Scaling Latent Variable Models

Alexander Smola

alex@smola.org
Thanks
• Variations on a Theme (collapsing or not)
• Scaling Problems
• Fast samplers
Variations on a Theme
Any distribution over exchangeable random variables can be written as conditionally independent.

\[ p(x_1, \ldots, x_n) = \int dp(\theta) \prod_{i=1}^{n} p(x_i | \theta) \]

Inference should be easy - \( \Theta | x_i \) and \( x_i | \Theta \)
Conjugates and Collapsing

• Exponential Family

\[ p(x|\theta) = \exp (\langle \phi(x), \theta \rangle - g(\theta)) \]

• Conjugate Prior

\[ p(\theta|\mu_0, m_0) = \exp (m_0 \langle \mu_0, \theta \rangle - m_0 g(\theta) - h(m_0\mu_0, m_0)) \]

• Posterior

\[ p(\theta|X, \mu_0, m_0) \propto \exp (\langle m_0\mu_0 + m\mu[X], \theta \rangle - (m_0 + m) g(\theta) - h(m_0\mu_0, m_0)) \]

• Collapsing the natural parameter

\[ p(X|\mu_0, m_0) = \exp (h(m_0\mu_0 + m\mu[X], m_0 + m) - h(m_0\mu_0, m_0)) \]
Conjugates and Collapsing

deFinetti collapsed representation
Clustering & Topic Models

Clustering

\[ \alpha \rightarrow \theta \rightarrow y \rightarrow x \]

Latent Dirichlet Allocation

\[ \alpha \rightarrow \theta \rightarrow y \rightarrow x \]

\[ \text{cluster probability} \]

\[ \text{cluster label} \]

\[ \text{instance} \]
Clustering & Topic Models

- **Clustering**
  - (0, 1) matrix

- **Topic Model**
  - Stochastic matrix

- **LSI**
  - Arbitrary matrices

- **Documents**

- **Membership**

- **Cluster/Topic Distributions**

Equation: $\text{Clustering: (0, 1) matrix} \times \text{membership} = \text{Documents}$
Clustering & Topic Models

Cluster/topic distributions \( \times \) membership = Documents

clustering: (0, 1) matrix

topic model: stochastic matrix

LSI: arbitrary matrices

estimate sample/optimize
V1 - Brute force maximization

- Integrate out latent parameters $\theta$ and $\psi$
  \[ p(X, Y | \alpha, \beta) \]
- Discrete maximization problem in $Y$
- Hard to implement
- Overfits a lot (mode is not a typical sample)
- Parallelization infeasible
V2 - Brute force maximization

- Integrate out latent parameters $y$
  \[ p(X, \psi, \theta | \alpha, \beta) \]
- Continuous nonconvex optimization problem in $\theta$ and $\psi$
- Solve by stochastic gradient descent over documents
- Easy to implement
- Does not overfit much
- Great for small datasets
- Parallelization difficult/impossible
- Memory storage/access is $O(TW)$ (this breaks for large models)
  - 1M words, 1000 topics = 4GB
  - Per document 1MFlops/iteration
V3 - Variational approximation

- Approximate intractable joint distribution by tractable factors
  \[
  \log p(x) \geq \log p(x) - D(q(y) || p(y|x)) = \int dq(y) [\log p(x) + \log p(y|x) - q(y)]
  = \int dq(y) \log p(x, y) + H[q]
  \]
- Alternating convex optimization problem
- Dominant cost is matrix matrix multiply
- Easy to implement
- Great for small topics/vocabulary
- Parallelization easy (aggregate statistics)
- Memory storage is \( O(TW) \) (this breaks for large models)
- Model not quite as good as sampling

Blei, Ng, Jordan
**V4 - Uncollapsed Sampling**

- Sample $y_{i|\text{rest}}$
  Can be done in parallel
- Sample $\theta|\text{rest}$ and $\psi|\text{rest}$
  Can be done in parallel
- Compatible with MapReduce
  (only aggregate statistics)
- Easy to implement
- Children can be conditionally independent*
- Memory storage is $O(TW)$
  (this breaks for large models)
- Mixes slowly

