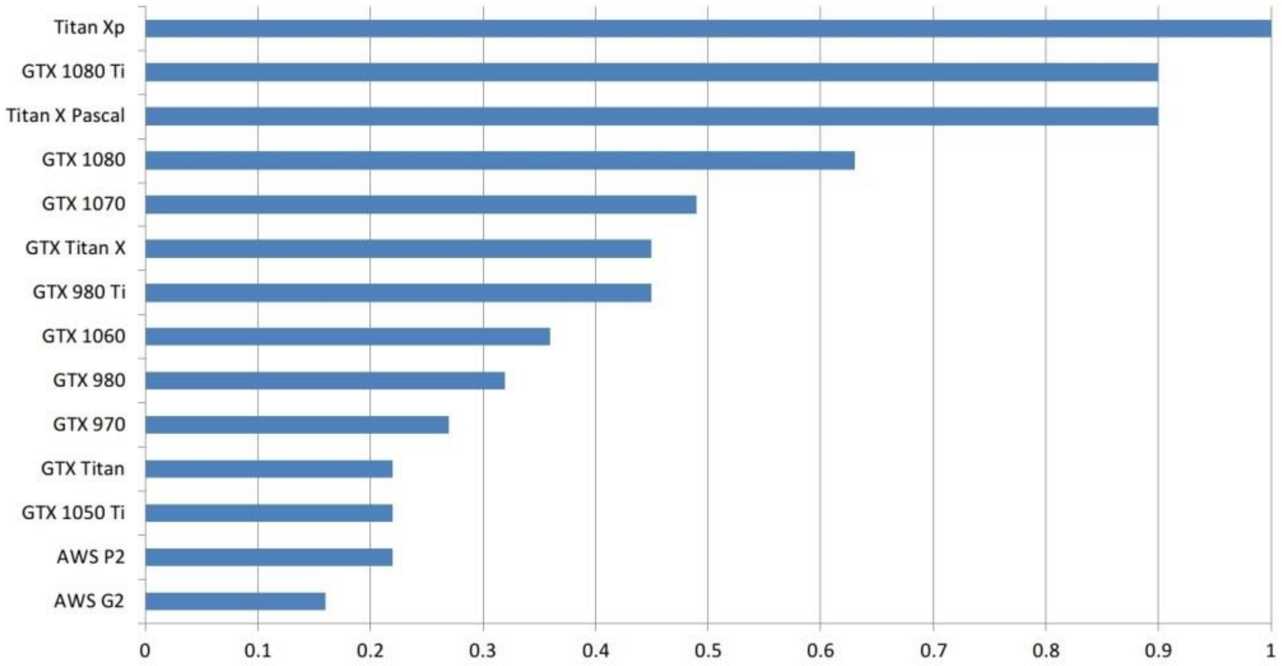
Cost analysis

- Memory
  - At least one batch of input sample
  - Parameters and their gradients
  - Feature maps (vectors) and the gradient of the loss w.r.t each layer
- How to save memory and handle big models?

