Machine learning for vision

Winter 2016

Roland Memisevic

Lecture 8, April 12, 2016

The "vision equation"

- The purpose of vision: Infer world properties (or hidden "causes"), $\boldsymbol{z}$, from an image, $\boldsymbol{x}$.
- We can express this with an *analysis*, *inference*, *encoder* or *backward* equation:

$$\boldsymbol{z} = g(\boldsymbol{x})$$

- Learning amounts to estimating the parameters of g from training data.

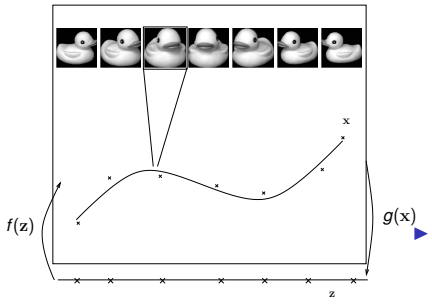
Latent variables and generative models

- In practice, it can be easier to express how images get formed *given* the causes, leading to the *synthesis*, or *decoder*, or *forward* equation:

$$\boldsymbol{x} = f(\boldsymbol{z})$$

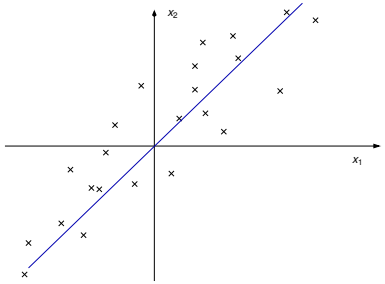
- It describes how images depend on the state of the world.
- $\boldsymbol{z}$ is called "latent variable" or "hidden variable", because unlike the image, $\boldsymbol{x}$, we do not observe it.

Manifold learning



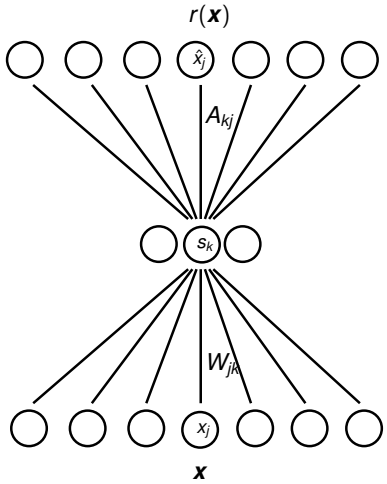
- When the dimensionality of the latent variables is smaller than the dimensionality of the data, then the data will be distributed along some lower-dimensional *manifold* in the dataspace.
- Learning the manifold is also known as *dimensionality reduction*.

Principal Components Analysis (PCA)



- If we assume the manifold to be *linear*, learning is easy and can done in closed form.
- (PCA can also be formulated as a probabilistic model.)

Autoencoders

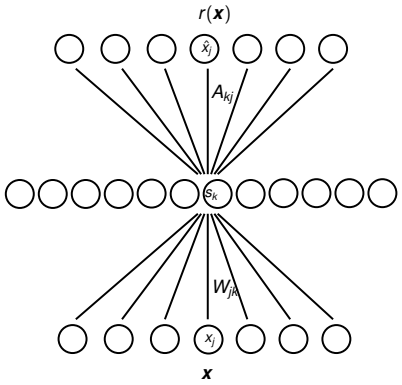


- Autoencoders are simple neural networks that are trained to reconstruct their input:

$$\text{cost} = \|r(\boldsymbol{x}) - \boldsymbol{x}\|^2$$

- The hidden layer is a bottleneck that forces the model to compress its input.
- Linear autoencoders implement a variation of PCA (Baldi, Hornik; 1989)

