Coordinate Ascent

Variational Inference

+ 

Variational EM
Reminders

- Lecture on Friday, Recitation on Monday
- Homework 4: MCMC
  - Out: Mon, Oct 24
  - Due: Fri, Nov 4 at 11:59pm
- Homework 5: Variational Inference
  - Out: Fri, Nov 4
  - Due: Wed, Nov 16 at 11:59pm
Q: Did you fix your slides by adding the log back into CAVI for the discrete factor graph?

A: Yes... (see Lecture 17 slides)
COORDINATE ASCENT
VARIATIONAL INFERENCE (CAVI)
Variational Inference

Whiteboard

– CAVI algorithm derivation
  • Chain rule decomposition of $\log p(x, z)$
  • Decomposing the entropy
  • Decomposing the ELBO
  • Derivatives and closed form solution
MAP INFERENCE AND VARIATIONAL INFERENCE
MAP Inference as Variational Inference

**Suppose:** We want a family $Q$ such that the variational inference solution:

$$
\hat{q}(z) = \operatorname{argmin}_{q \in Q} \text{KL}(q \parallel p)
$$

gives back a distribution that is a point mass on the MAP inference solution:

$$
\hat{q}(z) = \begin{cases} 
\hat{z} = \operatorname{argmax}_{z \in Z(x)} p(z \mid x) & \text{w/\text{prob. 1.0}} \\
\text{any other } z \in Z(x) & \text{w/\text{prob. 0.0}} 
\end{cases}
$$

**Question:** What is $Q$?

**Answer:**
MAP Inference as Variational Inference

**Suppose:** We want a family $Q$ such that the variational inference solution:

$$\hat{q}(z) = \text{argmin}_{q \in Q} \text{KL}(q \parallel p)$$

gives back a distribution that is a point mass on the MAP inference solution:

$$\hat{q}(z) = \begin{cases} \hat{z} = \text{argmax}_{z \in Z(x)} p(z \mid x) & \text{w/prob. 1.0} \\ \text{any other } z \in Z(x) & \text{w/prob. 0.0} \end{cases}$$

**Question:** What is $Q$?

**Answer:** Each member of $Q$, i.e. $q_{z'} \in Q$, is a point mass for some $z' \in Z(x)$. That is:

$$q_{z'}(z) = \begin{cases} 1 & \text{if } z' = z \\ 0 & \text{otherwise} \end{cases}$$

**Example:**

two vars $A$ and $B \in \{\text{red, blue}\}$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$p(a,b)$</th>
<th>$q_1(a,b)$</th>
<th>$q_2(a,b)$</th>
<th>$q_3(a,b)$</th>
<th>$q_r(a,b)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>red</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>red</td>
<td>blue</td>
<td>0.4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>blue</td>
<td>red</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>blue</td>
<td>blue</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$$\hat{q}_i = \text{argmin}_{q_i \in Q} \text{KL}(q_i \parallel p) = q_2$$

Q family
VARIATIONAL INFERENCE RESULTS
Mean-Field + Learning

How does variational inference fit into learning?

So far, we have two ways to learn parameters using variational inference:

**Maximum likelihood estimation:**
Optimize MLE over the parameters—optionally calling variational inference as a subroutine during learning

*Example:*
- train a CRF for tagging a tree given a sentence
- per-example gradient requires us to compute the marginals for that example
- instead of using BP to compute the marginals, use mean-field variational inference

*Example: Fully-Connected CRF*
*Example: Joint Parsing and Alignment*

**Variational Bayes:** treat the parameters as a variable in the model and run variational inference to estimate the distribution over those parameters

*Example:*
- learn the posterior distribution over the parameters in Gaussian LDA
- instead of sampling the topic assignments and parameters from a Gibbs sampler, use mean-field variational inference

*Example: Gaussian Mixture Model*
*Example: Uncollapsed LDA*
*Example: Collapsed LDA*
Factor Derivatives

