Recent Advances in Distributed Machine Learning

Tie-Yan Liu, Wei Chen, Taifeng Wang
Microsoft Research
AI is Making Fast Progress!

**Speech Recognition**

- In 2016, Microsoft’s speech recognition system achieved human parity on conversational data (word error rate: 5.9%)
- This result was powered by Microsoft Cognitive Toolkit (CNTK).
AI is Making Fast Progress!

Image Classification

Object Segmentation

ImageNet Winners and Errors (%)

COCO Segmentation Accuracy (%)
AI is Making Fast Progress!

Machine Translation

![BLEU4 Chart]

*Chinese->English*

- 2012.08 - 2012.12 - 2013.05 - 2013.11 - 2014.04 - 2014.11 - 2015.05 - 2016.04

- Google
- Youdao
- Baidu
- Microsoft

Skype Translator

---

2017/2/2

AAAI 2017 Tutorial

4
AI is Making Fast Progress!

**Atari Games**

Deep Q-nets

**Go**

AlphaGo

**Texas Hold’Em**

Libratus
The Big Trend Contributing to AI

1. **Big Data**
   - Signals, Information & Knowledge
   - Digital Representation of the World

2. **Big Compute**
   - Cloud Computing
   - Internet of Things
   - CPU/GPU/TPU/FPGA

3. **Big Model**
   - Statistical machine Learning (e.g., deep neural networks)
   - Symbolic learning
   - Billions or even trillions of parameters

4. **Artificial Intelligence**
   - Digital Life/Work
   - New Form of HCI
   - Reinvent Productivity & Business Process
   - Personal Agent
Big Data

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

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Typical training data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image classification</td>
<td>Millions of labeled images</td>
</tr>
<tr>
<td>Speech recognition</td>
<td>Thousands of hours of annotated voice data</td>
</tr>
<tr>
<td>Machine translation</td>
<td>Tens of millions of bilingual sentence pairs</td>
</tr>
<tr>
<td>Go playing</td>
<td>Tens of millions of expert moves</td>
</tr>
</tbody>
</table>
LightLDA: LDA with $10^6$ topics ($10^{11}$ parameters); More topics $\rightarrow$ better performance in ad selection and click predictions

DistBelief: DNN with $10^{10}$ weights; Deeper and larger networks $\rightarrow$ better performance in image classification.

Human brain: $10^{11}$ neurons and $10^{15}$ connections, much larger than any existing ML model.
Big Compute

- Large computer clusters and highly parallel computational architectures
How to well utilize computation resources to speed up the training of big model over big data?

Distributed Machine Learning
Distributed Machine Learning

- Distributed Systems
  - Data allocation
  - Sequential machine learning algorithms on single machine
  - Synchronization mechanism
- Model Aggregation
Outline of this Tutorial

1. Machine Learning: Basic Framework and Optimization Techniques
   - Machine learning framework
   - Machine learning models
   - Optimization algorithms: deterministic vs. stochastic
   - Theoretical analysis

   - Data parallelism vs. model parallelism
   - Synchronous vs. asynchronous parallelization
   - Data allocation
   - Model aggregation
   - Theoretical analysis

3. Distributed Machine Learning: Systems and Toolkits
   - MapReduce (Spark MLlib)
   - Parameter Server (DMTK/Multiverso)
   - Data Flow (TensorFlow)
1. Machine Learning: Basic Framework and Optimization Techniques
Machine Learning

Unsupervised

Training set

Supervised

Feature extraction

New Data

Machine learning algorithm

Grouping of objects based on some common characteristics

Predictive model

Annotated data
### Machine Learning Tasks

<table>
<thead>
<tr>
<th>Classification (e.g. image recognition)</th>
<th>Regression (e.g. click prediction)</th>
<th>Ranking (e.g. Web search)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{Y} = {0, 1}$</td>
<td>$\mathcal{Y} = \mathbb{R}$</td>
<td>$\mathcal{Y} = {0, 1, \ldots, K}$ or rank</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Model</strong></th>
<th>$f : \mathcal{X} \rightarrow \mathbb{R}$</th>
<th>$f : \mathcal{X} \rightarrow \mathbb{R}$</th>
<th>$f : \mathcal{X} \rightarrow \mathbb{R}$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Inferences</strong></th>
<th>$\text{sgn} \circ f(x)$</th>
<th>$f(x)$</th>
<th>$\pi \circ f(x_1, \ldots, x_m) = \text{Rank}(f(x_1), \ldots, f(x_m))$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Evaluation</strong></th>
<th>$\mathbb{I}_{[\text{sgn} \circ f(x) \neq y]}$</th>
<th>$(f(x) - y)^2$</th>
<th>$(1 - \text{NDCG or MAP})(\pi \circ f(x_1, \ldots, x_m), y_1, \ldots, y_m)$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Training Loss</strong></th>
<th>Hinge + Margin, Exponential, Cross-Entropy</th>
<th>$L_2$</th>
<th>Hinge + Pairwise, Piecewise, Exponential, Cross-Entropy</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Regularization</strong></th>
<th>$L_1, L_2, \text{etc.}$</th>
<th>$L_1, L_2, \text{etc.}$</th>
<th>$L_1, L_2, \text{etc.}$</th>
</tr>
</thead>
</table>

| **Optimization** | SGD, SCD, Newton method, BFGS, Interior-Point method, ADMM, etc. | | |
|------------------|------------------------------------------------------||
Machine Learning Models

Shallow Models

- Linear model:
  \[ f(x) = \sum_{j=1}^{d} w_j x_j \]

- Kernel model:
  \[ f(x) = \sum_{i=1}^{n} w_i k(x, x_i) \]

Deep Models

- Fully-connected Neural Networks
- Convolutional Neural networks
- Recurrent Neural Networks

\[ f \in \mathcal{F}_A^L(\sigma, n_1, \ldots n_{L-1}, K) \]

Simple, weak representation power
Complex, strong representation power
Fully-connected Neural Networks

Cost functions:
- Squared error
- Hinge loss
- Ranking loss

Activation functions:
- Sigmoid
- ReLU
Convolutional Neural Networks (CNN)

- Local connectivity
- Sharing weights
- Pooling (translation invariance)

\[ x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l} \]

Pooling stage

\[ y_{i,j} = f(a_{i,j}) \]

Non-linear stage

- e.g. \( f(a) = [a]_+ \)
- \( f(a) = \text{sigmoid}(a) \)

Convolutional stage

\[ a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l} \]

Only parameters

Input image

2017/2/2
Recurrent Neural Networks (RNN)

• Model a dynamic system driven by an external signal $x$
  • $A_t = f(Ux_t + WA_{t-1})$
  • Hidden node $A_{t-1}$ contains information about the whole past sequence
  • function $f(\cdot)$ maps the whole past sequence $(x_t, \ldots, x_1)$ to current state $A_t$
Optimization Framework for Machine Learning

**Problem:** Regularized Empirical Risk Minimization

\[
F(w) := \frac{1}{n} \sum f_i(w) + \lambda R(w)
\]

where \( f_i(w) = L(w; x_i, y_i) \) and \( \{x_i, y_i; i = 1, ..., n\} \) are i.i.d. sampled from \( P_{x,y} \).

**Goal:** Find a solution of (R-)ERM that converges to the optima, i.e.,

\[
\mathbb{E} \left| w^T - w^* \right|^2 \leq \epsilon(T), \text{ or } \mathbb{E} F(w^T) - F(w^*) \leq \epsilon(T) \quad \text{(For convex cases)}
\]

where \( w^* \) is \( \arg \min_w F(w) \) or \( \arg \min_w P(w) := \int_{x,y} L(w; x, y) dP(x, y) \)

\[
\min_{t=1, ..., T} \mathbb{E} \left| \nabla f(w_t) \right|^2 \text{ or } \frac{1}{T} \sum_t \mathbb{E} \left| \nabla f(w_t) \right|^2 \leq \epsilon(T) \quad \text{(For non-convex cases)}
\]
Optimization Techniques

1847: Gradient Descent

1940s: Linear Programming
- Convex Optimization
- BFGS
- ADMM

1950s: Nonlinear optimization
- Conjugate gradient
- Coordinate Descent (Quasi)-Newton
- Frank-Wolfe
- Recursive/Adaptive Algorithms (SGD)

1960s: Coordinate Descent

1970: Convex Optimization;
- Interior-Point
- BFGS;
- ADMM

1980s: Nesterov's Acceleration
- Primal-Dual

1983: Conjugate gradient
- Coordinate Descent
- (Quasi)-Newton
- Frank-Wolfe
- Recursive/Adaptive Algorithms (SGD)

1990s: Coordinate Descent

2000s: Stochastic BFGS
- DANE
- Async SCD
- CoCoA
- Async ADMM

2010s: Stochastic BFGS
- DANE
- Async SCD
- CoCoA
- Async ADMM

2011: SCD
- Hogwild!
- Mini-batch SGD
- Downpour SGD
- Parallel SGD

2013: Stochastic BFGS
- DANE
- Async SCD
- CoCoA
- Async ADMM

2015: Stochastic BFGS
- DANE
- Async SCD
- CoCoA
- Async ADMM

2016: ADMM+SVRG
- Frank-Wolfe+SVRG
- BFGS + SCD
- Newton + SDCA
- SASGD
- Graduated SGD
- DC-ASGD:

2017: Deterministic Algorithms
Stochastic Algorithms
Parallel Algorithms
Deterministic Optimization

First-order Methods

Primal

GD, CD, Projected Sub-GD, Frank-Wolfe, Nesterov's acceleration

Dual

Dual Problem Maximization, ADMM

Second-order Methods

Dual Newton’s Ascent

(Quasi-) Newton’s Method
**Stochastic Optimization**

**First-order Methods**
- GD, CD, Projected Sub-GD, Frank-Wolfe, Nesterov's acceleration
- SGD, SCD, SVRG, SAGA, FW+VR, ADMM+VR, ACC+SGD+VR, etc.

**Second-order Methods**
- Dual Problem Maximization, ADMM
- Dual Newton’s Ascent
- (Quasi-) Newton’s Method

**Stochastic (Quasi-) Newton’s Method**
Optimization Theory

- Conditions:
  - Strongly Convex, Convex, and Non-Convex
  - Smoothness and Continuity

- Convergence rate:
  - Linear
  - Super-linear
  - Sub-linear
Convexity

**Convex**: if the line segment between any two points on the graph of the function lies above or on the graph.

\[ f(x) - f(y) \geq \nabla f(y)^T (x - y) \]

\[ f(x) = x_1^4 + x_2^4 \]

**Strongly convex**: at least as steep as a quadratic function.

\[ f(x) - f(y) \geq \nabla f(y)^T (x - y) + \frac{\alpha}{2} \|x - y\|^2 \]

\[ f(x) = x_1^2 + x_2^2 \]

\[ f(\cdot) \] is $\alpha$-strongly convex only if $f(\cdot) - \frac{\alpha}{2} \|\cdot\|^2$ is convex.
Smoothness

**Smoothness**: a small change in the input will lead to also a small change in the function output.

- **β-smooth for differentiable functions**
  \[
  f(x) - f(y) \leq \nabla f(x)^T (x - y) + \frac{\beta}{2} \|x - y\|^2 \\
  f(x) - f(y) \geq \nabla f(y)^T (x - y) + \frac{\alpha}{2} \|x - y\|^2
  \]
  For \(x^* \in \arg\min f(x)\), we have
  \[
  \frac{\alpha}{2} \|x - x^*\|^2 \leq f(x) - f(x^*) \leq \frac{\beta}{2} \|x - x^*\|^2 \\
  (\alpha \leq \beta)
  \]

- **L-Lipschitz for non-differentiable functions**
  \[
  |f(x) - f(y)| \leq L \|x - y\|
  \]

\(f(\cdot)\) is \(\beta\)-smooth only if \(\nabla f(\cdot)\) is \(\beta\)-Lipschitz.
Convergence Rate

Does the error $\epsilon(T')$ decrease faster than $e^{-T}$?

