Lecture 8: Unsupervised Learning & EM Algorithm

Sam Roweis

November 2, 2004

Partially Unobserved Variables

- Certain variables $\mathbf{q}$ in our models may be *unobserved*, either at training time or at test time or both.
- If the are occasionally unobserved they are *missing data*. e.g. undefinied inputs, missing class labels, erroneous target values
- In this case, we define a new cost function in which we *integrate out* the missing values at training or test time:

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

- Variables which are *always* unobserved are called *latent variables* or sometimes *hidden variables*.

Recall: Missing Outputs

- Remember that you can think of unsupervised learning as supervised learning in which all the outputs are *missing*:
  - Clustering == classification with missing labels.
  - Dimensionality reduction == regression with missing targets.
- Density estimation is actually very general and encompasses the two problems above and a whole lot more.
- Today, let's focus on the idea of missing (unobserved) variables...

Latent Variables

- What should we 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 $p(\mathbf{z},\mathbf{y}|\mathbf{x}) = p(\mathbf{z}|\mathbf{y})p(\mathbf{y}|\mathbf{x},\mathbf{w})p(\mathbf{w})$.
- But if $\mathbf{z}$ is conditioned on, we need to model it: e.g. given $\mathbf{y},\mathbf{x}$ fit the model $p(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{y}|\mathbf{x},\mathbf{z})p(\mathbf{z})$
- Latent variables may appear naturally, from the structure of the problem. 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).

Why is Learning Harder with Hidden Variables?

- In fully observed settings, the probability model is a product thus the log likelihood is a sum where terms decouple.

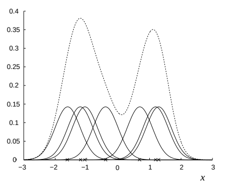
$$\begin{aligned}\ell(\theta;\mathcal{D}) &= \sum_n \log p(\mathbf{y}_n, \mathbf{x}_n|\theta) \\ &= \sum_n \log p(\mathbf{x}_n|\theta_x) + \sum_n \log p(\mathbf{y}_n|\mathbf{x}_n, \theta_y)\end{aligned}$$

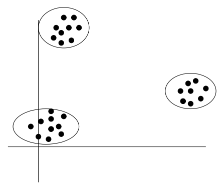
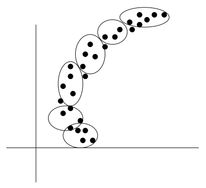
- With latent variables, the probability already contains a sum, so the log likelihood has all parameters coupled together:

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

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).



Learning with Latent Variables

- Likelihood $\ell(\theta) = \log \sum_{\mathbf{z}} p(\mathbf{z}|\theta_z) p(\mathbf{x}|\mathbf{z}, \theta_x)$ couples parameters:
- We can treat this as a black box probability function and just try to optimize the likelihood as a function of θ.
  We did this many times before by taking gradients.
- However, sometimes taking advantage of the latent variable structure can make parameter estimation easier.
- Good news: today 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 Densities

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

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

  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:

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

  these quantities are called *responsibilities*.
  You've seen them many times before; now you know their names!

Learning with Mixtures

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

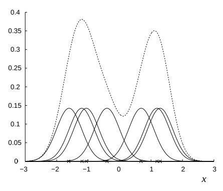
$$\ell(\theta) = \log p(\mathbf{x}|\theta) = \log \sum_k \alpha_k p_k(\mathbf{x}|\theta_k)$$
$$\frac{\partial \ell}{\partial \theta} = \frac{1}{p(\mathbf{x}|\theta)} \sum_k \alpha_k \frac{\partial p_k(\mathbf{x}|\theta_k)}{\partial \theta}$$
$$= \sum_k \alpha_k \frac{1}{p(\mathbf{x}|\theta)} p_k(\mathbf{x}|\theta_k)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

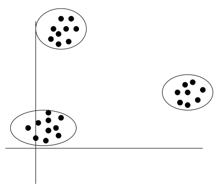
- In other words, the gradient is the *responsibility weighted sum* of the individual log likelihood gradients. (cf. MOEs)

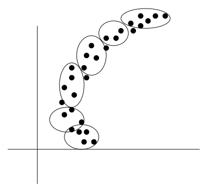
Clustering Example: Gaussian Mixture Models

- 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.

