4 Applications:
User Modeling and Graph Factorization

Amr Ahmed, Alexander J Smola, Markus Weimer
Yahoo! Research & UC Berkeley & ANU
Wrapping up

• Distributed inference in latent variable models
  – Star Synchronization
  – Delta aggregation
Wrapping up ...

Prior over topic mixtures

Topic-word distribution

\[ \alpha \]

\[ \theta \]

\[ z \]

\[ w \]

\[ \phi \]
Wrapping up ...

- Global variables
  - $\Phi$: Topic distribution over words

- Local variables
  - $\theta$: topic mixing vector
  - $Z$: topic indicator
Wrapping up ...

• Collapse global variables
  – \( \Phi \)
• Collapse local variables
  – \( \theta \)
• Couples all Zs
• Run collapsed sampler

\[
P(z_{di} = k|w_{di} = w, z_{-di}) \propto(n_{dk} + \alpha) \frac{n_{kw} + \beta}{n_k + W\beta}
\]
Wrapping up ...

Global State

\[ n_{kw}, n_k \]

Local counts (local state)

\[ P(z_{di} = k | w_{di} = w, z_{-di}) \propto \]

\[ (n_{dk} + \alpha) \frac{n_{kw} + \beta}{n_k + W\beta} \]

Global counts (global state)
Distributed Inference: LDA
Distributed Inference: LDA
Distributed Inference: LDA

Global State

\( n_{kw}, n_k \)
Distributed Inference: LDA

Global State

\( n_{kw}, n_k \)

Global replica

\( z \)

\( w \)

Global replica

\( z \)

\( w \)
General Architecture

• Star synchronization
  – Works when variables depend on each other via aggregates
    • Counts, sums, etc.
  – When state objects form an Abelian group
Template

• Fit most topic models in collapsed representation
  – Define the state (key, value) pairs
    • Mostly counts, sums, lists, hash tables
  – Define the +,- operations on a state object
  – Write your sampler
    • Input: document, state
    • Output:
      – Update document local variables
      – Update the global state

• Our API will take care of the rest
  – Synchronization, threading, distribution, etc
Distributed Inference: template

Global State: (key, value)
State Example: LDA

- **Alternative 1**
  - Key: (topic, word)
  - value: count
  - Operators:
    - +,- are trivially defines

- **Alternative 2**
  - Key: word
  - value: list of (topic, count)
    - Allows efficient samplers
  - Operators: sparse vector operations
    - Might need to delete and merge

\[
P(z_{di} = k | w_{di} = w, z_{-di}) \propto \frac{(n_{dk} + \alpha)(n_{kw} + \beta)}{n_k + W\beta}
\]
State Example: LDA

• You get the idea?
• Define the state to work with your sampler
• Define +,- for synchronization
• All details are abstracted form the synchronization logic
  – It just uses the +,- operators your just defined
  – Requires an iterator over state objects
Example 2: Multilingual LDA

- Each topic has a distribution over words
- Fits parallel documents
• Alternative 1
  – Key: (topic, language, word)
  – value: count
  – Operators: +,- are trivially defines

• Alternative 2
  – Key: word
  – value: list of (topic, language, count)
    • Allows writing efficient samplers
  – Operators: Sparse vector operations
    • Might need to delete and merge
State Example: Clustering

• Alternative 1
  – Key: Cluster ID
  – value:
    • Document counts
    • Parameter representation
      – Hash table: (word, count)

– Operations
  • Define +,- over each field
  • You write this code
  • Part of the application logic
  • You have to do it anyhow when:
    – Remove or add a document to a cluster
API Summary

• Template for distributed inference in latent variables models

• Two basic components
  – Document representation
    • You take care of that via Protocol Buffer
  – State representation
    • Key-value pairs
    • Value can be any object
      – Define +,- over that object
    • Provide an iterator over objects for the synchronizer
class stats{
public:
  virtual ~stats() { };
  virtual void from_str(const string& serialized_stats) = 0;
  virtual void to_str(string& serialized_stats) = 0;

  virtual void operator+=(stats& inp) = 0;
  virtual void operator-=(stats& inp) = 0;

  virtual int get_id() { return 0; }
  virtual void set_id(int) { }

  virtual void print() { }
};

typedef auto_ptr<stats> stats_ptr;
class stats_container{
public:
  virtual ~stats_container() { };

// copy operator
  virtual void from_stats_container(stats_container&) = 0;

// lock up operator, get stat object with a given id
  virtual stats_ptr get_stats(int id) = 0;

// update a state object with a give id
  virtual void update(int id, stats& delta) = 0;

virtual int size() = 0;

// iterator
  virtual bool has_next() = 0;
  virtual stats_ptr next() = 0;
  virtual void reset_iter() = 0;

virtual void print() = 0;
};
message LDA_document {
  optional string docID = 1;
  repeated uint32 body = 3 [packed=true]; // w
  repeated uint32 topic_assignment = 4 [packed=true]; // Z
  repeated uint32 topic_counts = 5 [packed=true]; // n_{dk}
}

message clustering_document {
  optional string docID = 1;
  repeated uint32 words = 2; // w
  repeated uint32 label = 3; // cluster assignment
}
class Model_Trainer {
public:
  virtual ~Model_Trainer() { };
  virtual void* read(google::protobuf::Message&) = 0;