Overcomplete autoencoders



- With overcomplete hiddens, the model will "cheat" and learn the identity.
- One solution: Denoising autoencoders corrupt the inputs during training, but train the model to reconstruct the original, uncorrupted inputs (Vincent et al. 2008):

$$\text{cost} = \|r(\boldsymbol{x} + \text{noise}) - \boldsymbol{x}\|^2$$

Latent variables and generative models

- To deal with ambiguities and uncertainties, we can re-phrase the forward equation as a *conditional probability*:

$$\boldsymbol{x} \sim p(\boldsymbol{x}|\boldsymbol{z})$$

in which case analysis follows from Bayes' rule

$$p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{x})}$$

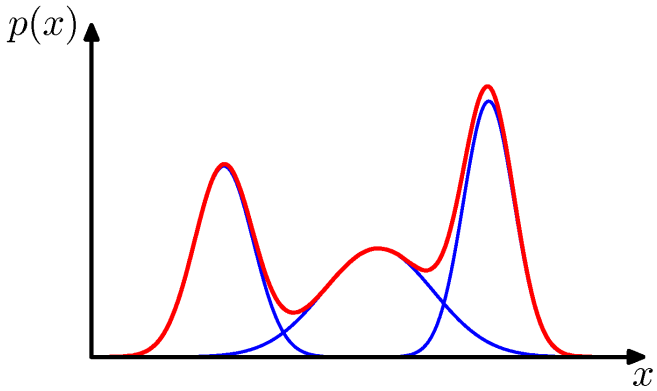
- We also need a *prior*, $p(\boldsymbol{z})$, over the latent variables.

Mixtures of Gaussians

$$p(\mathbf{x}) = \sum_k \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

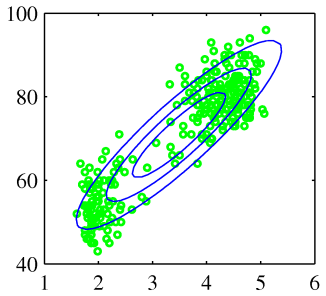
- $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ are parameters. π_k are called *mixing proportions*, each Gaussian is called a *mixture component*.
- The model is simply a weighted sum of Gaussians.
- It is much more powerful than a single Gaussian, because it can model *multi-modal* distributions:

Gaussian mixture models example

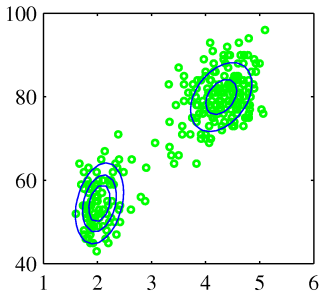


- A mixture of three Gaussians.

Gaussian mixture models example



Gaussian fit to some data.



Gaussian mixture fit to the data.

Gaussian mixture as latent variable model

- For $p(\mathbf{x})$ to be a probability distribution, we require

$$\sum_k \pi_k = 1 \quad \text{and} \quad \pi_k > 0 \quad \forall k$$

- Thus, we may interpret the π_k as probabilities themselves!
- This motivates introducing latent variables $\mathbf{z}$ and re-writing the model, equivalently, in terms of two distributions $p(\mathbf{z})$ and $p(\mathbf{z}|\mathbf{x})$ as follows:

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{z}) p(\mathbf{x}|\mathbf{z})$$

where $p(\mathbf{z}) = \prod_{k=1}^K \pi_k^{z_k}$ is a discrete distribution (use $\mathbf{z}$ in one-hot encoding), and $p(\mathbf{x}|z_k = 1) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is a conditional Gaussian.

Gaussian mixture models

- We can now think of the model as a generative process:
- To generate a data-point, first draw a mixture component, then draw the observation from a Gaussian, whose parameters depend on the component.
- To compute posteriors (hiddens given data), use Bayes' rule

$$p(\mathbf{z}_n|\mathbf{x}_n) = \frac{p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)}{\sum_{\mathbf{z}_n} p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)}$$

(which represent how likely a given observation $\mathbf{x}_n$ is to come from a particular mixture component).

- $p(z_{nk} = 1|\mathbf{x}_n)$ (abbreviate $\gamma(z_{kn})$) is usuall called *responsibility* of mixture component k.