- Equal to: linear convergence rate, e.g., $O(e^{-T})$
- Faster than: super-linear convergence rate, e.g., $O(e^{-T^2})$
- Slower than: sub-linear convergence rate, e.g., $O\left(\frac{1}{T}\right)$
1. Machine Learning:
Basic Framework and Optimization Techniques
Optimization Methods

• Deterministic Optimization
• Stochastic Optimization
Deterministic Optimization

First-order Methods

Second-order Methods

Primal

Dual

GD, PSG, FW, Nesterov’s acceleration
Gradient Descent [Cauchy 1847]

**Motivation:** to minimize the local first-order Taylor approximation of $f$

$$\min_x f(x) \approx \min_x f(x_t) + \nabla f(x_t)^T (x - x_t)$$

**Update rule:**

$$x_{t+1} = x_t - \eta \nabla f(x_t),$$

where $\eta > 0$ is a fixed step-size.
Convergence Rate of GD

Theorem 1: Assume the objective $f$ is convex and $\beta$-smooth on $\mathbb{R}^d$. With step size $\eta = \frac{1}{\beta}$, Gradient Descent satisfies:

$$f(x_{T+1}) - f(x^*) \leq \frac{2\beta ||x_1-x^*||^2}{T}.$$ 

Theorem 2: Assume the objective $f$ is $\alpha$-strongly convex and $\beta$-smooth on $\mathbb{R}^d$. With step size $\eta = \frac{2}{\alpha+\beta}$, Gradient Descent satisfies:

$$f(x_{T+1}) - f(x^*) \leq \frac{\beta}{2} \exp \left(-\frac{4T}{Q+1}\right) ||x_1 - x^*||^2,$$

where $Q = \frac{\beta}{\alpha}$. 
Projected Sub-gradient Descent (PSD)

Differences: 
\( a) \ x \in \mathcal{X}, \ b) \ gradient \Rightarrow sub-gradient \)

Update Rule:

\[
y_{t+1} = x_t - \eta g_t, \\
x_{t+1} = \Pi_{\mathcal{X}}(y_{t+1}),
\]

where \( g_t \in \partial f(x_t), \Pi_{\mathcal{X}}(x) = \arg\min_{y \in \mathcal{X}} \|x - y\| \)

Illustration:
Convergence Rate of PSG

<table>
<thead>
<tr>
<th>Smooth Setting</th>
<th>Convex</th>
<th>Strongly Convex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper Bound</td>
<td>$O\left(\frac{1}{T}\right)$</td>
<td>$O\left(\exp\left(-\frac{T}{Q}\right)\right)$</td>
</tr>
<tr>
<td>Lower Bound</td>
<td>$O\left(\frac{1}{\sqrt{T}}\right)$</td>
<td>$O\left(\frac{1}{T^{2}}\exp\left(-\frac{T}{Q}\right)\right)$</td>
</tr>
</tbody>
</table>

Condition number: $Q = \frac{\beta}{\alpha} \geq 1$
Nesterov’s Accelerated GD (NA) [Nesterov 1983]

Motivation:
To bridge the gap between the upper bound and the lower bound for the convergence rate in smooth setting.

Illustration:

Update Rule – Strongly Convex:
\[ y_{t+1} = x_t - \frac{1}{\beta} \nabla f(x_t), \]
\[ x_{t+1} = \left(1 + \frac{\sqrt{Q} - 1}{\sqrt{Q} + 1}\right) y_{t+1} - \frac{\sqrt{Q} - 1}{\sqrt{Q} + 1} y_t \]

Update Rule – Convex:
\[ y_{t+1} = x_t - \frac{1}{\beta} \nabla f(x_t), \]
\[ x_{t+1} = (1 - \gamma_t) y_{t+1} + \gamma_t y_t \]
where \( \lambda_0 = 0, \lambda_t = \frac{1+\lambda_t^2}{2}, \) and \( \gamma_t = \frac{1-\lambda_t}{\lambda_{t+1}} \)
Convergence Rate of NA

The Gap in Smooth Setting

<table>
<thead>
<tr>
<th>Smooth setting</th>
<th>Convex</th>
<th>Strongly Convex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper bound</td>
<td>$O\left(\frac{1}{T}\right)$</td>
<td>$exp\left(-\frac{T}{Q}\right)$</td>
</tr>
<tr>
<td>Lower Bound</td>
<td>$O\left(\frac{1}{T^2}\right)$</td>
<td>$exp\left(-\frac{T}{\sqrt{Q}}\right)$</td>
</tr>
</tbody>
</table>

Theorem 3: Assume the objective $f$ is **convex** and $\beta$-smooth on $R^d$. Then Nesterov’s Accelerated Gradient Descent satisfies:

$$f(y_t) - f(x^*) \leq \frac{2\beta \|x_1 - x^*\|^2}{t^2}.$$ 

Theorem 4: Assume the objective $f$ is **$\alpha$-strongly convex** and $\beta$-smooth on $R^d$. Then Nesterov’s Accelerated Gradient Descent satisfies:

$$f(y_t) - f(x^*) \leq \frac{\alpha + \beta}{2} \exp\left(-\frac{t-1}{\sqrt{Q}}\right) \|x_1 - x^*\|^2,$$

where $Q = \frac{\beta}{\alpha}$. 

2017/2/4

AAAI 2017 Tutorial 36
Frank-Wolfe \cite{frank1956algorithm}

**Motivation:** Projection step may be a bottleneck.

Recall Gradient Descent: to minimize the local first-order Taylor approximation of $f$

$$x_{t+1} = \arg\min_x f(x_t) + \nabla f(x_t)^\top (x - x_t)$$

**Frank-Wolfe (conditional gradient descent):**

$$y_t = \arg\min_{y \in \mathcal{X}} f(x_t) + \nabla f(x_t)^\top (y - x_t) = \arg\min_{y \in \mathcal{X}} \nabla f(x_t)^\top y$$

$$x_{t+1} = (1 - \gamma_t)x_t + \gamma_t y_t$$

Convergence Rate of Frank-Wolfe

Theorem 5: Assume the objective $f$ is convex and $\beta$-smooth w.r.t. some norm $\|L\|$, $R = \sup_{x,y\in X} \|x - y\|$. With combination weight $\gamma_s = \frac{2}{s+1}$, Frank-Wolfe satisfies:

$$f(x_T) - f(x^*) \leq \frac{2\beta R^2}{T + 1}$$

Frank-Wolfe algorithm has the same convergence rate with PGD. We may choose one of them considering whether the projection is expensive.
Deterministic Optimization

First-order Methods
- GD, PSG, FW, Nesterov’s acceleration

Second-order Methods
- (Quasi-) Newton’s Method
Newton’s Methods

Motivation:
To minimize the local second-order Taylor approximation of $f$:

$$
\min_x f(x) \approx \min_x f(x_t) + \nabla f(x_t)^\tau (x - x_t) + \frac{1}{2} (x - x_t)^\tau \nabla^2 f(x_t) (x - x_t) + o \left( \|x - x_t\|^2 \right).
$$

Update Rule: suppose $\nabla^2 f(x_t)$ is positive definite,

$$
x_{t+1} = x_t - [\nabla^2 f(x_t)]^{-1} \nabla f(x_t)
$$

A more dedicate step size, comparing to GD.
Convergence Rate of Newton’s Method

Theorem 6: Suppose the function $f$ is continuously differentiable, its derivative is 0 at its optimum $x^*$, and it has a second derivative at $x^*$, then the convergence is quadratic:

$$|x_t - x^*| \leq O \left(e^{-2T}\right)$$

**Advantage:**
We have a more accurate local approximation of the objective, the convergence is much faster.

**Disadvantage:**
We need to compute the inverse of Hessian, which is time/storage consuming.
Quasi Newton’s Methods

Main Idea:
The inverse of Hessian is approximated by rank one updates of the gradient evaluations, under the Quasi-Newton’s condition.

Denote $B_t = \nabla^2 f(x_t)$, $H_t = [\nabla^2 f(x_t)]^{-1}$

$B_{t+1} = B_t + aa^\tau + bb^\tau$

$$f(x) \approx f(x_t) + \nabla f(x_t)^\tau (x - x_t) + \frac{1}{2} (x - x_t)^\tau B_t (x - x_t)$$

$$\nabla f(x) \approx \nabla f(x_t) + B_t (x - x_t) \quad \text{//Calculate gradients for both sides}$$

$$B_t^{-1} y_t \approx s_t \quad \text{// Let } x = x_{t+1},$$

where $y_t = \nabla f(x_{t+1}) - \nabla f(x_t), s_t = x_{t+1} - x_t$
Quasi-Newton Algorithms

**BFGS (Broyden–Fletcher–Goldfarb–Shanno):**

\[
B_{t+1} = B_t + \frac{B_0}{y_t s_t} \left( y_t s_t \right)^\tau - \frac{B_t s_t (B_t s_t)^\tau}{s_t B_t s_t};
\]

\[
H_{t+1} = \left( I - \frac{s_t y_t^\tau}{y_t^\tau s_t} \right) H_t \left( I - \frac{y_t s_t^\tau}{y_t^\tau s_t} \right) + \frac{s_t s_t^\tau}{y_t^\tau s_t} \quad \text{// by Sherman-Morrison formula}
\]

**Other Algorithms:** DFP, Broyden, SR1, etc.

**Convergence rate:**
If the start point is close enough to a strict local minimum, quasi-Newton’s method has super-linear convergence rate.
# Summary – Deterministic Algorithms

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Convergence rates</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD</td>
<td>$O\left(\exp\left(-\frac{t}{Q}\right)\right)$</td>
</tr>
<tr>
<td>ACC-GD</td>
<td>$O\left(\exp\left(-\frac{t}{\sqrt{Q}}\right)\right)$</td>
</tr>
<tr>
<td>PGD</td>
<td>$O\left(\exp\left(-\frac{t}{Q}\right)\right)$</td>
</tr>
<tr>
<td>Frank-Wolfe</td>
<td>$O\left(\exp\left(-\frac{t}{Q}\right)\right)$</td>
</tr>
<tr>
<td>Newton</td>
<td>$O\left(\exp\left(-e^{2t}\right)\right)$</td>
</tr>
<tr>
<td>Quasi-Newton -BFGS</td>
<td>$O\left(\exp\left(-e^{2t}\right)\right)$</td>
</tr>
</tbody>
</table>

Strongly convex + smooth
Deterministic Optimization

First-order Methods

- GD, CD, PSG, FW, Nesterov’s acceleration

Second-order Methods

- (Quasi-) Newton’s Method

Dual Methods?
Primal-Dual: Lagrangian

**Primal Problem:**
\[
\begin{align*}
\min_{x} & \quad f_0(x) \\
\text{s.t.} & \quad f_i(x) \leq 0, \quad i = 1, \ldots, m \\
& \quad h_i(x) = 0, \quad j = 1, \ldots, p
\end{align*}
\]
Optimal value: \( p^* \)

**The Lagrangian:**
\[
L(x, \lambda, \nu) \triangleq f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{p} \nu_j h_j(x)
\]
where \( \lambda_i \geq 0, \nu_j \in \mathbb{R} \).