  // That is where you write your logic
  virtual void* sample(void* document) = 0;

  // Call in inference mode
  virtual void* test(void* document) = 0;

  // fold an update into the state
  virtual void* update(void* document) = 0;

  // time for synchronous operations
  virtual void* optimize(void*) = 0;

  // diagnostic
  virtual void* eval(void*, double&) = 0;

  // save
  virtual void write(void*) = 0;

  // need more iterations?
  virtual void iteration_done() = 0;
};
API Summary

• Current Yahoo! LDA release
  – Tightly integrates state, sampler and synchronization
  – Stay tuned for a new release with the new APIs
What Is next?

• Can we fit any model only with those asynchronous primitives?
  – No

• We need synchronous operations
  – Parameter optimization
    • EM style algorithm
  – Non-collapsed global variables
The Need for Synchronous Processing

Prior over topic mixtures

What if we need to optimize over $\alpha$?

Topic-word distribution
The Need for Synchronous Processing

- **E-Step**
  - Run *asynchronous* collapsed sampler as before

- **M-step**
  - Reach a barrier
  - Collect values needed to optimize \( \alpha \)
  - One machine optimizes \( \alpha \)
  - Broadcast value back
Distributed Sampling Cycle

Sample Z

Sample Z

Sample Z

Sample Z

Asynchronous

Optimize $\alpha$

Requires a reduction step

Synchronous
Distributed Sampling Cycle

- Sample Z
- Write counts
- Collect counts and optimize
- Read \( \alpha \)

This template will re-occur today in several applications.
• Application
  – Temporal Modeling of user interests
  – Multi-domain user personalization

• Asynchronous Distributed Optimization
  – Can we get rid of the synchronous step?
  – Asynchronous consensus
  – Factorizing Y!M graph
    • 200 Million users and 10 Billion edges
    • The largest published work on graph factorization
Modeling User Interests

**User-1**

- Dating
  - women
  - men
  - dating
  - singles
  - personals
  - seeking
  - match
- Baseball
  - League
  - baseball
  - basketball, doublehead
  - Bergesen
  - Griffey
  - bullpen
  - Greinke
- Celebrity
  - Snooki
  - Tom
  - Cruise
  - Katie
  - Holmes
  - Pinkett
  - Kudrow
  - Hollywood
- Health
  - skin
  - body
  - fingers
  - cells
  - toes
  - wrinkle
  - layers

**User-2**

- Baseball
  - Finances
  - Jobs
- Dating

**Finance**

- financial
- Thomson
- chart
- real
- Stock
- Trading
- currency
Multi-domain Personalization
Graph Factorization: Social Network
Characterizing User Interests

- Short term vs long-term

- Music
- Housing
- Buying a car
- Furniture
- Travel plans
Characterizing User Interests

- Short term vs long-term
- Latent
Problem formulation

**Input**
- Queries issued by the user or tags of watched content
- Snippet of page examined by user
- Time stamp of each action (day resolution)

**Output**
- Users’ daily distribution over interests
- Dynamic interest representation
- Online and scalable inference
- Language independent

Flight
London
Hotel
weather
classes
registration
housing
rent
School
Supplies
Loan
semester
## Problem formulation

### Input

- Queries issued by the user or tags of watched content
- Snippet of page examined by user
- Time stamp of each action (day resolution)

### Output

- Users’ daily distribution over interests
- Dynamic interest representation
- Online and scalable inference
- Language independent
Problem formulation

When to show a financing ad?
When to show a financing ad?
Problem formulation

When to show a financing ad?

- Flight
- London
- Hotel
- weather
- Travel
- classes
- registration
- housing
- rent
- School Supplies
- Loan
- semester

finance

Back to school
When to show a hotel ad?
When to show a hotel ad?
Problem formulation

Input

• Queries issued by the user or tags of watched content
• Snippet of page examined by user
• Time stamp of each action (day resolution)

Output

• Users’ daily distribution over interests
• Dynamic interest representation
• Online and scalable inference
• Language independent
1. Draw once $\Omega|\alpha \sim \text{Dir}(\alpha/K)$.
2. Draw each topic $\phi_k|\beta \sim \text{Dir}(\beta)$.
3. For each user $i$:
   (a) Draw topic proportions $\theta_i|\lambda, \Omega \sim \text{Dir}(\lambda \Omega)$.
   (b) For each word
      (a) Draw a topic $z_{ij}|\theta_d \sim \text{Mult}(\theta_i)$.
      (b) Draw a word $w_{ij}|z_{ij}, \phi \sim \text{Multi}(\phi_{z_{ij}})$. 
In Polya-Urn Representation

- Collapse multinomial variables: $\theta, \Omega$
- Fixed-dimensional Hierarchical Polya-Urn representation
  - Chinese restaurant franchise
Global topics trends