Log-probability:

\[
\log p(y) = \left[ \sum_c \log \psi_c(y_c) \right] - \log \sum_{y' \in Y} \prod_c \psi_c(y'_c)
\]  

(1)

Derivatives:

\[
\frac{\partial \log p(y)}{\partial \log \psi_c(y'_c)} = 1(y_c = y'_c) - \frac{p(y'_c)}{\psi_c(y'_c)}
\]  

(2)

\[
\frac{\partial \log p(y)}{\partial \psi_c(y'_c)} = 1(y_c = y'_c) - \frac{p(y'_c)}{\psi_c(y'_c)}
\]  

(3)

- We can compute these factor marginals using mean-field variational inference.
- Then we can run gradient descent to learn the parameters of the model.
Fully–Connected CRF

Model

\[ p(x|i) = \frac{1}{Z(i)} \exp(-E(x)) \]

\[ E(x) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j), \]

Inference

- Can do MCMC, but slow
- Instead use Variational Inference
- Then filter some variables for speed up

This is a fully connected graph!

Figures from Krähenbühl & Koltun (2011)
Fully–Connected CRF

Model

\[ p(x|i) = \frac{1}{Z(i)} \exp(-E(x)) \]

\[ E(x) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j), \]

This is a fully connected graph!

Inference

- Can do MCMC, but slow
- Instead use Variational Inference
- Then filter some variables for speed up

Follow-up Work (combine with CNN)

SEMANTIC IMAGE SEGMENTATION WITH DEEP CONVOLUTIONAL NETS AND FULLY CONNECTED CRFs

Liang-Chieh Chen
Univ. of California, Los Angeles
lcchen@cs.ucla.edu

George Papandreou
Google Inc.
gpapan@google.com

Iasonas Kokkinos
CentraleSupélec and INRIA
iasonas.kokkinos@enp.fr

Kevin Murphy
Google Inc.
kpmurphy@google.com

Alan L. Yuille
Univ. of California, Los Angeles
yuille@stat.ucla.edu

ABSTRACT

Deep Convolutional Neural Networks (DCNNs) have recently shown state of the art performance in high level vision tasks, such as image classification and object detection. This work brings together methods from DCNNs and probabilistic graphical models for addressing the task of pixel-level classification (also called "semantic image segmentation"). We show that responses at the final layer of DCNNs are not sufficiently localized for accurate object segmentation. This is due to the very invariance properties that make DCNNs good for high level tasks. We overcome this poor localization property of deep networks by combining the responses at the final DCNN layer with a fully connected Conditional Random Field (CRF). Qualitatively, our “DeepLab” system is able to localize segment boundaries at a level of accuracy which is beyond previous methods. Quantitatively, our method sets the new state-of-art at the PASCAL VOC-2012 semantic image segmentation task, reaching 71.6% IOU accuracy in the test set. We show how these results can be obtained efficiently: Careful network re-purposing and a novel application of the "hole" algorithm from the wavelet community allow dense computation of neural net responses at 8 frames per second on a modern GPU.

Figures from Krähenbühl & Koltun (2011)
### Fully–Connected CRF

#### Model

\[ p(x|i) = \frac{1}{Z(i)} \exp(-E(x)) \]

\[ E(x) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j) \]

This is a fully connected graph!

#### Inference

- Can do MCMC, but slow
- Instead use Variational Inference
- Then filter some variables for speed up

Figures from Krähenbühl & Koltun (2011)

Figure 1: Pixel-level classification with a fully connected CRF. (a) Input image from the MSRC-21 dataset. (b) The response of unary classifiers used by our models. (c) Classification produced by the Robust \( P^n \) CRF [9]. (d) Classification produced by MCMC inference [17] in a fully connected pixel-level CRF model; the algorithm was run for 36 hours and only partially converged for the bottom image. (e) Classification produced by our inference algorithm in the fully connected model in 0.2 seconds.

