Large Scale Distributed Deep Networks

Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc' Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Andrew Y. Ng

Google Brain

NIPS 2012

Summary by Justin Chen

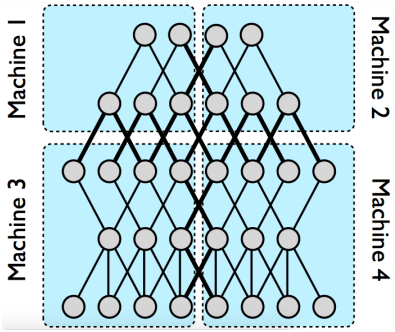
1. Introduction

- Limitation of training neural networks on GPUs is only small speed-up if model and large batches do not fit into GPU's memory
    - Need to reduce model size or batch size to reduce CPU-to-GPU transfer
    - Limits use on real-world data which is high dimensional
- **Contribution: DistBelief (later renamed to TensorFlow)**
    - **Model parallelism**
        - Utilizes multithreading on a single machine
        - Utilizes message passing across machines
        - Parallelism, synchronization, and communication abstracted away by framework
    - **Data parallelism**
        - Multiple copies on model used for optimizing a common objective
    - **Downpour SGD** - asynchronous SGD with adaptive learning rate
    - **Sandblaster L-BFGS** - distributed L-BFGS that uses data and mode parallelism

2. Previous Work

- Stale gradients for convex optimization
- Sparse gradients with lock-less asynchronous SGD on shared-memory architectures
- Parallelizing layers across clusters
- MapReduce and GraphLab ill-suited for iterative computations (training over epochs), and exploiting properties of computation graph (matrix computations)
- **Goal: distributed asynchronous training without assuming the problems to be convex, sparsity constraints, or constraining neural network architecture and utilizing data parallelism**

3. Model parallelism

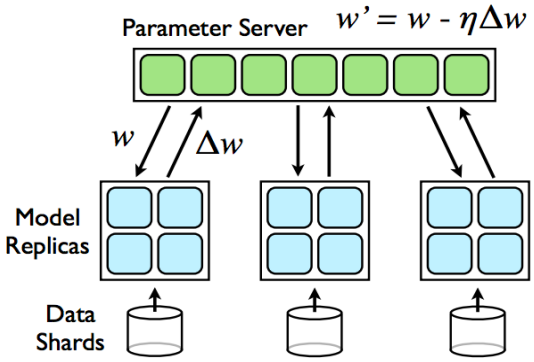


Model parallelism in DistBelief

- DistBelief framework allows:
  - user to define computation of each layer on each node in cluster and t
  - define the messages (activations, gradients, etc.) that should be communicated between these components
  - automatically parallelize across all available cores
  - manage communication, synchronization and data transfer between nodes during both training and inference
- Trade-off between training speed-up (convergence) and communication cost
- Sparse (local connectivity) components benefit more from distributed training than fully-connected components
- Suboptimal speed-up due to variance in computation time of individual machines
  - Must wait for slowest machine so can synchronize parameters

4. Distributed optimization algorithms

- Downpour SGD uses online training
- Sandblaster L-BFGs uses batch training
- Both
  - Use centralized shared parameter server for replicas to share parameters
  - Tolerate variance in processing speed of different model replicas
  - Tolerate wholesale failure of model replicas



Downpour SGD

- Original SGD is sequential making it impractical for distributed training

4.1. Distributed optimization algorithms

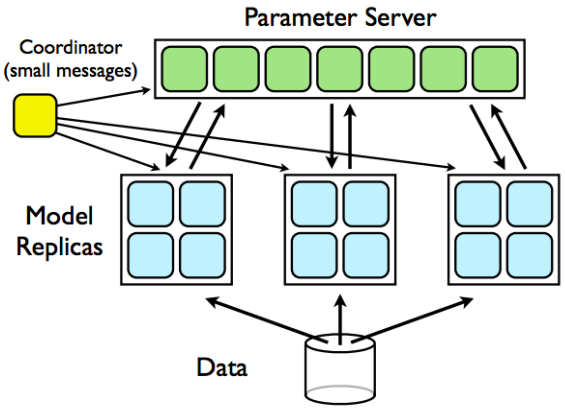
- **Downpour SGD** - Asynchronous SGD that uses multiple replicas of the model to train on even-sized subsets of the data
- Replicas share updated parameters via parameter server
- Model replicas run independently (blue blocks)
- **Parameter server** - server containing global model parameters
  - Divided into shards (green blocks) that run independently
    - Each shard is responsible for different part of model
- **Training procedure**:
  - Before each mini-batch, local model replica queries parameter server service for most recent parameters
    - Updates its local model
    - Trains locally
    - Computes **parameter gradient** - difference in updated parameters and current parameters
    - Sends parameter gradient to parameter server to update global model
  - Can reduce communication cost by only pulling parameters every n iterations, and pushing gradients every m steps
- **Disadvantages**:
  - Stale gradients – gradients computed w.r.t. outdated set of parameters
  - Asymmetric updates across parameter shards
  - Disordered updates across parameter shards
  - Inconsistent timestamps for parameters
  - No convergence guarantees nor reliability guarantees
- Adagrad improves robustness of Downpour SGD
  - Learning rate for individual parameters:

$$\eta_{i,K} = \gamma / \sqrt{\sum_{j=1}^{K} \Delta w_{i,j}^2}$$

γ constant scaling factor for all learning rates

$w_{i,j}^2$ gradient of i-th parameter on iteration K

  - Extends maximum number of mode replicas that can work simultaneously
  - Combined with warm startup training, stabilizes training of Downpour



Sandblaster L-BFGS

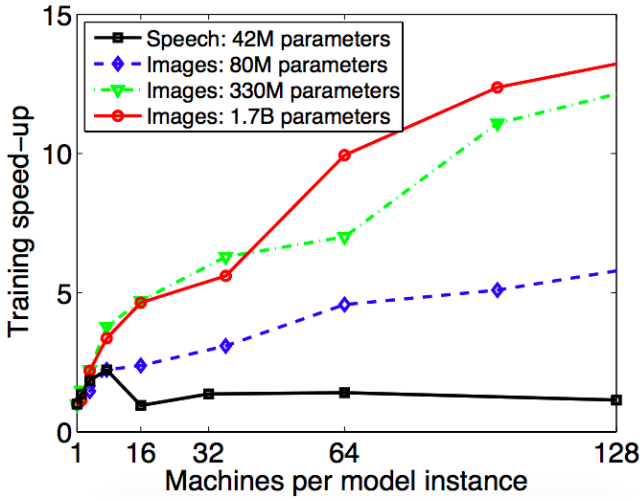
4.2. Sandblaster L-BFGS

- **Sandblaster** - distributed parameter storage and manipulation
- Batch optimization framework
- Coordinator issues commands for each parameter server shard
    - Dot product, scaling, coefficient-wise addition, multiplication
    - Results stored locally on shard instead of on parameter server
    - **Load balancing scheme**
        - Coordinator assigns each model replicas batch which is less than 1/N of total batch when model replica is available
        - Coordinator schedules multiple copies of outstanding data to multiple workers and takes the result from the one that finishes first
        - Only send accumulated gradients to server every few completes intervals

5. Experiments

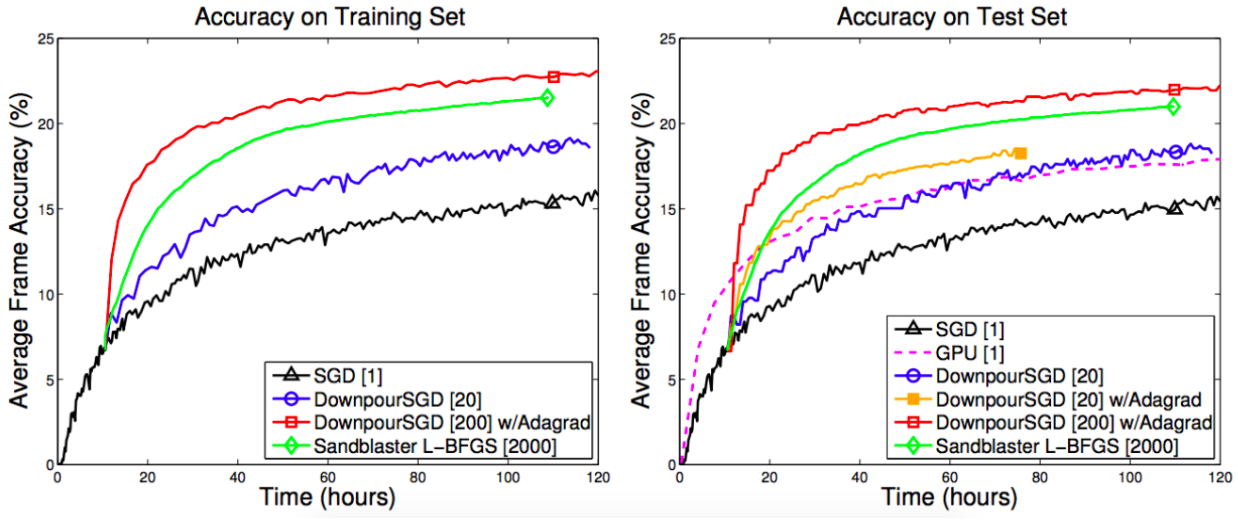
- Benchmarks: object recognition, and speech recognition
- Speech recognition
    - 5 layer, 4 hidden layers using sigmoid 2560 wide, softmax 8192 wide
    - Input 11 consecutive overlapping 25 ms frames of speech represented as a 40-dimensional vector of log-energy values
    - 1.1 billion samples
- Object recognition
    - ImageNet - 16 million images
        - Preprocessed to 100x100 pixel images
        - Convnet with 10x10 convolutions, pooling, and the local contrast normalization and experimented with architectures using 8 channels to 36 channels