Conditional Mixtures: MOEs Revisited

- Mixtures of Experts are also called conditional mixtures.
  Exactly like a class-conditional classification model, except the class is unobserved and so we sum it out:

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

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

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

Mixure of Gaussians Learning

- We can learn mixtures of Gaussians using gradient descent. For example, the gradients of the means:

$$\ell(\theta) = \log p(\mathbf{x}|\theta) = \log \sum_k \alpha_k p_k(\mathbf{x}|\theta_k)$$
$$\frac{\partial \ell}{\partial \theta} = \sum_k \alpha_k r_k \frac{\partial \ell_k}{\partial \theta_k} = \sum_k \alpha_k r_k \frac{\partial \log p_k(\mathbf{x}|\theta_k)}{\partial \theta}$$
$$\frac{\partial \ell}{\partial \mu_k} = -\sum_k \alpha_k r_k \Sigma_k^{-1}(\mathbf{x} - \mu_k)$$

- Gradients of covariance matrices are harder: require derivatives of log determinants and quadratic forms.
- Must ensure that mixing proportions α_k are positive and sum to unity and that covariance matrices are positive definite.

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:

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

Recap: Learning with Latent Variables

- With latent variables, the probability contains a sum, so the log likelihood has all parameters coupled together:

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

(we can also consider continuous $\mathbf{z}$ and replace $\sum$ with $\int$)
- If the latent variables were observed, parameters would decouple again and learning would be easy:

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

- One idea: ignore this fact, compute $\partial\ell/\partial\theta$, and do learning with a smart optimizer like conjugate gradient.
- Another idea: what if we use our current parameters to *guess* the values of the latent variables, and then do fully-observed learning? This back-and-forth trick might make optimization easier.

Expectation-Maximization (EM) Algorithm

- Iterative algorithm with two linked steps:
  E-step: fill in values of $\hat{\mathbf{z}}^t$ using $p(\mathbf{z}|\mathbf{x}, \theta^t)$.
  M-step: update parameters using $\theta^{t+1} \leftarrow \mathrm{argmax}\ \ell(\theta; \mathbf{x}, \hat{\mathbf{z}}^t)$.
- E-step involves inference, which we need to do at runtime anyway. M-step is no harder than in fully observed case.
- We will prove that this procedure monotonically improves ℓ (or leaves it unchanged). Thus it always converges to a local optimum of the likelihood (as any optimizer should).
- Note: EM is an optimization strategy for objective functions that can be interpreted as likelihoods in the presence of missing data.
- EM is *not* a cost function such as "maximum-likelihood". EM is *not* a model such as "mixture-of-Gaussians".

Complete & Incomplete Log Likelihoods

- Observed variables $\mathbf{x}$, latent variables $\mathbf{z}$, parameters θ:

$$\ell_c(\theta; \mathbf{x}, \mathbf{z}) = \log p(\mathbf{x}, \mathbf{z}|\theta)$$

is the *complete log likelihood*.
- Usually optimizing $\ell_c(\theta)$ given both $\mathbf{z}$ and $\mathbf{x}$ is straightforward. (e.g. class conditional Gaussian fitting, linear regression)
- With $\mathbf{z}$ unobserved, we need the log of a marginal probability:

$$\ell(\theta; \mathbf{x}) = \log p(\mathbf{x}|\theta) = \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}|\theta)$$

which is the *incomplete log likelihood*.