*for the right model
Integrate out latent parameters $\theta$ and $\psi$

Sample one topic assignment $y_{ij}|X,Y^{-ij}$ at a time from

$\frac{n^{-ij}(t,d) + \alpha_t}{n^{-i}(d) + \sum_t \alpha_t}$ \quad $\frac{n^{-ij}(t,w) + \beta_t}{n^{-i}(t) + \sum_t \beta_t}$

- Fast mixing
- Easy to implement
- Memory efficient
- Parallelization infeasible (variables lock each other)

Griffiths & Steyvers 2005
Integrate out latent parameters $\theta$ and $\psi$

Sample one topic assignment $y_{ij}|X,Y^{-ij}$ at a time from

- Fast mixing
- Easy to implement
- Memory efficient
- Parallelization infeasible (variables lock each other)

Griffiths & Steyvers 2005
- Collapsed sampler per machine
  \[
  \frac{n^{-ij}(t, d) + \alpha_t}{n^{-i}(d) + \sum_t \alpha_t} \quad \frac{n^{-ij}(t, w) + \beta_t}{n^{-i}(t) + \sum_t \beta_t}
  \]
- Defer synchronization between machines
  - no problem for \( n(t) \)
  - big problem for \( n(t,w) \)
- Easy to implement
- Can be memory efficient
- Easy parallelization
- Mixes slowly/worse likelihood

Asuncion, Smyth, Welling, ... UCI
Mimno, McCallum, ... UMass
• Collapsed sampler
\[
\frac{n^{-ij}(t, d) + \alpha_t}{n^{-i}(d) + \sum_t \alpha_t} \quad \frac{n^{-ij}(t, w) + \beta_t}{n^{-i}(t) + \sum_t \beta_t}
\]
• Make local copies of state
  • Implicit for multicore (delayed updates from samplers)
  • Explicit copies for multi-machine
• Not a hierarchical model (Welling, Asuncion, et al. 2008)
• Memory efficient (only need to view its own sufficient statistics)
• Multicore / Multi-machine
• Convergence speed depends on synchronizer quality
Integrate out latent $\theta$ and $\psi$

\[ p(X, Y | \alpha, \beta) \]

Chain conditional probabilities

\[ p(X, Y | \alpha, \beta) = \prod_{i=1}^{m} p(x_i, y_i | x_1, y_1, \ldots x_{i-1}, y_{i-1}, \alpha, \beta) \]

For each particle sample

\[ y_i \sim p(y_i | x_i, x_1, y_1, \ldots x_{i-1}, y_{i-1}, \alpha, \beta) \]

Reweight particle by next step data likelihood

\[ p(x_{i+1} | x_1, y_1, \ldots x_i, y_i, \alpha, \beta) \]

Resample particles if weight distribution is too uneven

Canini, Shi, Griffiths, 2009
Ahmed et al., 2011
• One pass through data
• Data sequential parallelization is open problem
• Nontrivial to implement
• Sampler is easy
• Inheritance tree through particles is messy
• Need to estimate data likelihood (integration over $y$), e.g. as part of sampler
• This is multiplicative update algorithm with log loss ...

Canini, Shi, Griffiths, 2009
Ahmed et al., 2011

V8 - Sequential Monte Carlo

• Integrate out latent $\theta$ and $\psi$
  $$p(X, Y|\alpha, \beta)$$
• Chain conditional probabilities
  $$p(X, Y|\alpha, \beta) = \prod_{i=1}^{m} p(x_i, y_i|x_1, y_1, \ldots x_{i-1}, y_{i-1}, \alpha, \beta)$$
• For each particle sample
  $$y_i \sim p(y_i|x_i, x_1, y_1, \ldots x_{i-1}, y_{i-1}, \alpha, \beta)$$
• Reweight particle by next step data likelihood
  $$p(x_{i+1}|x_1, y_1, \ldots x_i, y_i, \alpha, \beta)$$
• Resample particles if weight distribution is too uneven
<table>
<thead>
<tr>
<th></th>
<th>Uncollapsed</th>
<th>Variational approximation</th>
<th>Collapsed natural parameters</th>
<th>Collapsed topic assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optimization</strong></td>
<td>overfits too costly</td>
<td>easy parallelization big memory footprint</td>
<td>overfits too costly</td>
<td>easy to optimize big memory footprint difficult parallelization</td>
</tr>
<tr>
<td><strong>Sampling</strong></td>
<td>slow mixing conditionally independent</td>
<td>n.a.</td>
<td>approximate inference by delayed updates particle filtering sequential</td>
<td>sampling difficult</td>
</tr>
</tbody>
</table>
Scaling Problems
3 Problems