Model parallelism benchmarks:



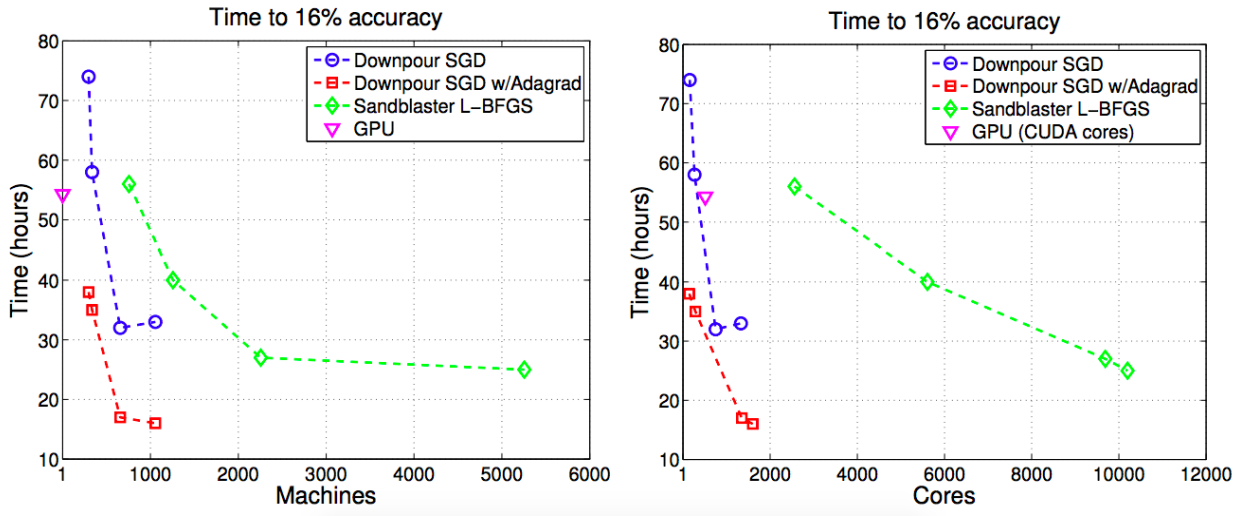
- Measure mean time to process a single mini-batch using vanilla SGD as a function of cluster size as seen in the graph
- 20 cores per machine
- Larger models benefit more from distributed training that smaller models because cost of communicating parameters is outweighed by amount of computation that can be completed per machine leading to net gain in performance
- Convnets benefit more from distributed training than densely connected layers

Optimization method comparisons:



Classification performance as a function of training time

- Baselines:
    - Downpour SGD with fixed learning rate
    - Downpour SGD with Adagrad
    - Sandblaster L-BFGS
- **Goal: obtain maximum test set accuracy in minimum amount of training time**
- Fastest is Downpour SGD using Adagrad with 200 model replicas (red curves)
- Given sufficient CPUs, Sandblaster L-BFGS and Downpour SGD using Adagrad can train substantially faster than a high performance GPU



Resource consumption for performance trade-off

- Arbitrarily chose 16% test set accuracy
- Measured time to reach 16% accuracy as a function of machine and utilized CPU cores
- Points closer to origin are preferable (less time with less resources)
- Downpour with Adagrad has best trade-off

Applications to ImageNet:

- Downpour SGD on 1.7 billion parameter convnet
- Cross-validated classification accuracy of over 15% on all 21,000 ImageNet categories

6. Conclusions

- Downpour SGD with Adagrad works well with 2000 CPU cores or less
- Surprising that Adagrad worked well with nonconvex problems and highly nonlinear deep networks
- **Conjecture** that Adagrad automatically **stabilizes volatile parameters** in asynchronous training with more fine-grained learning rate for each layer
- Clusters using just CPUs can outperform a GPU

Questions and Comments

- As seen in the first graph, image benefits substantially more than speech. I wonder if the nature of the compositionality of the data affects the ability to distribute training. They did mention that sparse connectivity benefits more than dense connections.
- Seems like Adagrad is also what enables them to use stale gradients