Parallel Implementation of Deep Learning Using MPI

CSE633 Parallel Algorithms (Spring 2014)

Instructor: Prof. Russ Miller

Team #13: Tianle Ma

Email: tianlema@buffalo.edu

May 7, 2014

Content

- Introduction to Deep Belief Network
- Parallel Implementation Using MPI
- Experiment Results and Analysis

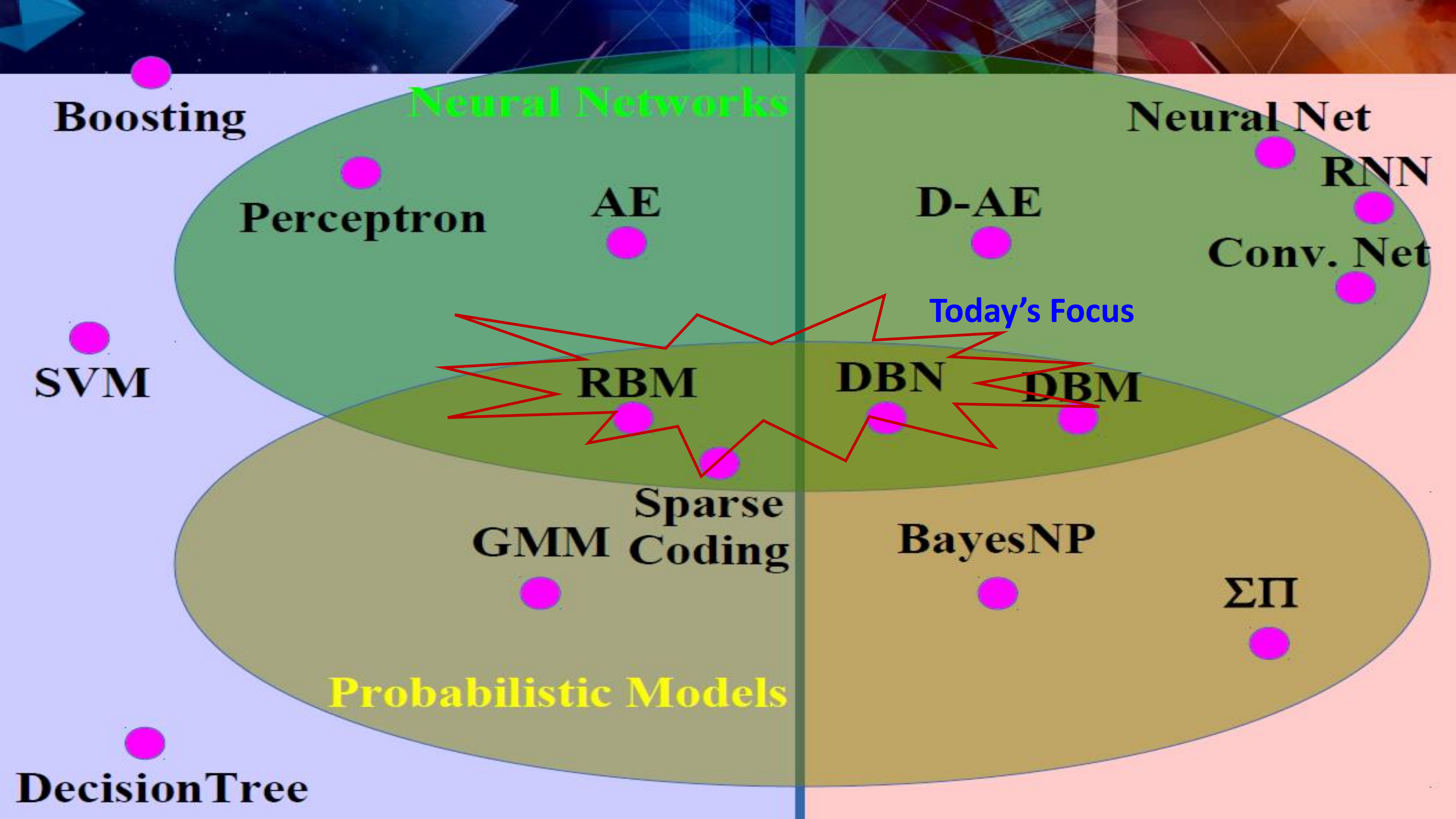
Why Deep Learning?

- **Deep Learning is a set of algorithms in machine learning that attempt to model high-level abstractions in data by using architectures composed of multiple non-linear transformations.**

- **It has been the hottest topic in speech recognition, computer vision, natural language processing, applied mathematics, … in the last 2 years**

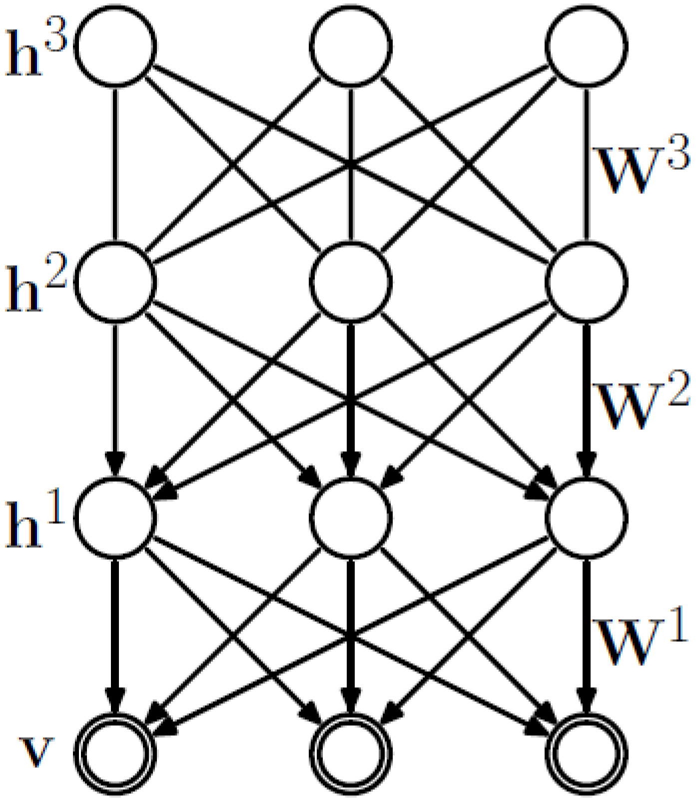
- **Deep Learning is about representing high-dimensional data**

- **It's deep if it has more than one stage of non-linear feature transformation**



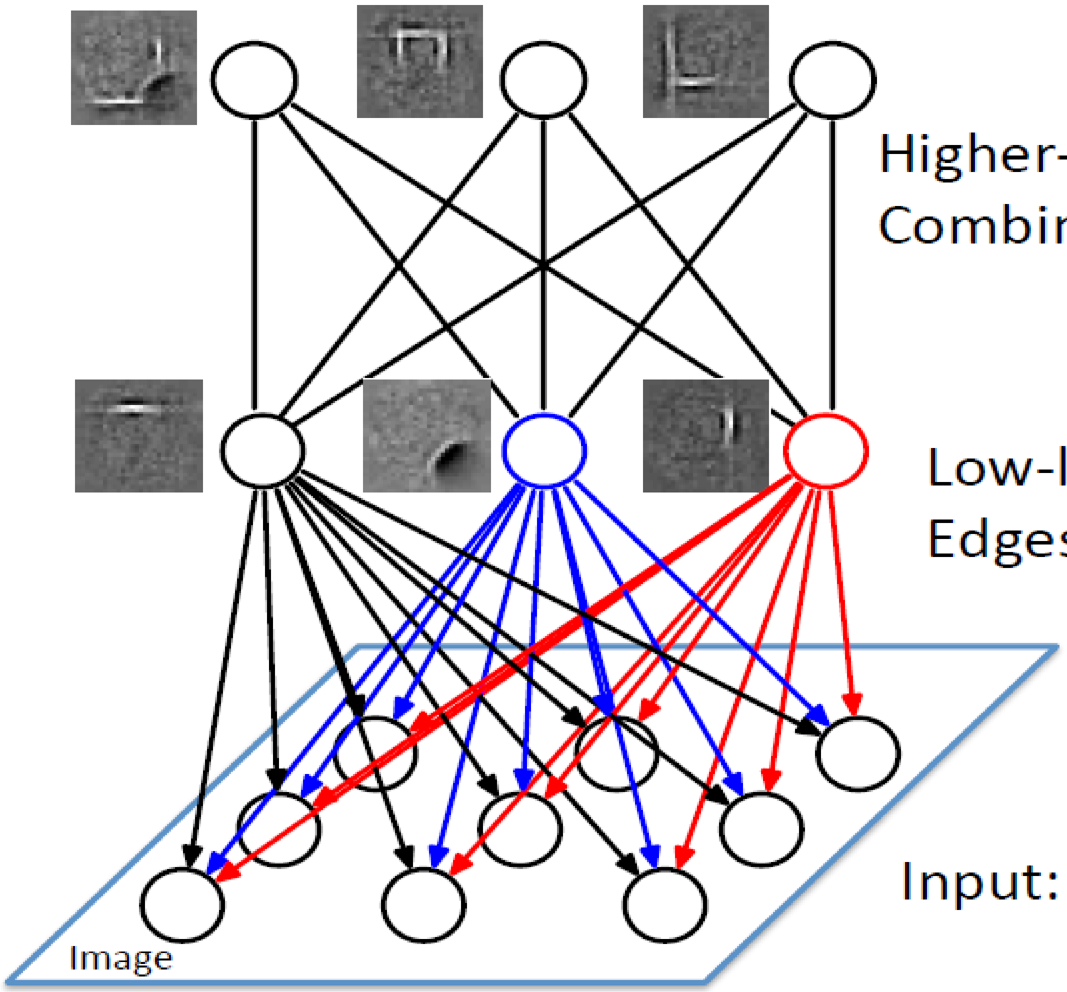
Neural Networks
Boosting
Neural Net
RNN
Perceptron
AE
D-AE
Conv. Net
Today's Focus
SVM
RBM
DBN
DBM
Sparse Coding
GMM
BayesNP
ΣΠ
Probabilistic Models
DecisionTree

Deep Belief Network



- Probabilistic Generative model.
- Contains multiple layers of nonlinear representation.
- Fast, greedy layer-wise pretraining algorithm.
- Inferring the states of the latent variables in highest layers is easy.

