Large Scale Distributed Deep Networks

Yifu Huang

School of Computer Science, Fudan University
huangyifu@fudan.edu.cn

COMP630030 Data Intensive Computing Report, 2013

Outline

1. Motivation
2. Software Framework: DistBelief
3. Distributed Algorithm
   - Downpour SGD
   - Sandblaster L-BFGS
4. Experiments
5. Discussion

Motivation

- Why I choose this paper [1]?
  - Google + Stanford
  - Jeffrey Dean + Andrew Y. Ng
- Why we need large scale distributed deep networks?
  - Large model can dramatically improve performance
    - Training examples + model parameters
  - Exist methods have limitations
    - GPU, MapReduce, GraphLab
- What can we learn from this paper?
  - Best parallelism design ideas for deep networks up to now
    - Model parallelism + data parallelism

Motivation

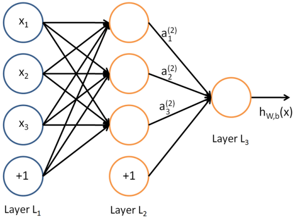
- Why I choose this paper [1]?
  - Google + Stanford
  - Jeffrey Dean + Andrew Y. Ng
- Why we need large scale distributed deep networks?
  - Large model can dramatically improve performance
    - Training examples + model parameters
  - Exist methods have limitations
    - GPU, MapReduce, GraphLab
- What can we learn from this paper?
  - Best parallelism design ideas for deep networks up to now
    - Model parallelism + data parallelism

Motivation

- Why I choose this paper [1]?
  - Google + Stanford
  - Jeffrey Dean + Andrew Y. Ng
- Why we need large scale distributed deep networks?
  - Large model can dramatically improve performance
    - Training examples + model parameters
  - Exist methods have limitations
    - GPU, MapReduce, GraphLab
- What can we learn from this paper?
  - Best parallelism design ideas for deep networks up to now
    - Model parallelism + data parallelism

Preliminaries

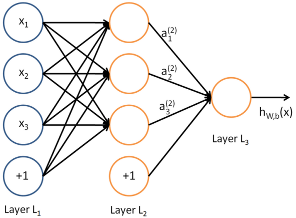
- Neural Networks [6]



- Deep Networks [7]
  - Multiple hidden layers
  - This will allow us to compute much more complex features of the input

Preliminaries

- Neural Networks [6]



- Deep Networks [7]
  - Multiple hidden layers
  - This will allow us to compute much more complex features of the input

Software Framework: DistBelief

- Model parallelism
  - "Inside" parallelism
  - Multi-thread + message passing -> large scale
- User defines
  - Computation in node, message upward/downward
- Framework manages
  - Synchronization, data transfer
- Performance depends on
  - Connectivity structure, computational needs

Software Framework: DistBelief

- Model parallelism
  - "Inside" parallelism
  - Multi-thread + message passing -> large scale
- User defines
  - Computation in node, message upward/downward
- Framework manages
  - Synchronization, data transfer
- Performance depends on
  - Connectivity structure, computational needs

Software Framework: DistBelief

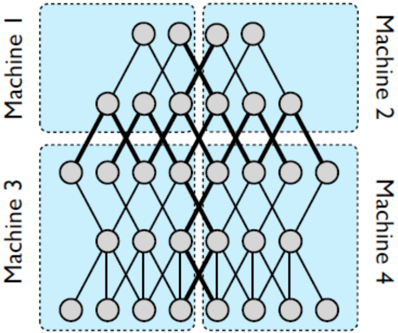
- Model parallelism
  - "Inside" parallelism
  - Multi-thread + message passing -> large scale
- User defines
  - Computation in node, message upward/downward
- Framework manages
  - Synchronization, data transfer
- Performance depends on
  - Connectivity structure, computational needs

Software Framework: DistBelief

- Model parallelism
  - "Inside" parallelism
  - Multi-thread + message passing -> large scale
- User defines
  - Computation in node, message upward/downward
- Framework manages
  - Synchronization, data transfer
- Performance depends on
  - Connectivity structure, computational needs

Software Framework: DistBelief (cont.)