A problem with hard constraints => A series of problems with soft constraints
ADMM

Separable objective with a constraint:
\[
\min_{x,z} f(x) + g(z) \\
\text{s.t. } Ax + Bz = c
\]

Augmented Lagrangian: \( \rho > 0 \)
\[
L_\rho(x, y, z) = f(x) + g(z) + y^T(Ax + Bz - c) + \left(\frac{\rho}{2}\right) \|Ax + Bz - c\|^2
\]

ADMM’s Partial Updates:
\[
\begin{align*}
  x^{t+1} &= \arg\min_x L_\rho(x, z^t, y^t) \quad \text{--------} x \text{ minimization} \\
  z^{t+1} &= \arg\min_z L_\rho(x^{t+1}, z, y^t) \quad \text{--------} y \text{ minimization} \\
  y^{t+1} &= y^t + \rho(Ax^{t+1} + Bz^{t+1} - c) \quad \text{--------} \text{dual ascent update}
\end{align*}
\]

For example, in DML:
\[
\min_{x_k, z} \sum_{k=1}^K f_k(x_k, D_k) \\
\text{s.t. } x_k = z, \forall k = 1, \ldots, K.
\]
Convergence Property of ADMM

• The general convergence:
  
  Theorem 6: Assume the extended real-valued objective \( f, g \) are proper, close, and convex, and the augmented Lagrangian has a saddle point, then ADMM satisfies:
  
  \[
  f(x^t) + g(z^t) \to p^* , y^t \to y^* , \text{as } t \to \infty.
  \]

• Convergence rate:
  
  • Convex: \( O \left( \frac{1}{\epsilon} \right) \)
  
  • Strongly-convex: \( \left( \sqrt{Q \log \frac{1}{\epsilon}} \right) \)

• ADMM variants:
  
  • with inexact minimization still converge under some conditions
  
  • with divergence penalty other than \( L^2 \) penalty, such as Bregman divergence may not converge
Primal-Dual: Dual function

Primal Problem:
\[
\begin{align*}
\min_{x} & \quad f_0(x) \\
\text{s.t.} & \quad f_i(x) \leq 0, i = 1, \ldots, m \\
& \quad h_i(x) = 0, j = 1, \ldots, p
\end{align*}
\]
Optimal value: \( p^* \)

The Lagrangian:
\[
L(x, \lambda, \nu) \triangleq f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{p} \nu_j h_i(x)
\]
where \( \lambda_i \geq 0, \nu_j \in \mathbb{R} \).

For feasible solutions,
\[
L(x, \lambda, \nu) \leq f_0(x), \forall x \in \mathcal{X}
\]

Take min over \( x \) to both side,
\[
\inf_{x \in \mathcal{X}} L(x, \lambda, \nu) \leq p^*
\]

The Lagrangian dual function:
\[
g(\lambda, \nu) \triangleq \inf_{x \in \mathcal{X}} L(x, \lambda, \nu) \leq p^*, \lambda_i \geq 0, \nu_j \in \mathbb{R}
\]

Property:
Since \( L \) is linear w.r.t. \( \lambda, \mu \), and \( \inf \) is concave-preserving, \( g(\lambda, \nu) \) is concave, even when the primal problem is not convex.
Primal-Dual: Dual Problem

**Primal Problem:**
\[
\min_x f_0(x) \\
\text{s.t. } f_i(x) \leq 0, i = 1, \ldots, m \\
h_i(x) = 0, j = 1, \ldots, p
\]
Optimal value: \( p^* \)

**Dual Problem:**
\[
\max_{\lambda, \nu} g(\lambda, \nu) \\
\text{s.t. } \lambda_i \geq 0, i = 1, \ldots, m
\]
The dual problem is convex. Optimal value: \( d^* \)

**The Lagrangian:**
\[
L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{p} \nu_j h_j(x)
\]
where \( \lambda_i \geq 0, \nu_j \in \mathbb{R} \).

**The best lower bound of \( p^* \)?

**The Lagrangian dual function:**
\[
g(\lambda, \nu) = \inf_{x \in \mathbb{X}} L(x, \lambda, \nu) \leq p^*, \lambda_i \geq 0, \nu_j \in \mathbb{R}
\]
Primal-Dual: Duality

Primal Problem:
\[
\min_x f_0(x) \\
\text{s.t. } f_i(x) \leq 0, i = 1, \ldots, m \\
h_i(x) = 0, j = 1, \ldots, p
\]
Optimal value: \( p^* \)

The Lagrangian:
\[
L(x, \lambda, \nu) \triangleq f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \sum_{j=1}^p \nu_j h_j(x)
\]
where \( \lambda_i \geq 0, \nu_j \in \mathbb{R} \).

The dual problem is concave. Optimal value: \( d^* \)

Dual Problem:
\[
\max_{\lambda, \nu} g(\lambda, \nu) \\
\text{s.t. } \lambda_i \geq 0, i = 1, \ldots, m
\]
The dual problem is concave. Optimal value: \( d^* \)

Weak duality always holds: \( d^* \leq p^* \).
When will strong duality \( d^* = p^* \) hold? E.g., Slater’s condition.
An necessary condition for differentiable problem: KKT

The Lagrangian dual function:
\[
g(\lambda, \nu) \triangleq \inf_{x \in X} L(x, \lambda, \nu) \leq p^*, \lambda_i \geq 0, \nu_j \in \mathbb{R}
\]
Property: concave, even when the primal problem is not convex.
Dual Methods

- In dual methods, we convert primal problem into its dual problem, and then implement first-order or second-order algorithms.
Optimization Methods

• Deterministic Optimization
• Stochastic Optimization
Stochastic Optimization

First-order Methods
- GD, PSG, FW, Nesterov’s acceleration
- SGD, SCD, SVRG, SAGA, FW+VR, ADMM+VR, ACC+SGD+VR, etc.

Second-order Methods
- Dual Problem Maximization, ADMM
- Dual Newton’s Ascent
- (Quasi-) Newton’s Method

SDCA
- Dual Problem Maximization, ADMM
- GD, PSG, FW, Nesterov’s acceleration

SDNA
- Dual Newton’s Ascent
- (Quasi-) Newton’s Method

Stochastic (Quasi-) Newton’s Method
Motivation

Example (Linear regression + GD):

Objective: \[ f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x) = \frac{1}{n} \sum_{i=1}^{n} (a_i x - b_i)^2, \ x \in \mathbb{R}^d \]

Update rule: \[ x_{t+1} = x_t - \eta \nabla f(x_t) = x_t - \frac{2n}{n} \sum_{i=1}^{n} a_i (a_i x - b_i) \]

The computation complexity for each iteration is linearly increasing with the data size \( n \) and the feature size \( d \), which are both large for big data!
Two Statistical Approaches

**Data Sampling:** Stochastic Gradient Descent (SGD)
\[ x_{t+1} = x_t - \eta_t \nabla f_i(x_t), \text{ where } \mathbb{E}_t \nabla f_i(x_t) = \nabla f(x_t) \]

**Feature Sampling:** Stochastic Coordinate Descent (SCD)
\[ x_{t+1} = x_t - \eta \nabla f_j(x_t), \text{ where } \mathbb{E}_t \nabla f_j(x_t) = \frac{1}{d} \nabla f(x_t) \]

Convergence Rate: deterministic → in expectation or with high probability
\[ \mathbb{E} f(x_T) - f(x^*) \leq \epsilon - \text{Convex} \]
\[ \min_{t=1,...,T} \mathbb{E} \left| \nabla f(x_t) \right|^2 \text{ or } \frac{1}{T} \sum_{t=1}^T \mathbb{E} \left| \nabla f(x_T) \right|^2 \leq \epsilon - \text{Nonconvex} \]
## Convergence Rate and Computational Complexity

**Overall Complexity**  
\[ (\varepsilon) = \text{Convergence Rate}^{-1}(\varepsilon) \times \text{Complexity of each iteration} \]

<table>
<thead>
<tr>
<th></th>
<th>Strongly Convex + Smooth</th>
<th>Convex + Smooth</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Convergence Rate</strong></td>
<td><strong>Complexity of each iteration</strong></td>
<td><strong>Overall Complexity</strong></td>
</tr>
<tr>
<td>GD</td>
<td>( O\left(\exp\left(-\frac{t}{Q}\right)\right) )</td>
<td>( O\left(n \cdot d\right))</td>
</tr>
<tr>
<td>SGD</td>
<td>( O\left(\frac{1}{t}\right) )</td>
<td>( O\left(d\right))</td>
</tr>
<tr>
<td>SCD</td>
<td>( O\left(\exp\left(-\frac{t}{d \cdot \max\left{Q_j\right}}\right)\right) ) (For separable cases)</td>
<td>( O\left(n\right))</td>
</tr>
</tbody>
</table>

1. When data size \( n \) is very large, SGD is faster than GD.
2. SCD is faster than GD for separable cases. \( \max\left\{\beta_j\right\} \leq \beta \leq d\beta_j, \max\left\{Q_j\right\} \leq Q \leq dQ_j \)
Reason for Convergence Rate Reduction

Error Induction in GD:

\[
\begin{align*}
||x_{t+1} - x^*||^2 &= ||x_t - x^*||^2 - \eta_t \nabla f(x_t)^T (x_t - x^*) + \eta_t^2 ||\nabla f(x_t)||^2 \\
&\leq \left(1 - \frac{2\eta\beta\alpha}{\beta + \alpha}\right)||x_t - x^*||^2 + \eta \left(\eta - \frac{2}{\beta + \alpha}\right) ||\nabla f(x_t)||^2
\end{align*}
\]

Let \( \eta - \frac{2}{\beta + \alpha} \leq 0 \), we can get \( A_{t+1} = aA_t \), where \( 0 \leq a \leq 1 \)

**Linear Convergence**
Reason for Convergence Rate Reduction

Error Induction in SCD:

\[ \mathbb{E}_j \| x_{t+1} - x^* \|^2 \]
\[ = \| x_t - x^* \|^2 - \eta_t \mathbb{E}_j \nabla f(x_t)^T (x_t - x^*) + \eta_t^2 \mathbb{E}_j \nabla f(x_t)^2 \]
\[ \leq \| x_t - x^* \|^2 - \eta_t \| \nabla f(x_t) \| (x_t - x^*) + \frac{\eta_t^2}{d} \| \nabla f(x_t) \|^2 \]

---- similar to GD

\[ A_{t+1} = aA_t, \text{ where } 0 \leq a \leq 1 \]

**Linear Convergence**
Reason for Convergence Rate Reduction

Error Induction in SGD:

\[
\mathbb{E}_{i_t} ||x_{t+1} - x^*||^2 \\
= ||x_t - x^*||^2 - \eta_t \mathbb{E}_{i_t} \nabla f_i(x_t)^T (x_t - x^*) + \eta_t^2 \mathbb{E}_{i_t} ||\nabla f_i(x_t)||^2 \\
\leq (1 - 2\eta_t \alpha)||x_t - x^*||^2 + \eta_t^2 b
\]

Usually, we assume \( \mathbb{E}_i ||\nabla f_i(x_t)||^2 \leq b \)

\[
A_{t+1} = aA_t + \eta_t^2 b, \text{ where } 0 < a < 1, b \geq 0
\]

\( \eta \) should be decreasing

Sublinear Convergence
Stochastic Optimization

First-order Methods:
- GD, PSG, FW, Nesterov’s acceleration
- SGD, SCD, SVRG, SAGA, FW+VR, ADMM+VR, ACC+SGD+VR, etc.