Deep Belief Network



Internal representations capture higher-order statistical structure

Higher-level features: Combination of edges

Low-level features: Edges

Built from **unlabeled** inputs.

Input: Pixels

(Hinton et.al. Neural Computation 2006)

Deep Belief Network

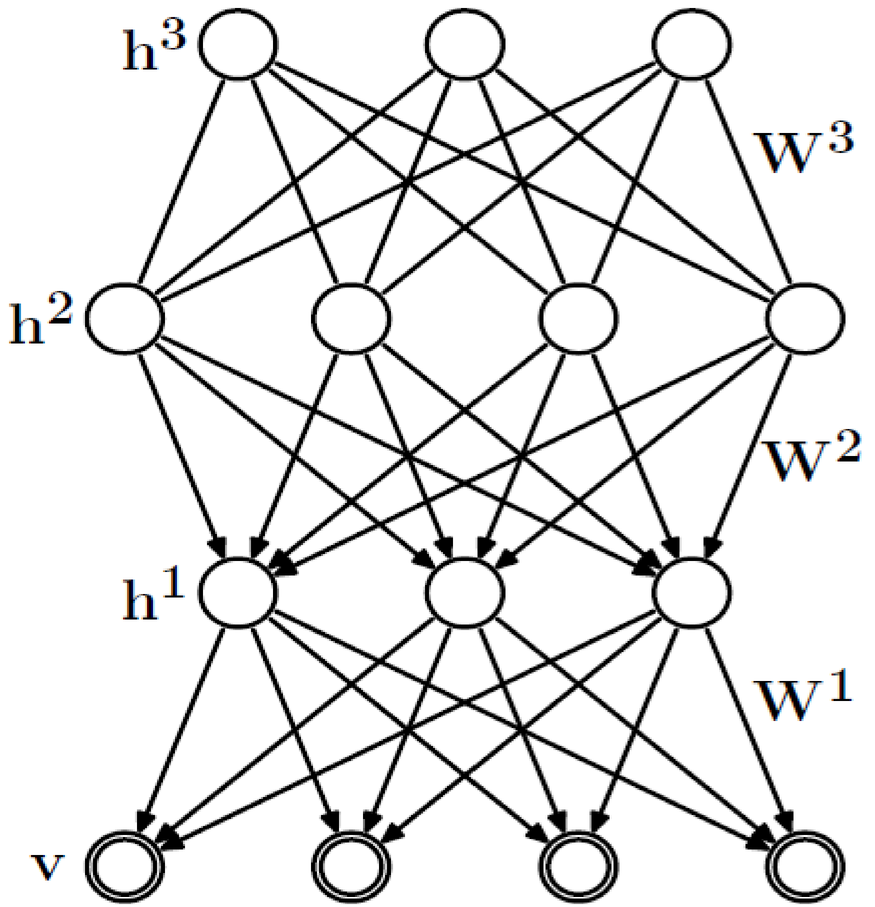
Approximate Inference

Stacked Restricted Boltzmann Machines

$Q(\mathbf{h}^3|\mathbf{h}^2)$

$Q(\mathbf{h}^2|\mathbf{h}^1)$

$Q(\mathbf{h}^1|\mathbf{v})$



Generative Process

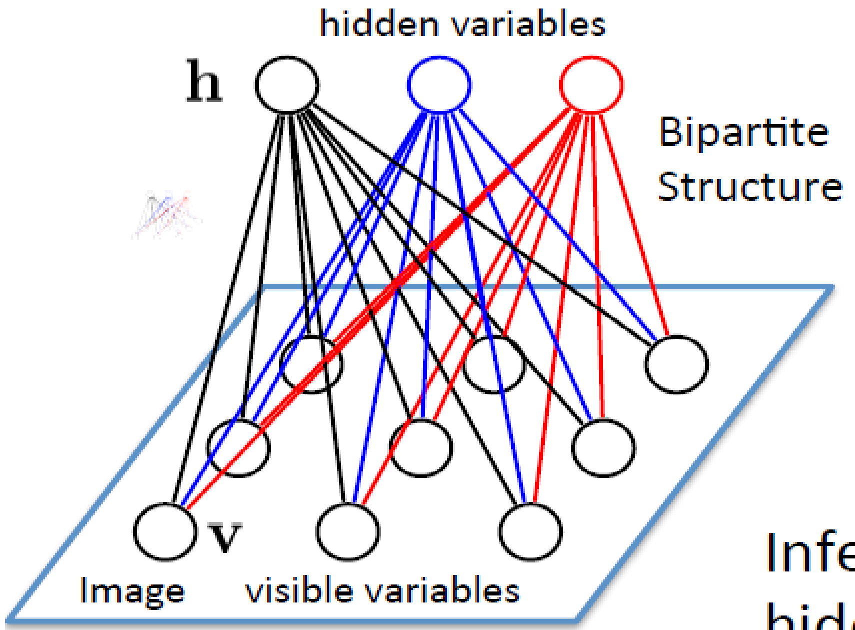
$P(\mathbf{h}^2, \mathbf{h}^3)$

$P(\mathbf{h}^1|\mathbf{h}^2)$

$P(\mathbf{v}|\mathbf{h}^1)$

$$Q(\mathbf{h}^t|\mathbf{h}^{t-1}) = \prod_j \sigma\left(\sum_i W^t h_i^{t-1}\right) \qquad P(\mathbf{h}^{t-1}|\mathbf{h}^t) = \prod_j \sigma\left(\sum_i W^t h_i^t\right)$$

Restricted Boltzmann Machines



$$p(v,h) = \frac{1}{Z} e^{h^T W v + b^T v + a^T h}$$

Restricted: No interaction between hidden variables

Inferring the distribution over the hidden variables is easy:

$$\underbrace{P(\mathbf{h}|\mathbf{v}) = \prod_j P(h_j|\mathbf{v})}_{\text{Factorizes: Easy to compute}} \quad P(h_j = 1|\mathbf{v}) = \frac{1}{1 + \exp(-\sum_i W_{ij} v_i - a_j)}$$

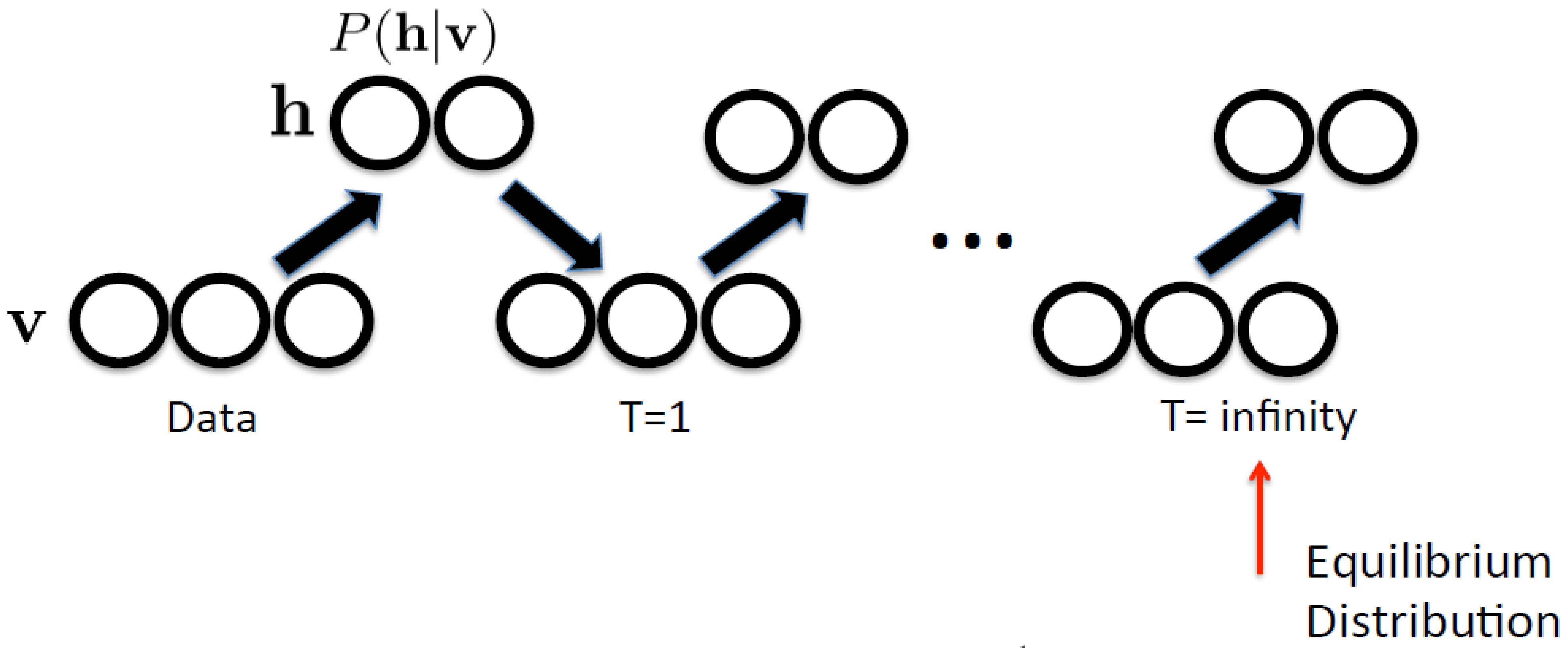
Similarly:

$$P(\mathbf{v}|\mathbf{h}) = \prod_i P(v_i|\mathbf{h}) \quad P(v_i = 1|\mathbf{h}) = \frac{1}{1 + \exp(-\sum_j W_{ij} h_j - b_i)}$$

Markov random fields, Boltzmann machines, log-linear models.