- An example of model parallelism in DistBelief



Distributed Algorithm

- Data parallelism
  - "Outside" parallelism
  - Multiple model instances optimize a single objective -> high speed
- A centralized sharded parameter server
  - Different model replicas retrieve/update their own parameters
- Load balance, robust
  - Tolerate variance in the processing speed of different model replicas
  - The wholesale failure may be taken offline or restart at random

Distributed Algorithm

- Data parallelism
  - "Outside" parallelism
  - Multiple model instances optimize a single objective -> high speed
- A centralized sharded parameter server
  - Different model replicas retrieve/update their own parameters
- Load balance, robust
  - Tolerate variance in the processing speed of different model replicas
  - The wholesale failure may be taken offline or restart at random

Distributed Algorithm

- Data parallelism
  - "Outside" parallelism
  - Multiple model instances optimize a single objective -> high speed
- A centralized sharded parameter server
  - Different model replicas retrieve/update their own parameters
- Load balance, robust
  - Tolerate variance in the processing speed of different model replicas
  - The wholesale failure may be taken offline or restart at random

Outline

1 Motivation

2 Software Framework: DistBelief

3 Distributed Algorithm
- Downpour SGD
- Sandblaster L-BFGS

4 Experiments

5 Discussion

Downpour SGD

- SGD [8]
  - Minimize the object function $F(\omega)$
  - Update parameters $\omega' = \omega - \eta \Delta \omega$
  - asynchronous SGD [3]
- Downpour
  - Massive parameters retrieved and updated
  - 
- Adagrad learning rate [4]
  - $\eta_{i,K} = \gamma / \sqrt{\sum_{j=1}^{K} \Delta \omega_{i,j}^2}$
  - Improve both robust and scale

Downpour SGD

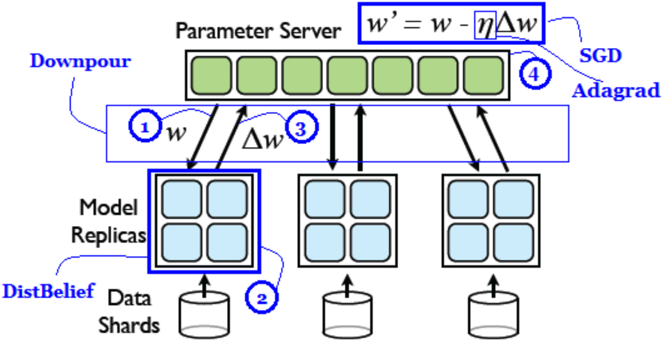
- SGD [8]
  - Minimize the object function $F(\omega)$
  - Update parameters $\omega' = \omega - \eta \Delta \omega$
  - asynchronous SGD [3]
- Downpour
  - Massive parameters retrieved and updated
  - 
- Adagrad learning rate [4]
  - $\eta_{i,K} = \gamma / \sqrt{\sum_{j=1}^{K} \Delta \omega_{i,j}^2}$
  - Improve both robust and scale

Downpour SGD

- SGD [8]
  - Minimize the object function $F(\omega)$
  - Update parameters $\omega' = \omega - \eta \Delta \omega$
  - asynchronous SGD [3]
- Downpour
  - Massive parameters retrieved and updated
  - 
- Adagrad learning rate [4]
  - $\eta_{i,K} = \gamma / \sqrt{\sum_{j=1}^{K} \Delta \omega_{i,j}^2}$
  - Improve both robust and scale

Downpour SGD (cont.)

- Algorithm visualization



Downpour SGD (cont.)

- Algorithm pseudo code

Algorithm 1.1: DOWNPOURSGDCLIENT($\alpha, n_{fetch}, n_{push}$)

```
procedure STARTASYNCHRONOUSLYFETCHINGPARAMETERS(parameters)
  parameters ← GETPARAMETERSFROMPARAMSERVER()

procedure STARTASYNCHRONOUSLYPUSHINGGRADIENTS(accruedgradients)
  SENDGRADIENTSTOPARAMSERVER(accruedgradients)
  accruedgradients ← 0

main
  global parameters, accruedgradients
  step ← 0                                                          1
  accruedgradients ← 0
  while true
       { if (step mod n_fetch) == 0
       {   then STARTASYNCHRONOUSLYFETCHINGPARAMETERS(parameters)
       { data ← GETNEXTMINIBATCH()
       { gradient ← COMPUTEGRADIENT(parameters, data)
    do { accruedgradients ← accruedgradients + gradient          2     3
       { parameters ← parameters − α ∗ gradient
       { if (step mod n_push) == 0
       {   then STARTASYNCHRONOUSLYPUSHINGGRADIENTS(accruedgradients)
       { step ← step + 1
```