Second-order Methods:
- Dual Problem Maximization, ADMM
- Dual Newton’s Ascent
- Dual Newton’s Ascent
- (Quasi-) Newton’s Method

Stochastic (Quasi-) Newton’s Method
Stochastic (Quasi-) Newton’s Method
[Byrd,Hansen,Nocedal&Singer,2015]

Update Rule:
• Stochastic mini-batch Quasi-Newton:
  \[ x^{t+1} = x^t - \eta_t H_t \cdot \frac{1}{b} \sum_{i \in B_t} \nabla f_i(x^t) \]
• Hessian updating:
  \[ \hat{\nabla^2} F(x^t) = \frac{1}{bh} \sum_{i \in B_{H,t}} \nabla^2 f_i(x^t); s^t = (x^t - x^{t-1}); y^t = \hat{\nabla^2} F(x^t)(x^t - x^{t-1}) \]
  \[ H \leftarrow (I - \rho_j s_j y_j^T) H (I - \rho_j y_j s_j^T) + \rho_j s_j s_j^T \]

Convergence Rate: \( O\left(\frac{1}{T}\right) \)
Assumption 1: strongly convex + smooth
Assumption 2: \( \mathbb{E}\|\nabla f_i(w^t)\|^2 \leq G^2 \)
SDCA [Shalev-Shwartz & Zhang, 2013]

\[
\begin{align*}
\min_{w \in \mathbb{R}^d} F(w) &= \frac{1}{n} \sum_{i=1}^{n} \phi_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2 \\
\max_{\alpha \in \mathbb{R}^n} D(\alpha) &= \frac{1}{n} \sum_{i=1}^{n} -\phi^*_i(-\alpha_i) - \frac{\lambda n}{2} \|\frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_i x_i\|_2^2,
\end{align*}
\]

where \(\phi^*(u) = \max_z (zu - \phi_i(z))\)

If we define \(w(\alpha) = \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_i x_i\), then it is known that \(w(\alpha^*) = w^*, P(w^*) = D(\alpha^*)\).

According to Fenchel’s duality theorem

**Update Rule:**

1. Randomly pick \(i\), find \(\Delta \alpha_i\) to maximize \(-\phi^*_i(-\alpha^t_{i-1} + \Delta \alpha_i) - \frac{\lambda n}{2} \|w^t_{-1} + \frac{1}{\lambda n} \Delta \alpha_i x_i\|_2^2\)
2. \(\alpha^t = \alpha^{t-1} + \Delta \alpha_i e_i\)
3. \(w^t = w^{t-1} + \frac{1}{\lambda n} \Delta \alpha_i x_i\)
Convergence Rate of SDCA

Assumption: \( \phi_i \) is convex and \( L \)-Lipschitz or \( \beta \) – smooth

It is well known that if \( \phi_i(\alpha) \) is \( \beta \)-smooth, then \( \phi_i^*(u) \) is \( 1/\beta \)-strongly convex.

Stopping Criterion: Duality gap \( F(w(\alpha)) - D(\alpha) \leq \epsilon \)

<table>
<thead>
<tr>
<th>Complexity</th>
<th>( L )-Lipschitz</th>
<th>( \beta )-smooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDCA</td>
<td>( O\left( n + \frac{L^2}{\lambda \epsilon} \right) )</td>
<td>( O\left( \left( n + \frac{\beta}{\lambda} \right) \log \frac{1}{\epsilon} \right) )</td>
</tr>
</tbody>
</table>

where \( \lambda \) is the regularization term coefficient.
**SDNA [??]**

\[
\min_{w \in \mathbb{R}^d} F(w) = \frac{1}{n} \sum_{i=1}^{n} \phi_i(w^T x_i) + \frac{\lambda}{2} g(w)
\]

\[
\max_{\alpha \in \mathbb{R}^n} D(\alpha) = \frac{1}{n} \sum_{i=1}^{n} -\phi_i(-\alpha_i) - \lambda g^* \left( \frac{1}{\lambda n} X \alpha \right),
\]

where \( \phi^*(u) = \max_z (zu - \phi_i(z)) \), \( g^*(u) = \max_z (zu - g(z)) \) and \( X = (x_1, \ldots, x_n) \) with each \( x_i \in \mathbb{R}^d \)

**Update Rule**

Generate a random set of blocks \( S_t \sim \hat{S} \), and compute

\[
\Delta \alpha^t = \arg \min_{\alpha \in \mathbb{R}^n} \left\langle (X^T w^t)_{S_t}, \alpha \right\rangle + \frac{1}{2\lambda n} \alpha^T (X^T X)_{S_t} \alpha + \sum_{i \in S_t} \phi_i^*(-\alpha_i^t - h_i)
\]

Primal Update: \( w^t = \nabla g^*(\bar{\alpha}^t) \)

Dual Update: \( \alpha^{t+1} = \alpha^t + (\Delta \alpha^t)_{S_k} \)

Average Update: \( \bar{\alpha}^{t+1} = \bar{\alpha}^t + \frac{1}{\lambda n} \sum_{i \in S_t} \Delta \alpha_i^t \alpha_i \)

Second-order approximation. Newton’s type learning rate.
Convergence Rate of SDNA

SDNA:  \[ \mathbb{E}[F(w^k) - D(\alpha^k)] \leq \frac{(1-c_2)^k}{c_2} (D(\alpha^*) - D(\alpha^0)) \]  \[ \Rightarrow \mathcal{O}\left(\frac{1}{c_2} \cdot \log \frac{1}{\epsilon}\right) \]

SDCA:  \[ \mathbb{E}[F(w^k) - D(\alpha^k)] \leq \frac{(1-c_1)^k}{c_1} (D(\alpha^*) - D(\alpha^0)) \]  \[ \Rightarrow \mathcal{O}\left(\frac{1}{c_1} \cdot \log \frac{1}{\epsilon}\right) \]

**Theorem:** If the sampling methods is uniform, then \( c_1 \leq c_2 \).

SDCA converges faster than SDCA.
Stochastic Optimization

First-order Methods
- GD, PSG, FW, Nesterov's acceleration
- SGD, SCD, SVRG, SAGA, FW+VR, ADMM+VR, ACC+SGD+VR, etc.

Second-order Methods
- Dual Problem Maximization, ADMM
- Dual Newton's Ascent
- (Quasi-) Newton's Method

1. SQN ~ SGD
2. SDCA converge linearly in smooth case
3. SDNA > SDCA

Stochastic (Quasi-) Newton's Method
Data Parallelism vs. Model Parallelism

1. Partition the training data
2. Parallel training on different machines
3. Synchronize the local updates
4. Refresh local model with new parameters, then go to 2.

1. Partition the model into multiple local workers
2. For every sample, the local workers collaborate with each other to perform the optimization
Distributed Machine Learning Architectures

Dataflow

- Flexible in modeling dependency
- Lack of good abstraction

Parameter Server

- Support hybrid parallelism and fine-grained parallelization, particularly for deep learning
- Good balance between high-level abstraction and low-level flexibility in implementation

Asynchronous

Synchronous

Data Parallelism

- High-level abstractions (MapReduce)
- Lack of flexibility in modeling complex dependency

Model Parallelism

Irregular Parallelism

- Petuum, DMLC
- TensorFlow

Spark

Iterative MapReduce

2017/2/2
Evolution of Distributed ML Architectures

Iterative MapReduce

- Use MapReduce / AllReduce to sync parameters among workers
- Only synchronous update
- Example: Spark and other derived systems
Evolution of Distributed ML Architectures

- Parameter server (PS) based solution is proposed to support:
  - Asynchronous update
  - Different mechanisms for model aggregation, especially in asynchronous manner
  - Model parallelism

- Example:
  - Google’s DistBelief;
  - Petuum
  - Multiverso
Evolution of Distributed ML Architectures

Dataflow based solution is proposed to support:

- Irregular parallelism (e.g., hybrid data- and model-parallelism), particularly in deep learning
- Both high-level abstraction and low-level flexibility in implementation

Example:
- TensorFlow

Task scheduling & execution based on:
1. Data dependency
2. Resource availability
Data Parallelism

• Optimization under different parallelization mechanisms
  • Synchronous vs. Asynchronous vs. Lock-free

• Aggregation method
  • Consensus based on model average

• Data allocation strategy
  • Shuffling + Partition
  • Sampling
Parallelization Mechanism: BSP

*Bulk Synchronous Parallel*
Example Algorithm: BSP-SGD

\[
\min_w \sum_{k=1}^{K} L_k(w)
\]

\[
\begin{align*}
    w_k^t &= w^t \\
    \Delta w_k^t &= -\eta_t \nabla L_k(w_k^t) \\
    w^{t+1} &= w^t + \sum_k \Delta w_k^t
\end{align*}
\]

Minibatch SGD [Dekel et. al., 2013]

**Communication:** every iteration

**Convergence Rate:** (convex and smooth case)

\[
O\left(\frac{1}{\sqrt{PT}}\right)
\]

**Speedup:** \(O(\sqrt{P})\)
Parallelization Mechanism: ADMM

**Alternating Direction Method of Multipliers**

$$\min_w \sum_{k=1}^K L_k(w)$$

s.t. \( w_k - z = 0, k = 1, \ldots, K \)

\[
\begin{align*}
    w_k &= \arg \min_{w_k} L_k(w) \\
    w_{k}^{t+1} &= \arg \min_{w_k} \sum_k \left( L_k(w_k) + (\lambda_k^t)^T (w_k - z^t) + \frac{\rho}{2} \|w_k - z^t\|_2^2 \right) \\
    z^{t+1} &= \frac{1}{K} \sum_k (w_k^{t+1} + \frac{1}{\rho} \lambda_k^t) \\
    \lambda_k^{t+1} &= \lambda_k^t + \rho (w_k^{t+1} - z^{t+1})
\end{align*}
\]

A global consensus problem
Parallelization Mechanism: MA

Model Average

$$\min_w \sum_{k=1}^{K} L_k(w)$$

$$w_k = \arg \min L_k(w)$$

$$z^{t+1} = \frac{1}{N} \sum_k w_k^t$$

$$w_k^{t+1} = z^{t+1}$$

Another global consensus problem
Parallelization Mechanism: Elastic Averaging SGD

- **MA**: each worker can only explore several steps (1 in most cases) and then parameter server will force to sync with local workers.
- **EASGD** motivation: Cancel the compulsive synchronization by parameter server, and allow local workers for more exploration.
- **EASGD** main idea: link the parameters of workers with an elastic force (i.e. a global variable) and allow local variables to fluctuate from the center variable.