Approximate ML Learning for RBMs

Run Markov chain (alternating Gibbs Sampling):

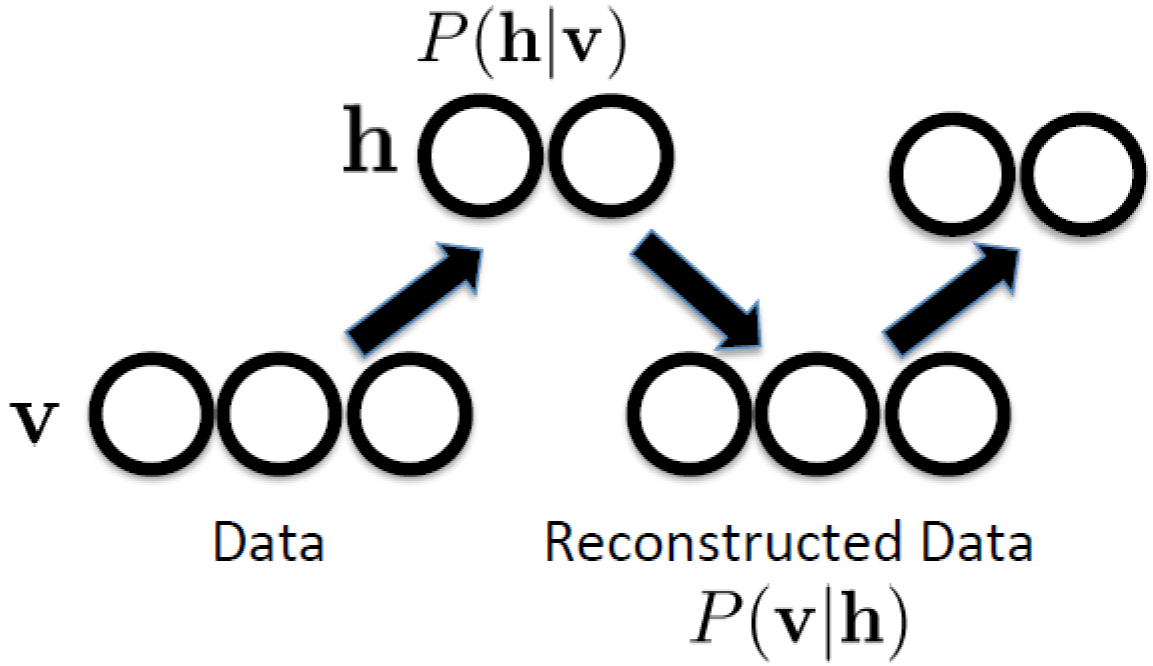


$$P(\mathbf{h}|\mathbf{v}) = \prod_j P(h_j|\mathbf{v}) \quad P(h_j = 1|\mathbf{v}) = \frac{1}{1 + \exp(-\sum_i W_{ij}v_i - a_j)}$$

$$P(\mathbf{v}|\mathbf{h}) = \prod_i P(v_i|\mathbf{h}) \quad P(v_i = 1|\mathbf{h}) = \frac{1}{1 + \exp(-\sum_j W_{ij}h_j - b_i)}$$

Contrastive Divergence

A quick way to learn RBM:



- Start with a training vector on the visible units.
- Update all the hidden units in parallel.
- Update the all the visible units in parallel to get a "reconstruction".
- Update the hidden units again.

Update model parameters:

$$\Delta W_{ij} = \mathrm{E}_{P_{data}}[v_i h_j] - \mathrm{E}_{P_1}[v_i h_j]$$

Algorithm1 RBMupdate (v_0, ϵ, W, b, c)

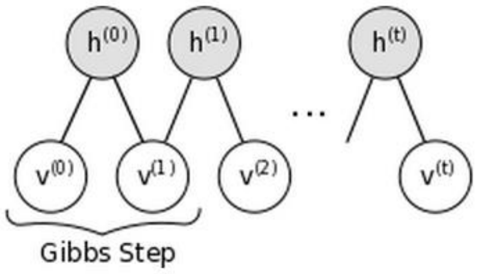
Go Up

- **for all** hidden units i do

  Compute $Q(h_{0i}|v_0) = \mathrm{sigm}(b_i + \sum_j W_{ij} v_{0j})$ (for binomial units)

  Sample h_{0i} from $Q(h_{0i}|v_0)$

- **end for**



Go Down

- **for all** hidden units j do

  Compute $\mathrm{P}(v_{1j}|h_0) = \mathrm{sigm}(c_j + \sum_i W_{ij} h_{0i})$ (for binomial units)

  Sample v_{1j} from $\mathrm{P}(v_{1j}|h_0)$

- **end for**

Go Up

- **for all** hidden units i do

  Compute $Q(h_{0i}|v_0) = \mathrm{sigm}(b_i + \sum_j W_{ij} v_{0j})$ (for binomial units)

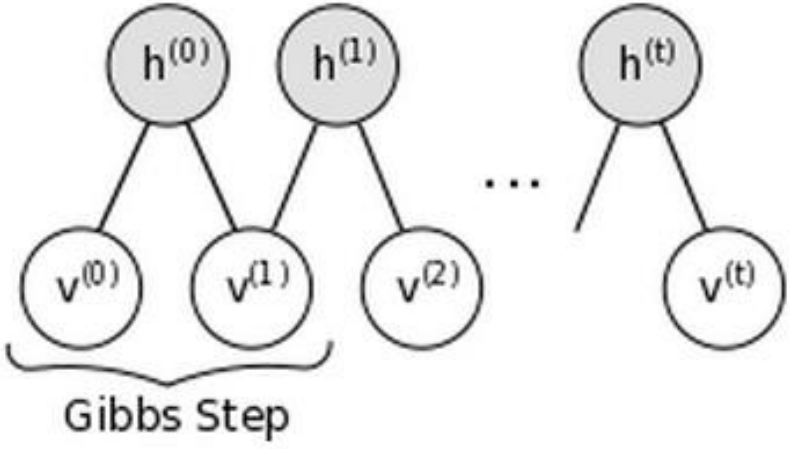
  Sample h_{0i} from $Q(h_{0i}|v_0)$

- **end for**

Algorithm1 RBMupdate (v_0, ϵ, W, b, c)

- $W \leftarrow W - \epsilon(h_0 v_0' - Q(h_1 = 1|v_1)v_1')$
- $b \leftarrow b - \epsilon(h_0 - Q(h_1 = 1|v_1))$
- $c \leftarrow c - \epsilon(v_0 - v_1)$

Update model parameters



Feature representation

Contrastive Divergence

Algorithm2 PreTrainDBN $(x, \epsilon, L, n, W, b)$

- Initialize $b^0 = 0$
- **for** $l = 1\ to\ L$ do

  Initialize $W^l = 0, b^l = 0$

  while not stopping criterion do

  $g^0 = x$

  for $i = 1\ to\ l - 1$ do

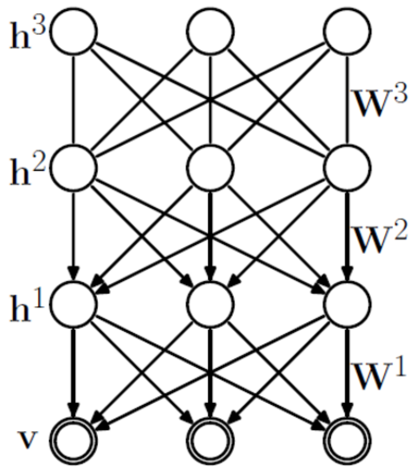
  Sample g^i from $Q(g^i | g^{i-1})$

  end for

  RBMupdate $(g^{l-1}, \epsilon, W^l, b^l, b^{l-1})$

  end while
- **end for**

Stacked RBMs!