Food Chicken

Topic word-distributions

User-specific topics trends (mixing-vector)

User interactions: queries, keyword from pages viewed

Recipe Chocolate Pizza Food Chicken Milk Butter Powder

Car Blue Book Kelley Prices Small Speed large

job Career Business Assistant Hiring Part-time Receptio nist

Bank Online Credit Card debt portfolio Finance Chase
Generative Process

- For each user interaction
  - Choose an intent from local distribution
    - Sample word from the topic’s word-distribution
    - Choose a new intent \( \propto \lambda \)
    - Sample a new intent from the global distribution
      - Sample word from the new topic word-distribution
Generative Process

- For each user interaction
  - Choose an intent from local distribution
    - Sample word from the topic’s word-distribution
  - Choose a new intent $\sim \lambda$
    - Sample a new intent from the global distribution
      - Sample word from the new topic word-distribution
For each user interaction:
- Choose an intent from local distribution
  - Sample word from the topic’s word-distribution
- Choose a new intent $\sim \lambda$
  - Sample a new intent from the global distribution
    - Sample word from the new topic word-distribution
Generative Process

• For each user interaction
  • Choose an intent from local distribution
    • Sample word from the topic’s word-distribution
  • Choose a new intent $\propto \lambda$
    • Sample a new intent from the global distribution
      • Sample word from the new topic word-distribution
Generative Process

• For each user interaction
  • Choose an intent from local distribution
    • Sample word from topic’s word-distribution
  • Choose a new intent $\propto \lambda$
    • Sample a new intent from the global distribution
  • Sample from word the new topic word-distribution
**Problems**

- Static Model
- Does not evolve user’s interests
- Does not evolve the global trend of interests
- Does not evolve interest’s distribution over terms
At time t

Food  Chicken  pizza  millage

At time t+1

Recipe  Chocolate  Pizza  Food  Chicken  Milk  Butter  Powder

Car  Blue  Book  Kelley  Prices  Small  Speed  large

job  Career  Business  Assistant  Hiring  Part-time  Receptio  

nist

Bank  Online  Credit  Card  debt  portfolio  Finance  Chase

Build a dynamic model

Connect each level using a RCRP
Which time kernel to use at each level?
At time t

- Popular topics at time t are likely to be popular at time t+1
  \(- \phi_{k,t+1} \) is likely to smoothly evolve from \( \phi_{k,t} \)

At time t+1

- Pseudo counts

Food
  - Chicken
  - Pizza
  - Millage

Car
  - Speed
  - Offer
  - Camry
  - Accord
  - Career

Job
  - Online
  - Credit
  - Card
  - Debt
  - Portfolio
  - Finance
  - Chase

Recipe
  - Chocolate
  - Pizza
  - Food
  - Chicken
  - Milk
  - Butter
  - Powder

Decay factor

\( \exp \frac{-1}{\lambda} \)
Observation 1

- Popular topics at time $t$ are likely to be popular at time $t+1$.

- $\phi_{k,t+1}$ is likely to smoothly evolve from $\phi_{k,t}$.
At time t

- Food
- Chicken
- Pizza
- Millage

At time t+1

- Observations
  - User prior at time t+1 is a mixture of the user short and long term interest

How do we get a prior that captures both long and short term interest?

Observation 2

- User prior at time t+1 is a mixture of the user short and long term interest
Generative Process

- For each user interaction
  - Choose an intent from local distribution
    - Sample word from the topic’s word-distribution
  - Choose a new intent $\propto \lambda$
    - Sample a new intent from the global distribution
      - Sample word from the new topic word-distribution
Polya-Urn RCRF Process
1. Draw once $\Omega^t | \alpha, \tilde{m}^t \sim \text{Dir}(\tilde{m}^t + \alpha/K)$.

2. Draw each topic, $\phi_k^t | \beta, \tilde{\beta}_k^t \sim \text{Dir}(\tilde{\beta}_k^t + \beta)$.

3. For each user $i$:
   (a) Draw topic proportions $\theta_i^t | \lambda, \Omega^t, \tilde{n}_i^t \sim \text{Dir}(\lambda \Omega^t + \tilde{n}_i^t)$.
   (b) For each word
      (a) Draw a topic $z_{in}^t | \theta_i^t \sim \text{Mult}(\theta_i^t)$.
      (b) Draw a word $w_{in}^t | z_{ij}^t, \phi^t \sim \text{Multi}(\phi_{z_{ij}^t})$.

---

At time $t$  

At time $t+1$
1. Draw once $\Omega^t|\alpha, \tilde{m}^t \sim \text{Dir}\left(\tilde{m}^t + \alpha/K\right)$.