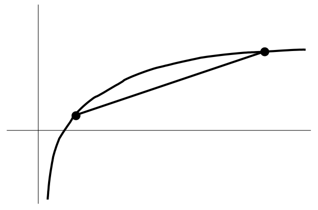
Expected Complete Log Likelihood

- For *any* distribution $q(\mathbf{z})$ define *expected complete log likelihood*:

$$\ell_q(\theta;\mathbf{x}) = \langle \ell_c(\theta;\mathbf{x},\mathbf{z})\rangle_q \equiv \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x},\mathbf{z}|\theta)$$

- Amazing fact: $\ell(\theta) \geq \ell_q(\theta) + \mathcal{H}(q)$ because of concavity of $\log$:

$$\begin{aligned}
\ell(\theta;\mathbf{x}) &= \log p(\mathbf{x}|\theta)\\
&= \log \sum_{\mathbf{z}} p(\mathbf{x},\mathbf{z}|\theta)\\
&= \log \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \frac{p(\mathbf{x},\mathbf{z}|\theta)}{q(\mathbf{z}|\mathbf{x})}\\
&\geq \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x},\mathbf{z}|\theta)}{q(\mathbf{z}|\mathbf{x})}
\end{aligned}$$



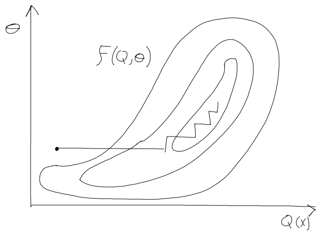
- Where the inequality is called *Jensen's inequality*.
  (It is only true for distributions: $\sum q(\mathbf{z}) = 1;\ q(\mathbf{z}) > 0$.)

Lower Bounds and Free Energy

- For fixed data $\mathbf{x}$, define a functional called the *free energy*:

$$F(q,\theta) \equiv \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x},\mathbf{z}|\theta)}{q(\mathbf{z}|\mathbf{x})} \quad \leq \ell(\theta)$$

- The EM algorithm is coordinate-ascent on F:
  E-step: $q^{t+1} = \mathrm{argmax}_q \quad F(q,\theta^t)$
  M-step: $\theta^{t+1} = \mathrm{argmax}_\theta \quad F(q^{t+1},\theta^t)$



M-step: maximization of expected ℓ_c

- Note that the free energy breaks into two terms:

$$\begin{aligned}
F(q,\theta) &= \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x},\mathbf{z}|\theta)}{q(\mathbf{z}|\mathbf{x})}\\
&= \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x},\mathbf{z}|\theta) - \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log q(\mathbf{z}|\mathbf{x})\\
&= \ell_q(\theta;\mathbf{x}) + \mathcal{H}(q)
\end{aligned}$$

  (this is where its name comes from)
- The first term is the expected complete log likelihood (energy) and the second term, which does not depend on θ, is the entropy.
- Thus, in the M-step, maximizing with respect to θ for fixed q we only need to consider the first term:

$$\theta^{t+1} = \mathrm{argmax}_\theta\ \ell_q(\theta;\mathbf{x}) = \mathrm{argmax}_\theta \sum_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x},\mathbf{z}|\theta)$$

E-step: inferring latent posterior

- Claim: the optimim setting of q in the E-step is:

$$q^{t+1} = p(\mathbf{z}|\mathbf{x},\theta^t)$$

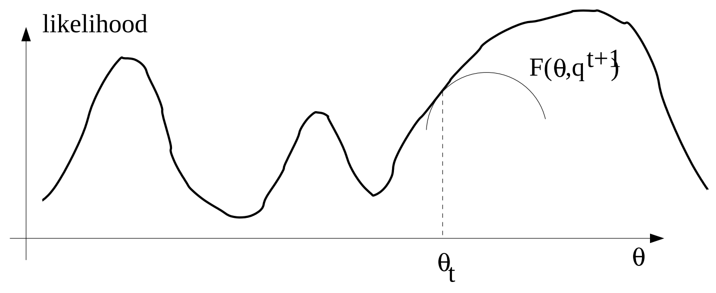
- This is the posterior distribution over the latent variables given the data and the parameters. Often we need this at test time anyway (e.g. to perform classification).
- Proof (easy): this setting saturates the bound $\ell(\theta;\mathbf{x}) \geq F(q,\theta)$

$$\begin{aligned}
F(p(\mathbf{z}|\mathbf{x},\theta^t),\theta^t) &= \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x},\theta^t) \log \frac{p(\mathbf{x},\mathbf{z}|\theta^t)}{p(\mathbf{z}|\mathbf{x},\theta^t)}\\
&= \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x},\theta^t) \log p(\mathbf{x}|\theta^t)\\
&= \log p(\mathbf{x}|\theta^t) \textstyle\sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{x},\theta^t)\\
&= \ell(\theta;\mathbf{x}) \cdot 1
\end{aligned}$$

- Can also show this result using variational calculus or the fact that $\ell(\theta) - F(q,\theta) = \mathrm{KL}[q||p(\mathbf{z}|\mathbf{x},\theta)]$

EM Constructs Sequential Convex Lower Bounds

- Consider the likelihood function and the function $F(q^{t+1}, \cdot)$.



