Lecture 8:

Latent Variable Models

Sam Roweis

January 28, 2004

Partially Unobserved (Missing) Variables

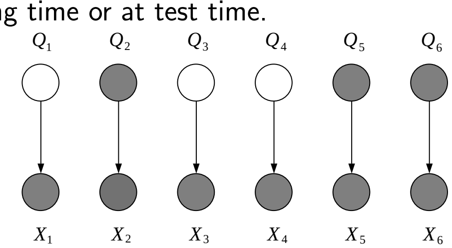
- If variables are occasionally unobserved they are *missing data*.
  e.g. undefinied inputs, missing class labels, erroneous target values
- In this case, we can still model the joint distribution, but we define a new cost function in which we *sum out* or *marginalize* the missing values at training or test time:

$$\ell(\theta; \mathcal{D}) = \sum_{\text{complete}} \log p(\mathbf{x}^c, \mathbf{y}^c|\theta) + \sum_{\text{missing}} \log p(\mathbf{x}^m|\theta)$$
$$= \sum_{\text{complete}} \log p(\mathbf{x}^c, \mathbf{y}^c|\theta) + \sum_{\text{missing}} \log \sum_{\mathbf{y}} p(\mathbf{x}^m, \mathbf{y}|\theta)$$

[Recall that $p(x) = \sum_q p(x, q)$.]

Unobserved Variables

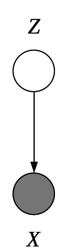
- Certain variables Q in our models may be *unobserved*,
  either some of the time or always,
  either at training time or at test time.



Graphically, we will use shading to indicate observation.

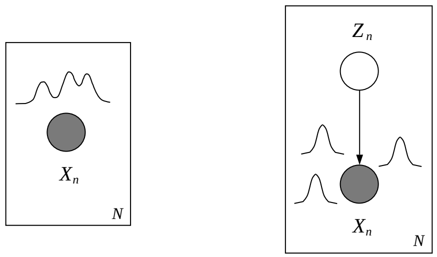
Latent Variables

- What to do when a variable $\mathbf{z}$ is *always* unobserved?
  Depends on where it appears in our model. If we never condition on it when computing the probability of the variables we *do* observe, then we can just forget about it and integrate it out.
  e.g. given $\mathbf{y}, \mathbf{x}$ fit the model $p(\mathbf{z}, \mathbf{y}|\mathbf{x}) = p(\mathbf{z}|\mathbf{y})p(\mathbf{y}|\mathbf{x}, \mathbf{w})p(\mathbf{w})$.
  (In other words if it is a leaf node.)
- But if $\mathbf{z}$ is conditioned on, we need to model it:
  e.g. given $\mathbf{y}, \mathbf{x}$ fit the model $p(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{y}|\mathbf{x}, \mathbf{z})p(\mathbf{z})$



Where Do Latent Variables Come From?

- Latent variables may appear naturally, from the structure of the problem, because something wasn't measured, because of faulty sensors, occlusion, privacy, etc.
- But also, we may want to *intentionally* introduce latent variables to model complex dependencies between variables without looking at the dependencies between them directly.
  This can actually simplify the model (e.g. mixtures).



(a) (b)

Why is Learning Harder?

- In fully observed iid settings, the probability model is a product thus the log likelihood is a sum where terms decouple.
  (At least for directed models.)

$$\begin{aligned}\ell(\theta; \mathcal{D}) &= \log p(\mathbf{x}, \mathbf{z}|\theta) \\ &= \log p(\mathbf{z}|\theta_z) + \log p(\mathbf{x}|\mathbf{z}, \theta_x)\end{aligned}$$

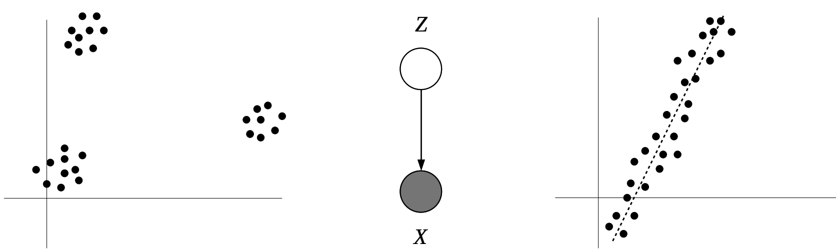
- With latent variables, the probability already contains a sum, so the log likelihood has all parameters coupled together via $\log \sum()$:

$$\begin{aligned}\ell(\theta; \mathcal{D}) &= \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}|\theta) \\ &= \log \sum_{\mathbf{z}} p(\mathbf{z}|\theta_z) p(\mathbf{x}|\mathbf{z}, \theta_x)\end{aligned}$$

(Just as with the partition function in undirected models.)