2. Draw each topic, $\phi_k^t|\beta, \tilde{\beta}_k^t \sim \text{Dir}(\tilde{\beta}_k^t + \beta)$.

3. For each user $i$:
   (a) Draw topic proportions $\theta_i^t|\lambda, \Omega^t, \tilde{n}_i^t \sim \text{Dir}(\lambda\Omega^t + \tilde{n}_i^t)$.
   (b) For each word
      (a) Draw a topic $z_{in}^t|\theta_i^t \sim \text{Mult}(\theta_i^t)$.
      (b) Draw a word $w_{in}^t | z_{ij}^t, \phi^t \sim \text{Multi}(\phi_{z_{ij}^t})$.

$$\tilde{\beta}_{kw}^t = \sum_{h=1}^{t-1} \exp \frac{h-t}{\kappa_0} n_{kw}^h$$
1. Draw once $\Omega^t|\alpha, \tilde{m}^t \sim \text{Dir}(\tilde{m}^t + \alpha/K)$.
2. Draw each topic, $\phi_k^t|\beta, \tilde{\beta}_k^t \sim \text{Dir}(\tilde{\beta}_k^t + \beta)$.
3. For each user $i$:
   (a) Draw topic proportions $\theta_i^t|\lambda, \Omega^t, \tilde{n}_i^t \sim \text{Dir}(\lambda\Omega^t + \tilde{n}_i^t)$.
   (b) For each word
      (a) Draw a topic $z_{i\in}^t|\theta_i^t \sim \text{Mult}(\theta_i^t)$.
      (b) Draw a word $w_{i\in}^t|z_{i\in}^t, \phi^t \sim \text{Multi}(\phi^t_{z_{i\in}^t})$. 

Food
Chicken
Pizza
Millage
1. Draw once $\Omega^t | \alpha$, $\tilde{m}^t \sim \text{Dir}(\tilde{m}^t + \alpha/K)$.
2. Draw each topic, $\phi^t_k | \beta$, $\tilde{\beta}^t_k \sim \text{Dir}(\tilde{\beta}^t_k + \beta)$.
3. For each user $i$:
   (a) Draw topic proportions $\theta^t_i | \lambda$, $\Omega^t$, $\tilde{n}_i^t \sim \text{Dir}(\lambda \Omega^t + \tilde{n}_i^t)$.
   (b) For each word
       (a) Draw a topic $z^t_{in} | \theta^t_i \sim \text{Mult}(\theta^t_i)$.
       (b) Draw a word $w^t_{in} | z^t_{ij}, \phi^t \sim \text{Multi}(\phi^t_{z^t_{ij}})$.
1. Draw once $\Omega^t|\alpha, \tilde{m}^t \sim \text{Dir} \left( \tilde{m}^t + \alpha/K \right)$.
2. Draw each topic, $\phi_k^t|\beta, \tilde{\beta}_k^t \sim \text{Dir}(\tilde{\beta}_k^t + \beta)$.
3. For each user $i$:
   (a) Draw topic proportions $\theta_i^t|\lambda, \Omega^t, \tilde{\mathbf{n}}_i^t \sim \text{Dir}(\lambda \Omega^t + \tilde{\mathbf{n}}_i^t)$.
   (b) For each word
      (a) Draw a topic $z_{in}^t|\theta_i^t \sim \text{Mult}(\theta_i^t)$.
      (b) Draw a word $w_{in}^t|z_{ij}^t, \phi^t \sim \text{Multi}(\phi_{z_{ij}^t})$.

**Topics evolve over time?** ✔

**User’s intent evolve over time?** ✔

**Capture long and term interests of users?** ✔
Online Distributed Inference

Work Flow
Work Flow

today

User interactions

User interactions

User interactions

User interactions

User interactions

System state

Daily Update (inference)

new Users’ models

Current Users’ models

Hundred of millions
Online Scalable Inference

• Online algorithm
  – Greedy 1-particle filtering algorithm
  – Works well in practice
  – Collapse all multinomials except $\Omega_t$
    • This makes distributed inference easier
  – At each time $t$:
    \[
P(\Omega^t, z^t | \tilde{n}^t, \tilde{\beta}^t, \tilde{m}^t)
    \]

• Distributed scalable implementation
  – Used first part architecture as a subroutine
  – Added synchronous sampling capabilities
Distributed Inference (at time t)
Distributed Inference (at time $t$)

Collapse all multinomial
Except $\Omega$
After collapsing

Use Star-Synchronization

client

Car speed offer camry accord career

client

Food Chicken Pizza millage
Fully Collapsed

Shared memory

$\Omega_t$

Recipe Chocolate Pizza Food Chicken Milk Butter Powder

Car Blue Book Kelley Prices Small Speed large

job Career Business Assistant Hiring Part-time Receptionist

Bank Online Credit Card debt portfolio Finance Chase

Car speed offer

Food Chicken Pizza millage

client

client
Semi-Collapsed

\[ P(z_{ij}^t = k | w_{ij}^t = w, \Omega^t, \tilde{n}_i^t) \]

\[ \propto \left( n_{ik}^{t,-j} + \tilde{n}_{ik}^t + \lambda \Omega^t \right) \frac{n_{kw}^{t,-j} + \tilde{\beta}_{kw}^t + \beta}{\sum_l n_{kl}^{t,-j} + \tilde{\beta}_{kl}^t + \beta} \]
Distributed Sampling Cycle

