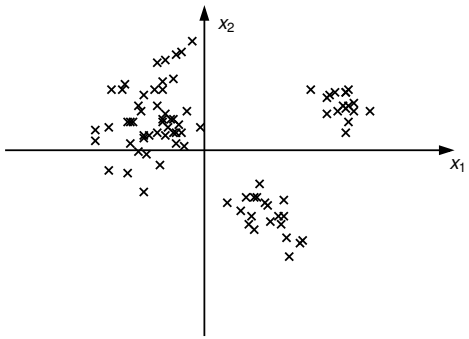
Machine learning for vision

Fall 2013

Roland Memisevic

Lecture 9, October 29, 2013

Classic K-means clustering



- K-means is traditionally a clustering algorithm.
- **Learning:** Fit K prototypes $\boldsymbol{\mu}_k$ (the rows of some matrix, W) to training data-points $\boldsymbol{x}_n$.
- **Inference:** Given a point, find the nearest prototype.

Classic K-means clustering

- Define $\boldsymbol{s}_n$ as the one-hot encoding of the discrete variable representing the index of the nearest cluster center for $\boldsymbol{x}_n$.
- It is also useful to think of a matrix S with entries s_{nk}, holding the one-hot vectors in its rows.
- Assume we *knew* the cluster assignments $\boldsymbol{s}_n$ for each point $\boldsymbol{x}_n$.
- The K-means objective function measures the *average distance between points* $\boldsymbol{x}$ *and their representatives*:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} s_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$$

Classic K-means clustering

- *Learning* amounts to finding *both* the prototypes $\boldsymbol{\mu}_k$ *and* the assignments $\boldsymbol{s}_n$ for each point, so as to minimize J.
- This seems like a tricky optimization problem, because the $\boldsymbol{s}_n$ are discrete and the $\boldsymbol{\mu}_k$ are continuous.
- But learning gets easy if we decouple learning the $\boldsymbol{s}_n$ from learning the $\boldsymbol{\mu}_k$.
- This gives rise to a *block coordinate-descent method*, which is a special case of the *EM-algorithm* for training mixtures of Gaussians.

Classic K-means clustering

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} s_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$$

Finding the optimal $\boldsymbol{s}_n$

- Given the $\boldsymbol{\mu}_k$, we can optimize all the $\boldsymbol{s}_n$ independently, because the objective is just the sum over n.
- But the squared error will be smallest if we set $s_{nk} = 1$ for whichever $\boldsymbol{\mu}_k$ is *closest*.
- Formally, to optimize all $\boldsymbol{s}_n$, given the set of $\boldsymbol{\mu}_k$, set:

$$s_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\boldsymbol{x}_n - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise.} \end{cases}$$

Classic K-means clustering

Finding the optimal $\boldsymbol{\mu}_k$

- Given S, J is a quadratic function of $\boldsymbol{\mu}_k$ which we can minimize by setting the derivative to zero:

$$2 \sum_{n=1}^{N} s_{nk}(\boldsymbol{x}_n - \boldsymbol{\mu}_k) = 0$$

- Solving for $\boldsymbol{\mu}_k$ yields:

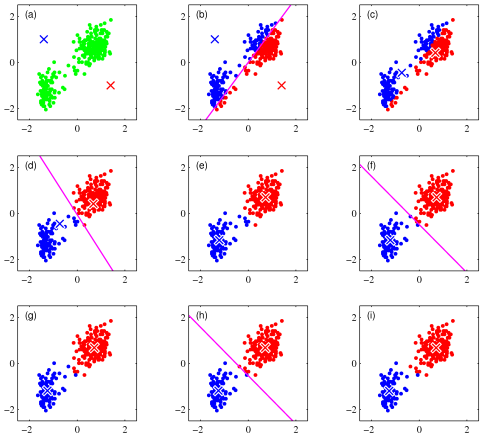
$$\boldsymbol{\mu}_k = \frac{\sum_n s_{nk} \boldsymbol{x}_n}{\sum_n s_{nk}}$$

- This solution has a simple interpretation: Set $\boldsymbol{\mu}_k$ to the mean of all points currently assigned to cluster k.