Clustering vs. Classification
Latent Factor Models vs. Regression

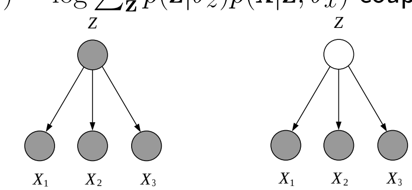
- You can think of clustering as the problem of classification with missing class labels.



- You can think of factor models (such as factor analysis, PCA, ICA, etc.) as linear or nonlinear regression with missing inputs.

Learning with Latent Variables

- Likelihood $\ell(\theta; \mathcal{D}) = \log \sum_{\mathbf{z}} p(\mathbf{z}|\theta_z) p(\mathbf{x}|\mathbf{z}, \theta_x)$ couples parameters:

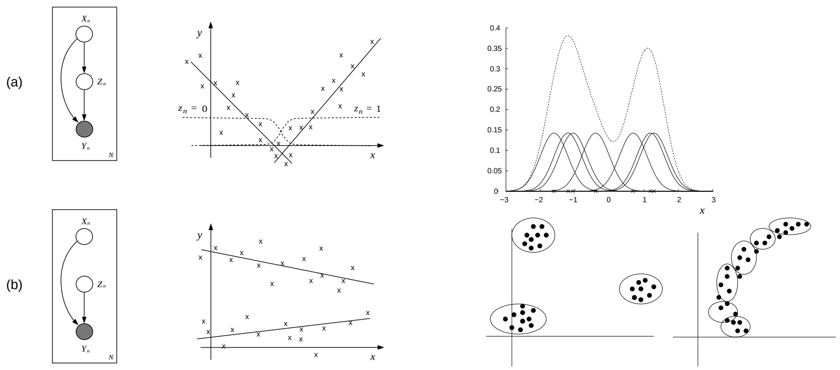


(a) (b)

- We can treat this as a black box probability function and just try to optimize the likelihood as a function of θ (e.g. gradient descent). However, sometimes taking advantage of the latent variable structure can make parameter estimation easier.
- Good news: soon we will see the *EM algorithm* which allows us to treat learning with latent variables using fully observed tools.
- Basic trick: guess the values you don't know.
  Basic math: use convexity to lower bound the likelihood.

Mixture Models

- Most basic latent variable model with a single discrete node z.
- Allows different submodels (experts) to contribute to the (conditional) density model in different parts of the space.
- Divide and conquer idea: use simple parts to build complex models. (e.g. multimodal densities, or piecewise-linear regressions).



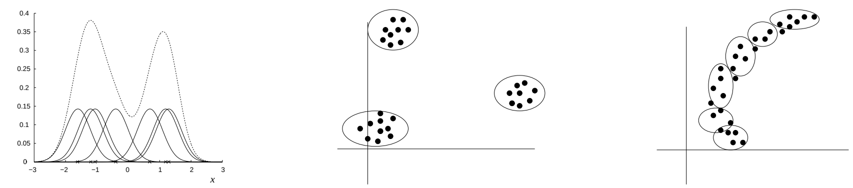
Clustering Example: Gaussian Mixture Models

- Consider a mixture of K Gaussian components:

$$p(\mathbf{x}|\theta) = \sum_k \alpha_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)$$

$$p(z = k|\mathbf{x}, \theta) = \frac{\alpha_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)}{\sum_j \alpha_j \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)}$$

$$\ell(\theta; \mathcal{D}) = \sum_n \log \sum_k \alpha_k \mathcal{N}(\mathbf{x}^n|\mu_k, \Sigma_k)$$



- Density model: $p(x|\theta)$ is a familiarity signal.
  Clustering: $p(z|\mathbf{x}, \theta)$ is the assignment rule, $-\ell(\theta)$ is the cost.

Mixture Densities

- Exactly like a classification model but the class is unobserved and so we sum it out. What we get is a perfectly valid density:

$$p(\mathbf{x}|\theta) = \sum_{k=1}^{K} p(z = k|\theta_z) p(\mathbf{x}|z = k, \theta_k)$$
$$= \sum_k \alpha_k p_k(\mathbf{x}|\theta_k)$$

  where the "mixing proportions" add to one: $\sum_k \alpha_k = 1$.
- We can use Bayes' rule to compute the posterior probability of the mixture component given some data:

$$p(z = k|\mathbf{x}, \theta) = \frac{\alpha_k p_k(\mathbf{x}|\theta_k)}{\sum_j \alpha_j p_j(\mathbf{x}|\theta_j)}$$

  these quantities are called *responsibilities*.