Sample $\Omega_t$

Requires a reduction step
Distributed Sampling Cycle

Sample Z For users

Write counts

Collect counts and sample Ω

Barrier

Do nothing

Read Ω from

Sample Z For users

Write counts

Do nothing

Read Ω from

Sample Z For users

Write counts

Do nothing

Read Ω from

Sample Z For users

Write counts

Do nothing

Read Ω from
Experimental Results

• Tasks is predicting **convergence** in display advertising

• Use two datasets
  – 6 weeks of user history
  – Last week responses to Ads are used for **testing**

• Baseline:
  – User **raw data** as features
  – **Static** topic model

<table>
<thead>
<tr>
<th>dataset</th>
<th># days</th>
<th># users</th>
<th># campaigns</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>56</td>
<td>13.34M</td>
<td>241</td>
<td>242GB</td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>33.5M</td>
<td>216</td>
<td>435GB</td>
</tr>
</tbody>
</table>
Interpretability

User-1

- Dating:
  - women
  - men
  - dating
  - singles
  - personals
  - seeking
  - match

- Baseball:
  - League
  - baseball
  - basketball
  - doubleheader
  - Griffey
  - bullpen
  - Greinke

- Celebrity:
  - Snooki
  - Tom
  - Cruise
  - Katie
  - Holmes
  - Pinkett
  - Kudrow
  - Hollywood

User-2

- Health:
  - skin
  - body
  - fingers
  - cells
  - toes
  - wrinkle
  - layers

- Jobs:
  - job
  - career
  - business
  - assistant
  - hiring
  - part-time
  - receptionist

- Finance:
  - financial
  - Thomson
  - chart
  - real
  - Stock
  - Trading
  - currency
Performance in Display Advertising

Dataset-2

ROC

Number of conversions

Number of conversions:
- >1000
- [1000, 600]
- [600, 400]
- [400, 200]
- [200, 100]
- [100, 60]
- [60, 40]
- [40, 20]
- <20

Graph showing ROC values for different number of conversions.
## Performance in Display Advertising

### Weighted ROC measure

<table>
<thead>
<tr>
<th></th>
<th>base</th>
<th>TLDA</th>
<th>TLDA+base</th>
<th>LDA+base</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset 1</td>
<td>54.40</td>
<td>55.78</td>
<td><strong>56.94</strong></td>
<td>55.80</td>
</tr>
<tr>
<td>dataset 2</td>
<td>57.03</td>
<td>57.70</td>
<td><strong>60.38</strong></td>
<td>58.54</td>
</tr>
</tbody>
</table>

### Effect of number of topics

<table>
<thead>
<tr>
<th></th>
<th>topics</th>
<th>TLDA</th>
<th>TLDA + base</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset 1</td>
<td>50</td>
<td>55.32</td>
<td>56.01</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>55.5</td>
<td>56.56</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td><strong>55.8</strong></td>
<td><strong>56.94</strong></td>
</tr>
<tr>
<td>dataset 2</td>
<td>50</td>
<td>59.10</td>
<td>60.40</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td><strong>59.14</strong></td>
<td><strong>60.60</strong></td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>58.7</td>
<td>60.38</td>
</tr>
</tbody>
</table>

Static Batch models
How Does It Scale?

2 Billion instances with 5M vocabulary using 1000 machines
one iteration took ~ 3.8 minutes
Application

Multi-Domain Personalization
Multi-domain Personalization

• Intuition
  – We observe user interaction with news and movies
  – Can we predict his music taste?

• Interaction definition
  – A bag of words describing objects user interacts with in a given domain
Example

User Meta Profile

User Music Profile
- Personalized Music

User News Profile
- Personalized News
Example

User Meta Profile

User Music Profile
User News Profile
User Movie Profile
The Model

A user’s interaction with a domain is a bag of words.
A topic is a mixture of words.

User’s prior interest in a domain is
\[ \alpha = \log(1 + \exp(\lambda_d x_u)) \]

Each user has a meta-profile:
\[ x_u \in \mathbb{R}^k \]

Each domain has a latent matrix:
\[ \lambda_d \in \mathbb{R}^{k \times t_d} \]

Slide credit Yucheng Low
The Model

\[ \text{sgt}(x) = \log(1 + \exp(x)) \]

User Meta Profile
\[ x_u \in \mathbb{R}^k \]

\[ \lambda_{\text{music}} \]

User Music Profile
\[ \text{sgt}(\lambda_{\text{music}} x_u) \]

\[ \lambda_{\text{news}} \]

User News Profile
\[ \text{sgt}(\lambda_{\text{news}} x_u) \]

\[ \lambda_{\text{movie}} \]

User Movie Profile
\[ \text{sgt}(\lambda_{\text{movie}} x_u) \]

Slide credit Yucheng Low
Music
Topic->word table

News
Topic->word table

Movies
Topic->word table

λ₁

λ₂

λ₃

$w \in W_{u,d}$
Inference and Learning

User u’s interaction with domain \( p \)

E-Step: sample local variables

M-Step
Optimize via barrier

Collapse and synchronize

E-Step:
sample local variables

\( \beta \) → \( \phi_d \)