Unsupervised Learning

Learn Latent Variables (Higher level Feature Representations)

Algorithm3 FineTuneDBN $(x, y, \epsilon, L, n, W, b)$

- $\mu^0(x) = x$
- **for** $l = 1\ to\ L$ do

$$\mu^l(x) = \mathrm{E}\left[g^i \middle| g^{i-1} = \mu^{l-1}(x)\right]$$
$$= \mathrm{sigm}(b_j^l + \sum_k W_{jk}^l\, \mu_k^l(x))$$ (for binomial units)

- **end for**
- Network output function: $f(x) = V(\mu^l(x)', 1)'$
- Use Stochastic Gradient Descent to iteratively minimize cost function $C(f(x), y)$ (Back Propagation)

Supervised Learning

Fine tune Model parameters

Learning Deep Belief Network

Step1:

Unsupervised generative pre-training of stacked RBMs (Greedy layer wise training)

Step2:

Supervised fine-tuning (Back Propagation)

How many parameters to learn?

$$\sum_{i=0}^{L} n_i n_{i+1} + \sum_{i=0}^{L+1} n_i$$

Can We SCALE UP?

- Deep learning methods have higher capacity and have the potential to model data better.

- More features always improve performance unless data is scarce.

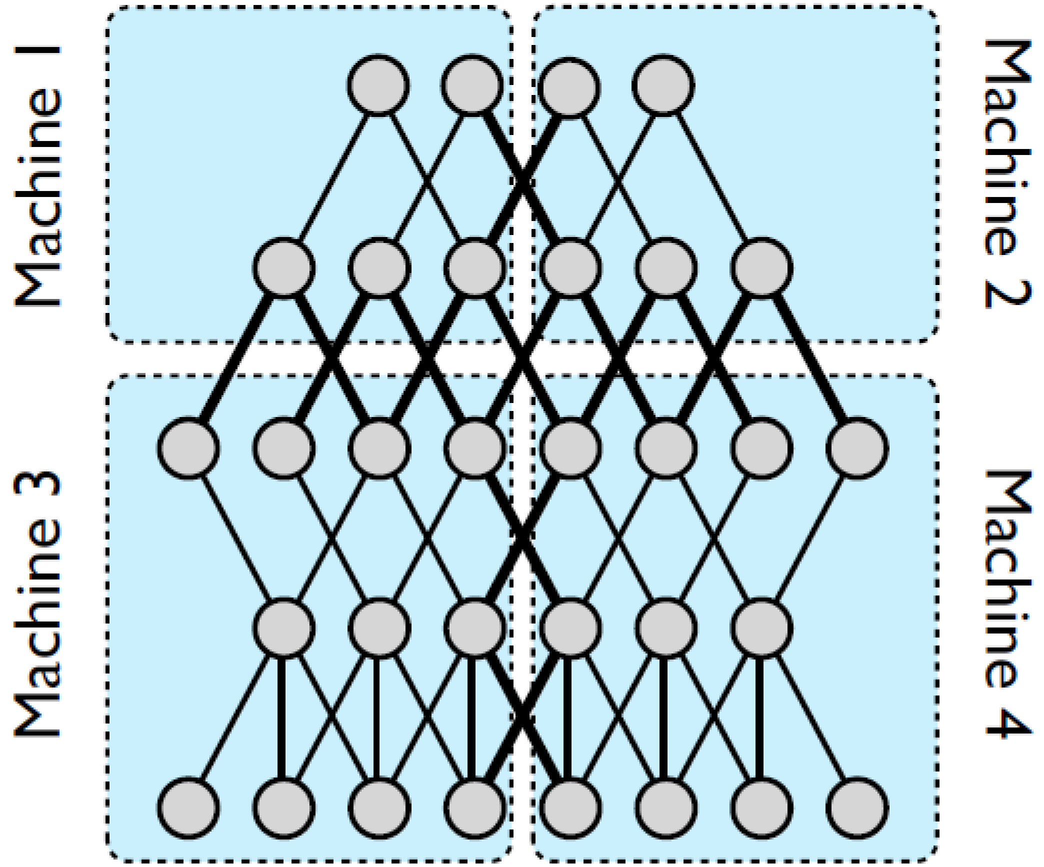
- Given lots of data and lots of machines, can we scale up deep learning methods?

MapReduce?	No!
MPI?	Yes!

Model Parallelism

- For large models, partition the model across several machines.
- Models with local connectivity structures tend to be more amenable to extensive distribution than fully-connected structures, given their **lower communication costs**.
- Need to manage **communication**, **synchronization**, and **data transfer** between machines.

Reference Implementation:
Google DistBelief



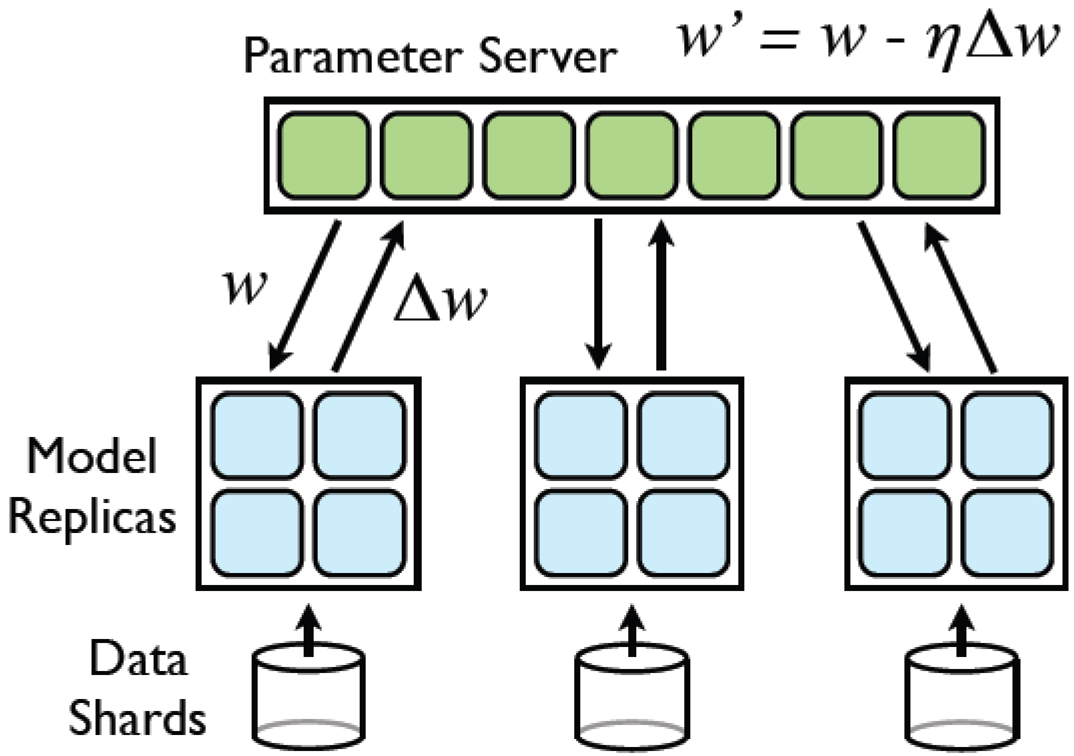
Data Parallelism (Asynchronous SGD)

Fetch parameters
- Before processing each batch, a model replica asks the Parameter Sever for an updated copy of its model parameters;

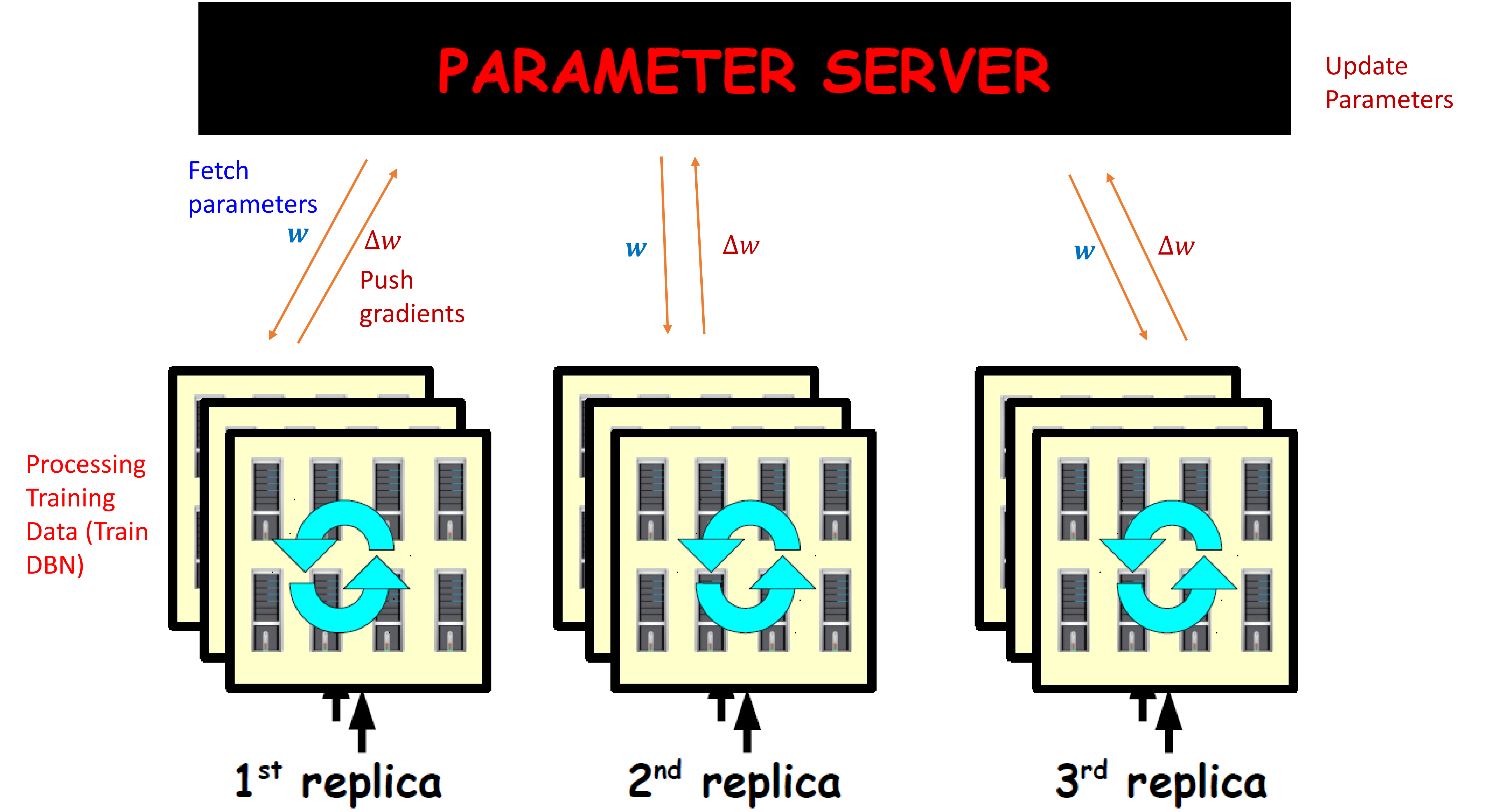
Train DBN
- Compute a parameter gradient.