[Zhang et.al. NIPS 2015]

Parallelization Mechanism: ASP

Asynchronous Parallel

Workers push update to parameter server and/or pull latest parameter back without waiting for others.
Example Algorithm: ASGD

Center Parameter Update: \( \hat{w}_{t+\tau+1} = \hat{w}_{t+\tau} - \eta_t \cdot g_t \)
where \( g_t = \nabla f(\hat{w}_t) \)
(Clock \( t \) is defined according to the global update.)

Communicate every iteration.

Local: Gradient Calculation

Theoretical Analysis [Agarwal&Duchi,2011]:

Convergence Rate: \( O \left( \frac{\tau}{T} + \frac{1}{\sqrt{T}} + \frac{(\tau+1)^2 \log T}{T} \right) \)

Speedup: \( \tau = O(T^{1/4}) \), then it is the same as serial case \( O(1/\sqrt{T}) \).
Parallelization Mechanism: AADMM

**Asynchronous ADMM**

- Partial barrier and bounded delay
- Each local machine updates $\omega$ using the most recent $z$ received from the server
- The server update $z$ after receiving $S$ not-too-old updates from the local machines.
Parallelization Mechanism: Hogwild!

Goal: \( \min f(x) = \sum_{e \in E} f_e(x_e) \) \( \Rightarrow \) hypergraph \( G(V, E) \).
Each subvector \( x_e \) induces an edge consisting of some nodes (training instances).

Three sparse parameter:
\[ \Omega = \max_{e \in E} |e|, \Delta = \max_{1 \leq v \leq n} \frac{\max_{e \in E} |\{e \in E : v \in e\}|}{|E|}, \rho = \max_{e \in E} \frac{\max_{\hat{e} \in E : e \cap \hat{e} \neq \emptyset}|e|}{|E|} \]

Algorithm 1 HOGWILD! update for individual processors

1: loop
2: Sample \( e \) uniformly at random from \( E \)
3: Read current state \( x_e \) and evaluate \( G_e(x) \)
4: for \( v \in e \) do \( x_v \leftarrow x_v - \gamma b_v^T G_e(x) \)
5: end loop

Theoretical Analysis: with-lock, constant learning rate
If \( \tau = o(n^{1/4}) \) as \( \rho \) and \( \Delta \) are typically both \( o\left(\frac{1}{n}\right) \), it can achieve linear speedup. [Niu et al, NIPS-11]
Parallelization Mechanism: Cyclades

Motivation: introduce no conflicts between cores during the asynchronous parallel execution.

\[ G_u \]  
\[ G_c \]

\[ E_u: \text{the number of edges of } G_u \]
\[ \Delta: \text{the max vertex degree of } G_c \]

Cyclades samples updates \( u_i \) and finds conflict-groups.

Allocation: processing all the conflicting updates within the same core

Processing a batch is completed.

Theoretical Results: Cyclades on \( P = O \left( \frac{n}{\Delta \Delta_L} \right) \) cores, with batch sizes \( B = \frac{(1-\epsilon)n}{\Delta} \) can execute \( T = c \cdot n \) updates, for any constant \( c \geq 1 \), selected uniformly at random with replacement, in time \( O \left( \frac{E_u \lambda}{P} \cdot \log^2 n \right) \) with high probability, where \( \lambda \) is a constant.

[Pan, et al, NIPS-16]

2017/2/2

AAA1 2017 Tutorial 85
Data Allocation Strategy

• In theory - full access
  • Online accessing: $i_t \sim P$ i.i.d
  • Full access + with replacement sampling

• In practice - random shuffling + data partition + local sampling with replacement
  • One-shot SGD [Zhang et. al. 2012]
    • Shuffling + partition: statistical similarity
  • Other algorithms e.g. ADMM has convergence guarantee
Data Parallelism vs. Model Parallelism

1. Partition the training data
2. Parallel training on different machines
3. Synchronize the local updates
4. Refresh local model with new parameters, then go to 2.

1. Partition the model into multiple local workers
2. For every sample, the local workers collaborate with each other to perform the optimization
Model Parallelism

• Synchronous model parallelization

• Asynchronous model slicing

• Model parallelization with model block scheduling
Synchronous Model Parallelization

- The global model is partitioned into $K$ sub-models without overlap.
- The sub-models are distributed over $K$ local workers and serve as their local models.
- In each mini-batch, the local workers compute the gradients of the local weights by back propagation.
Synchronous Model Parallelization

• High communication cost: huge intermediate data
  • SGD-like algorithms require intermediate results for every data sample to be transferred between machines.
  • CNN: $O(10^9)$
    • $10^2$ imgs/mini-batch $\times 10^5$ patches/img $\times 10$ filters/patch $\times 10$ layers

• Sensitive to communication delay & machine failure
  • Speed differences among machines $\rightarrow$ slow down training.
  • Machine failure $\rightarrow$ break down training.
Asynchronous Model Slicing

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

- Parameters in the slice and hidden-node activations triggered by the slice are updated. When updating $Slice_1$, previous information about $Slice_2$ is reused.
- Other activations are retrieved from historical storage in local machine.
  - e.g., activations in DNN, Doc-topic table in LDA.

- When updating $Slice_2$, previous information about $Slice_1$ is reused.

$w_{i,j} \in slice_1$

$t_1$

$t_2$

Intermediate Data

Multiverso server

Client

Server

Stochastic Coordinate Descent (SCD)
Asynchronous Model Slicing

SCD and SGD have the same convergence rate for $\lambda$-strongly convex problem:

$$O\left(\frac{1}{\lambda \epsilon}\right)$$ iterations to obtain $\mathbb{E}[f(w^t) - f(w^*)] \leq \epsilon$.

### 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>
Model parallelization with model block scheduling

- Petuum’s Model Parallel Solution

- STRADS: Structure-aware Dynamic Scheduler [Jin Kyu Kim, et.al., 2016]
  - Scheduling on big model training
    - Blocks of variables are dispatched to workers based on a selection process
      - Consider the parameters’ converge status
    - Typically applicable to large scale logistic regression, matrix factorization and LDA model.
## DML Trends Overview

*Mostly about Data Parallelism*

<table>
<thead>
<tr>
<th>Components</th>
<th>Basic Research</th>
<th>Advanced Research</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential algorithms</td>
<td>Convex</td>
<td>Non-convex, faster algorithm</td>
</tr>
<tr>
<td>Data Allocation</td>
<td>Gap between theory and practice</td>
<td>Theoretical analysis of practical data allocation</td>
</tr>
<tr>
<td>Synchronization</td>
<td>BSP, ASP, etc.</td>
<td>Handling communication delay</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Model average</td>
<td>Other alternatives</td>
</tr>
<tr>
<td>Theory</td>
<td>Convergence</td>
<td>Generalization</td>
</tr>
</tbody>
</table>
Distributed ML Recent Advances

• Faster Stochastic Optimization Methods
• Non-convex Optimization
• Theoretical Analysis of Practical Data Allocation
• Handling Communication Delay
• Alternative Aggregation Methods
• Generalization Theory
SVRG

Initialize $\tilde{x}_0$
Iterate: for $s=1,2...$

\[ \tilde{x} = \tilde{x}_{s-1} \]
\[ \tilde{u} = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\tilde{x}) \]
\[ x_0 = \tilde{x} \]
Iterate: for $t = 1,2,...,m$
Randomly pick $i_t \in \{1,...,n\}$ and update weight
\[ x_t = x_{t-1} - \eta (\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{u}) \]
end
Option 1: set $\tilde{x}_s = x_m$
Option 2: set $\tilde{x}_s = x_t$ for randomly chosen $t$. end

**Extra computation:**
1. Full gradient $\tilde{u}$
2. Random gradient of $\tilde{x}$

VR-regularized Gradient:
\[ v_t = \nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{u} \]
\[ E_{i_t} ||v_t||^2 \leq 4L (f(x_{t-1}) - f(x^*) + f(\tilde{x}) - f(x^*)) \]
With $x_t \rightarrow x^*$, $E_{i_t} ||v_t||^2 \rightarrow 0$.

**Error Induction:**
\[ E[f(\tilde{x}_s) - f(x^*)] \leq \left[ \frac{1}{\alpha \eta (1 - 2\beta \eta)m} + \frac{2\beta \eta}{1 - 2\beta \eta} \right] E[f(\tilde{x}_{s-1}) - f(x^*)] \]
With $\eta = \frac{c}{\beta}, m = O(\frac{\beta}{\alpha})$

**Linear Convergence**

[Johnson&Zhang 2013]
Other VR Methods

SVRG: \[ x_{t+1} = x_t - \eta \left[ \nabla f_i(x_t) - \nabla f_i(\bar{x}) + \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\bar{x}) \right] \]

SAG: \[ x_{t+1} = x_t - \eta \left[ \frac{n}{n} \nabla f_i(x_t) - \nabla f_i(\phi^t_i) + \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\phi^t_i) \right] \] [Schmidt, Roux & Bach, 2013]

SAGA: \[ x_{t+1} = x_t - \eta \left[ \nabla f_i(x_t) - \nabla f_i(\phi^t_i) + \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\phi^t_i) \right] \] [Defazio, Bach & Lacoste-Julien, 2014]

Linear Convergence rate
Large storage: \( O(n) \)
Various Accelerations

- **SVRF** [Hazan&Luo, 2016]: Frank-Wolfe + VR
  convex: $O \left( \frac{1}{\epsilon^2} \right) \rightarrow O \left( \frac{1}{\epsilon^{1.5}} \right)$
  strongly convex: (Frank-Wolfe) $O \left( \frac{1}{\epsilon} \right) \rightarrow O \left( \log \frac{1}{\epsilon} \right)$

- **SVRG-ADMM** [Zheng&Kwok, 2016]: ADMM + VR
  strongly convex: (SCAS-ADMM) $O \left( \frac{1}{\epsilon} \right) \rightarrow O \left( \log \frac{1}{\epsilon} \right)$

- **APCG** [Lin, Lu&Xiao, 2014]: SCD + Prox + Nesterov
  strongly convex: $O \left( \left( nd + nd \cdot \max \frac{\beta_j}{\alpha} \right) \log \frac{1}{\epsilon} \right) \rightarrow O \left( \left( nd + nd \cdot \sqrt{\max \frac{\beta_j}{\alpha}} \right) \log \frac{1}{\epsilon} \right)$