User u’s interaction with domain \( p \)

\( \lambda_p \) → \( \alpha \)

\( \theta \) → \( z \) → \( w \)
Distributed Sampling Cycle

Sample $Z, x$ For users
Sample $Z, x$ For users
Sample $Z, x$ For users
Sample $Z, x$ For users

Optimize $\lambda$
Requires a reduction step
Distributed Sampling Cycle

Sample Z, x
For users

Write statistics

Collect and optimize

Read

Sample Z, x
For users

Write statistics

Do nothing

Read

Sample Z, x
For users

Write statistics

Do nothing

Read

Sample Z, x
For users

Write statistics

Do nothing

Read

Barrier

Barrier
Results

- 2 domain dataset.
  Frontpage and News clicks of 5.6 million users.
  **Frontpage/News:** Article text for each click.
- Measure gain relative to independent models on each domain
Results

% Gain in Cosine Similarity

- Front Page: 25 Features (blue), 50 Features (brown)
- News: 25 Features (blue), 50 Features (brown)
Distributed Inference Revisited
To collapse or not to collapse?

• Not collapsing
  – Keeps conditional independence
    • Good for parallelization
    • Requires synchronous sampling
  – Might mix slowly

• Collapsing
  – Mixes faster
  – Hinder parallelism
  – Use star-synchronization
    • Works well if sibling depends on each others via aggregates
    • Requires asynchronous communication
Inference Primitive

- Collapse a variable
  - *Star synchronization* for the sufficient statistics

- Sampling a variable
  - Local
    - Sample it locally (possibly using the *synchronized statistics*)
  - Shared
    - *Synchronous sampling* using a barrier

- Optimizing a variable
  - Same as in the shared variable case
  - Ex. Conditional topic models
Asynchronous Optimization
Asynchronous Processing

• Needed when
  – Ex: Optimizing a global variable
• Mostly requires a barrier
• Advantages
  – Easy to program
  – Well-understood reusable templates
• Disadvantages
  – The curse of the last reducer
  – You are as fast as the slowest machine!
Asynchronous Processing

- Needed when
  - Ex: Optimize a global variable
- Mostly requires a barrier
- Advantages
  - Easy to program
  - Well-understood reusable template
- Disadvantages
  - The curse of the last reducer
  - You are as fast as the slowest machine!
Asynchronous Optimization

Graph Factorization
Graph Factorization Problem

• Factor a graph into low rank components
• Assign a latent vector $Z_i \in \mathcal{R}^k$ with each node
• Optimize:

$$f(Y, Z, \lambda) = \frac{1}{2} \sum_{(i,j) \in E} (Y_{ij} - \langle Z_i, Z_j \rangle)^2 + \frac{\lambda}{2} \sum_i n_i \|Z_i\|^2$$

Observed value over edges

Predicted value

Regularization
Single-Machine Algorithm

- Just use stochastic gradient decent (SGD)

\[
\frac{\partial f}{\partial Z_i} = - \sum_{j \in \mathcal{N}(i)} (Y_{ij} - \langle Z_i, Z_j \rangle) Z_j + \lambda n_i Z_i
\]

- Cycle until convergence
  - Read a node, \( i \)
  - Update its latent factor

\[
Z_i \leftarrow Z_i - \eta \left( \frac{\partial f}{\partial Z_i} \right)
\]
Problem Scale

- Yahoo IM and Mail graphs
- Nodes are users
- Edges represent (log) number of messages
- 200 Million vertices
- 10 Billion edges
Challenges

• Parameter storage
  – Too much for a single machine

• Approach
  – Distribute the graph over machines
    • How to partition the nodes?
  – Synchronization
    • How to synchronize replicated nodes
  – Communication
    • How to accommodate network topology
Can we solve the problem with similar ideas to what we have covered?
Partition and Replicate
Partition and Replicate

- Cycle until convergence
  - Read a node, $i$
  - Update its latent factor

$$Z_i \leftarrow Z_i - \eta \left( \frac{\partial f}{\partial Z_i} \right)$$
Partition and Replicate

- **Problem**
  - Some neighbors are missing

- **Solution**
  - Replicate and synchronize
    - *Borrowed* vs. owned nodes
• Formulation
  – Introduce local copies
    • A factor per node X
  – Tie across machines
    • Introduce global factor Z
    • Penalizes deviations
Formulation

• Original problem

\[ f(Y, Z, \lambda) = \frac{1}{2} \sum_{(i,j) \in E} (Y_{ij} - \langle Z_i, Z_j \rangle)^2 + \frac{\lambda}{2} \sum_i n_i \| Z_i \|^2 \]

• Relaxed problem

\[ \sum_{k=1}^{K} f_k(Y, X^{(k)}, \lambda) + \frac{1}{2} \sum_{k=1}^{K} \left[ \mu \sum_{i \in V_k} \| Z_i - X_i^{(k)} \|^2 \right] \]