Regression Example: Mixtures of Experts

- Also called conditional mixtures. Exactly like a class-conditional model but the class is unobserved and so we sum it out again:

$$p(\mathbf{y}|\mathbf{x}, \theta) = \sum_{k=1}^{K} p(z = k|\mathbf{x}, \theta_z) p(\mathbf{y}|z = k, \mathbf{x}, \theta_k)$$
$$= \sum_k \alpha_k(\mathbf{x}|\theta_z) p_k(\mathbf{y}|\mathbf{x}, \theta_k)$$

  where $\sum_k \alpha_k(\mathbf{x}) = 1 \quad \forall \mathbf{x}$.
- Harder: must learn $\alpha(\mathbf{x})$ (unless chose z independent of $\mathbf{x}$).
- We can still use Bayes' rule to compute the posterior probability of the mixture component given some data:

$$p(z = k|\mathbf{x}, \mathbf{y}, \theta) = \frac{\alpha_k(\mathbf{x}) p_k(\mathbf{y}|\mathbf{x}, \theta_k)}{\sum_j \alpha_j(\mathbf{x}) p_j(\mathbf{y}|\mathbf{x}, \theta_j)}$$

  this function is often called the *gating function*.

Example: Mixture of Linear Regression Experts

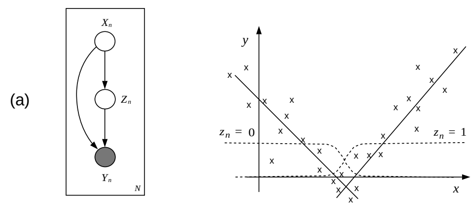
- Each expert generates data according to a linear function of the input plus additive Gaussian noise:

$$p(y|\mathbf{x},\theta) = \sum_k \alpha_k(\mathbf{x})\mathcal{N}(y|\beta_k^\top \mathbf{x}, \sigma_k^2)$$

- The "gate" function can be a softmax classification machine:

$$\alpha_k(\mathbf{x}) = p(z=k|\mathbf{x}) = \frac{e^{\eta_k^\top \mathbf{x}}}{\sum_j e^{\eta_j^\top \mathbf{x}}}$$

- Remember: we are not modeling the density of the inputs $\mathbf{x}$.



Parameter Constraints

- If we want to use general optimizations (e.g. conjugate gradient) to learn latent variable models, we often have to make sure parameters respect certain constraints. (e.g. $\sum_k \alpha_k = 1$, Σ_k pos.definite).
- A good trick is to reparameterize these quantities in terms of unconstrained values. For mixing proportions, use the softmax:

$$\alpha_k = \frac{\exp(q_k)}{\sum_j \exp(q_j)}$$

- For covariance matrices, use the Cholesky decomposition:

$$\Sigma^{-1} = A^\top A$$

$$|\Sigma|^{-1/2} = \prod_i A_{ii}$$

where A is upper diagonal with positive diagonal:

$$A_{ii} = \exp(r_i) > 0 \qquad A_{ij} = a_{ij} \quad (j > i) \qquad A_{ij} = 0 \quad (j < i)$$

Gradient Learning with Mixtures

- We can learn mixture densities using gradient descent on the likelihood as usual. The gradients are quite interesting:

$$\ell(\theta) = \log p(\mathbf{x}|\theta) = \log \sum_k \alpha_k p_k(\mathbf{x}|\theta_k)$$

$$\frac{\partial \ell}{\partial \theta} = \frac{1}{p(\mathbf{x}|\theta)} \sum_k \alpha_k \frac{\partial p_k(\mathbf{x}|\theta_k)}{\partial \theta}$$

$$= \sum_k \alpha_k \frac{1}{p(\mathbf{x}|\theta)} p_k(\mathbf{x}|\theta_k) \frac{\partial \log p_k(\mathbf{x}|\theta_k)}{\partial \theta}$$

$$= \sum_k \alpha_k \frac{p_k(\mathbf{x}|\theta_k)}{p(\mathbf{x}|\theta)} \frac{\partial \ell_k}{\partial \theta_k} = \sum_k \alpha_k r_k \frac{\partial \ell_k}{\partial \theta_k}$$

- In other words, the gradient is the *responsibility weighted sum* of the individual log likelihood gradients.

Logsum

- Often you can easily compute $b_k = \log p(\mathbf{x}|z=k,\theta_k)$, but it will be very negative, say -10^6 or smaller.
- Now, to compute $\ell = \log p(\mathbf{x}|\theta)$ you need to compute $\log \sum_k e^{b_k}$. (e.g. for calculating responsibilities at test time or for learning)
- Careful! Do not compute this by doing `log(sum(exp(b)))`. You will get underflow and an incorrect answer.
- Instead do this:
  - Add a constant exponent B to all the values b_k such that the largest value comes close to the maxiumum exponent allowed by machine precision: `B = MAXEXPONENT-log(K)-max(b)`.
  - Compute `log(sum(exp(b+B)))-B`.
- Example: if $\log p(x|z=1) = -120$ and $\log p(x|z=2) = -120$, what is $\log p(x) = \log\left[p(x|z=1) + p(x|z=2)\right]$? Answer: $\log[2e^{-120}] = -120 + \log 2$.