The (variational) EM algorithm

- We could use gradient-based for learning, but this view inspired a procedure (EM), that minimizes a sequence of lower bounds:

$$\begin{aligned} L := \sum_n \log p(\mathbf{x}_n) &= \sum_n \log \sum_{\mathbf{z}_n} p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n) \\ &= \sum_n \log \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \frac{p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)}{q(\mathbf{z}_n)} \\ &\geq \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log \frac{p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)}{q(\mathbf{z}_n)} \\ &:= \mathcal{L} \end{aligned}$$

where we use **Jensen's inequality**:
$\log \sum_i a_i b_i \geq \sum_i a_i \log b_i$ if $\forall i : a_i > 0$ and $\sum_i a_i = 1$

- As we shall see, separately optimizing $\mathcal{L}$ and the q's is easy.

The EM algorithm for Gaussian mixtures

- To avoid clutter, it is convenient to rewrite

$$\mathcal{L} = \sum_{nk} q_{nk} \log p(\mathbf{x}_n|\mathbf{z}_n = k)p(\mathbf{z}_n = k) - \sum_{nk} q_{nk} \log q_{nk}$$

where we abbreviate $q_{nk} = q(\mathbf{z}_n = k)$

- Note that the first term of $\mathcal{L}$ is the expectation of $p(\mathbf{x}_n, \mathbf{z}_n)$ with respect to $q(\mathbf{z}_n)$. It is commonly referred to as "expected complete log-likelihood".
- This is the only term that depends on model parameters. We can set derivatives to zero to get closed-form solutions:

Optimizing $\mathcal{L}$

- For the means:

$$\frac{\partial \mathcal{L}}{\partial \mu_k} = \sum_n q_{nk} \boldsymbol{\Sigma}_k (\mathbf{x}_n - \boldsymbol{\mu}_k) = 0 \Leftrightarrow \boldsymbol{\mu}_k = \frac{\sum_n q_{nk} \mathbf{x}_n}{\sum_n q_{nk}}$$

- For the variances:

$$\boldsymbol{\Sigma}_k = \frac{\sum_n q_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}}{\sum_n q_{nk}}$$

- For the mixing proportions:

$$\pi_k = p(\mathbf{z}_k) = \frac{\sum_n q_{nk}}{N}$$

The (variational) EM algorithm

- Before optimizing the q's, let us ask the following question: *What is the gap between L and $\mathcal{L}$?*
- To answer this question, rewrite $\mathcal{L}$ differently:

$$\begin{aligned}
\mathcal{L} &= \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log \frac{p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)}{q(\mathbf{z}_n)} \\
&= \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n|\mathbf{x}_n)p(\mathbf{x}_n)}{q(\mathbf{z}_n)} \\
&= \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n|\mathbf{x}_n)}{q(\mathbf{z}_n)} + \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log p(\mathbf{x}_n) \\
&= -\sum_n \mathrm{KL}\left(q(\mathbf{z}_n) \,||\, p(\mathbf{z}_n|\mathbf{x}_n) \right) + L
\end{aligned}$$

- (Neal, Hinton 1998)

The (variational) EM algorithm

- The Kullback-Leibler divergence (KL-divergence)

$$\mathrm{KL}\left(p_1(\mathbf{z}) \,||\, p_2(\mathbf{z}) \right) = \sum_{\mathbf{z}} p_1(\mathbf{z}) \log \frac{p_1(\mathbf{z})}{p_2(\mathbf{z})}$$

  measures the similarity between two probability distributions p_1 and p_2.
- It is always non-negative, and it is zero only for identical distributions.
- This means that $\mathcal{L}$ will be equal to L if we set $q(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{x}_n)$!
- It also suggests that just improving $q(\mathbf{z}_n)$ may work, too (and cross our fingers...)

