Making Super-Large Scale Machine Learning Possible

Tie-Yan Liu
Principle Researcher/Research Manager
Microsoft Research Asia
Era of Big Data and Big Model

- Search engine index: $10^{10}$ pages ($10^{12}$ tokens)
- Search engine logs: $10^{12}$ impressions and $10^9$ clicks every year

- Social networks: $10^9$ nodes and $10^{12}$ edges

- Peacock: LDA with $10^5$ topics ($10^{10}$ parameters); More topics $\rightarrow$ better performance in click predictions

- DistBelief: DNN with $10^{10}$ weights; Deeper and larger networks $\rightarrow$ better performance with sufficient training data.

- Human brain: $10^{11}$ neurons and $10^{15}$ connections, much larger than any existing ML model.
Existing Approach to Big Machine Learning

- Parallelization of existing machine learning algorithms using either MapReduce or Parameter Server

### Iterative MapReduce / AllReduce
- Only synchronous updates (BSP, MA, ADMM), poor efficiency on heterogeneous clusters
- Only data parallelism, cannot handle big models

### Parameter Server
- Support asynchronous updates; better efficiency on heterogeneous clusters
- Support model parallelism, but inefficient, especially on heterogeneous clusters.
- Only support fixed-structure models
- “sum”, “average”, and “addition” as atomic aggregation operations
Iterative MAP-Reduce
BSP, ADMM and Model Average

\[
\min_w \sum_{i=1}^{N} L_i(w)
\]

\[
w_t^{t+1} = w_t + \sum \Delta w_i^t
\]

\[
\Delta w_i^t = \eta_t \nabla L_i(w_t^t)
\]

\[
w_t^t = w_t
\]

\[
\min_w \sum_{i=1}^{N} L_i(w)
\]

\[
\text{s.t. } w_i - z = 0, i = 1, \ldots, N
\]

\[
w_t^{t+1} = \arg \min_{w_i} \left\{ \sum_{i} (L_i(w_i) + (\lambda_i^t)^r(w_i - z^t) + \frac{\rho}{2} \|w_i - z^t\|_2^2) \right\}
\]

\[
z^{t+1} = \frac{1}{N} \sum_{i=1}^{N} w_i^{t+1}
\]

\[
w_i^{t+1} = z^{t+1}
\]

\[
\lambda_i^{t+1} = \lambda_i^t + \rho (w_i^{t+1} - z^{t+1})
\]
Parameter Server
ASP: Asynchronous Parallel

Workers push update to parameter server and/or pull latest parameter back

1. $\Delta \omega$
2. $\omega$

Worker 1
Worker 2
Worker 3
Worker 4
SSP: Stale Synchronous Parallel

Workers push update to parameter server and/or pull latest parameter back. When staleness=4, worker 3 will wait here for worker 1 to catch up.

Finished Iteration# = 2

Finished Iteration# = 6
Model Parallelism
Limitations of Existing Approaches

• Scalability
  • Hard to train a topic model with millions of topics, or a DNN model with trillions of weights.

• Efficiency
  • 2+ days for 3000 CPU cores to finish the training of Peacock LDA.
  • 3 days for 16,000 CPU cores to finish the training of DistBelief DNN.

• Flexibility
  • Not many other big models beyond LDA and DNN were extensively studied in the literature.
Desirable System for Big Machine Learning

- Web data (trillions of tokens)
- Click logs (trillions of impressions)
- Social networks (trillions of edges)

Data Scalability

- LDA (millions of topics)
- CNN (trillions of activations)
- DNN (trillions of edge weights)
- Word embedding (millions of words)

Model Scalability

- Gradient boosting trees
- Decision trees / Random forest
- Ensemble models

Flexibility

- Almost linear speed up, even on heterogeneous clusters
- Reasonable training time even for big data and big model

Efficiency
How to Achieve It?

Algorithmic Innovation
• Machine learning algorithms themselves need to have sufficiently high efficiency and throughout.
• Existing design/implementation of machine learning algorithms might not have considered this request; redesign/re-implementation might be needed.

System Innovation
• One needs to leverage the full power of distributed system, and pursue almost linear scale out/speed up.
• New distributed training paradigm needs to be invented in order to revolve the bottle neck of existing distributed machine learning systems.
Case Studies

• **LightLDA**: Highly efficient LDA algorithm (with $O(1)$ amortized per-token sampling complexity) by using multiplicative factorization.