Global

$\mu_j$

$j \in [k]$

Local

$x_i$

$i \in [m]$

mean

variance

cluster weight

data

cluster ID

Yahoo!
3 Problems

Global

\[ \mu_j \]
\[ j \in [k] \]

Local

\[ x_i \]
\[ i \in [m] \]

\[ z_i \]

global state

data

local state
3 Problems

too big for single machine

Global

Local

μ_j

j ∈ [k]

x_i

i ∈ [m]

huge

only local
3 Problems

Vanilla LDA

global state

local state

data

global state
3 Problems

Vanilla LDA

global state

local state

data

global state

User profiling
3 Problems

- Local state is too large
- Global state is too large
- Does not fit into memory
- Network load & barriers
- Does not fit into memory
3 Problems

- Local state is too large
- Global state is too large
- Stream local data from disk
- Network load & barriers
- Does not fit into memory
3 Problems

- Local state is too large
- Global state is too large
- Network load & barriers does not fit into memory
- Stream local data from disk
- Asynchronous synchronization
- Does not fit into memory
3 Problems

Local state is too large

Global state is too large

- Stream local data from disk
- Asynchronous synchronization
- Partial view
Distribution

Global

\[ \mu_j \]

\( j \in [k] \)

Replica

\[ \mu_{jl} \]

\( j \in [k] \)

\[ xi \]

\( i \in [m] \)

\[ zi \]

\( l \in [p] \)

processor local state

global

replica
Distribution

Global

$\mu_j$

$j \in [k]$

Replica

$\mu_{jl}$

$j \in [k]$

$\times_i$

$i \in [m]$

$L \in [p]$

Processor Local State

Cluster

Rack
Synchronization

- Child updates local state
  - Start with common state
  - Child stores old and new state
  - Parent keeps global state
- Transmit differences asynchronously
  - Inverse element for difference
  - Abelian group for commutativity (sum, log-sum, cyclic group, exponential families)
Synchronization

- Naive approach (dumb master)
  - Global is only (key,value) storage
  - Local node needs to lock/read/write/unlock master
  - Needs a 4 TCP/IP roundtrips - latency bound
- Better solution (smart master)
  - Client sends message to master / in queue / master incorporates it
  - Master sends message to client / in queue / client incorporates it
  - Bandwidth bound (>10x speedup in practice)

\[
\begin{align*}
\delta & \leftarrow x - x^{\text{old}} \\
x^{\text{old}} & \leftarrow x \\
x^{\text{global}} & \leftarrow x^{\text{global}} + \delta \\
x & \leftarrow x + (x^{\text{global}} - x^{\text{old}}) \\
x^{\text{old}} & \leftarrow x^{\text{global}}
\end{align*}
\]
Distribution

- Dedicated server for variables
- Insufficient bandwidth (hotspots)
- Insufficient memory
- Select server e.g. via consistent hashing

\[ m(x) = \arg\min_{m \in M} h(x, m) \]
Distribution & fault tolerance

- Storage is $O(1/k)$ per machine
- Communication is $O(1)$ per machine
- Fast snapshots $O(1/k)$ per machine (stop sync and dump state per vertex)

$$m(x) = \arg\min_{m \in M} h(x, m)$$
Distribution & fault tolerance

- Storage is $O(1/k)$ per machine
- Communication is $O(1)$ per machine
- Fast snapshots $O(1/k)$ per machine (stop sync and dump state per vertex)

$$m(x) = \arg\min_{m \in M} h(x, m)$$
Distribution & fault tolerance

- Storage is $O(1/k)$ per machine
- Communication is $O(1)$ per machine
- Fast snapshots $O(1/k)$ per machine (stop sync and dump state per vertex)
- $O(k)$ open connections per machine
- $O(1/k)$ throughput per machine

$$m(x) = \arg\min_{m \in M} h(x, m)$$
Synchronization

- Data rate between machines is $O(1/k)$
- Machines operate asynchronously (barrier free)