Figure 2: Convergence analysis. (a) KL-divergence of the mean field approximation during successive iterations of the inference algorithm, averaged across 94 images from the MSRC-21 dataset. (b) Visualization of convergence on distributions for two class labels over an image from the dataset.
Figure 2: An example of a Chinese-English sentence pair with parses, word alignments, and a subset of the full optimal ITG derivation, including one totally unsynchronized bispan ($b_4$), one partially synchronized bispan ($b_7$), and and fully synchronized bispan ($b_8$). The inset provides some examples of active synchronization features (see Section 4.3) on these bispans. On this example, the monolingual English parser erroneously attached the lower PP to the VP headed by established, and the non-syntactic ITG word aligner misaligned 等 to such instead of to etc. Our joint model corrected both of these mistakes because it was rewarded for the synchronization of the two NPs joined by $b_8$. 

Figures from Burkett et al. (2010)
Joint Parsing and Alignment with Weakly Synchronized Grammars

Figures from Burkett & Klein (ACL 2013 tutorial)

Table 1: Parsing results. Our joint model has the highest reported F$_1$ for English-Chinese bilingual parsing.

<table>
<thead>
<tr>
<th></th>
<th>Test Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ch F$_1$</td>
</tr>
<tr>
<td>Monolingual</td>
<td>83.6</td>
</tr>
<tr>
<td>Reranker</td>
<td>86.0</td>
</tr>
<tr>
<td>Joint</td>
<td>85.7</td>
</tr>
</tbody>
</table>

Table 2: Word alignment results. Our joint model has the highest reported F$_1$ for English-Chinese word alignment.

<table>
<thead>
<tr>
<th></th>
<th>Test Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
</tr>
<tr>
<td>HMM</td>
<td>86.0</td>
</tr>
<tr>
<td>ITG</td>
<td>86.8</td>
</tr>
<tr>
<td>Joint</td>
<td>85.5</td>
</tr>
</tbody>
</table>
Mean-Field + Learning

How does variational inference fit into learning?

So far, we have two ways to learn parameters using variational inference:

**Maximum likelihood estimation:**
Optimize MLE over the parameters—optionally calling variational inference as a subroutine during learning

*Example:*
- train a CRF for tagging a tree given a sentence
- per-example gradient requires us to compute the marginals for that example
- instead of using BP to compute the marginals, use mean-field variational inference

*Example:* Fully-Connected CRF
*Example:* Joint Parsing and Alignment

**Variational Bayes:** treat the parameters as a variable in the model and run variational inference to estimate the distribution over those parameters

*Example:*
- learn the posterior distribution over the parameters in Gaussian LDA
- instead of sampling the topic assignments and parameters from a Gibbs sampler, use mean-field variational inference

*Example:* Gaussian Mixture Model
*Example:* Uncollapsed LDA
*Example:* Collapsed LDA
Variational Inference & Nonconvexity

• ELBO is a non-convex objective function
• Below shows 10 random initializations of CAVI for Gaussian Mixture Model
• Parameters with higher ELBO are closer to true posterior

Figure 2: Different initializations may lead CAVI to find different local optima of the ELBO.
Variational Bayesian LDA

• Explicit Variational Inference
Variational Bayesian LDA

• Explicit Variational Inference

Standard VB inference upper bounds the negative log marginal likelihood \(-\log p(x|\alpha, \beta)\) using the variational free energy:

\[
-\log p(x|\alpha, \beta) \leq \widetilde{F}(\tilde{q}(z, \theta, \phi)) = E_{\tilde{q}}[-\log p(x, z, \phi, \theta|\alpha, \beta)] - \mathcal{H}(\tilde{q}(z, \theta, \phi))
\]  
(2)

with \(\tilde{q}(z, \theta, \phi)\) an approximate posterior, \(\mathcal{H}(\tilde{q}(z, \theta, \phi)) = E_{\tilde{q}}[-\log \tilde{q}(z, \theta, \phi)]\) the variational entropy, and \(\tilde{q}(z, \theta, \phi)\) assumed to be fully factorized:

\[
\tilde{q}(z, \theta, \phi) = \prod_{ij} \tilde{q}(z_{ij} | \tilde{\gamma}_{ij}) \prod_{j} \tilde{q}(\theta_{j} | \tilde{\alpha}_{j}) \prod_{k} \tilde{q}(\phi_{k} | \tilde{\beta}_{k})
\]  
(3)