- **Accelerated Mini-Batch Prox-SVRG** [Nitanda, 2014]: SGD + VR + Prox + Nesterov
  strongly convex: (SVRG) $O \left( \left( n + \frac{n \beta}{\alpha} \right) \log \frac{1}{\epsilon} \right) \rightarrow O \left( \left( n + \frac{n \beta}{n+\sqrt{\beta}} \right) \log \frac{1}{\epsilon} \right)$

- **MRBCD** [Zhao et. al., 2014]: SCD + VR + SGD
  smooth and strongly convex: (RBCD) $O \left( \frac{\max j \beta_j}{\alpha} \cdot \log \frac{1}{\epsilon} \right) \rightarrow O \left( nd + \frac{d \max j \beta_j}{\alpha} \cdot \log \frac{1}{\epsilon} \right)$

- **AASGD** [Meng et. al., 2016]: SGD + SCD + VR + Nesterov + Async
Asynchronous Accelerated SGD (AASGD)

- Variance Reduction (VR)
- Coordinate Sampling (CS)
- Nesterov’s Acceleration (NA)
- Asynchronous Parallelism

Key problem: Delayed gradients

\[ Model_{t+1} = Update (Model_t, Gradient_{t-\tau}) \]

Worker:
- Gradient Aggregation
- Master Parameter Update

Master:
- Gradient computing
- Push gradients
- Pull Master parameters
Convergence Rate of SASGD

\[
O\left(\frac{1}{\epsilon \mu_n}\right) + O\left(\left(\frac{mn + mL_{\text{res}}}{\mu_n}\right) \log \frac{1}{\epsilon}\right) = O\left(\left(\frac{mn + n mL_{\text{res}}/\mu_n}{n + \sqrt{L_{\text{res}}/\mu_n \sqrt{L_{\text{res}}/L_{\max}}}}\right) \log \frac{1}{\epsilon}\right)
\]

\(n\): data size
\(m\): # coordinate block
\(\mu_n\): convex coefficient
\(L, T\): smooth coefficients, \(T \geq L\)
Speedup of AASGD

Under some conditions for the delay and the data sparsity

\[ O\left( \frac{1}{\varepsilon \mu_n} \right) \quad O\left( mn + \frac{nmL_{res}/\mu_n}{n + \sqrt{L_{res}/\mu_n L_{res}/L_{max}}} \log \frac{1}{\varepsilon} \right) \quad O\left( mn + \frac{nmL_{res}/\mu}{n + \sqrt{L_{res}/\mu_n L_{res}/(\tau L_{max})}} \log \frac{1}{\varepsilon} \right) \]

- **n**: data size
- **m**: # coordinate block
- **\( \mu_n \)**: convex coefficient
- **\( L, T \)**: smooth coefficients, \( T \geq L \)
- **\( \tau \)**: upper bound of the independent delays
Distribute ML Recent Advances

• Faster Stochastic Optimization Methods
• Non-convex Optimization
• Theoretical Analysis of Practical Data Allocation
• Handling Communication Delay
• Alternative Aggregation Methods
• Generalization Theory
Non-convex Optimization

• Tasks:
  • Training of deep neural networks
  • Inference in graphical models
  • Unsupervised learning models, such as topic models, dictionary learning.

• In general, the global optimization of a non-convex function is NP-hard.

• Approaches:
  • Under some assumptions on the input for some models, we can design specific polynomial-time algorithms.
  • Robust, generic algorithms: simulated annealing, Bayesian optimization usually perform well in practice.
  • Attempt to reach a stationary point.
  • Graduated Optimization
Attempt to Reach a Stationary Point

• **SGD** [Ghadimi & Lan, 2015]
  
  Lipschitz: $O\left(n \left(\frac{L}{\epsilon} + \frac{L\sigma^2}{\epsilon^2}\right)\right)$

• **SVRG** [Allen-Zhu & Hazan, 2016]
  
  Lipschitz: $O\left(\frac{2}{n^3} \left(\frac{L}{\epsilon} + \frac{L\sigma^2}{\epsilon^2}\right)\right)$

• **APG** [Li & Lin, 2015]: Prox + Nesterov
  
  modification: descent $\Rightarrow$ sufficient descent
  
  Lipschitz: $O\left(\frac{1}{\sqrt{\epsilon}}\right)$

• **SAGA, PSG, Frank-Wolfe** [Sashank et. al., 2016]:
  
  Smooth: keep the same convergence rates with that in convex case.

Measure:

$$\min_{t=1,\ldots,T} \mathbb{E}||\nabla f(w_t)||^2 \text{ or } \frac{1}{T} \sum_t \mathbb{E}||\nabla f(w_t)||^2 \leq \epsilon(T)$$
Graduated Optimization

• Graduated optimization approach:
  1. Generate a coarse-grained version of the problem by a local smoothing operation:
  2. Solve the coarse-grained version
  3. Gradually refining the problem, with the solution of the previous stage as an initial point.

Given an L-Lipschitz function \( f: \mathbb{R}^d \rightarrow \mathbb{R} \) define its \( \delta \)-smooth version to be [Elad Hazan et. al. 2016]

\[
\hat{f}_\delta(x) = E_{u \sim B(0,1)}[f(x + \delta u)]
\]
Graduated Optimization

**Generalized mollifiers** $\{T_\sigma f : \sigma > 0\}$: [Caglar Gulcehre, Yoshua Bengio, et al, 2016]

$$\lim_{\sigma \to 0} T_\sigma f = f_0 \text{ is an identity function,}$$

$$\lim_{\sigma \to 0} T_\sigma f = f,$$

$$\frac{\partial (T_\sigma f)(x)}{\partial x} \text{ exists, } \forall x, \sigma > 0$$

**Noisy mollifier**: $(T_\sigma f)(x) = E_\xi [\phi(x, \xi_\sigma)]$

**Noisy Mollifiers for NN**: the random variable $\xi_\sigma$ yields noise to activations in each layer, except the output layer.

As $\sigma$ becomes larger:

- The nonlinearities $\to$ Linear
- Complex input-output mapping $\to$ Identity,
- Stochastic depth
Distribute ML Recent Advances

• Faster Stochastic Optimization Methods
• Non-convex Optimization
• Theoretical Analysis of Practical Data Allocation
• Handling Communication Delay
• Alternative Aggregation Methods
• Generalization Theory
Recall: Gap between Theory and Practice

• In theory - *full access*
  • Online accessing: $i_t \sim P$ i.i.d
  • Full access + with replacement sampling

• In practice - *random shuffling + data partition + local re-shuffling when the data is processed*
  • One-shot SGD [Zhang *et. al.* 2012]
    • Shuffling + partition: statistical similarity
  • Other algorithms e.g. ADMM has convergence guarantee
Without-Replacement Sampling

Objective function:

\[ F(w) = \frac{1}{m} \sum_{i=1}^{m} f_{\sigma(i)}(w) \]

where \( \sigma \) be a permutation over \( \{1, \ldots, m\} \) chosen uniformly.

Average loss with respect to the first \( t-1 \) and last \( m-t+1 \) loss functions:

\[ F_{1:t-1}(\cdot) = \frac{1}{t-1} \sum_{i=1}^{t-1} f_{\sigma(i)}(\cdot), \quad F_{t:m} = \frac{1}{m-t+1} \sum_{i=t}^{m} f_{\sigma(i)}(\cdot) \]

Theorem 1: Suppose the algorithm has a regret bound \( R_T \), and sequentially processes \( f_{\sigma(1)}(\cdot), \ldots, f_{\sigma(T)}(\cdot) \) where \( \sigma \) is a random permutation on \( \{1, \ldots, m\} \). Then in expectation over \( \sigma \),

\[
E \left[ \frac{1}{T} \sum_{t=1}^{T} F(w_T) - F(w^*) \right] \leq \frac{R_T}{T} + \frac{1}{mT} \sum_{t=2}^{T} (t-2)E[F_{1:t-1}(w_T) - F_{t:m}(w_t)]
\]

[Ohad Shamir NIPS 2016]
Convergence Rate via Transductive Rademacher Complexity

Definition: Let $\mathcal{V}$ be the set of vectors $\mathbf{v} = (v_1, ..., v_m)$. Let $s, u$ be positive integers such that $s + u = m$, and denote $p := \frac{su}{(s+u)^2} \in (0, 1/2)$. We define the **transductive Rademacher Complexity**

$$R_{s,u}(\mathcal{V}) = \left(\frac{1}{s} + \frac{1}{u}\right) \cdot E_{r_1,...,r_m} \left[ \sup_{\mathbf{v} \in \mathcal{V}} \sum_{i=1}^{m} r_i v_i \right],$$

where $r_1, ..., r_m$ are i.i.d. random variables such that

$$r_i = \begin{cases} 1 & \text{w.r.t } p \\ -1 & \text{w.r.t } p \\ 0 & \text{w.r.t } 1 - 2p \end{cases}$$

Theorem 2: Suppose that each $w_t$ is chosen from a fixed domain $\mathcal{W}$, that the algorithm has a regret bound $R_T$, and that $\sup_{i} |f_i(w)| \leq B$. Then in expectation over $\sigma$,

$$E\left[T \sum_{t=1}^{T} \left( F_{w_t} - F_{w_t}^* \right) \right] \leq R_T + \frac{1}{m} \sum_{t=1}^{T} R_{s,u}(\mathcal{V}) + \frac{2B}{m},$$

for linear model

Typically, $R_T = O(B \sqrt{T})$

Up to constants, the convergence rate of without-replacement is the same as that of with replacement. However, how about the distributed ML?
Distribute ML Recent Advances

• Faster Stochastic Optimization Methods
• Non-convex Optimization
• Theoretical Analysis of Practical Data Allocation
• Handling Communication Delay
• Alternative Aggregation Methods
• Generalization Theory
Challenges in Data-Parallel Training

- Sequential SGD:
  \[ w_{t+\tau+1} = w_{t+\tau} - \eta \cdot g(w_{t+\tau}) \]

- Async SGD:
  \[ w_{t+\tau+1} = w_{t+\tau} - \eta \cdot g(w_t) \]
SSP: Stale Synchronous Parallel

SSP: Bounded Staleness and Clocks

Convergence Rate: $O\left(\frac{2P(s+1)}{T}\right)$, where $P$ is the number of workers. They claimed that $2P(s + 1) \leq \tau$.

- SSP defines the delay based on clock on each local worker. And there will be no delay larger than $\tau$ based on this kind of logic.
- But later on people define delay based on parameter server side update times. To be more specific, delay means # of update happened on the PS side from the when local mode is got to when the local update is applied on PS.