Be careful: 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 $\log \sum_k e^{b_k}$.
- 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 $\log p(x|z = 1) = -420$ and $\log p(x|z = 2) = -420$, what is $\log p(x) = \log [p(x|z = 1) + p(x|z = 2)]$?
  Answer: $\log[2e^{-420}] = -420 + \log 2$.

Partially Hidden Data

- Of course, we can learn when there are missing (hidden) variables on some cases and not on others.
- In this case the cost function was:

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

- Now you can think of this in a new way: in the E-step we estimate the hidden variables on the incomplete cases only.
- The M-step optimizes the log likelihood on the complete data plus the expected likelihood on the incomplete data using the E-step.

Example: Mixtures of Gaussians

- Recall: a mixture of K Gaussians:
  $p(\mathbf{x}|\theta) = \sum_k \alpha_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)$
  $\ell(\theta; \mathcal{D}) = \sum_n \log \sum_k \alpha_k \mathcal{N}(\mathbf{x}^n|\mu_k, \Sigma_k)$
- Learning with EM algorithm:

$$\mathbf{E-step}: \quad p^t_{kn} = \mathcal{N}(\mathbf{x}^n|\mu^t_k, \Sigma^t_k)$$

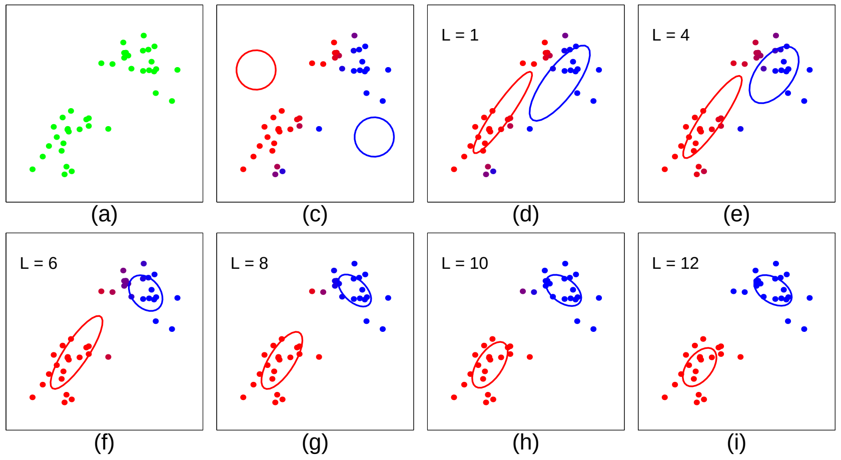
$$q^{t+1}_{kn} = p(z{=}k|\mathbf{x}^n, \theta^t) = \frac{\alpha^t_k p^t_{kn}}{\sum_j \alpha^t_j p^t_{kn}}$$

$$\mathbf{M-step}: \quad \mu^{t+1}_k = \frac{\sum_n q^{t+1}_{kn}\mathbf{x}^n}{\sum_n q^{t+1}_{kn}}$$

$$\Sigma^{t+1}_k = \frac{\sum_n q^{t+1}_{kn}(\mathbf{x}^n - \mu^{t+1}_k)(\mathbf{x}^n - \mu^{t+1}_k)^\top}{\sum_n q^{t+1}_{kn}}$$

$$\alpha^{t+1}_k = \frac{1}{M}\sum_n q^{t+1}_{kn}$$

EM for MoG



Compare: K-Means

- The EM algorithm for mixtures of Gaussians is just like a soft version of the K-means algorithm with fixed priors and covariance.
- Instead of "hard assignment" in the E-step, we do "soft assignment" based on the softmax of the squared distance from each point to each cluster.
- Each centre is then moved to the *weighted mean* of the data, with weights given by soft assignments. In K-means, the weights are 0 or 1.