• Local problem

\[ f_k(Y, X^{(k)}, \lambda) = \frac{1}{2} \left[ \sum_{(i,j) \in E, \ i,j \in V_k} (Y_{ij} - \langle X_i^{(k)}, X_j^{(k)} \rangle)^2 + \lambda \sum_{i \in V_k} n_i \| X_i^{(k)} \|^2 \right] \]
Synchronous Algorithms

• Optimize joint objective over $X,Z$

• Local parameter updates
  – Run SGD until convergence

\[
\min_{X^{(k)}} f_k(Y, X^{(k)}, \lambda) + \frac{1}{2} \mu \sum_{i \in V_k} \| Z_i - X_i^{(k)} \|^2
\]

Fit the data

Minimize deviation

• Global parameter updates

\[
\min_{Z} \frac{1}{2} \sum_{k=1}^{K} \left[ \mu \sum_{i \in V_k} \| Z_i - X_i^{(k)} \|^2 \right]
\]
Synchronous Algorithms

Global state
Distributed
shared memory

1- We only store replicated nodes
2- The global state is distributed across machines
3- each machine keeps track of the global copy of its owned variables
Step 1: Push global variables

Global state
Distributed
shared memory

\[ X^{(k)} \]
Step 2: Local Optimization

Global state
Distributed
shared memory

\[
\text{minimize}_{X^{(k)}} f_k(Y, X^{(k)}, \lambda) + \frac{1}{2} \mu \sum_{i \in V_k} \|Z_i - X^{(k)}_i\|^2
\]

\[Z\]
Step 3: Push and average

Global state
Distributed
shared memory

\[
\minimize_{\mathbf{Z}} \frac{1}{2} \sum_{k=1}^{K} \left[ \mu \sum_{i \in V_k} \| \mathbf{Z}_i - \mathbf{X}_i^{(k)} \|^2 \right]
\]
Step 3: Push and average

Global state
Distributed
shared memory

$Z$

$X^{(k)}$

$X^{(k)}$

$X^{(k)}$
Summary of Asynchronous Algorithms

- An improvement over standard Map-Reduce
- Curse of the last reducer
- You are as fast as the slowest machine
  - Optimize local variables
  - Barrier
  - Optimize global variables
  - Barrier
- Can we do better?
An Asynchronous Algorithm

- Conceptual idea
  - Optimize $X$ and $Z$ jointly

\[
\sum_{k=1}^{K} f_k(Y, X^{(k)}, \lambda) + \frac{1}{2} \sum_{k=1}^{K} \left[ \mu \sum_{i \in V_k} \| Z_i - X_i^{(k)} \|^2 \right]
\]

- User SGD over $(X,Z)$
- Pick a local node
- Do a gradient step over corresponding $X,Z$!
Conceptual Idea

\[
\sum_{k=1}^{K} f_k(Y, X^{(k)}, \lambda) + \frac{1}{2} \sum_{k=1}^{K} \left[ \mu \sum_{i \in V_k} \| Z_i - X_i^{(k)} \|^2 \right]
\]

\[
\frac{\partial f}{\partial Z_i} \left[ X_i^{(k)} \right] = \mu (Z_i - X_i^{(k)}).
\]

Cache the global variables
Locally (Asynchronous updates)

We don’t have global copy locally
Parallel Updates

Global state
Distributed
shared memory

$Z$

$X^{(k)}$

Indicate A borrowed node
Form other partitions

Last cached value of the global variable
Parallel Asynchronous Updates

Global state
Distributed
shared memory

\[ Z \]

\[
\frac{\partial f}{\partial X^{(k)}_i} = -\sum_{j \in N(i)} (Y_{ij} - \langle X^{(k)}_i, X^{(k)}_j \rangle) X^{(k)}_j + \lambda n_i X^{(k)}_i + \mu (X^{(k)}_i - Z^{(k)}_i). 
\]

- Receive local copy \( X_i \) from \( k \)
- Update \( Z_i \)
- Send back new \( Z_i \) to \( k \)

Synchronization thread Send
- Cycle through nodes
- Send local copy to DSM

Computation thread
- Cycle through nodes
- Update local copies

Synchronization thread receive
- Received \( Z_i \) from DSM
- Update cached copy
Convergence

• Can be reduced to lock-free parallel SGD [Hogwild]

• Convergence is affected by
  – Synchronization rate
    • Time needed to refresh the local version of the global variable
    • Number of replicated nodes
Summary of Asynchronous

- Continuously update local variables $X$ (via SGD)
- Continuously send local variables to global
- Continuously update global variable $Z$ (via SGD)
- Continuously send & overwrite global variables to local

\[
\sum_{k=1}^{K} f_k(Y, X^{(k)}, \lambda) + \frac{1}{2} \sum_{k=1}^{K} \left[ \mu \sum_{i \in V_k} \| Z_i - X_i^{(k)} \|^2 \right]
\]
Convergence

Full Dataset: 200M nodes

Objective Function vs. Time in minutes (linear scale)