• **Distributed Word Embedding**: Highly scalable word embedding algorithm by using histogram-based data sampler.
Latent Dirichlet Allocation (LDA)

For document $d$, sample a topic distribution $\theta_d$ from a Dirichlet distribution with parameter $\alpha$.

Sample a word distribution $\varphi_k$ for each topic $k$ from a Dirichlet distribution with parameter $\beta$.

For each token $i$ in document $d$

- Sample a specific topic $z_{di}$ from topic distribution $\theta_d$.
- Sample a word from word distribution $\varphi_{z_{di}}$.

[Blei, et al. 2003]
Collapsed Gibbs Sampling

• Sampling from a closed-form conditional probability of topics, by integrating out $\theta$ and $\varphi$:

$$p(k) = p(z_{di} = k|\text{rest}) \propto \frac{n_{kw}^{-di} + \beta_w}{n_{k}^{-di} + \bar{\beta}} (n_{kd}^{-di} + \alpha_k)$$

- $n_{k}^{-di}$: number of tokens assigned to topic $k$ (excluding $z_{di}$, $w_{di}$ in count)
- $n_{kw}^{-di}$: number of tokens with word $w$ assigned to topic $k$ (excluding $z_{di}$, $w_{di}$ in count)
- $n_{kd}^{-di}$: number of tokens in document $d$ assigned to topic $k$ (excluding $z_{di}$, $w_{di}$ in count).

Per-token sampling complexity proportional to the number of topics: $O(K)$, thus hard to scale up to large number of topics.
Reduce Complexity by Amortizing Computations

**Alias Table** [Walker, 1977]
- Build alias table for some terms in $p(k)$ and reuse it across many tokens (introducing approximation error)

**Metropolis Hastings** [Hastings, 1970]
- Handle approximation error using a rejection procedure.
  - Given original $p(k)$ and its approximation $q(k)$
  - Sample according to $q(k)$ followed by a rejection procedure based on the difference between $q(k)$ and $p(k)$
    - $r \sim U(0,1)$, $s \rightarrow t$
    - Accept $t$ as next state if $r < \min\left\{1, \frac{p(t)q(s)}{p(s)q(t)}\right\}$.
  - Stationary distribution of the above Markov chain is exactly $p(k)$; mixing rate depends on the difference between $p(k)$ and $q(k)$.

**Alias Table construction**: transform non-uniform distribution to uniform in $O(K)$ time; sample from uniform distribution in $O(1)$ time.
## Amortizability

<table>
<thead>
<tr>
<th>Terms</th>
<th>$n_{kd}$</th>
<th>$n_{kw}$</th>
<th>$n_{kd} \cdot n_{kw}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias table construction</td>
<td>For each document $d$, in $O(L_d)$ time</td>
<td>For each word, in $O(KV)$ time</td>
<td>For each document and word, in $O(L_d V)$ time</td>
</tr>
<tr>
<td>Reused for</td>
<td>Only tokens in document $d$</td>
<td>All documents</td>
<td>Only tokens in document $d$</td>
</tr>
<tr>
<td>Amortized O(1)?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
SparseLDA  [Yao, et al. 2009]

• Decompose \( p(k) \) into additive terms, then sample the terms using the mixture approach

\[
p(z_{di} = k|\text{rest}) \propto \frac{\alpha_k \beta_w}{n^+_k + \beta} + \frac{n^+_{kd} \beta_w}{n^+_k + \beta} + \frac{n^+_{kw}(n^+_{kd} + \alpha_k)}{n^+_k + \beta}
\]

Amortizable \( \Rightarrow O(1) \)

Unamortizable but sparse \( \Rightarrow O(K_w) \)

Per-token complexity: \( O(K_w) \ll O(K) \)

\( K_w \): number of topics word \( w \) belongs to

Non-zero elements in word-topic table \( \{n^+_{kw}\} \)
AliasLDA  [Li, et al. 2014]

• Decompose $p(k)$ into additive terms, then sample the terms using the mixture approach

\[
p(z_{di} = k | \text{rest}) \propto \frac{n_{kd}^{-di} (n_{kw}^{-di} + \beta_w)}{n_k^{-di} + \beta} + \frac{\alpha_k (n_{kw}^{-di} + \beta_w)}{n_k^{-di} + \beta}
\]

Unamortizable but sparse $\Rightarrow O(K_d)$
Amortizable $\Rightarrow O(1)$

Non-zero elements in doc-topic table $\{n_{kd}^{-di}\}$

Per-token complexity: $O(K_d) \ll O(K_w) \ll O(K)$

$K_d$: number of topics document $d$ contains
LightLDA

• Factorize $p(k)$ into *multiplicative* terms, instead of decomposing it into *additive* terms
  • Separate $n_{kd}^{-di}$ and $n_{kw}^{-di}$ into different terms, so as to avoid the issue of unamortizability.
  • All terms after factorization only contain either $n_{kd}^{-di}$, $n_{kw}^{-di}$, or constant, thus a $O(1)$ sampling complexity can be achieved by Alias and MH methods.