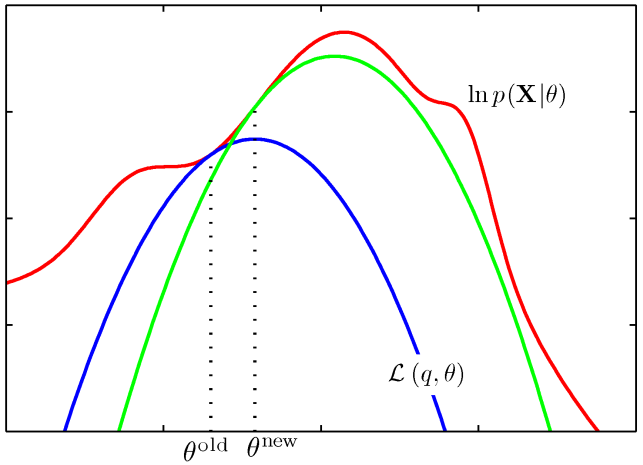
The EM algorithm for Gaussian mixtures

- After updating the parameters, setting $q(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{x}_n)$ will make the bound $\mathcal{L}$ on L tight again.
- Recall that $p(\mathbf{z}_n|\mathbf{x}_n) = (\gamma(z_{nk}))$ is easy to compute using Bayes' rule.
- Thus, we have a simple, iterative two-step algorithm:

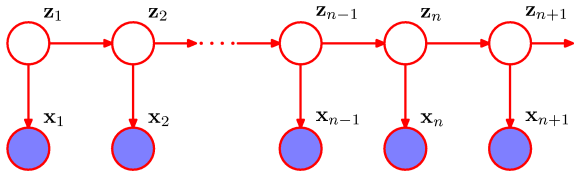
EM algorithm

1. E-step: Evaluate the posteriors $p(\mathbf{z}_n|\mathbf{x}_n)$.
2. M-step: Optimize $\mathcal{L}$ with respect to the model parameters, keeping $q(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{x}_n)$ fixed.

EM as optimizing a sequence of (tight) lower bounds



Hidden Markov Models



$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \sum_{\mathbf{z}_1, \ldots, \mathbf{z}_N} p(\mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{z}_1, \ldots, \mathbf{z}_N)$$

with

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{z}_1, \ldots, \mathbf{z}_N) = p(\mathbf{z}_1) \prod_{n=2}^{N} p(\mathbf{z}_n|\mathbf{z}_{n-1}) \prod_{m=1}^{N} p(\mathbf{x}_m|\mathbf{z}_m)$$

- Exact inference of the posterior (and EM training) is trickier than in a simple MoG, but do-able using dynamic programming.

Variational EM

- In a more powerful/interesting/realistic model than a simple mixture model, we may not be able to infer exact posteriors to tighten the lower bound.
- Idea (see e.g. Helmholtz machine, or more recently NVIL and VAE): Use neural networks for the synthesis model and for approximating the corresponding posteriors $q(\boldsymbol{z}|\boldsymbol{x})$.
- Then iteratively optimize the model and inference networks.

Variational autoencoder

- Let us rewrite $\mathcal{L}$ again:

$$\begin{aligned}
\mathcal{L} &= \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log \frac{p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)}{q(\mathbf{z}_n)} \\
&= \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log p(\mathbf{x}_n|\mathbf{z}_n) + \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n)}{q(\mathbf{z}_n)} \\
&= \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log p(\mathbf{x}_n|\mathbf{z}_n) - \sum_n \mathrm{KL}\left(q(\mathbf{z}_n) \,||\, p(\mathbf{z}_n) \right)
\end{aligned}$$

- The second term is easy (if we define q and p appropriately.
- For the first term we can do a sampling approximation
- (Note that $q(\mathbf{z}_n)$ really means $q(\mathbf{z}_n|\mathbf{x}_n)$)

Generative models for actual generation

- VAE's allow us to generate data by sampling from the prior.
- VAE suffer from some unexplained issues, like collapsing decoder-weights for hiddens that satisfy the KL term.
- An alternative are Generative Adversarial Networks (Goodfellow et al 2014) where the generator network is combined with a recognition network trained to tell real data from false data.

What is wrong with unsupervised learning?

- Despite 30+ years of research unsupervised learning never took off.
- Will its time still come (like it has for neural nets)? Or is it fundamentally flawed?
- One hypothesis why UL is the wrong approach (but transfer learning should work):
- **There is too much structure in natural data, more than is relevant for humans. Teasing out *the structure* in natural data, as attempted by UL, may simply be overkill**.