Outline

1. Motivation
2. Software Framework: DistBelief
3. Distributed Algorithm
   - Downpour SGD
   - Sandblaster L-BFGS
4. Experiments
5. Discussion

Sandblaster L-BFGS

- BFGS [9]
  - An iterative method for solving unconstrained nonlinear optimization
  - Compute an approximation to the Hessian matrix B
  - Limited-memory BFGS [10]
- Sandblaster
  - Massive commands issued by coordinator

- Load banancing scheme
  - Dynamic work assigned by coordinator

Sandblaster L-BFGS

- BFGS [9]
  - An iterative method for solving unconstrained nonlinear optimization
  - Compute an approximation to the Hessian matrix B
  - Limited-memory BFGS [10]
- Sandblaster
  - Massive commands issued by coordinator
  - 

- Load banancing scheme
  - Dynamic work assigned by coordinator

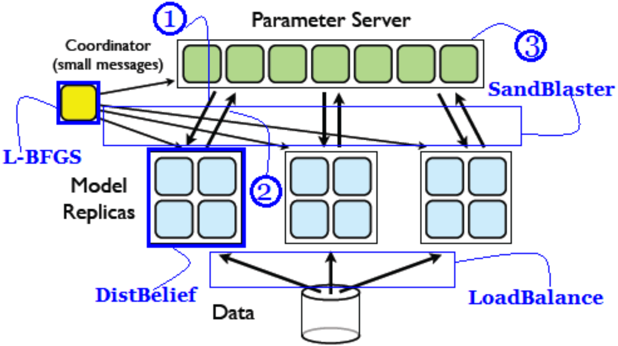
Sandblaster L-BFGS

- BFGS [9]
  - An iterative method for solving unconstrained nonlinear optimization
  - Compute an approximation to the Hessian matrix B
  - Limited-memory BFGS [10]
- Sandblaster
  - Massive commands issued by coordinator

- Load banancing scheme
  - Dynamic work assigned by coordinator

Sandblaster L-BFGS (cont.)

- Algorithm visualization



Sandblaster L-BFGS (cont.)

- Algorithm pseudo code

Algorithm 1.2: SANDBLASTERLBFGS()

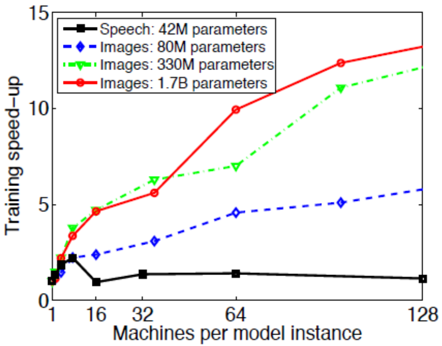
```
procedure REPLICA.PROCESSPORTION(portion)
  if (!hasParametersForStep)
    then parameters ← GETPARAMETERSFROMPARAMSERVER()
  data ← GETDATAPORTION(portion)
  gradient ← COMPUTEGRADIENT(parameters, data)
  localAccruedGradients ← localAccruedGradients + gradient

procedure PARAMETERSERVER.PERFORMOPERATION(operation)
  PerformOperation

main
  step ← 0
  while true
  do {
      comment: PS: ParameterServer
      PS.accruedgradients ← 0
      while (batchProcessed < batchSize)
      do {
          for all (modelReplicas) comment: Loop is parallel and asynchronous
            {
              if (modelReplicaAvailable)                                        1  2
                then { REPLICA.PROCESSPORTION(modelReplica)
                       batchProcessed ← batchProcessed + portion
              if (modelReplicaWorkDone and timeToSendGradients)
                then { SENDGRADIENTS(modelReplica)
                       PS.accruedGradients ← PS.accruedGradients + gradient
            }
         }
      COMPUTELBFGSDIRECTION(PS.Gradients, PS.History, PS.Direction)
      LINESEARCH(PS.Parameters, PS.Direction)                                   3
      PS.UPDATEPARAMETERS(PS.parameters, PS.accruedGradients)
      step ← step + 1
     }
```