$$\begin{aligned}
\mathbf{E-step}: \quad & d_{kn}^t = \frac{1}{2}(\mathbf{x}^n - \mu_k^t)^\top \Sigma^{-1}(\mathbf{x}^n - \mu_k^t) \\
& q_{kn}^{t+1} = \frac{\exp(-d_{kn}^t)}{\sum_j \exp(-d_{jn}^t)} = p(c_n^t = k | \mathbf{x}^n, \mu^t) \\
\mathbf{M-step}: \quad & \mu_k^{t+1} = \frac{\sum_n q_{kn}^{t+1}\mathbf{x}^n}{\sum_n q_{kn}^{t+1}}
\end{aligned}$$

Derivation of M-step

- Expected complete log likelihood $\ell_q(\theta; \mathcal{D})$:

$$\sum_n \sum_k q_{kn} \left[\log \alpha_k - \frac{1}{2}(\mathbf{x}^n - \mu_k^{t+1})^\top \Sigma_k^{-1}(\mathbf{x}^n - \mu_k^{t+1}) - \frac{1}{2}\log|2\pi\Sigma_k| \right]$$

- For fixed q we can optimize the parameters:

$$\begin{aligned}
\frac{\partial \ell_q}{\partial \mu_k} &= \Sigma_k^{-1} \sum_n q_{kn}(\mathbf{x}^n - \mu_k) \\
\frac{\partial \ell_q}{\partial \Sigma_k^{-1}} &= \frac{1}{2}\sum_n q_{kn}\left[\Sigma_k^\top - (\mathbf{x}^n - \mu_k^{t+1})(\mathbf{x}^n - \mu_k^{t+1})^\top\right] \\
\frac{\partial \ell_q}{\partial \alpha_k} &= \frac{1}{\alpha_k}\sum_n q_{kn} - \lambda \qquad (\lambda = M)
\end{aligned}$$

- Fact: $\frac{\partial \log|A^{-1}|}{\partial A^{-1}} = A^\top$ and $\frac{\partial \mathbf{x}^\top A \mathbf{x}}{\partial A} = \mathbf{x}\mathbf{x}^\top$

Recap: EM Algorithm

- A way of maximizing likelihood function for latent variable models. Finds ML parameters when the original (hard) problem can be broken up into two (easy) pieces:
  1. Estimate some "missing" or "unobserved" data from observed data and current parameters.
  2. Using this "complete" data, find the maximum likelihood parameter estimates.
- Alternate between filling in the latent variables using our best guess (posterior) and updating the paramters based on this guess:
  E-step: $q^{t+1} = p(\mathbf{z}|\mathbf{x}, \theta^t)$
  M-step: $\theta^{t+1} = \operatorname{argmax}_\theta \sum_\mathbf{z} q(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{z}|\theta)$
- In the M-step we optimize a lower bound on the likelihood. In the E-step we close the gap, making bound=likelihood.

A Report Card for EM

- Some good things about EM:
  - no learning rate parameter
  - very fast for low dimensions
  - each iteration guaranteed to improve likelihood
  - adapts unused units rapidly
- Some bad things about EM:
  - can get stuck in local minima
  - both steps require considering *all* explanations of the data which is an exponential amount of work in the dimension of θ
- EM is typically used with mixture models, for example mixtures of Gaussians or mixtures of experts. The "missing" data are the labels showing which sub-model generated each datapoint.
  Very common: also used to train HMMs, Boltzmann machines, ...

Variants

- Sparse EM:
  Do not recompute exactly the posterior probability on each data point under all models, because it is almost zero.
  Instead keep an "active list" which you update every once in a while.
- Generalized (Incomplete) EM: It might be hard to find the ML parameters in the M-step, even given the completed data. We can still make progress by doing an M-step that improves the likelihood a bit (e.g. gradient step).