Push gradients
- Send the parameter gradient to the server.

Parameter Sever applies the gradient to the current value of the model parameters.



Divide the data into a number of subsets and run a copy of the model on each of the subsets.


PARAMETER SERVER
Update Parameters
Fetch parameters
w
Δw
Push gradients
w
Δw
w
Δw
Processing Training Data (Train DBN)
1st replica
2nd replica
3rd replica

Algorithm4 Asynchronous SGD(α)

- **Procedure** *StartFetchingParameters*($parameters$)

  $parameters \leftarrow$ *GetParametersFromParameterSever();* MPI_Get(parameters, … , win);

- **Procedure** *StartPushingGradients*($gradients$)

  SendGradientsToParameterSever(gradients); MPI_Put(parameters, … , win);

- **Main**

  Global $parameters, gradients$

  while not stopping criterion **do**

  StartFetchingParameters($parameters$)

  $data \leftarrow$ *GetNextMiniBatch()*

  $gradients \leftarrow ComputeGradient(parameters, data)$ Train DBN
  Takes lots of time

  $parmeters \leftarrow parameters + \alpha * gradients$ Update the parameters

  StartPushingGradients($gradients$)

  end while

Experiment

➢MNIST Handwritten Dataset

- $28 \times 28 = 784\, pixels$
- 60000 training images
- 10000 test images

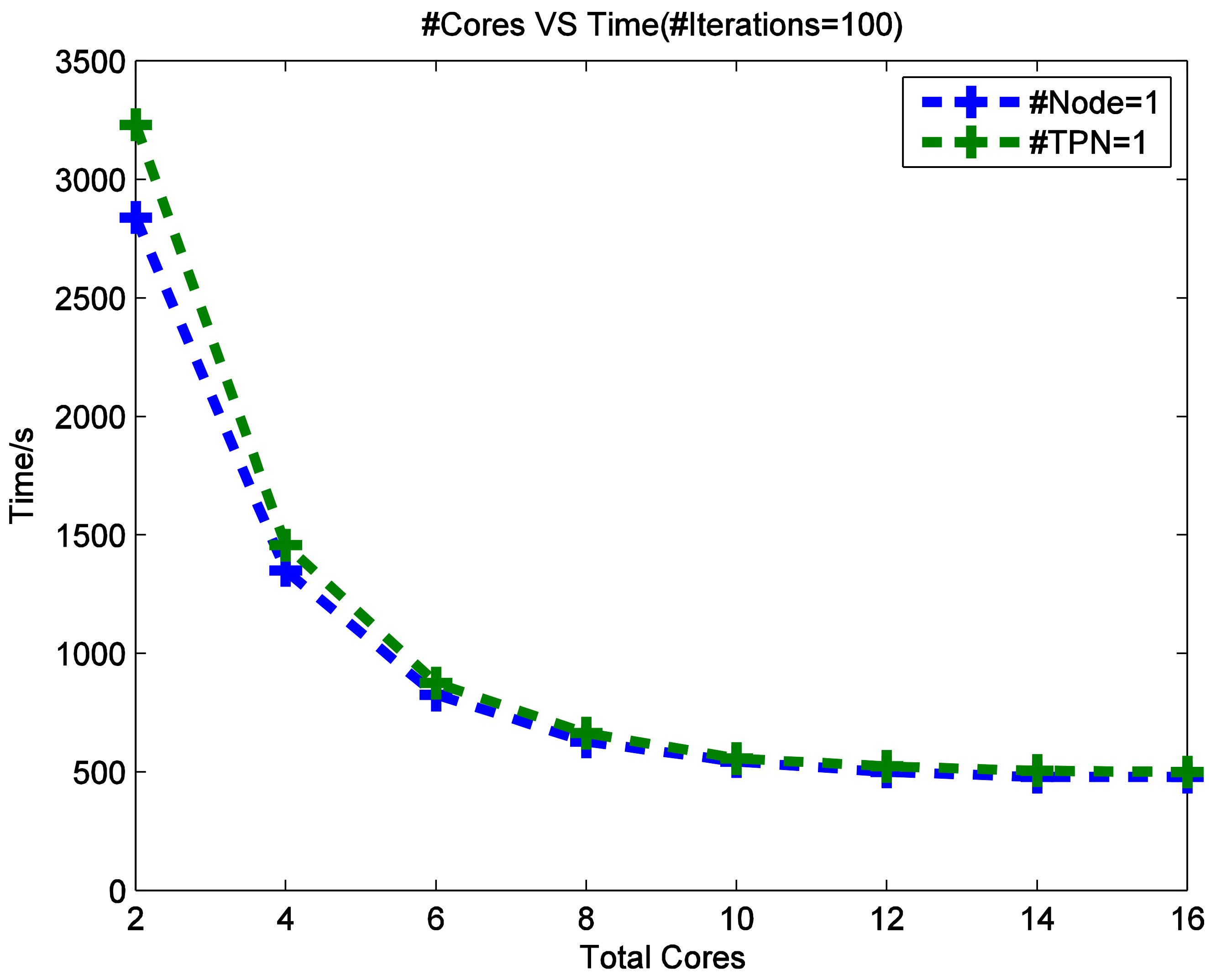
➢Partition the training data into data shards for each model replica for parallelism.

Cores VS Time (#Iterations = 100)

- Equally divide the training data into #Total Cores partitions (**Balanced Partitions**).

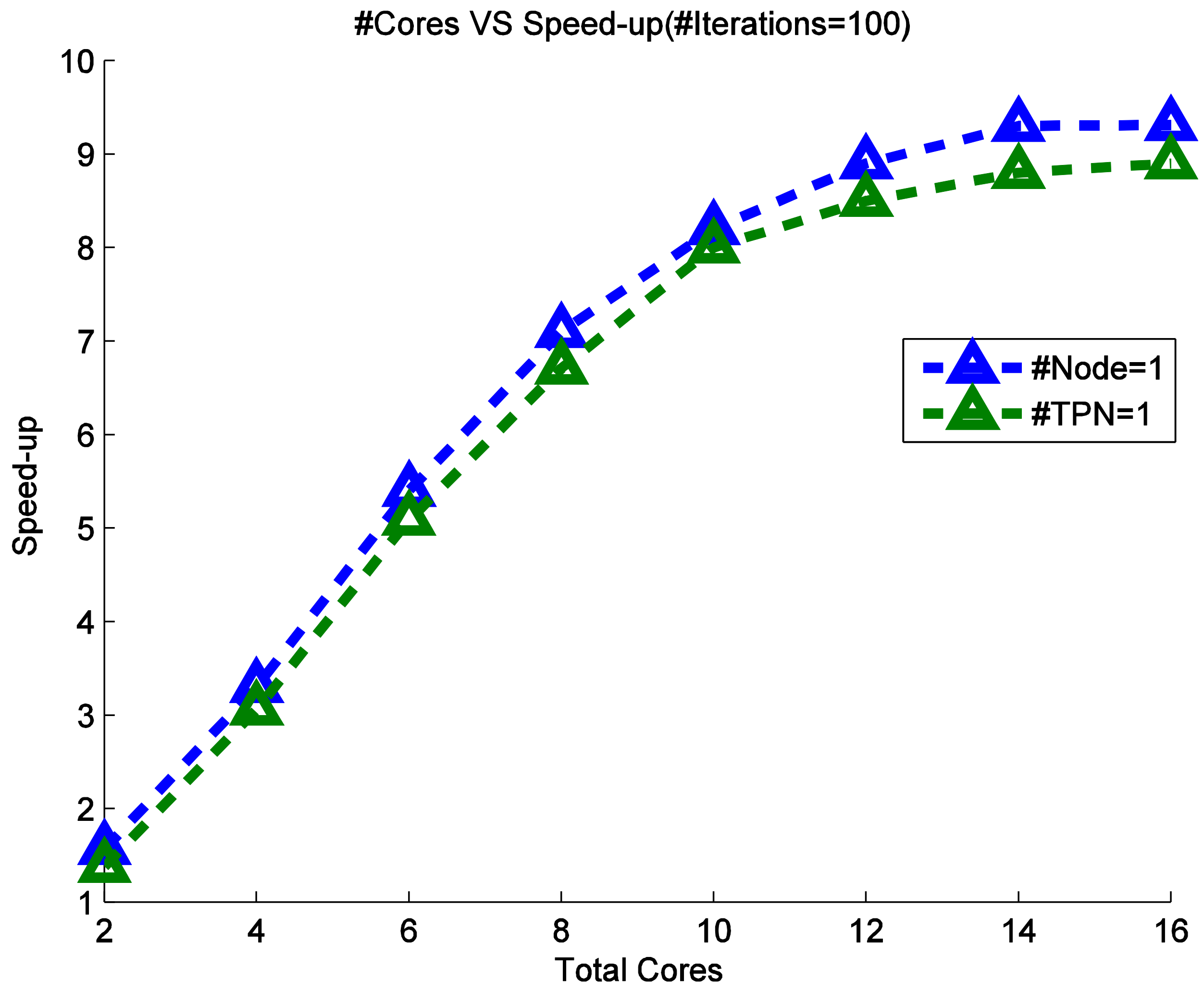
- The smaller training data, the less training time which is dominate in the total time.