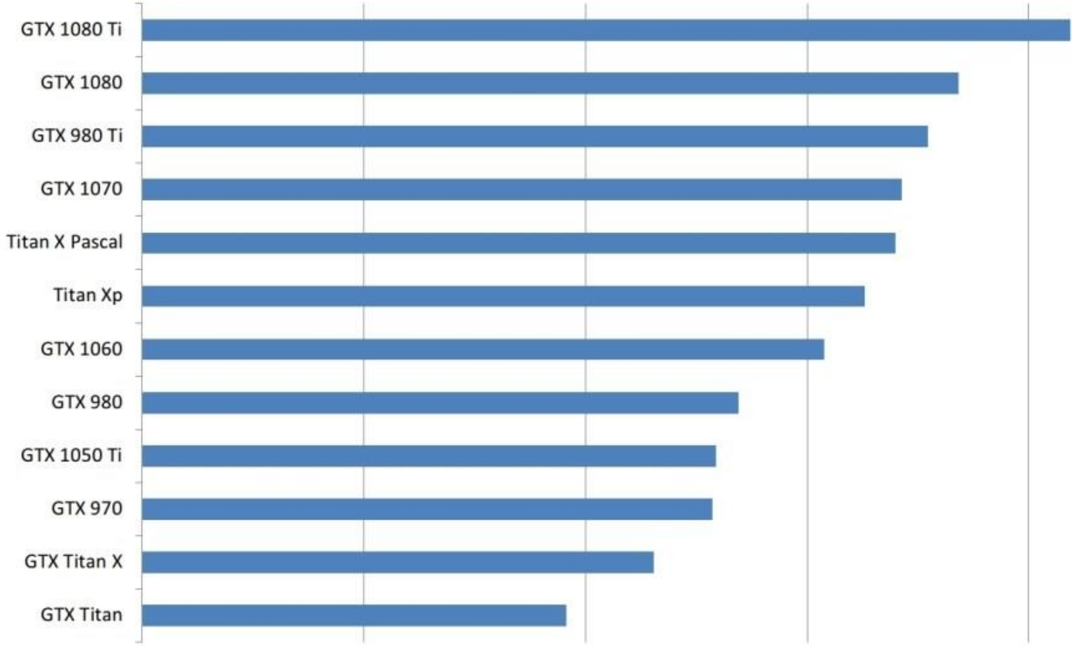
Approaches

- Faster and bigger GPUs
  - >= 3GB

performance comparisons



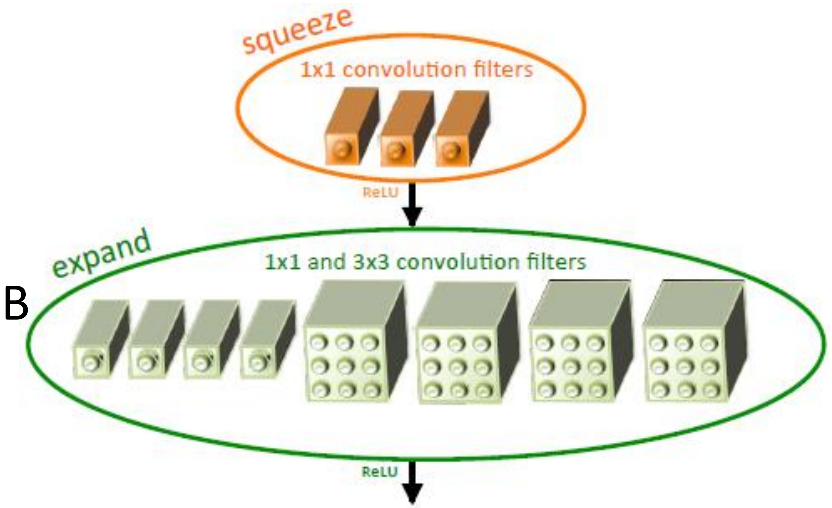
Normalized cost efficiency



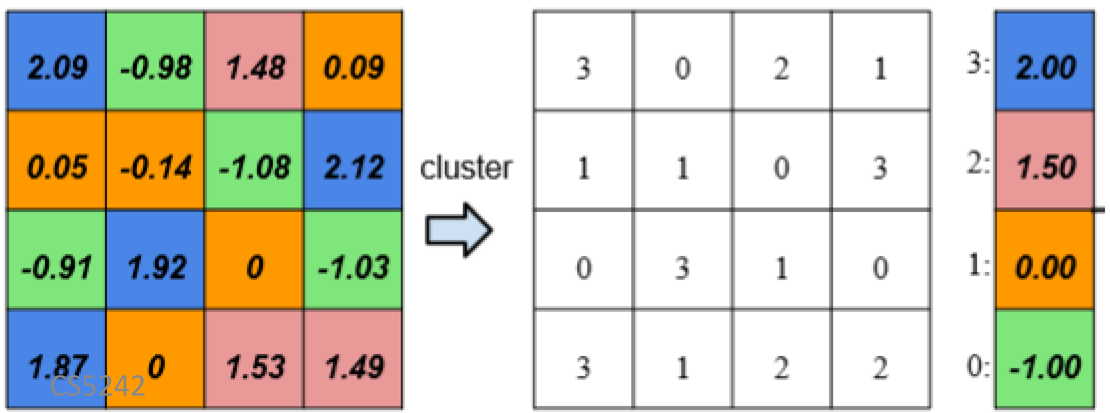
Source from: http://timdettmers.com/2017/04/09/which-gpu-for-deep-learning/

Approaches

- Model compression
  - Fewer filters, smaller filter size, bottleneck layers
    - SqueezeNet[1] (for AlexNet): 50X fewer parameters, <0.5MB model size
  - Lower precision parameters, and parameter quantization (for Inference)
    - Float32->float16->floatX (x<16) [2]
    - Cluster each weight matrix into 16 groups and represent each value by group ID (4bits) [3]
  - Memory VS Speed
    - Reduce memory
    - May speed up



Source from [1]