• The mixture approach does not naturally work for multiplicative factorization - we use a cycling approach instead.

[Yuan, et al. 2015]
Multiplicative Factorization

\[ p(z_{di} = k|rest) \propto \frac{n_{kw}^{-di} + \beta_w}{n_k^{-di} + \bar{\beta}} \left( n_{kd}^{-di} + \alpha_k \right) \]

Amortizable: \( O(1) \)

Per-token complexity: \( O(1) \)

Other tricks: (1) sparsified alias table to further reduce the sampling complexity of \( p_1(k) \); (2) fully leverage in-memory intermediate result to simply the sampling complexity of \( p_2(k) \).
**Experimental Results (Single-core)**

With a single core only, LightLDA uses 20 hours to train 10K topics from ~1B tokens (PubMed). With a commodity machine of 20 cores, LightLDA can finish training in 2 hours. This single-machine capability is equivalent to (if not beyond) a medium-size cluster of SparseLDA or AliasLDA.

LightLDA achieves **better** log-likelihood than SparseLDA and AliasLDA in **much shorter time**!

NYTimes Dataset
- 300K documents

PubMed Dataset
- 8.2M documents
- 738M tokens
Case Studies

• **LightLDA**: Highly efficient LDA algorithm (with O(1) amortized per-token sampling complexity) by using multiplicative factorization.

• **Distributed Word Embedding**: Highly scalable word embedding algorithm by using histogram-based data sampler.
Word Embedding

Native Discrete Representation

**Word:** 1-of-N vector

\[
\{w_1, w_2, \ldots, w_i, \ldots, w_{N-1}, w_N\} \\
\{0, 0, \ldots, 1, \ldots, 0, 0\}
\]

Training data: text corpus
Sliding window

... as a candidate for the Illinois state senate Obama had ...

\[w_{i-2} w_{i-1} w_{i} w_{i+1} w_{i+2}\]

- \(V:\) vocabulary size
- 1-of-V word representation: \(w_{i+2} = (0, \ldots, 0, 1, \ldots, 0)^T\)

Natural Languages

Representations in Continuous Space

Deep Learning

- State-of-the-art machine learning methods require data to be in a continuous space
- Continuous representation eases text understanding, inference, and reasoning
Word2Vec (Skip-Gram)

**Promising Accuracy on analogical reasoning**

- Evaluate linear regularity of word embedding, e.g., the *accuracy* of [China–Beijing+Tokyo] = [Japan]?

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Questions</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mikolov</td>
<td>19544</td>
<td>31.30%</td>
</tr>
</tbody>
</table>

- **Training data:** enwiki9
- **Dimension of word embedding:** 100
Training Word Embedding Using Entire Web

• Challenge: Web data are simply too large to copy, store, and process!

WIKIPEDIA

~ 1 billion words

Tier-0: ~15 trillion words
(~150 trillion word pairs, ~1PB data)

~1000 machines with 1TB disk are required to store training data; and
~5000 machines with 200GB memory to support in-memory training.
SGD Training for Word2Vec (Skip-Gram)

• Skip-gram training is based on stochastic gradient descent (SGD)
  • Read one word pair from the training corpus
  • Compute gradient for this pair, and update the model
  • Repeat this process until the model converges (after many epochs)

• SGD converges and is an unbiased estimate of gradient descent
  • When the training instances (word pairs) are i.i.d. sampled.
  • Under this assumption, only the distribution matters, but not necessarily the raw data set.
Histogram-based Sampler

• Obtain empirical distribution (word pair histogram) of the training corpus using MapReduce at the beginning of the training process.