Solution
- Schedule message pairs
- Communicate with $r$ random machines simultaneously

local

r=1

global
Synchronization

- Data rate between machines is $O(1/k)$
- Machines operate asynchronously (barrier free)

Solution
- Schedule message pairs
- Communicate with $r$ random machines simultaneously

local

$\Rightarrow$

r=1

global

$\Rightarrow$
Synchronization

- Data rate between machines is $O(1/k)$
- Machines operate asynchronously (barrier free)
- Solution
  - Schedule message pairs
  - Communicate with $r$ random machines simultaneously
Synchronization

- Data rate between machines is $O(1/k)$
- Machines operate asynchronously (barrier free)

Solution
- Schedule message pairs
- Communicate with $r$ random machines simultaneously

$0.78 < \text{eff.} < 0.89$
Synchronization

• Data rate between machines is $O(1/k)$
• Machines operate asynchronously (barrier free)

Solution
• Schedule message pairs
• Communicate with $r$ random machines simultaneously
• Use Luby-Rackoff PRPG for load balancing

Efficiency guarantee

$$1 - e^{-r} \sum_{i=0}^{r} \left[ 1 - \frac{i}{r} \right] \frac{r^i}{i!} \leq \text{Eff} \leq 1 - e^{-r}$$

4 simultaneous connections are sufficient
Scalability Analysis

Fixed #machines=100

Linearly scaling #machines: 100, 300, ...

Ideal
Samplers
Sampling

• Brute force sampling over large number of items is expensive
  • Ideally want work to scale with entropy of distribution over labels.
  • Sparsity of distribution typically only known after seeing the instances
  • Decompose (dense) probability into dense invariant and sparse variable terms
  • Use fast proposal distribution & rejection sampling
Exploiting Sparsity

- Decomposition (Mimno & McCallum, 2009)
  Only need to update **sparse** terms per word

\[
p(t|w_{ij}) \propto \beta_w \frac{\alpha_t}{n(t) + \beta} + \beta_w \frac{n(t, d = i)}{n(t) + \beta} + \frac{n(t, w = w_{ij}) [n(t, d = i) + \alpha_t]}{n(t) + \beta}
\]

- Does not work for clustering (too many factors)
Exploiting Sparsity

- **Context LDA** (Petterson et al., 2009)
  The smoothers are word and topic dependent

\[
p(t|w_{ij}) \propto \beta(w, t) \frac{\alpha_t}{n(t) + \beta(t)} + \bar{\beta}(w, t) \frac{n(t, d = i)}{n(t) + \beta(t)} + \frac{n(t, w = w_{ij}) [n(t, d = i) + \alpha_t]}{n(t) + \beta(t)}
\]

- Simple sparse factorization doesn’t work
- Use Cauchy Schwartz to upper-bound first term

\[
\sum_t \beta(w, t) \frac{\alpha_t}{n(t) + \beta(t)} \leq \|\beta(w, \cdot)\| \left\| \frac{\alpha}{n(\cdot) + \beta(\cdot)} \right\|
\]
Collapsed vs Variational

- **Memory requirements (1k topics, 2M words)**
  - Variational inference: **8GB RAM (no sparsity)**
  - Collapsed sampler: **1.5GB RAM (rare words)**
- **Burn-in & sparsity exploit saves a lot**

![Sampling Speed-up (1000 topics)]

**unif | doc | doc, word**

- Cauchy Schwartz bound
- Multilingual LDA
- Word context
- Smoothing over time
Fast Proposal

- In reality sparsity often not true for real proposal
- Guess sparse proxy
- In the storylines model this are the entities
Summary

- Sampling can be much faster/cheaper than dense optimization
- Use the computer architecture to scale up algorithms
- Problem-specific samplers (use sparsity)