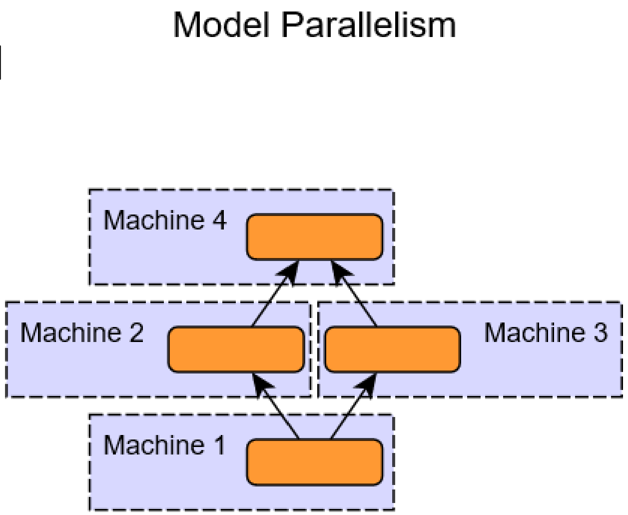
Source from [2]

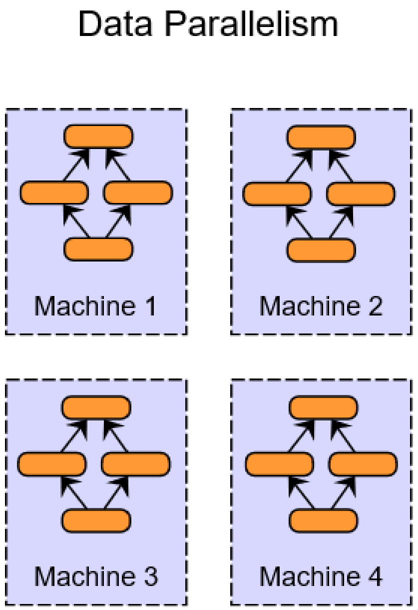
Approaches

- Faster SGD
  - Adaptive learning rate
    - AdaGrad, AdaDelta, Adam
  - Momentum
  - Speed VS memory

Approaches

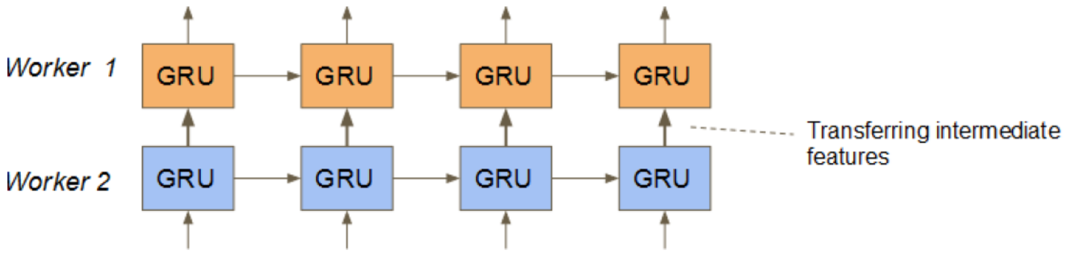
- Distributed training
  - Each computing instance is called a worker
    - A GPU node/A CPU thread
    - In the same server or different servers
  - Partition the model
    - layers onto different workers
    - Each worker stores partial of the model
      - Less memory
      - Faster BP? (may not)
  - Partition the data
    - Samples onto different workers
    - Each worker uses smaller batchsize
      - Less memory
      - Faster BP

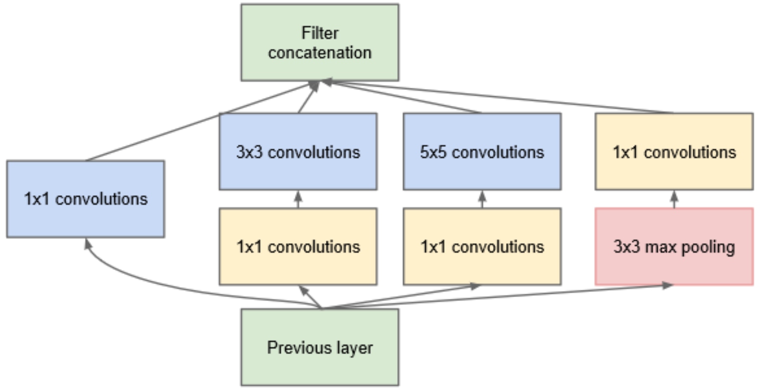




Model partitioning/parallelism

- Partitioning the layers
  - RNN
    - Top stack of GRU to worker 1
    - Bottom stack of GRU to worker 2
  - Inception
    - one path per worker