• Train word embedding model using SGD, by sampling from the empirical distribution instead of the original text corpus, for an arbitrary number of epochs when needed.
Histogram Re-shape

- Smoothed histogram to handle truncation bias in limited number of sampling

\[
\begin{align*}
\min & \quad \sum_i \frac{1}{H(w_i)} |\epsilon_i|^2 \\
\text{s. t.} & \quad H(w_i) + \epsilon_i \geq T, \quad \forall i. \\
& \quad \sum_i \epsilon_i = 0
\end{align*}
\]

Similar to original empirical histogram \( \text{(relative change is minimized)} \)

Each word has at least \( T \) counts

The total count remains unchanged

To satisfy hard constraint \( H(w_i) + \epsilon_i \geq T \), for those pairs whose \( H(w_i) < T \), the modification \( \epsilon_i \) is lower bounded and the minimization of the loss function will push \( \epsilon_i = T - H(w_i) \).

For those pairs whose \( H(w_i) \geq T \), the optimal solution \( \epsilon_i \) will be proportional to \( H(w_i) \), i.e.,

\[
\epsilon_i = \frac{-H(w_i)}{\sum_{H(w_i) \geq T} H(w_i)} \sum_{H(w_i) < T} T - H(w_i)
\]
Experimental Results

Accuracy Curve on Analogical Reasoning Task

Accuracy of Histogram Based Sampler
Accuracy of Standard WordVec
A New Distributed ML Framework

Multiverso Server

Efficiency - Hybrid Model Store
- Aggregation of model updates
- Send model to clients

Flexibility – Customizable Model Representation and Aggregations

Client updates
Client requests
Server responses
Message queue

U1 U2 G1 U5 G3 G2 ...

Communicator Thread

Update Parameter
Get Parameter

Multiverso Client

Efficiency – Automatic Pipelining
- Pre-fetch parameter for next data block
- Automatic pipelining

Scalability – Model Scheduling
- Aggregation of model updates
- Send model to clients

Parameter request
Server responses
Client updates
Update cache

Communicator Thread

Local Model Store

Intermediate Data store

Training threads

Pre-fetch parameter for next data block

D1 D2 D3 ...

Training queue

Client updates

Efficiency – Hybrid Model Store
- Aggregation of model updates
- Send model to clients

Flexibility – Customizable Model Representation and Aggregations

Client updates
Client requests
Server responses
Message queue

U1 U2 G1 U5 G3 G2 ...

Communicator Thread

Update Parameter
Get Parameter

Multiverso Server

Efficiency - Hybrid Model Store
- Aggregation of model updates
- Send model to clients

Flexibility – Customizable Model Representation and Aggregations

Client updates
Client requests
Server responses
Message queue

U1 U2 G1 U5 G3 G2 ...

Communicator Thread

Update Parameter
Get Parameter

Multiverso Client

Efficiency – Automatic Pipelining
- Pre-fetch parameter for next data block
- Automatic pipelining

Scalability – Model Scheduling
- Aggregation of model updates
- Send model to clients

Parameter request
Server responses
Client updates
Update cache

Communicator Thread

Local Model Store

Intermediate Data store

Training threads

Pre-fetch parameter for next data block

D1 D2 D3 ...

Training queue

Client updates

Efficiency – Hybrid Model Store
- Aggregation of model updates
- Send model to clients

Flexibility – Customizable Model Representation and Aggregations

Client updates
Client requests
Server responses
Message queue

U1 U2 G1 U5 G3 G2 ...

Communicator Thread

Update Parameter
Get Parameter
Scalability: Problem with Model Parallelism

- SGD-like algorithms require intermediate results for every data sample to be transferred between machines.

- High comm cost: huge intermediate data
  - LDA: $O(10^9)$
    - $10^6$ docs/data block $\times$ $10^3$ tokens/doc
  - CNN: $O(10^9)$
    - $10^2$ imgs/mini-batch $\times$ $10^5$ patches/img $\times$ 10 filters/patch $\times$ 10 layers

- Sensitive to comm delay & machine failure
  - Speed differences among machines $\rightarrow$ slow down training.
  - Machine failure $\rightarrow$ break down training.
Scalability: Tackle the Challenges

• Model parallelism might be necessary from system perspective
  • Ensure the same behavior of distributed training with single machine training

• However, it is not necessary from machine learning perspective
  • Machine learning is statistical: achieving similar results (in large probability) is enough, not necessarily preserving exactly the same behaviors.

• Our proposal
  • Change gradient descent to (block) coordinate descent
  • Allow one-round communication delay
Scalability: Model Scheduling

- Model slices are pulled from server and updated in a round robin fashion.

Server

Client

Parameters in the slice and hidden-node activations triggered by the slice are updated.

\[ w_{i,j} \in \text{slice}_1 \]

Other activations are retrieved from historical storage in local machine.

\[ w_{i,j} \in \text{slice}_2 \]

Intermediate Data

- Stochastic (Block) Coordinate Descent (SCD)

When updating Slice_1, previous information about Slice_2 is reused.