Classic K-means clustering

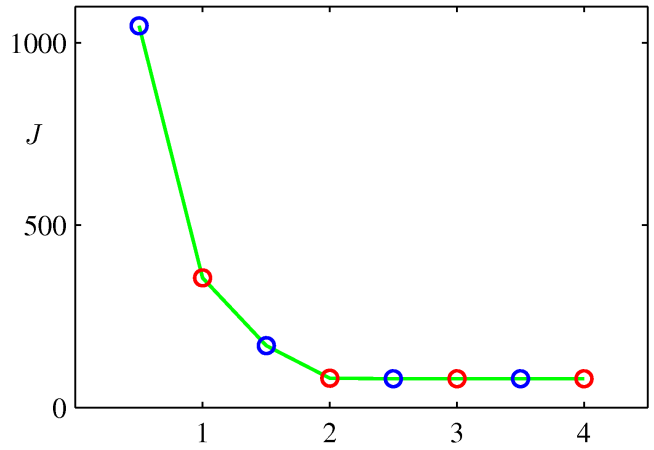
- Learning amounts to iterating inference for the $\boldsymbol{s}_n$, and adapting the parameters $\boldsymbol{\mu}_k$ until there are no more changes.
- This training procedure always converges: J is positive, and every step either decreases it or leaves it unchanged.
- But there can be local minima.
- One way to deal with this is to try multiple runs with different initializations for the parameters $\boldsymbol{\mu}_k$ and to pick the solution with the lowest final cost.

Classic K-means example ($K = 2$)



this and most of the following images from: (Bishop, 2006)

The value of J as learning progresses



K-means inference

- Given the trained model, we can infer the cluster-center for a new test-data point $\boldsymbol{x}$ not seen during training, by finding the nearest $\boldsymbol{\mu}_k$ like during training:

$$s_k(\boldsymbol{x}) = \begin{cases} 1 & \text{if } k = \arg\min_j \|\boldsymbol{x}_n - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise.} \end{cases}$$

- The set of all K prototypes $\boldsymbol{\mu}_k$ is called *codebook*.
- Clustering and K-means are also known as *vector quantization*.

K-means

The probably most important property of K-means is that it distributes cluster-centers in space, such that the cluster center density is roughly proportional to the data density.

So it will resolve high-density regions well, at the cost of low-density regions.

K-means via online learning

- The reconstruction error for training point $\boldsymbol{x}$ may be written

$$E(W) = \frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{w}_{s(\boldsymbol{x})}\right)^2$$

- Its gradient is

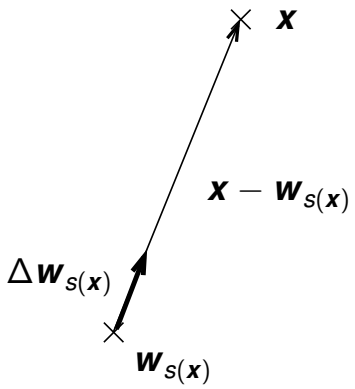
$$\frac{\partial E(W)}{\partial \boldsymbol{w}_i} = -\left(\boldsymbol{x} - \boldsymbol{w}_{s(\boldsymbol{x})}\right)\delta_{s(\boldsymbol{x}),i}$$

- So we can use the online learning rule:

$$\boldsymbol{w}_{s(\boldsymbol{x})} \leftarrow \boldsymbol{w}_{s(\boldsymbol{x})} + \eta\left(\boldsymbol{x} - \boldsymbol{w}_{s(\boldsymbol{x})}\right)$$

- (Here, it is easier to think of $s(x)$ as index rather than one-hot vector.)

Geometry of online K-means



- $\Delta \boldsymbol{w}_{s(\boldsymbol{x})} = \eta(\boldsymbol{x} - \boldsymbol{w}_{s(\boldsymbol{x})})$ moves the winning weight vector towards the observation.