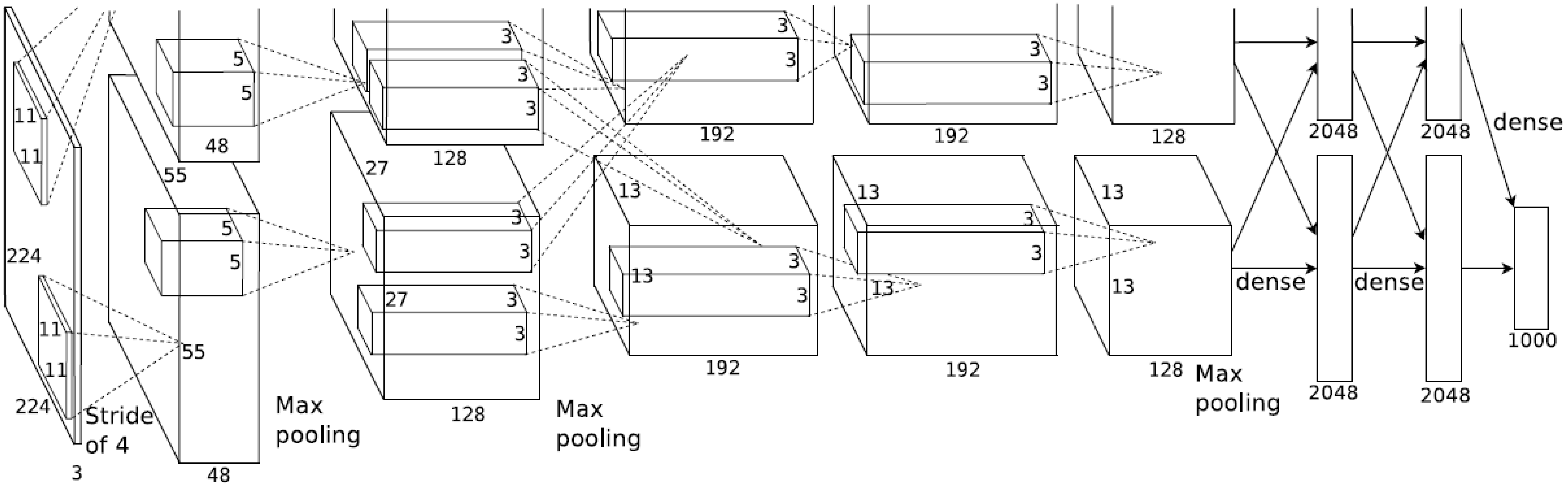




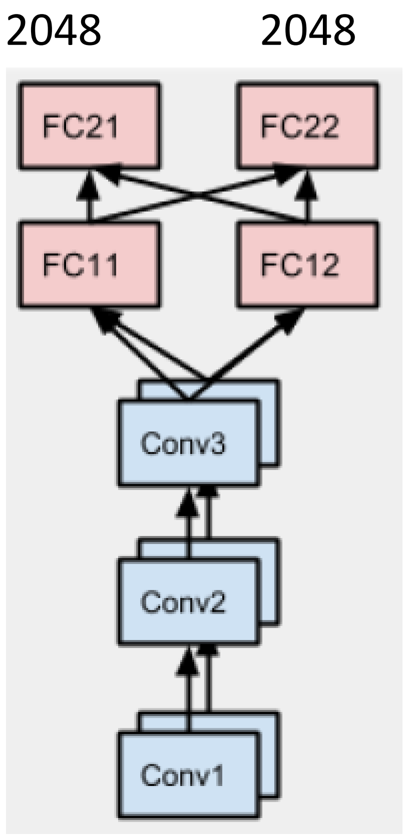
(b) Inception module with dimension reductions

Model partitioning/parallelism

- Partition the filters of conv layer
  - Original AlexNet [4]
- Partition the neurons of FC layer
  - AlexNet using 4 GPUs [5]



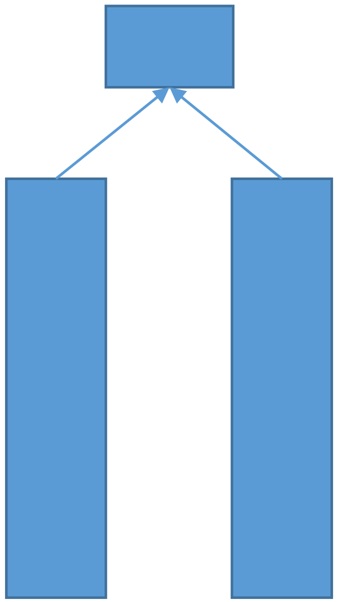
Source from [4]