When updating Slice_2, previous information about Slice_1 is reused.

Timeline

2015/11/7

MLA 2015
Scalability: Model Scheduling

**Theoretical guarantee**
SCD and SGD have the similar convergence rate for $\lambda$-strongly convex problem; and both lead to local optima for non-convex problems.

**Practical efficiency**
- Lower comm cost (only model is transferred)

<table>
<thead>
<tr>
<th></th>
<th>Model Parallelism</th>
<th>Model Scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA</td>
<td>Data $\sim O(10^9)$</td>
<td>Model $\sim O(10^7)$</td>
</tr>
<tr>
<td>CNN</td>
<td>Data $\sim O(10^9)$</td>
<td>Model $\sim O(10^4)$</td>
</tr>
</tbody>
</table>

- Robust to comm delay & machine failure

<table>
<thead>
<tr>
<th></th>
<th>Model Parallelism</th>
<th>Model Scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updates</td>
<td>Synchronous</td>
<td>Asynchronous</td>
</tr>
</tbody>
</table>
Efficiency: Hybrid Model Store

Typical scenarios

- Huge sparse model
  - Example: topic model
  - Dense format is prohibitively large and unnecessary
- Screwed model access
  - Example: word embedding
  - 0.1% terms are used in 90% training samples

Goal: High memory usage + model access speed

Multi-tier storage

- Separate storage of terms with different access frequencies
- High cache hit rate
- Balance between memory usage and access speed

Hybrid Store format

- Frequent term → topic vector is sparse → Hash table O(K)
- Rare term → topic vector is dense → Dense Array O(1)
Efficiency: Adaptive Pipelining

- Adaptively determine the optimal setting to match learning algorithms, disk speed, CPU/GPU speed, and network speed.

Perfect pipelining:
\[
T_1 = T_2 = T_3 = T_4
\]

Adaptive pipelining:
\[
\min_{\Delta D, \Delta M, N} \sum_{i,j} |T_i - T_j|^2
\]

- Online algorithm to adjust \(\Delta D, \Delta M, N\).
- Efficient optimization since all \(T_i\)'s are monotone functions w.r.t. \(\Delta D, \Delta M, N\).

Cluster situations:
- Network speed: \(S_N\)
- CPU/GPU speed: \(S_C\)
- Disk speed: \(S_D\)

Multiverso settings
- Data block size: \(\Delta D\)
- Model slice size: \(\Delta M\)
- Number of threads: \(N\)

Load data block from disk
\[
T_1 = \Delta D / S_D
\]

Load model slice from server
\[
T_2 = \Delta M / S_N
\]

Send aggregated updates to server:
\[
T_4 = \Delta M / S_N
\]

Train on data block and generate updates for model slice:
\[
T_3 = f(\Delta M, \Delta D, S_C, N)
\]
Flexibility: Customizable Model Representation and Aggregations

• Beyond matrix-form models and sum/average aggregation operators.

```csharp
Interface IAggregation
{
    public bool Aggregate(void* models, enum agg_type);
}

Class ParallelModel: IAggregation
{
    public virtual bool Aggregate(void* models, void* inter_data, enum agg_type);
    private void* _models;//model parameters
    private void* _inter_data;//intermediate variables
}
//Pre-defined models data structure in Multiverso:
//Matrix (sparse/dense), Trees.

//Pre-defined aggregation operations:
//Weighted sum, Average, Voting, Max, Min, Histogram merge.
```

For DNN/Logistic Regression/LDA:
• models = (sparse) matrix
• agg_type = Sum/Average

For FastRank/Decision trees:
• models = trees (with split point information) + histogram
• agg_type = max info gain/histogram merge

For Ensemble Models:
• models = trees + (sparse) matrix + ...
• agg_type = voting/max/min/weighted sum

For other algorithms, one can implement their own model data structures and aggregation operators.
Flexibility: Plug-in Mode