\(\tilde{q}(z_{ij} | \tilde{\gamma}_{ij})\) is multinomial with parameters \(\tilde{\gamma}_{ij}\) and \(\tilde{q}(\theta_{j} | \tilde{\alpha}_{j}), \tilde{q}(\phi_{k} | \tilde{\beta}_{k})\) are Dirichlet with parameters \(\tilde{\alpha}_{j}\) and \(\tilde{\beta}_{k}\) respectively. Optimizing \(\widetilde{F}(\tilde{q})\) with respect to the variational parameters gives us a set of updates guaranteed to improve \(\widetilde{F}(\tilde{q})\) at each iteration and converges to a local minimum:

\[
\tilde{\alpha}_{jk} = \alpha + \sum_{i} \tilde{\gamma}_{ijk}
\]  
(4)

\[
\tilde{\beta}_{kw} = \beta + \sum_{ij} 1(x_{ij} = w)\tilde{\gamma}_{ijk}
\]  
(5)

\[
\tilde{\gamma}_{ijk} \propto \exp \left( \Psi(\tilde{\alpha}_{jk}) + \Psi(\tilde{\beta}_{kw}) - \Psi(\sum_{w} \tilde{\beta}_{kw}) \right)
\]  
(6)

where \(\Psi(y) = \frac{\partial \log \Gamma(y)}{\partial y}\) is the digamma function and \(1\) is the indicator function.
For each topic $k \in \{1, \ldots, K\}$:  
\[ \phi_k \sim \text{Dir}(\beta) \]  
\text{[draw distribution over words]}

For each document $m \in \{1, \ldots, M\}$  
\[ \theta_m \sim \text{Dir}(\alpha) \]  
\text{[draw distribution over topics]}

For each word $n \in \{1, \ldots, N_m\}$  
\[ z_{mn} \sim \text{Mult}(1, \theta_m) \]  
\text{[draw topic assignment]}

\[ x_{mn} \sim \phi_{z_{mi}} \]  
\text{[draw word]}

In general, we typically choose the distribution $q_\theta(z_t)$ to be the same form as $p_\alpha(z_t \mid \text{parents}(z_t))$ in the original distribution

Ex:
- if $p_\alpha(z_t \mid \text{parents}(z_t))$ is Dirichlet, then let $q_\theta(z_t)$ be Dirichlet
- if $p_\alpha(z_t \mid \text{parents}(z_t))$ is Categorical, then let $q_\theta(z_t)$ be Categorical
- if $p_\alpha(z_t \mid \text{parents}(z_t))$ is Gaussian, then let $q_\theta(z_t)$ be Gaussian

$q(z; \psi) = \prod_{ij} \text{i}^{z_{ij}} \prod_{j} \text{j}^{z_{ij}} \prod_{k} \text{k}^{z_{ik}}$  
\( (3) \)
Collapsed Variational Bayesian LDA

• Collapsed Variational Inference

\[
\begin{align*}
\text{Dirichlet} & \quad \alpha \\
\text{Document-specific topic distribution} & \quad \theta_m \\
\text{Topic assignment} & \quad \sim \text{Approximate with } q \\
\text{Observed word} & \quad x_{mn} \\
\text{Integrated out} & \\
\text{Topic} & \quad \phi_k \\
\text{Dirichlet} & \quad \beta
\end{align*}
\]
Collapsed Variational Bayesian LDA

- **First row:** test set per word log probabilities as functions of numbers of iterations for VB, CVB and Gibbs.
- **Second row:** histograms of final test set per word log probabilities across 50 random initializations.

Figure 1: Left: results for KOS. Right: results for NIPS.