Each local worker has its clock
C : current clock
S : the max delay between slowest and fastest workers

[AAAI 2017 Tutorial] [NIPS 2013]
AdaDelay: Delay Adaptive Distributed Stochastic Convex Optimization

• Motivation:
  • Larger delay gets smaller learning rate
  • Extend AdaGrad to asynchronous parallelism

• AdaDelay Learning Rate:
  • $\alpha(t, \tau_t) = \left( L + \eta(t, \tau_t) \right)^{-1}$,
  • $\eta_j(t, \tau_t) = c_j \sqrt{(t + \tau_t)}$
  • $c_j = \frac{1}{t} \sum_{i=1}^{t} \frac{i}{i + \tau_i} g_j^2(i - \tau_i) : \text{accumulation of the weighted delayed gradients on feature } j$

• Convergence Rate:
  • Assumption: delay is uniform distributed $\mathbb{E}\tau_t = \tau$
  • $O\left(\frac{\tau}{\sqrt{T}} + \frac{\tau \log T}{T}\right)$

• A similar work: Delay-Tolerant Algorithms for Asynchronous Distributed Online Learning (NIPS 2014)

Delay Compensation

\[ g(w_{t+\tau}) = g(w_t) + \nabla g(w_t) \cdot (w_{t+\tau} - w_t) + O(\|w_{t+\tau} - w_t\|^2) \]

\[ \nabla g(w_t) \text{ corresponds to the Hessian matrix} \]

**Theorem:** Assume that loss function \( L(x, y; w) \) is cross-entropy loss for neural networks, then the second-order derivatives \( \frac{\partial^2}{\partial w^2} L(X, Y; w) \) can be estimated in an unbiased manner by the outer-product of first-order derivatives, i.e.,

\[ E(y|x, w) \frac{\partial^2}{\partial w^2} L(X, Y; w) = E(y|x, w) \left( \frac{\partial}{\partial w} L(X, Y; w) \cdot \frac{\partial}{\partial w} L(X, Y; w)^\tau \right) \]
Delay Compensated ASGD (DC-ASGD)

ASGD: \( w_{t+\tau+1} = w_{t+\tau} - \eta g(w_t) \)

DC-ASGD:
\[
 w_{t+\tau+1} = w_{t+\tau} - \eta g(w_t) - \lambda (g(w_t) \cdot g(w_t)^T)(w_{t+\tau} - w_t)
\]

A work that directly targets to handle the delay, and it is experimentally effective and with convergence analysis.

Training curve for a Resnet DNN model for cifar-10

https://arxiv.org/abs/1609.08326
Asynchrony begets Momentum

Momentum SGD:
\[ w_{t+1} = w_t - \alpha_t \nabla_w f(w_t; z_{i_t}) + \mu (w_t - w_{t-1}) \]

Asynchronous SGD:
\[ w_{t+1} = w_t - \alpha_t \nabla_w f(w_{t-\tau_t}; z_{i_t}), \text{ where } \tau_t \sim Q \]

They are equivalent, under the expectation over certain distribution \( Q \).

Theorem: Let the staleness distribution be geometric on \( \{0, 1, \cdots\} \) with parameter \( 1 - \mu \). The expected update with learning rate \( \alpha \) of ASGD takes the momentum form:
\[ \mathbb{E}[w_{t+1} - w_t] = \mu \mathbb{E}[w_t - w_{t-1}] - (1 - \mu) \alpha \mathbb{E}\nabla_w f(w_t) \]

Consider \( M \) asynchronous workers and \( C_t \sim \text{Exp}(\lambda) \), where \( W_t \) denotes the time it takes step \( t \) to finish, then
\[ \mathbb{E}[w_{t+1} - w_t] = \left( 1 - \frac{1}{M} \right) \mathbb{E}[w_t - w_{t-1}] - \frac{\alpha}{M} \mathbb{E}\nabla_w f(w_t) \]

[Mitliagkas, et al, Allerton-16]
Distribute ML Recent Advances

• Faster Stochastic Optimization Methods
• Non-convex Optimization
• Theoretical Analysis of Practical Data Allocation
• Handling Communication Delay
• Alternative Aggregation Methods
• Generalization Theory
Ensemble-Compression based Aggregation

Ensemble model

1/K

Compress

Compressed model (=Model’)

Model 1

Model K

Ensemble model

1/K

Communicate and Aggregation

Model

SGD → Model 1

SGD → Model 2

... → Model K

Model’

SGD → ...

SGD → ...

SGD → ...

[Sun et. al. 2016]
PV-Tree: Voting based Aggregation

1. Globally select \( k \) features from all local proposed features by majority voting.
2. Collect the histogram for these \( k \) features for global best split selection.

Steps in parallel

FastRank

Communication cost

Find best split point \( O(k \times \#\text{bin}) \)

Split the tree nodes No
Distribute ML Recent Advances

- Faster Stochastic Optimization Methods
- Non-convex Optimization
- Theoretical Analysis of Practical Data Allocation
- Handling Communication Delay
- Alternative Aggregation Methods
- Generalization Theory
Motivation

**Generalization Ability of Machine Learning Algorithms**

\[
\min_{f \in F} R(f) = E_{Z \sim P} l(f, Z)
\]

\[
\min_{f \in F} R_S(f) = \frac{1}{n} \sum_{i=1}^{n} l(f, Z_i)
\]

- \( P \) is unknown
- No closed-form solution
- Optimization algorithms: Convergence rate

Optimization algorithm’s generalization error?
Tradeoffs in Large Scale Learning \cite{Buttou&Bousquet, NIPS08}

Error decomposition:
\[
\epsilon = \epsilon_{app} + \epsilon_{est} + \epsilon_{opt} \leq c \left( \epsilon_{app} + \left( \frac{d}{n} \log \left( \frac{n}{d} \right) \right)^\alpha + \rho \right)
\]

Tradeoffs:
\[
\rho \sim \left( \frac{d}{n} \log \left( \frac{n}{d} \right) \right)^\alpha \rightarrow n \sim \frac{d}{\rho^\alpha} \log \left( \frac{1}{\rho} \right)
\]

\(d\): VC dimension
\(\alpha\): noise condition

---

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cost of one iteration</th>
<th>Iterations to reach (\rho)</th>
<th>Time to reach accuracy (\rho)</th>
<th>Time to reach (\epsilon \leq c (\epsilon_{app} + \epsilon))</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD</td>
<td>(O(nd))</td>
<td>(O(\kappa \log \frac{1}{\rho}))</td>
<td>(O(n \kappa \log \frac{1}{\rho}))</td>
<td>(O\left( \frac{d^2 \kappa^2}{\epsilon^{1/\alpha}} \log^2 \frac{1}{\epsilon} \right))</td>
</tr>
<tr>
<td>2GD</td>
<td>(O(d^2 + nd))</td>
<td>(O(\log \log \frac{1}{\rho}))</td>
<td>(O((d^2 + nd) \log \log \frac{1}{\rho}))</td>
<td>(O\left( \frac{d^2 \kappa^2}{\epsilon^{1/\alpha}} \log^2 \frac{1}{\epsilon} \log \log \frac{1}{\epsilon} \right))</td>
</tr>
<tr>
<td>SGD</td>
<td>(O(d))</td>
<td>(\frac{\nu \kappa^2}{\rho} + O\left( \frac{1}{\rho} \right))</td>
<td>(O\left( \frac{\nu \kappa^2}{\rho} \right))</td>
<td>(O\left( \frac{d \nu \kappa^2}{\epsilon} \right))</td>
</tr>
<tr>
<td>2SGD</td>
<td>(O(d^2))</td>
<td>(\frac{\nu}{\rho} + O\left( \frac{1}{\rho} \right))</td>
<td>(O\left( \frac{d \nu}{\rho} \right))</td>
<td>(O\left( \frac{d \nu}{\epsilon} \right))</td>
</tr>
</tbody>
</table>

[Hardt et.al., ICML-16]

[Qi Meng et.al. AAAI-17]
3. Distributed Machine Learning: Systems and Toolkits
Distributed Machine Learning Architectures

- **Asynchronous**
  - Flexible in modeling dependency
  - Lack of good abstraction

- **Synchronous**
  - High-level abstractions (MapReduce)
  - Lack of flexibility in modeling complex dependency

- **Data Parallelism**
  - Iterative MapReduce

- **Model Parallelism**
  - Parameter Server
  - Petuum, DMLC

- **Irregular Parallelism**
  - Dataflow
  - TensorFlow

- **Support hybrid parallelism and fine-grained parallelization, particularly for deep learning**
- **Good balance between high-level abstraction and low-level flexibility in implementation**
Representative Systems and Toolkits

• Iterative MapReduce – Spark MLlib

• Parameter Server – DMTK

• Data Flow – TensorFlow
What is Apache Spark?

- Apache Spark is a fast and general engine for large-scale data processing
  - 100x faster than Hadoop
Motivation

• *Iterative* machine learning and *interactive* data mining
• MapReduce is inefficient in *reusing* intermediate results
• Spark is for fast in memory cluster computing
Programming Model

• Resilient Distributed Datasets (RDDs)
  • Read-only, collection of distributed objects, can be rebuilt

• RDD Parallel Operations
  • Transformation and action on RDD
    • *map / filter / reduce / ...*
MLLib

• Spark’s machine learning library

• Algorithms:
  • Classification: logistic regression, naïve Bayes
  • Regression: generalized linear regression, isotonic regression
  • Decision trees, random forests, and gradient-boosted trees
  • Recommendation: alternating least squares (ALS)
  • Clustering: K-means, Gaussian mixtures
  • Topic modeling: latent Dirichlet allocation
Execution

1. **Driver**
   - Initialize weight
   - Read data partition
   - Calculate local gradient
   - Read data partition
   - Calculate local gradient

2. **Mapper**
   - Communication
   - Synchronization

3. **Aggregator**
   - Aggregate global gradient
   - Update weights

4. **Big Model**
   - Next iteration
Pros and Cons

• Pros
  • Simple interface and clean logic (MapReduce)
  • High level programming language (Python, Scala, Java, SparkR)
  • Convergence guarantee

• Cons
  • Not flexible enough for detailed optimization, e.g., pipelining
  • Only support synchronous mode, might be very inefficient on large-scale heterogeneous cluster (when there are stragglers)
  • Only support data parallelism, unclear how to support big models
Representative Systems

• Iterative MapReduce - Spark

• Parameter Server – DMTK

• Data Flow – TensorFlow
Parameter Server

• A specialized distributed key-value store for Machine Learning
• A group of servers managing shared parameters for a group of data-parallel workers
## Comparison with KV Store in DataBase

<table>
<thead>
<tr>
<th>KV store</th>
<th>DataBase</th>
<th>Distributed ML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data model</td>
<td>Any data</td>
<td>(Multi-d) (Sparse) array</td>
</tr>
<tr>
<td>Write access</td>
<td>Write/Update/Append</td>
<td>associative and commutative combiner</td>
</tr>
<tr>
<td>Consistency</td>
<td>Strictly consistent</td>
<td>ASP is accepted</td>
</tr>
<tr>
<td>Usage</td>
<td>Highly available online service</td>
<td>Offline data training</td>
</tr>
<tr>
<td>Replication</td>
<td>Must have</td>
<td>Typically no</td>
</tr>
<tr>
<td>Example</td>
<td>Dynamo/Redis/Memcached</td>
<td>Multiverso/PS/Petuum</td>
</tr>
</tbody>
</table>
Parameter Server for Distributed ML

- Scalable topic models and deep neural networks
- General purpose parameter server

<table>
<thead>
<tr>
<th>Year</th>
<th>LDA</th>
<th>YahooLDA</th>
<th>Peacock</th>
<th>DistBelief</th>
<th>Adam</th>
<th>MxNet</th>
<th>Paddle</th>
<th>Piccolo</th>
<th>Petuum</th>
<th>PS</th>
<th>Multiverso</th>
<th>GeePS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2015</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Asynchronous training
Break the Dependency

- Asynchronous

Approximation in two aspects:
1) Model replicas run independently of each other
2) PS shards run independently of one another
PS for DNN with GPU Cluster

• The mismatch in speed between GPU compute and network interconnects makes it extremely difficult to support data parallelism via a parameter server (ADAM 2014)

• GeePS
  • Big model – model swapping
  • PS is still in CPU memory
DMTK/Multiverso

• A distributed machine learning framework that supports
• [http://github.com/Microsoft/dmtk](http://github.com/Microsoft/dmtk)

- **Big Data**
  - Both synchronous and asynchronous data parallelism with a unified mechanism.
- **Big Model**
  - Super big models, through model scheduling and adaptive pipelining.
- **Flexibility**
  - Supports various cluster like MPI/Yarn/Azure.
- **Efficiency**
  - Training big models on big data with reasonable time.
Unified Data Parallelism Interface

Unified Interface:
- **Add(index \( i \), delta \( \Delta W_i \), coeff \( c \))**:
  - \( W^{(t+1)} = W^{(t)} + \sum_{i=1}^{n} c \cdot \Delta W_i \),
  - \( c = 1 \) for BSP/ASP/SSP;
  - \( c = 1/n \) for MA/ADMM.
- **SetDelay(delay)**:
  - \( delay = 0 \): BSP, MA, ADMM;
  - \( delay > 0 \): SSP;
  - \( delay = \infty \): ASP.