- After about **10** partitions, **the training data is small enough, the training time is not dominate in the total time, so the speed-up is not increasing linearly.**



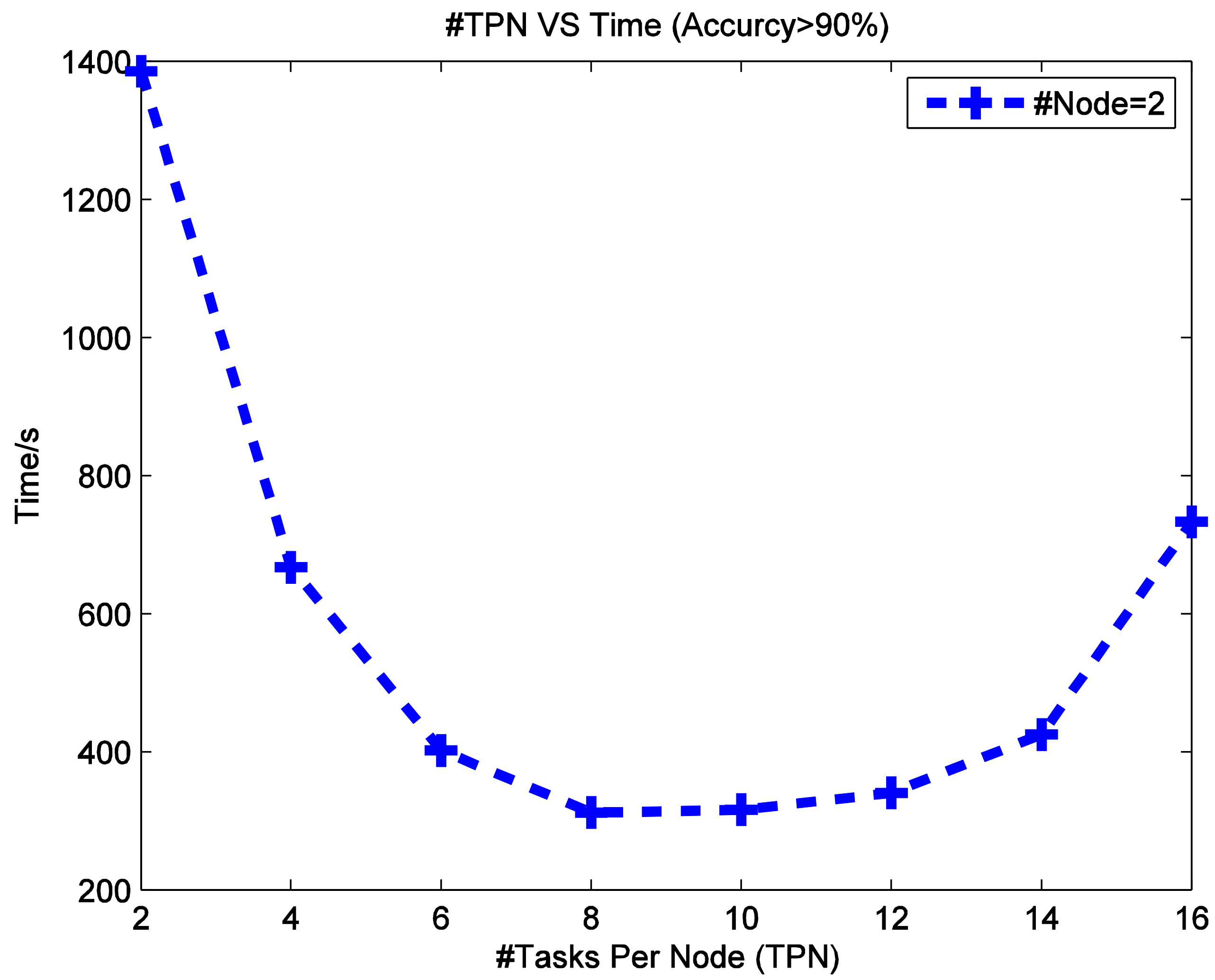
Cores VS Speed-up (#Iterations = 100)

- Equally divide the training data into #Total Cores partitions (**Balanced Partitions**).
- The smaller training data, the less training time which is dominate in the total time.
- After about **10** partitions, **the training data is small enough, the training time is not dominate in the total time, so the speed-up is not increasing linearly.**



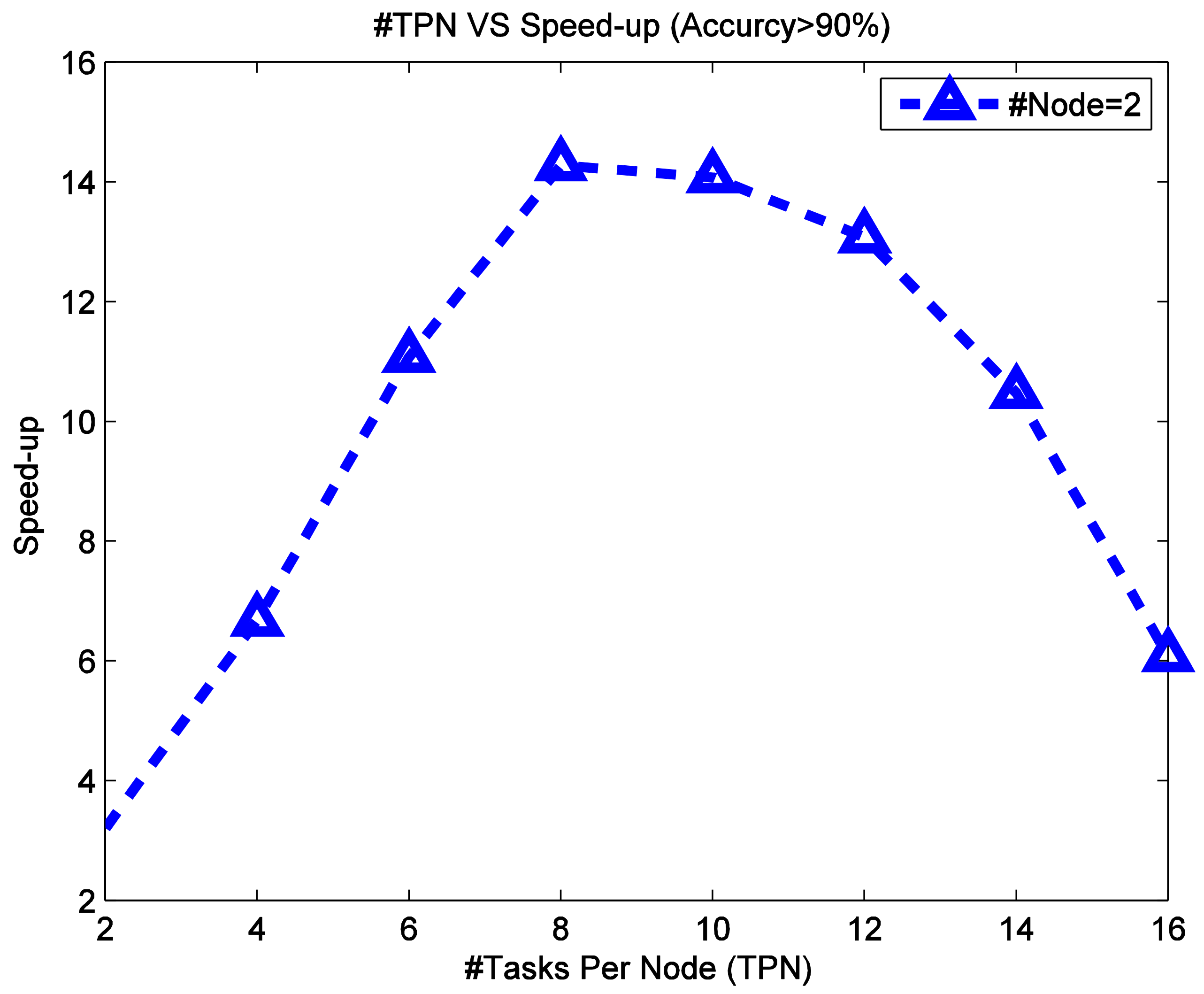
Fixing #Node= 2, Accuracy > 90%

- ➢ Time begins increasing after about 20 data partitions.
- ➢ Reason:
- **Data partition becomes too small and insufficient to learn the model parameters.**
- **So it needs more iterations to get the same accuracy.**