First row: test set per word log probabilities as functions of numbers of iterations for VB, CVB and Gibbs.
Second row: histograms of final test set per word log probabilities across 50 random initializations.
Online Variational Bayes for LDA

Figure 1: Top: Perplexity on held-out Wikipedia documents as a function of number of documents analyzed, i.e., the number of E steps. Online VB run on 3.3 million unique Wikipedia articles is compared with online VB run on 98,000 Wikipedia articles and with the batch algorithm run on the same 98,000 articles. The online algorithms converge much faster than the batch algorithm does.

Bottom: Evolution of a topic about business as online LDA sees more and more documents. To this end, we develop an online variational Bayes algorithm for latent Dirichlet allocation (LDA), one of the simplest topic models and one on which many others are based. Our algorithm is based on online stochastic optimization, which has been shown to produce good parameter estimates dramatically faster than batch algorithms on large datasets.
Online Variational Bayes for LDA

Algorithm 1 Batch variational Bayes for LDA

Initialize $\lambda$ randomly.

while relative improvement in $\mathcal{L}(w, \phi, \gamma, \lambda) > 0.00001$ do

$E$ step:

for $d = 1$ to $D$ do

Initialize $\gamma_{dk} = 1$. (The constant 1 is arbitrary.)

repeat

Set $\phi_{dwk} \propto \exp\{\mathbb{E}_q[\log \theta_{dk}] + \mathbb{E}_q[\log \beta_{kw}]\}$

Set $\gamma_{dk} = \alpha + \sum_w \phi_{dwk} n_{dw}$

until $\frac{1}{K} \sum_k |\text{change in } \gamma_{dk}| < 0.00001$

end for

$M$ step:

Set $\lambda_{kw} = \eta + \sum_d n_{dw} \phi_{dwk}$

end while

Algorithm 2 Online variational Bayes for LDA

Define $\rho_t \triangleq (\tau_0 + t)^{-\kappa}$.

Initialize $\lambda$ randomly.

for $t = 0$ to $\infty$ do

$E$ step:

Initialize $\gamma_{tk} = 1$. (The constant 1 is arbitrary.)

repeat

Set $\phi_{twk} \propto \exp\{\mathbb{E}_q[\log \theta_{tk}] + \mathbb{E}_q[\log \beta_{kw}]\}$

Set $\gamma_{tk} = \alpha + \sum_w \phi_{twk} n_{tw}$

until $\frac{1}{K} \sum_k |\text{change in } \gamma_{tk}| < 0.00001$

$M$ step:

Compute $\tilde{\lambda}_{kw} = \eta + D n_{tw} \phi_{twk}$

Set $\lambda = (1 - \rho_t)\lambda + \rho_t \tilde{\lambda}$.

end for

Figure 2: Held-out perplexity obtained on the Nature (left) and Wikipedia (right) corpora as a function of CPU time. For moderately large mini-batch sizes, online LDA finds solutions as good as those that the batch LDA finds, but with much less computation. When fit to a 10,000-document subset of the training corpus batch LDA’s speed improves, but its performance suffers.

Figures from Hoffman et al. (2010)
HIDDEN STATE CRFS
Case Study: Object Recognition

Data consists of images $x$ and labels $y$. 

- pigeon
- rhinoceros
- leopard
- llama
Case Study: Object Recognition

Data consists of images $x$ and labels $y$.

- Preprocess data into “patches”
- Posit a latent labeling $z$ describing the object’s parts (e.g. head, leg, tail, torso, grass)
- Define graphical model with these latent variables in mind
- $z$ is not observed at train or test time
Case Study: Object Recognition

Data consists of images $x$ and labels $y$.

- Preprocess data into “patches”
- Posit a latent labeling $z$ describing the object’s parts (e.g. head, leg, tail, torso, grass)
- Define graphical model with these latent variables in mind
- $z$ is not observed at train or test time
Case Study: Object Recognition

Data consists of images $x$ and labels $y$.