C++/C/python/Lua interfaces are supported, so that many algorithm lib users can benefit from using this tool. E.g. parallel Theano / Torch to work in multiple node scenario.
Model Slicing

Each document is fixed onto a worker machine; there is no need to share document-topic table across machines.

Model are cut into slices, and we send model to data instead of sending data to model during training.

- Different model slices are pulled from parameter server and trained in a round robin fashion; data samples and intermediate results are locally stored.
- Lower communication cost than standard model parallelism: data are much larger than model and model becomes increasingly sparse during the training process.

Dictionary is shuffled to ensure load balancing at model-slice level;
BOW representations are sorted according to word ID to facilitate sliced data sweeping.
Communication Delay Allows Pipelining

Use pipelining to overlap data/model loading and computation.
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)
Customizable Model Representation and Aggregations

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

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

Class ParallelModel: IAggregation
{
    public virtual bool Aggregate(void* models, 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.
Rich Parallel Optimization Libraries

Asynchronous algorithms

- Hogwild! & Downpour SGD (ASGD)
- Stale Synchronous Parallel
- EASGD Async version
- DC-ASGD
- ...

Synchronous algorithms

- Parallel SGD
- ADMM
- Model Average
- BMUF
- ...

2017/2/2
AAAI 2017 Tutorial
GPU support

• Client side pipeline on GPU side to enhance throughput
• Hide communication latency.
Representative Systems

• Iterative MapReduce - Spark

• Parameter Server – DMTK

• Data Flow – TensorFlow
Overview

• Data Flow
• Graph represent mathematical operations
• Support wide variety hardware.
  • E.g. CPU, GPU, mobile phone
• Support data and model parallelism
  • can parallel using hybrid devices
• python and C++ front end
• Automatic gradient computation
• Partial Execution
Computational Graph

• Describe mathematical computation with a direct graph of nodes & edges
  • Nodes in the graph represent mathematical operations (included variables)
  • Edges describe the i/o relationships between nodes, carry dynamically-sized multidimensional data arrays (tensors)

```python
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))
W = tf.Variable(tf.random_uniform([784, 100], -1, 1))
x = tf.placeholder(name="x")
relu = tf.nn.relu(tf.matmul(W, x) + b)
C = [...]  # 100-D vector, init to zeroes
# 784x100 matrix w/rnd vals
# Placeholder for input
# Relu(Wx+b)
# Cost computed as a function
# of Relu
```
Execution

• **Single-Device** Execution (on single machine)
  • executed in order (that respects the dependencies between nodes)

• **Multi-Device** Execution (on single machine)
  • node placement (Greedy approach) (also support self-defined devices)
  • cross-device communication
    • Add virtual send/recv node at border

• **Distributed** Execution
  • similar to multi-device execution
  • cross-device communication is based on gRpc
  • fault tolerance (checkpoint)
Node Placement

• Run a simulated execution
  • *Greedy* heuristics
  • Start with the *sources* of the graph
  • For each node, choose the device where the node’s operation would *finish the soonest* (estimate by cost model)

• Cost model
  • *Computation* time
  • *Communication* time

• But it seems this algorithm is not used in code base
  • “The placement algorithm is an area of ongoing development within the system”
Data Parallelism and Model Parallelism
Synchronous Replica Coordination

(a) Asynchronous replication
(b) Synchronous replication
(c) Synchronous w/ backup worker

Figure from TensorFlow’s white paper.
Parameter Server vs. TensorFlow

- TensorFlow represent “Parameter Server” as parts of Dataflow graph

<table>
<thead>
<tr>
<th>Architecture</th>
<th>TensorFlow</th>
<th>Parameter Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataflow graph</td>
<td></td>
<td>Distributed key-value store</td>
</tr>
<tr>
<td>Shared state</td>
<td>Represented as vertex(Operator) in graph, like computation</td>
<td>Stored in PS</td>
</tr>
<tr>
<td>Parameter placement</td>
<td>Can be specified by user</td>
<td>Implemented as a hash function in PS</td>
</tr>
<tr>
<td>Consistency control logic</td>
<td>Implemented with high-level language by combining existing operations</td>
<td>Implemented sync logic in server side</td>
</tr>
<tr>
<td>Optimization algorithm</td>
<td>Implemented as operator python optimizer library</td>
<td>Implemented update logic in server side</td>
</tr>
</tbody>
</table>
Overall Structure

• **TensorFlow**
  - In architecture level, TensorFlow implements a dataflow models to present both local computations and distributed extensions.

- **Computation Infrastructure**
  - Parallel execution engine based on gRPC, packet message, communication, etc.

- **Computation allocation method**
  - Manual assign to devices vs. automatic conduct optimized allocation

- **parameter server node logic**
  - Inherent all research output from parameter server side.
Real Examples over Different Systems

• Machine learning systems
  • Spark MLlib
  • DMTK - Parameter server
  • TensorFlow

• Tasks
  • Logistic regression over three different systems
  • DNN on DMTK+CNTK & TensorFlow (Resnet).
Example – Logistic Regression

val points = spark.textFile(...) .map(parsePoint).persist()

var w = Vector.random(D) // random initial vector

for (i <- 1 to ITERATIONS) {
  val gradient = points.map{ p =>
    p.x * (1/(1+exp(-p.y*(w dot p.x))))-1)*p.y
  }.reduce((a,b) => a+b)
  w -= gradient
}

Create RDD

Map operation: compute gradient

Reduce operation: aggregate gradient
import multiverso as mv
import numpy as np

mv.init(sync=true, updater="adagrad")

feature = np.random.rand(m, n) // synthetic data
label = np.random.rand(n)

model = mv.create_table()

for iter in range(0, ITRATIONS):
    w = model.get()
    gradient = (label - sigmoid(w * feature)) * feature
    model.add(gradient)

mv.shutdown()
Example – Logistic Regression

import tensorflow as tf

cluster_spec = tf.train.ClusterSpec({
    "ps": ["ps0:2222"],
    "worker": ["worker0:2222", "worker1:2222", "worker2:2222"]})
Server = tf.train.Server(cluster, job_name, task_index)

X = tf.placeholder(tf.float32, data_shape)
Y = tf.placeholder(tf.float32, label_shape)

if job_name == "ps":
    server.join()
elif job_name == "worker":
    with tf.device(tf.train.replica_device_setter(cluster = cluster_spec)):
        W = tf.Variable(tf.random_normal(weight_shape))
        loss = tf.reduce_mean(
            tf.nn.softmax_cross_entropy_with_logits(tf.matmul(X, W), Y))
        global_step = tf.Variable(0)
        train_op = tf.train.AdagradOptimizer(0.01).minimize(loss, global_step=global_step)
        init_op = tf.global_variables_initializer()

sv = tf.train.Supervisor(init_op = init_op, global_step= global_step)
with sv.managed_session(server.target) as sess:
    while not sv.should_stop() and step < 1000:
        _, step = sess.run([train_op, global_step])
sv.stop()
Example – DMTK+CNTK on ResNet [K. He, et.al., 2016]

Create model

Training loop

Setting hyper-parameter

Create distributed learner for parallel training

Test loop
Example – Tensorflow ResNet

```python
# Create a cluster from the parameter server and worker hosts.
cluster = tf.train.ClusterSpec({'worker': worker_hosts})
create server instance
# Create a server.
server = tf.train.Server(cluster,
    job_name=job_name,
    task_index=task_index)

with tf.device('/job:worker/task:%d' % (args.task_index),
    ps_device='/job:worker/task:0/cpu:0',
    cluster=cluster):
    feature = tf.placeholder(tf.float32, data_shape)
    label = tf.placeholder(tf.int32, label_shape)
    global_step = tf.Variable(0, name='global_step', trainable=False)
    optimizer = tf.train.GradientDescentOptimizer(args.lr)
    replicas_to_aggregate = len(worker_hosts)
    global_opt = tf.train.SyncReplicasOptimizer(
        optimizer,
        replicas_to_aggregate=replicas_to_aggregate,
        total_num_replicas=len(worker_hosts),
        replica_id=task_index,
        name='mnist_async_replicas')
    subMinibatchGradients = []
    # build model in local used GPUs
    for i in used_gpus:
        with tf.device('/gpu:%d' % i):
            with tf.name_scope('replica_%d' % (i)) as scope:
                # build resnet model
                last_layer = build_resnet_model(feature)
                cross_entropy =
                    tf.nn.sparse_softmax_cross_entropy_with_logits(
                        logits=last_layer, label)
                loss = tf.reduce_mean(cross_entropy)
                if is_chief:
                    # Chief worker will start the chief queue runner and call the init op
                    print('Starting chief queue runner and running init_op')
                    sv.start_queue_runners(sess, [chief_queue_runner])
                if is_chief:
                    sess.run(init_op)
                # start to train
                sess.run([train_step, global_step],
                    feed_dict={feature: feature_in, label: label_in})
```
Summary on Distributed Machine Learning Systems

• Flexibility and User Friendliness help the AI developers
  • Enclose all distributed algorithm details in high level abstractions to let user just use distributed training
  • Expose simple but strong interface to advanced users to let them design their own distributed algorithm

• Efficiency is key for solving big learning tasks
  • Leverage advanced hardware and software, e.g., GPU/RDMA
  • Use efficient sync up logic, e.g., Asynchronous algorithm + pipelining
  • Choose framework with less overhead cost, e.g., MapReduce v.s., parameter server

• Parallel Optimization method determine the accuracy
  • Apply advanced optimization algorithm in distributed training
  • SGD / SCD / Variance Reduction / Delay handling
Thanks

Contact: tyliu@microsoft.com