Fixing #Node= 2, Accuracy > 90%

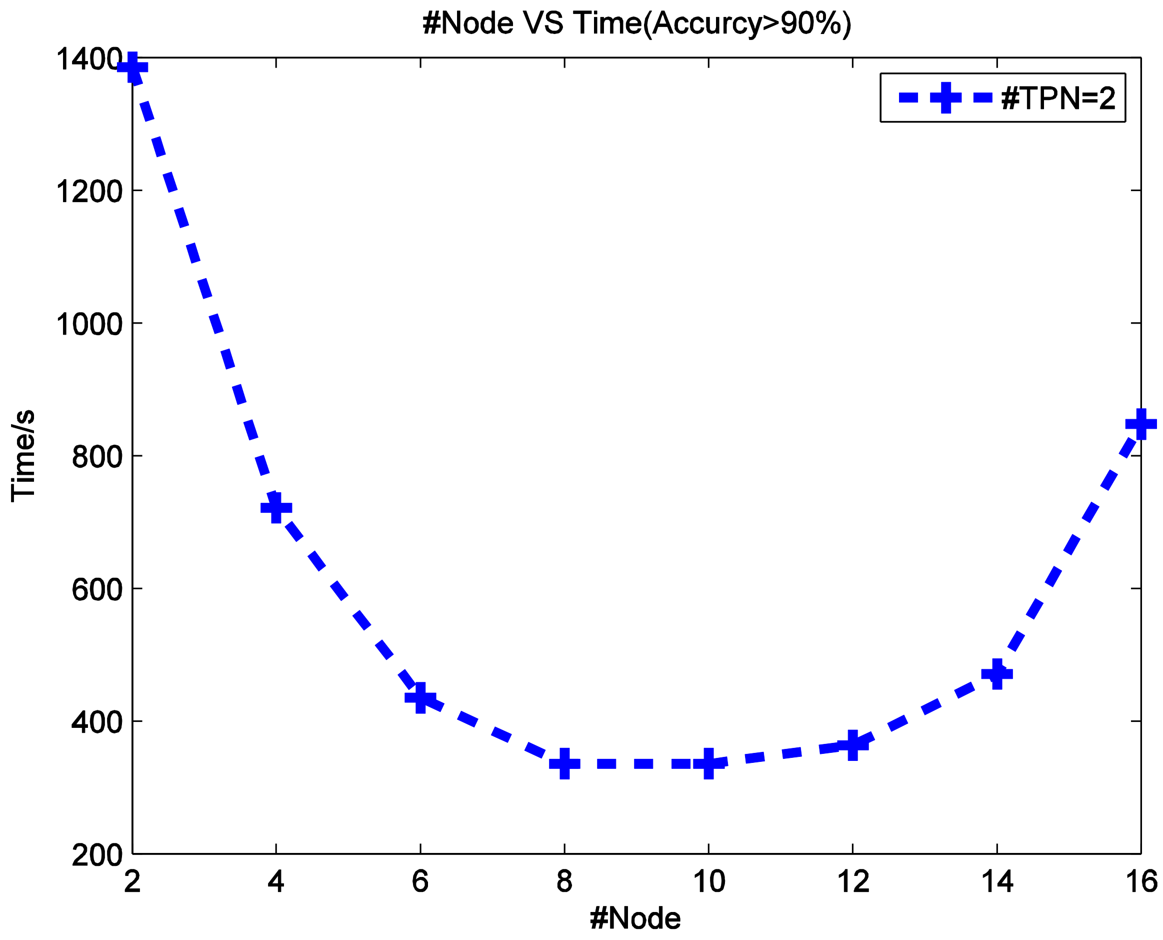
- ➢ Speed-up begins decreasing after about 20 data partitions.
- ➢ Reason:
- **Data partition becomes too small and insufficient to learn the model parameters.**
- **So it needs more iterations to get the same accuracy.**



Fixing #Tasks Per Node (TPN) = 2

- Time begins increasing after about 20 data partitions.
- Reason:
- **Data partition becomes too small and insufficient to learn the model parameters.**
- **So it needs more iterations to get the same accuracy.**