- Preprocess data into “patches”
- Posit a latent labeling $z$ describing the object’s parts (e.g. head, leg, tail, torso, grass)
- Define graphical model with these latent variables in mind
- $z$ is not observed at train or test time
Hidden-state CRFs

Data: \[ D = \{ x^{(n)}, y^{(n)} \}_{n=1}^N \]

Joint model: \[ p_\theta(y, z \mid x) = \frac{1}{Z(x, \theta)} \prod_\alpha \psi_\alpha(y_\alpha, z_\alpha, x) \]

Marginalized model: \[ p_\theta(y \mid x) = \sum_z p_\theta(y, z \mid x) \]
Hidden-state CRFs

Data: \( \mathcal{D} = \{ x^{(n)}, y^{(n)} \}_{n=1}^{N} \)

Joint model: \( p_{\theta}(y, z | x) = \frac{1}{Z(x, \theta)} \prod_{\alpha} \psi_{\alpha}(y_{\alpha}, z_{\alpha}, x) \)

Marginalized model: \( p_{\theta}(y | x) = \sum_{z} p_{\theta}(y, z | x) \)

We can train using gradient based methods:
(the values \( x \) are omitted below for clarity)

\[
\frac{d\ell(\theta | \mathcal{D})}{d\theta} = \sum_{n=1}^{N} \left( \mathbb{E}_{z \sim p_{\theta}(\cdot | y^{(n)})} [f_{j}(y^{(n)}, z)] - \mathbb{E}_{y, z \sim p_{\theta}(\cdot, \cdot)} [f_{j}(y, z)] \right)
\]

\[
= \sum_{n=1}^{N} \sum_{\alpha} \left( \sum_{z_{\alpha}} p_{\theta}(z_{\alpha} | y^{(n)}) f_{\alpha, j}(y^{(n)}_{\alpha}, z_{\alpha}) - \sum_{y_{\alpha}, z_{\alpha}} p_{\theta}(y_{\alpha}, z_{\alpha}) f_{\alpha, j}(y_{\alpha}, z_{\alpha}) \right)
\]

Inference on clamped factor graph

Inference on full factor graph
GAUSSIAN MIXTURE MODEL (GMM)
**Gaussian Mixture Model**

**Data:** \[ \mathcal{D} = \{x^{(i)}\}_{i=1}^{N} \text{ where } x^{(i)} \in \mathbb{R}^{M} \]

**Generative Story:**
- \[ z \sim \text{Categorical}(\phi) \]
- \[ x \sim \text{Gaussian}(\mu_{z}, \Sigma_{z}) \]

**Model:**

- **Joint:** \[ p(x, z; \phi, \mu, \Sigma) = p(x|z; \mu, \Sigma)p(z; \phi) \]
- **Marginal:** \[ p(x; \phi, \mu, \Sigma) = \sum_{z=1}^{K} p(x|z; \mu, \Sigma)p(z; \phi) \]

- **(Marginal) Log-likelihood:**
  \[
  \ell(\phi, \mu, \Sigma) = \log \prod_{i=1}^{N} p(x^{(i)}; \phi, \mu, \Sigma) \\
  = \sum_{i=1}^{N} \log \sum_{z=1}^{K} p(x^{(i)}|z; \mu, \Sigma)p(z; \phi)
  \]
### Mixture-Model

**Data:** \[ \mathcal{D} = \{ \mathbf{x}^{(i)} \}_{i=1}^{N} \text{ where } \mathbf{x}^{(i)} \in \mathbb{R}^M \]

**Generative Story:** \[ z \sim \text{Categorical}(\phi) \]
\[ \mathbf{x} \sim p_\theta(\cdot|z) \]

**Model:**

Joint: \[ p_{\theta,\phi}(\mathbf{x}, z) = p_\theta(\mathbf{x}|z)p_\phi(z) \]

Marginal: \[ p_{\theta,\phi}(\mathbf{x}) = \sum_{z=1}^{K} p_\theta(\mathbf{x}|z)p_\phi(z) \]