Source from [5]

Model partitioning/parallelism

- Not widely used?
  - Synchronization overhead
  - Complex to implement
  - Only works for models with multiple paths
    - like Siamese network



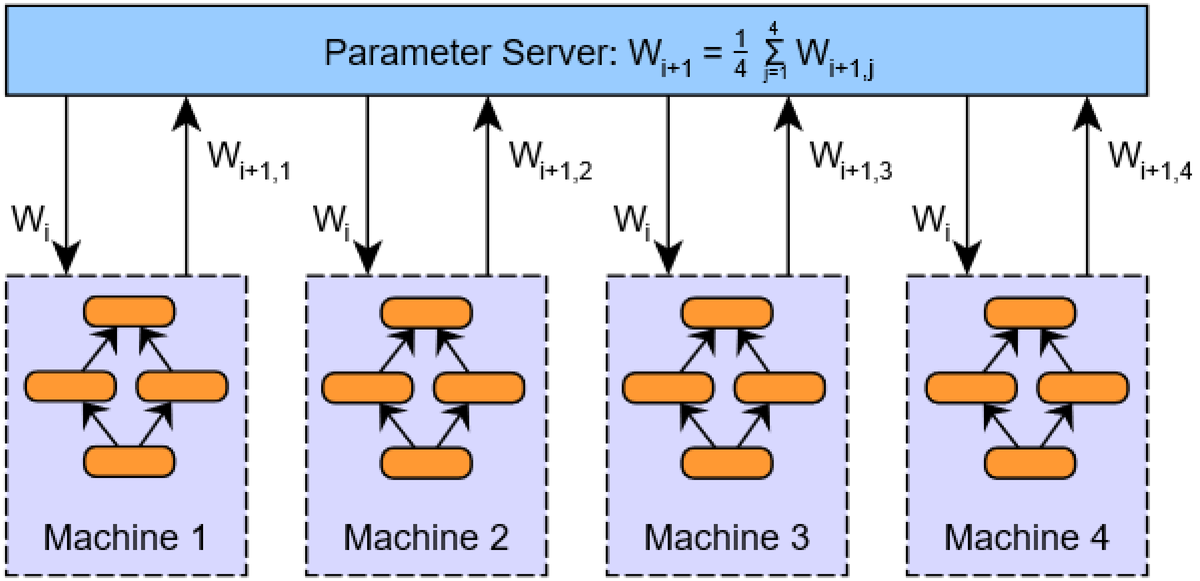
Siamese network

Data partitioning/parallelism

- Replicate the models onto each worker
- Partition the dataset over all workers
- Parameter server framework
  - Each worker conducts BP over its own data
  - A (logical) central server conducts the SGD
- Consistency
  - Synchronous
  - Asynchronous