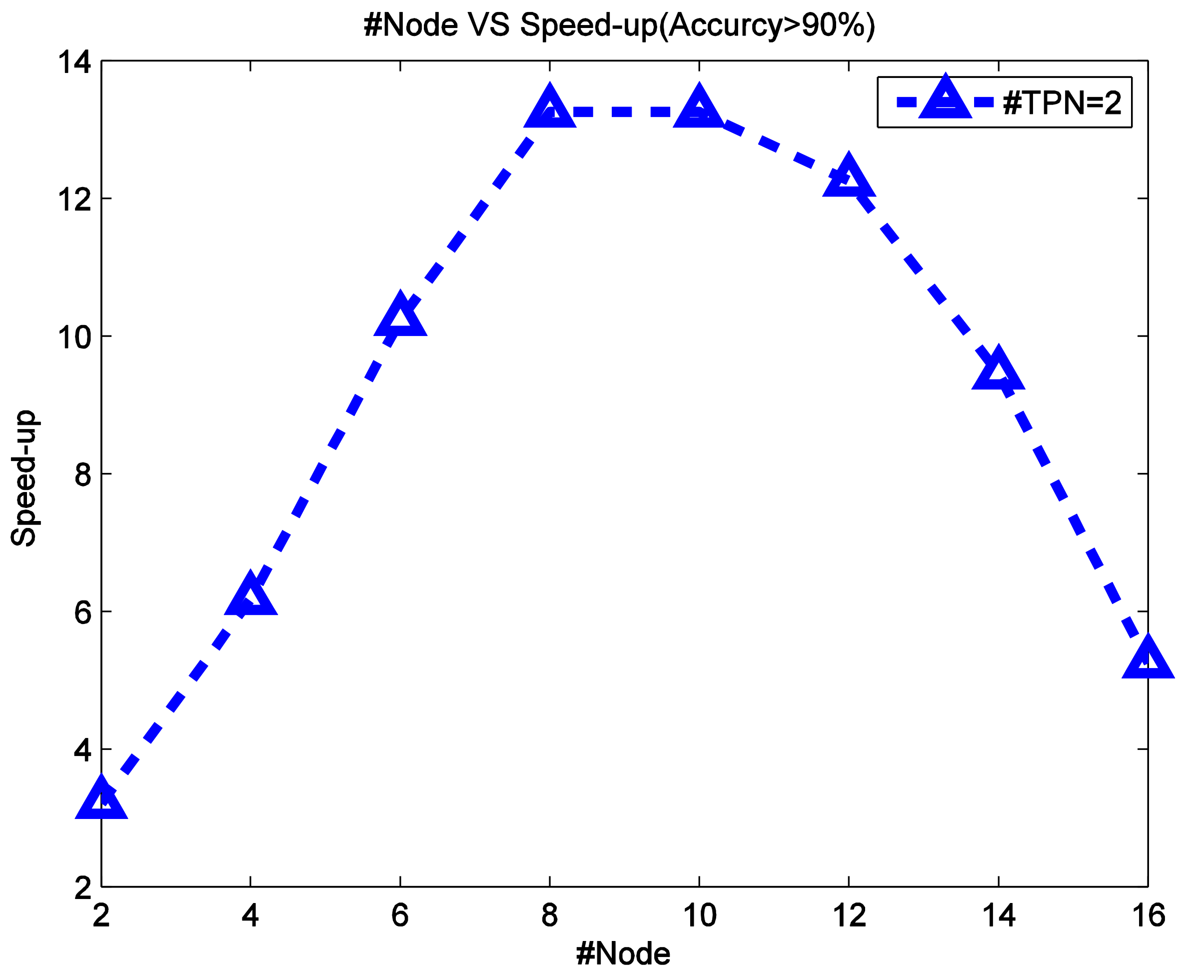
Similar to the results of fixing #Nodes



Fixing #Tasks Per Node (TPN) = 2

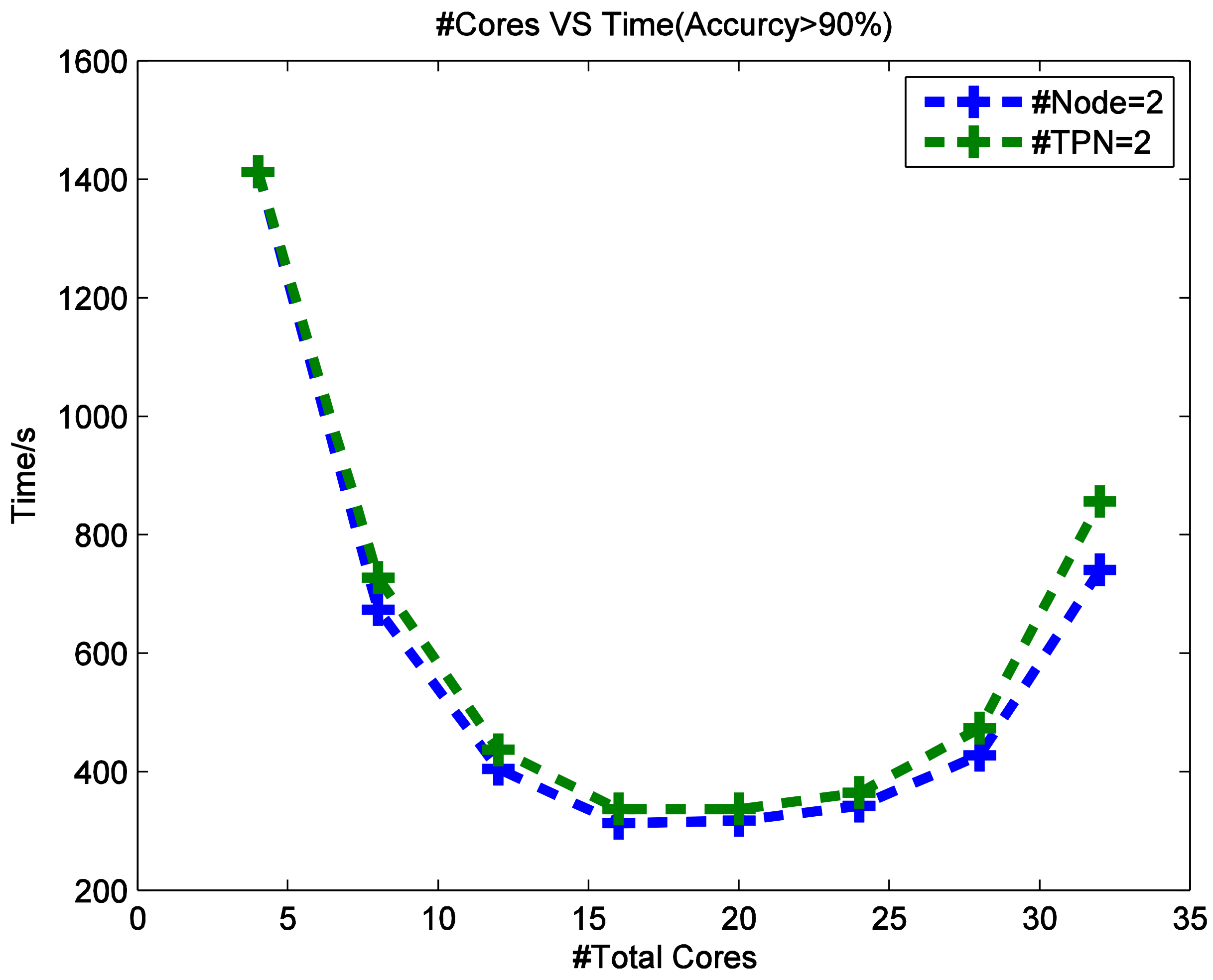
- Speed-up begins decreasing after about 20 data partitions.
- Reason:
  - **Data partition becomes too small and insufficient to learn the model parameters.**
  - **So it needs more iterations to get the same accuracy.**

Similar to the results of fixing #Nodes, slightly different.



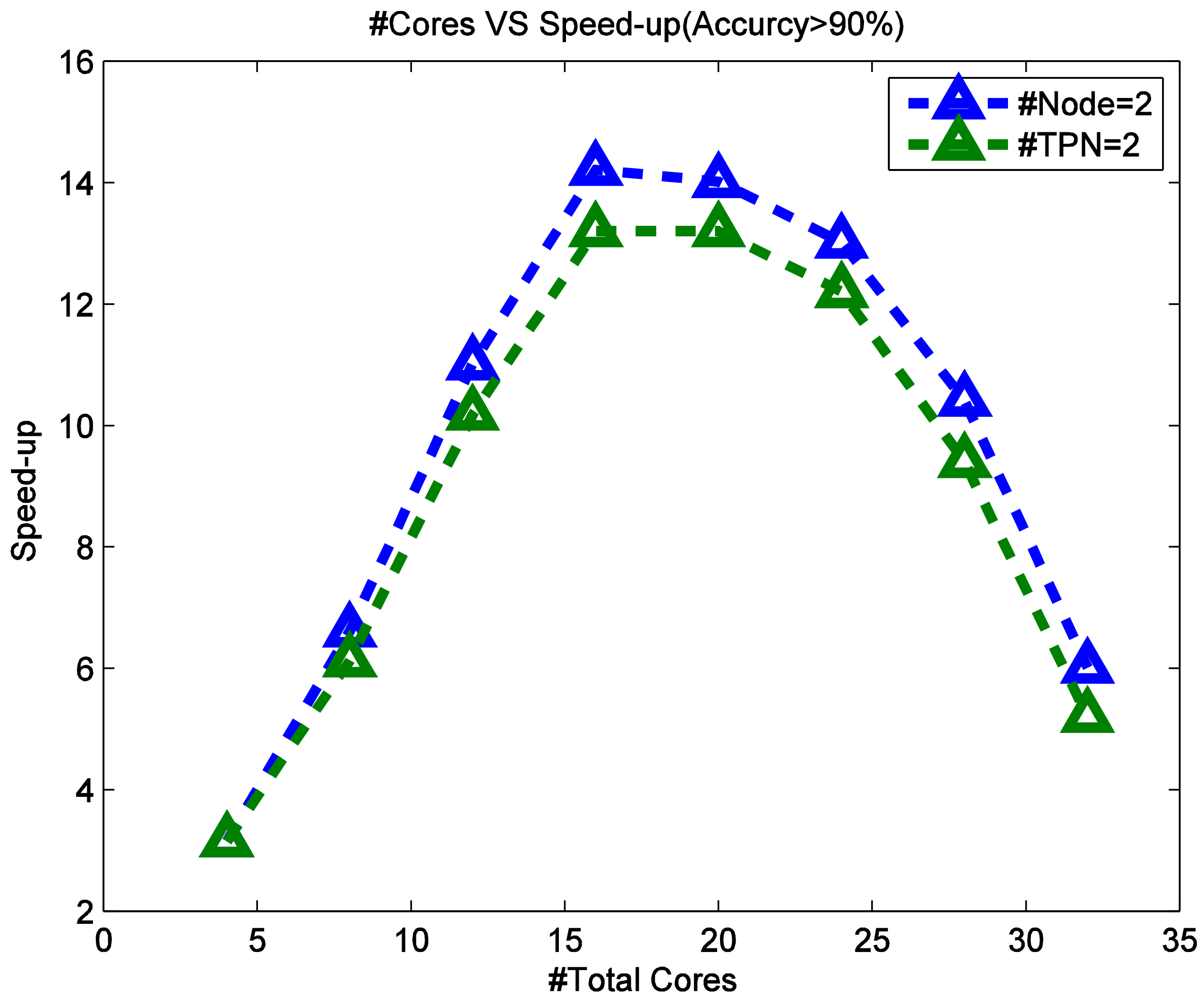
#Total Cores VS Time (Accuracy > 90%)

- Time begins increasing after about 20 data partitions.
- Reason:
  - **Data partition becomes too small and insufficient to learn the model parameters.**
  - **So it needs more iterations to get the same accuracy.**
- **Inter-communication** cost is higher than **intra-communication** cost.



#Total Cores VS Speed-up (Accuracy > 90%)

- Speed begins decreasing after about 20 data partitions.
- Reason:
  - **Data partition becomes too small and insufficient to learn the model parameters.**
  - **So it needs more iterations to get the same accuracy.**
- **Inter-communication** cost is higher than **intra-communication** cost.



Results

Table1 Fixing Nodes

#Node	#TPN	#Total Cores	Time/s	Speed-up
2	2	4	1413	3.1552
2	4	8	674	6.6152
2	6	12	405	11.0152
2	8	16	314	14.2152
2	10	20	318	14.0152
2	12	24	342	13.0152
2	14	28	428	10.4152
2	16	32	741	6.0152

Table2 Fixing Tasks Per Nodes (TPN)

#Node	#TPN	#Total Cores	Time/s	Speed-up
2	2	4	1411	3.1552
4	2	8	728	6.1254
6	2	12	438	10.1854
8	2	16	337	13.2054
10	2	20	337	13.2054
12	2	24	365	12.2054
14	2	28	474	9.4054
16	2	32	856	5.2054

Inter-communication costs between nodes are higher than intra-communication costs between nodes.

Conclusion

- There is a tradeoff between communication costs and computation costs. Inter-communication costs > Intra-communication costs

- When each data partition is big, the training time of DBN dominates. The speed-up on CCR using MPI is approximately linear.

- When the partition becomes small enough, it's insufficient to train sophisticated DBN model. To achieve certain accuracy, it needs more iterations. The performance could become significant worse when the partition is too small. It depends on the datasets. The bigger dataset, the more amenable to extensive distribution and the more obvious speed-up.

- In general, using MPI framework to distribute large deep neural network is a good choice. The efficiency and scalability have been proved in industrial practice.

Reference

- Russ Miller, Laurence Boxer. *Algorithms Sequential & Parallel: A Unified Approach, 3rd edition*, 2012

- Marc Snir, Steve Otto, etc. *MPI – The Complete Reference, 2nd Edition*, 1998

- Jeffrey Dean, Greg S. Corrado, etc. *Large Scale Distributed Deep Networks*. NIPS, 2012

- Yoshua Bengio. *Learn Deep Architecture for AI*. Foundations and Trends in Machine Learning, 2009

- *http://deeplearning.net/tutorial/*


Thank You
for Your
Time