(Marginal) Log-likelihood:

\[ \ell(\theta) = \log \prod_{i=1}^{N} p_{\theta,\phi}(\mathbf{x}^{(i)}) \]
\[ = \sum_{i=1}^{N} \log \sum_{z=1}^{K} p_\theta(\mathbf{x}^{(i)}|z)p_\phi(z) \]
### Mixture-Model

#### Data:
\[
D = \{ x^{(i)} \}_{i=1}^{N} \text{ where } x^{(i)} \in \mathbb{R}^{M}
\]

#### Generative Story:
\[
z \sim \text{Categorical}(\phi)
\]
\[
x \sim p_\theta(\cdot | z)
\]

#### Model:

**Joint:**
\[
p_{\theta,\phi}(x, z) = p_\theta(x | z) p_\phi(z)
\]

**Marginal:**
\[
p_{\theta,\phi}(x) = \sum_{z=1}^{K} p_{\theta}(x | z) p_\phi(z)
\]

**(Marginal) Log-likelihood:**
\[
\ell(\theta) = \log \prod_{i=1}^{N} p_{\theta,\phi}(x^{(i)})
\]
\[
= \sum_{i=1}^{N} \log \sum_{z=1}^{K} p_\theta(x^{(i)} | z)p_\phi(z)
\]

This could be any arbitrary distribution parameterized by \( \theta \).

Today we’re thinking about the case where it is a Multivariate Gaussian.
Learning a Mixture Model

**Supervised Learning:** The parameters decouple!

\[
\mathcal{D} = \{(\mathbf{x}^{(i)}, z^{(i)})\}_{i=1}^{N}
\]

\[
\theta^*, \phi^* = \arg\max_{\theta, \phi} \sum_{i=1}^{N} \log p_{\theta}(\mathbf{x}^{(i)}|z^{(i)}) p_{\phi}(z^{(i)})
\]

\[
\theta^* = \arg\max_{\theta} \sum_{i=1}^{N} \log p_{\theta}(\mathbf{x}^{(i)}|z^{(i)})
\]

\[
\phi^* = \arg\max_{\theta} \sum_{i=1}^{N} \log p_{\phi}(z^{(i)})
\]

**Unsupervised Learning:** Parameters are coupled by marginalization.

\[
\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}
\]

\[
\theta^*, \phi^* = \arg\max_{\theta, \phi} \sum_{i=1}^{N} \log \sum_{z=1}^{K} p_{\theta}(\mathbf{x}^{(i)}|z) p_{\phi}(z)
\]
Learning a Mixture Model

**Supervised Learning:** The parameters decouple!

\[ \mathcal{D} = \{(x^{(i)}, z^{(i)})\}_{i=1}^{N} \]

Training certainly isn’t as simple as the supervised case.

In many cases, we could still use some black-box optimization method (e.g. Newton-Raphson) to solve this coupled optimization problem.

This lecture is about a more problem-specific method: EM.

**Unsupervised Learning:** Parameters are coupled by marginalization.

\[ \mathcal{D} = \{x^{(i)}\}_{i=1}^{N} \]

\[ \theta^*, \phi^* = \arg\max_{\theta, \phi} \sum_{i=1}^{N} \log \sum_{z=1}^{K} p_{\theta}(x^{(i)}|z)p_{\phi}(z) \]
EXPECTATION MAXIMIZATION
Hard Expectation-Maximization

• Initialize parameters randomly
• while not converged

1. E-Step:
   Set the latent variables to the values that maximizes likelihood, treating parameters as observed

2. M-Step:
   Set the parameters to the values that maximizes likelihood, treating latent variables as observed

Estimate unobserved variables
MLE given the estimated values of unobserved variables
(Soft) Expectation-Maximization

- Initialize parameters randomly
- while not converged
  1. **E-Step:**
     Create one training example for each possible value of the latent variables
     Weight each example according to model’s confidence
     Treat parameters as observed
  2. **M-Step:**
     Set the parameters to the values that maximizes likelihood
     Treat pseudo-counts from above as observed