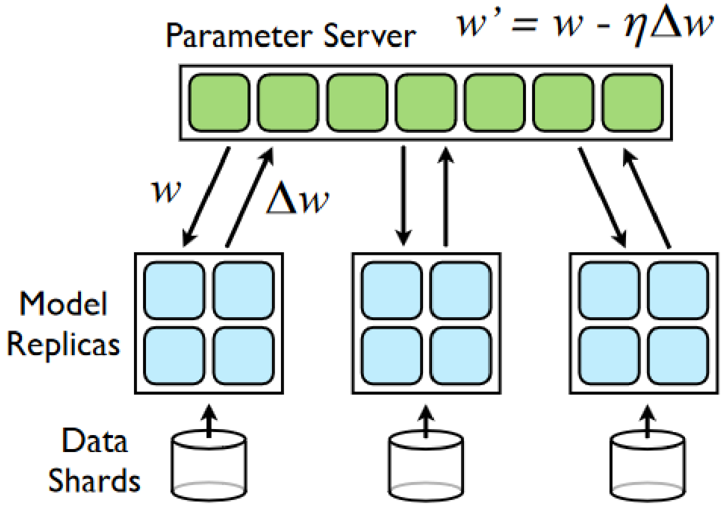
Synchronous

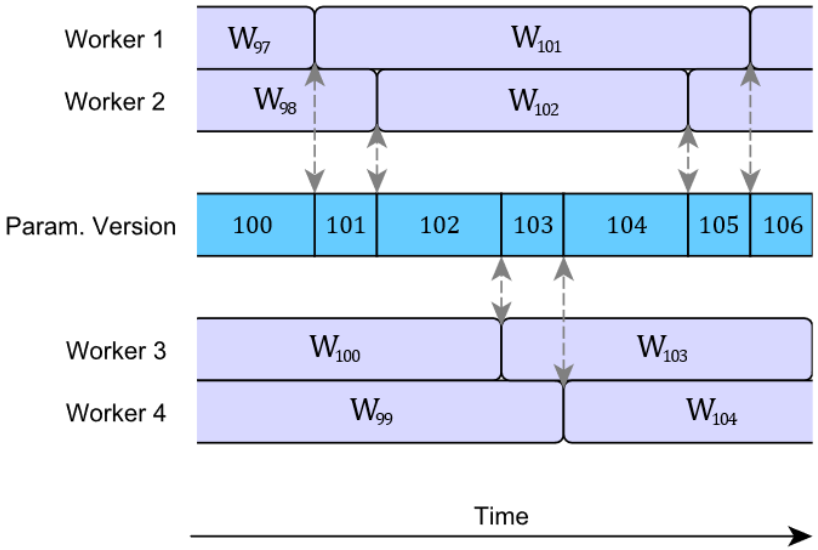
- For each iteration
  - server waits until it receives gradients from all workers
  - Server broadcasts the new version of gradients to workers
- Communication VS computation
  - AlexNet, VGG
  - InceptionNet
  - Large mini-batch [6,7]
- Easy to implement
  - For single node with 3-8 GPUs



Asynchronous

- Server updates the parameters
  - Once it receives the gradients from any worker
- Workers run asynchronously
  - Gradient staleness





$\vdots$

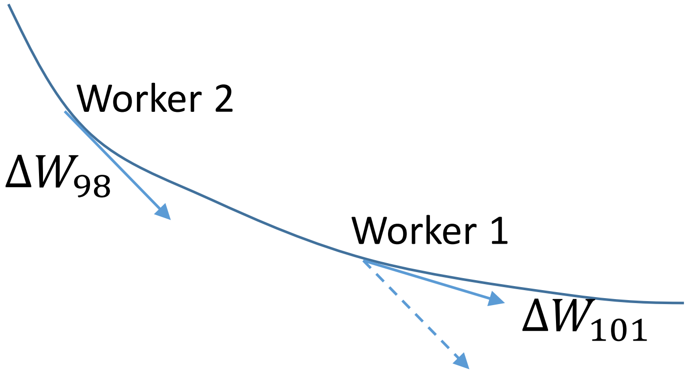
$W_{101}=W_{100}-\lambda\Delta W_{97}$

$W_{102}=W_{101}-\lambda\Delta W_{98}$

$W_{103}=W_{102}-\lambda\Delta W_{100}$

$W_{104}=W_{103}-\lambda\Delta W_{99}$

$\vdots$



Asynchronous

- Backup workers [8]
  - Do update once the server receives gradients from γ (< 1) workers
- Bounded Staleness [9]
  - Stop the fast workers and let them wait for a while
- Staleness-aware learning rate [10]

Reference

- [1] F. Iandola, S. Han, M. Moskewicz, K. Ashraf, W. J. Dally, K. Keutzer, “SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and < 0.5MB Model Size”, arXiv 16.
- [2] Matthieu Courbariaux, Yoshua Bengio, Jean-Pierre David. Training deep neural networks with low precision multiplications 2015. https://arxiv.org/abs/1412.7024
- [3] S. Han, H. Mao, W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding”, ICLR’16.
- [4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. 2012
- [5] One weird trick for parallelizing convolutional neural networks
- Alex Krizhevsky. 2014. https://arxiv.org/abs/1404.5997
- [6] Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He Tech report, arXiv, June 2017
- [7] Large Batch Training of Convolutional Networks. Yang You, Igor Gitman, Boris Ginsburg. 2017. https://arxiv.org/abs/1708.03888
- [8] Revisiting Distributed Synchronous SGD. Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, Rafal Jozefowicz. 2017. https://arxiv.org/abs/1604.00981
- [9] More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A. Gibson, Greg Ganger, Eric P. Xing. 2013. NIPS
- [10] Staleness-aware Async-SGD for Distributed Deep Learning. Wei Zhang, Suyog Gupta, Xiangru Lian, Ji Liu. 2016. https://arxiv.org/abs/1511.05950