- Scenario: existing codebase; model is dense and can fit into local machine memory.
- Examples: CNTK, CNN for image classification.

```
//Initial parameter server
print("Initializing parameter server ...");
_multiversoWrapper = new MultiversoWrapper();

_multiversoWrapper->Init(adapterID, strConfig);
print("Connected to the parameter server. \n");
Mode Test:

Get model from parameter server finished;

_getNet()->GetModelFromMultiverso();

for (int i = 0; i < EvalNumInBatch(); i++)
{   
        _convNet->GetProp(i, _test ? PASS_TEST : PASS_TRAIN);
        _convNet->GetCost(batchCost);
        
        if (_test)
        {
        _convNet->GetProp(PASS_TRAIN);
        _convNet->updateWeigths();
        
        if (i & _sendInterv == 0) _convNet->SubmitModelToMultiverso(); //submit the update parameter finished
        
        if (i & _getInterv == 0)
        _convNet->GetModelFromMultiverso(); //Get model from parameter server
        
        }
} //Training batch finished

if (_test)
{   
        //force to sync up in the last steps
        _convNet->SubmitModelToMultiverso();
        _convNet->GetModelFromMultiverso();
        
        }
```
Flexibility: Embedded Mode

- Scenario: model exceeds single machine memory; sparse model training (only a small subset of model parameters are used when training a data block)
- Examples: LightLDA, Word Embedding, Logistic Regression.

![Diagram of Multiverso architecture]

User needs to define:
- Data block & model slices
- Train logic for one data block
- Model parsing and update logics

Multiverso client manages the pipelining of the following procedures:
- Training threads to obtain model updates
- Parameter loading thread to fetch model slices
- Local aggregation thread to aggregate and send out updates

Project template integrated with Visual Studio to assist algorithm developer.
Record Breaking: Model Size & Training Speed

• Topic Models:

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Data Scale</th>
<th>Model Scale</th>
<th>#Core</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed LightLDA</td>
<td>$10^{11}$</td>
<td>$10^{13}$</td>
<td>384</td>
<td>60 hrs</td>
</tr>
<tr>
<td>Peacock LDA (Tencent)</td>
<td>$10^9$</td>
<td>$10^{10}$</td>
<td>3,000</td>
<td>50 hrs</td>
</tr>
<tr>
<td>Alias LDA (Google, Baidu, CMU)</td>
<td>$10^{10}$</td>
<td>$10^{10}$</td>
<td>10,000</td>
<td>70 hrs</td>
</tr>
</tbody>
</table>

• Word2vec:

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Data Scale</th>
<th>Model Scale</th>
<th>#Core</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Word Embedding</td>
<td>$10^{11}$</td>
<td>$10^{10}$</td>
<td>96</td>
<td>40 hrs</td>
</tr>
<tr>
<td>Word2Vec (Google)</td>
<td>$10^{11}$</td>
<td>$10^8$</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## Rich Learning Algorithms on Multiverso

### Our New Platform

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Model</th>
<th>Data</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LightLDA</strong></td>
<td>20M vocab, 1M topics (largest topic model)</td>
<td>200B tokens (Bing web chunk)</td>
<td>60 hrs on 24 machines (nearly linear speed-up)</td>
</tr>
<tr>
<td><strong>Word2Vec</strong></td>
<td>10M vocab, 1000 dim (largest word embedding)</td>
<td>200B samples (Bing web chunk)</td>
<td>40 hrs on 8 machines (nearly linear speed-up)</td>
</tr>
<tr>
<td><strong>GBDT</strong></td>
<td>3000 trees (120-node) (GBDT)</td>
<td>7M records (Bing HRS data)</td>
<td>3 hrs on 8 machines (4x of speed-up)</td>
</tr>
<tr>
<td><strong>LSTM</strong></td>
<td>20M parameters (4 hidden layer, LSTM)</td>
<td>375 hrs speech data (Win phone data)</td>
<td>11180 on 4 GPU (3.8x speed-up)</td>
</tr>
<tr>
<td><strong>CNN</strong></td>
<td>60M parameters (AlexNet)</td>
<td>2M images (ImageNet 1K dataset)</td>
<td>2 hrs on 16 GPU cards (12x speed-up)</td>
</tr>
<tr>
<td><strong>Online FTRL</strong></td>
<td>800M parameters (Logistic Regression)</td>
<td>6.4B impressions (Bing Ads click log)</td>
<td>2400s on 24 machines (12x speed-up)</td>
</tr>
</tbody>
</table>
Open Source

• Releasing to Github
  • https://github.com/Microsoft/multiverso
  • Containing a parameter server based framework, LightLDA and distributed word embedding

• Next steps:
  • Release more distributed machine learning algorithms, and new features of Multiverso.
Future Research

• Data exchange vs. model exchange
• Data server vs. parameter server
• Adaptive communication filters
• Automatic hyper-parameter tuning
• Machine learning for distributed machine learning
Thanks!

tyliu@microsoft.com
http://research.microsoft.com/users/tyliu/