Estimate unobserved variables

MLE given the estimated values of unobserved variables
Algorithm 1 Hard EM for GMMs

1: procedure HARDEM(D = \{x^{(i)}\}_{i=1}^{N})
2: Randomly initialize parameters, \(\phi, \mu, \Sigma\)
3: while not converged do
4: 
5: M-Step:

\[
\phi_k \leftarrow \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(z^{(i)} = k), \forall k
\]

\[
\mu_k \leftarrow \frac{\sum_{i=1}^{N} \mathbb{I}(z^{(i)} = k)x^{(i)}}{\sum_{i=1}^{N} \mathbb{I}(z^{(i)} = k)}, \forall k
\]

\[
\Sigma_k \leftarrow \frac{\sum_{i=1}^{N} \mathbb{I}(z^{(i)} = k)(x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T}{\sum_{i=1}^{N} \mathbb{I}(z^{(i)} = k)}, \forall k
\]

6: return \((\phi, \mu, \Sigma)\)

---

Algorithm 1 Soft EM for GMMs

1: procedure SOFTEM(D = \{x^{(i)}\}_{i=1}^{N})
2: Randomly initialize parameters, \(\phi, \mu, \Sigma\)
3: while not converged do
4: 
5: M-Step:

\[
c_k^{(i)} \leftarrow p(z^{(i)} = k|x^{(i)}; \phi, \mu, \Sigma)
\]

\[
\phi_k \leftarrow \frac{1}{N} \sum_{i=1}^{N} c_k^{(i)}, \forall k
\]

\[
\mu_k \leftarrow \frac{\sum_{i=1}^{N} c_k^{(i)} x^{(i)}}{\sum_{i=1}^{N} c_k^{(i)}}, \forall k
\]

\[
\Sigma_k \leftarrow \frac{\sum_{i=1}^{N} c_k^{(i)} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T}{\sum_{i=1}^{N} c_k^{(i)}}, \forall k
\]

6: return \((\phi, \mu, \Sigma)\)
Posterior Inference for Mixture Model

We obtain the posterior $p(z^{(i)} = k | x^{(i)}; \phi, \mu, \Sigma)$ as follows:

$$
p(z^{(i)} = k | x^{(i)}; \phi, \mu, \Sigma) = \frac{p(x^{(i)} | z^{(i)} = k; \mu, \Sigma)p(z^{(i)} = k; \phi)}{\sum_{j=1}^{K} p(x^{(i)} | z^{(i)} = j; \mu, \Sigma)p(z^{(i)} = j; \phi)}
$$

(1)
EXAMPLE: K-MEANS VS GMM
Example: K-Means
Example: K-Means
Example: K-Means

Clustering with K-Means (k=3, iter=0)
Example: K-Means

Clustering with K-Means (k=3, iter=1)
Example: K-Means

Clustering with K-Means (k=3, iter=2)
Example: K-Means

Clustering with K-Means (k=3, iter=3)
Example: K-Means
Example: K-Means

Clustering with K-Means (k=3, iter=5)
Example: GMM
Example: GMM
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=0)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=1)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=2)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=3)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=4)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=5)
Example: GMM

Clustering with GMM (k=3, init= random, cov= spherical, iter=6)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=7)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=8)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=9)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=10)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=11)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=12)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=13)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=14)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=15)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=16)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=17)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=18)
Example: GMM

Clustering with GMM (k=3, init=random, cov=spherical, iter=19)
K-Means vs. GMM

Convergence:
K-Means tends to converge much faster than a GMM.

Speed:
Each iteration of K-Means is computationally less intensive than each iteration of a GMM.

Initialization:
To initialize a GMM, we typically first run K-Means and use the resulting cluster centers as the means of the Gaussian components.

Output:
A GMM yields a probability distribution over the cluster assignment for each point; whereas K-Means gives a single hard assignment.