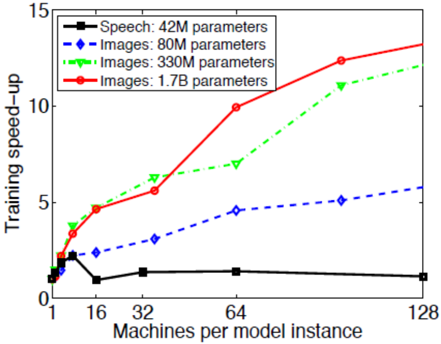
Experiments

- Setup
  - Object recognition in still images [5]
  - Acoustic processing for speech recognition [2]
- Model parallelism benchmarks



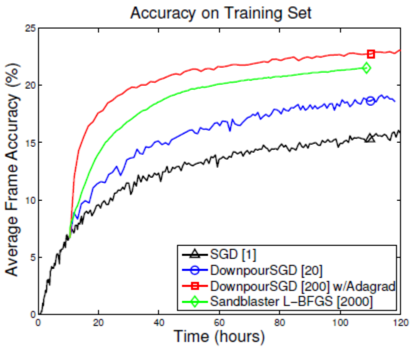
Experiments

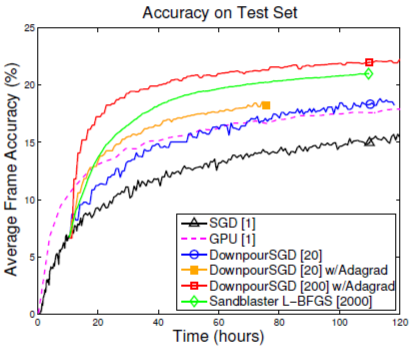
- Setup
  - Object recognition in still images [5]
  - Acoustic processing for speech recognition [2]
- Model parallelism benchmarks



Experiments (cont.)

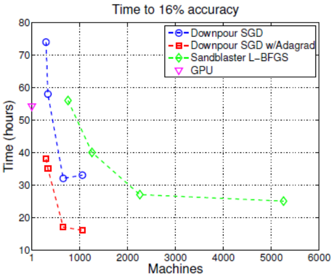
- Optimization method comparisons

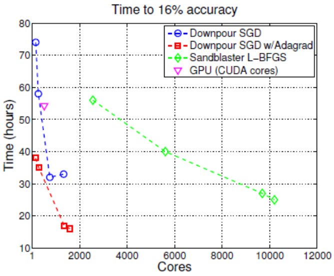




Experiments (cont.)

- Optimization method comparisons





- Application to ImageNet [2]
  - This network achieved a cross-validated classification accuracy of over 15%, a relative improvement over 60% from the best performance we are aware of on the 21k category ImageNet classification task

Discussion

- Contributions
  - Increase the scale and speed of deep networks training
- Drawbacks
  - There is no guarantee that parameters are consistent

Discussion

- Contributions
  - Increase the scale and speed of deep networks training
- Drawbacks
  - There is no guarantee that parameters are consistent

References I

- [1] Large Scale Distributed Deep Networks. NIPS. 2012.
- [2] Building High-level Features Using Large Scale Unsupervised Learning. ICML. 2012.
- [3] Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. NIPS. 2011.
- [4] Adaptive subgradient methods for online learning and stochastic optimization. JMLR. 2011.
- [5] Improving the speed of neural networks on cpus. NIPS. 2011.
- [6] http://ufldl.stanford.edu/wiki/index.php/Neural_Networks
- [7] http://ufldl.stanford.edu/wiki/index.php/Deep_Networks:_Overview
- [8] http://ufldl.stanford.edu/wiki/index.php/Gradient_checking_and_advanced_optimization

References II

- [9] http://en.wikipedia.org/wiki/BFGS
- [10] http://en.wikipedia.org/wiki/Limited-memory_BFGS