- Asynchronous optimization
- Synchronous optimization
Convergence
Scalability

![Graph showing scalability over the number of nodes in millions. The graph compares time per epoch for single machine and multi-machine asynchronous operations. The number of machines scaling linearly is 4, 8, 16, ...]
Solution Quality

32M Nodes

- Multi-Machine Asynchronous (32 machines)
- Single machine

Average test error vs. Time in minutes (Log Scale)
Practical Considerations

• How to partition the graph?
  – We want to minimize the number of borrowed nodes
    • Affect convergence
    • Increases the number of deviation penalties
  – Take each machine capacity into consideration
    • Store owned nodes
    • Borrowed nodes
    • Cached copies of relevant global variables

• Network Optimization
  – Take network topology into account
Graph Partition

• Find a set of minimally overlapped partitions
  “Decompose the graph to minimize number of vertices + neighbors per partition”
  – NP hard problem by itself [WSDM 2012]

• Under capacity constraints

• We just scratched the surface here
  – Simple greedy algorithm
  – Hierarchal extension
  – LSH and random baselines
• Intuitively
  – Add each node to where its neighbors are!
• Maintain a set of open partitions
  – Store the borrowed and owned nodes in each partition
• For each vertex $v$
  – For each partition $p$
    • We want to make sure that $N(v)$ are in the same partition
    • Add $N(v) / \text{Owned}(p)$ to borrowed of $p$
  – Select $p$ with minimum number of borrowed nodes
• For each vertex $v$
  • For each partition $p$
    • We want to make sure that $N(v)$ are in the same partition
    • Add $N(v) / \text{Owned}(p)$ to borrowed of $p$
  • Select $p$ with minimum number of borrowed nodes
Hierarchical Extension

• Two step approach
  – First run greedy with small number of partitions
  – Second, run greedy over the first level partitions

• Time is proportional to number of open partitions
  – Divide and conquer
Baselines

• Radom
• LSH-based
  – LSH over adjacency matrix
  – Related to shingle-based graph compression approaches
• Metrics
  – Time to perform partitioning
  – Quality of partitions
    • Number of borrowed nodes
    • Time to perform a full synchronization cycle
The Effect of Partitioning Quality

<table>
<thead>
<tr>
<th>Method</th>
<th>Total borrowed nodes (millions)</th>
<th>Partitioning time (minutes)</th>
<th>Sync time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat</td>
<td>252.31</td>
<td>166</td>
<td>71.5</td>
</tr>
<tr>
<td>Hierarchical</td>
<td>392.33</td>
<td>48.67</td>
<td>85.9</td>
</tr>
<tr>
<td>Hier-LSH</td>
<td>640.67</td>
<td>17.8</td>
<td>136.1</td>
</tr>
<tr>
<td>Hier-Random</td>
<td>720.88</td>
<td>11.6</td>
<td>145.2</td>
</tr>
</tbody>
</table>
The Effect of Partitioning Quality

Effect of partitioning on performance

- Heirarchical
- Flat

Objective function vs. Time in minutes
Network Optimization

V1 — Machine 1.6
V2 — Machine 1.3
V3 — Machine 2.4
V4 — Machine 2.1
V5 — Machine 1.5

Rack 1
- Machine 1.3
- Machine 1.5
- Machine 1.6

Rack 2
- Machine 2.1
- Machine 2.4
- Machine 2.6
Network Optimization

• We only know the layout at run time
  – Inverse network bandwidth $D$

• Inter-partitions communication
  – Communication requirement $C$
  – The more overlap, the higher is $C$

• Solve a quadratic assignment problem

\[
T(\pi) = \sum_{kl} C_{kl} D_{\pi(k)\pi(l)} = \sum_{kl} C_{kl} \sum_{uv} \pi_{ku} \pi_{lv} D_{uv} = \text{tr} C \pi D \pi^T
\]
Sync time without QAP

Histogram of Sync time with QAP disabled
Sync time with QAP
Summary

• Model as consensus problem
• Synchronous algorithms
  – Curse of the last reducer
• Asynchronous algorithm
  – Asynchronous parallel updates
  – Network topology optimization
  – Overlapping partitions
Future Directions
Future Directions

- Theoretical bounds and guarantees
- Non-parametric models
  - Learning structure from data
- Working under communication constraints
- A new release of Yahoo! LDA
- More applications
  - Citation analysis
    - Graph factorization + LDA
Questions?
Sampling $\Omega$

- Introduce auxiliary variable $m_{kt}$
  - How many times the global distribution was visited
  - $P(m_{kt}^t|n_{1k}^t, \cdots, n_{ik}^t, \cdots) \sim \text{AnotniaK}$

$$P(\Omega^t|m^t, \tilde{m}^t) \sim \text{Dir} (\tilde{m}^t + m^t + \alpha/K)$$
Distributed Sampling Cycle

Sample Z For users
Sample Z For users
Sample Z For users
Sample Z For users

Write counts
Write counts
Write counts
Write counts

Collect counts and sample $\Omega$
Do nothing
Do nothing
Do nothing

Barrier

Barrier

Read $\Omega$ from
Read $\Omega$ from
Read $\Omega$ from
Read $\Omega$ from