Online K-means and Hebbian learning

- We can interpret the online k-means updates as:

  Hebb-rule + competition + unlearning

- To this end write the update as

$$\Delta \boldsymbol{w}_k = \eta \delta_{ks(\boldsymbol{x})} (\boldsymbol{x} - \boldsymbol{w}_k)$$

  where

$$\delta_{ks(\boldsymbol{x})} = \begin{cases} 1 & \text{if } s(\boldsymbol{x}) = k \\ 0 & \text{else} \end{cases}$$

  is the "post-synaptic activity" determined by competition ("winner takes all" rule)

- There are two learning terms:

K-means and Hebbian learning

1. A Hebbian term:

$$\delta_{ks(\boldsymbol{x})} \boldsymbol{x}$$

2. An "unlearning" term:

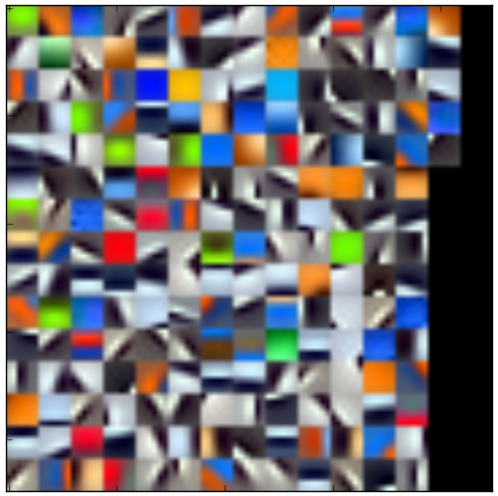
$$-\delta_{ks(\boldsymbol{x})} \boldsymbol{w}_k$$

- The positive term decreases the energy near the data.
- The unlearning term increases the energy everywhere.
- "Hebb-rule + competition + unlearning" are present (not surprisingly) in a wide variety of learning algorithms, including contrastive divergence learning for RBMs.

Hebbian K-means in 9 lines of code

```
import numpy
def kmeans(W, X, numepochs, learningrate=0.01, batchsize=100):
    X2 = (X**2).sum(1)[:, None]
    for epoch in range(numepochs):
        for i in range(0, X.shape[0], batchsize):
            D = -2*numpy.dot(W, X[i:i+batchsize,:].T) + (W**2).sum(1)[:, None] + X2[i:i+batchsize].T
            S = (D==D.min(0)[None,:]).astype("float").T
            W += learningrate * (numpy.dot(S.T, X[i:i+batchsize,:]) - S.sum(0)[:, None] * W)
    return W
```

K-means features learned from natural image patches



Self-organizing maps

- We obtain the *self-organizing map* (SOM) aka *Kohonen network* by changing the k-means update from

$$\Delta \boldsymbol{w}_k = \eta \delta_{ks(\boldsymbol{x})} \left(\boldsymbol{x} - \boldsymbol{w}_k \right)$$

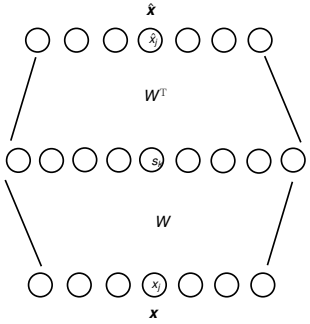
  into

$$\Delta \boldsymbol{w}_k = \eta h_{ks(\boldsymbol{x})} \left(\boldsymbol{x} - \boldsymbol{w}_k \right)$$

  where h_{kj} is some smooth neighborhood function, that will let hiddens near the winning hidden learn, too.
- This requires hiddens to be arranged in space in some way (commonly 2-D).

K-means as autoencoder

$$\begin{aligned} \boldsymbol{s} &= \mathrm{wta}(W^{\mathrm{T}} \boldsymbol{x}) \\ \boldsymbol{x} &= W \boldsymbol{s} \end{aligned}$$



- $E = \sum_i \| \boldsymbol{x}_i - W \mathrm{wta} \left(W^{\mathrm{T}} \boldsymbol{x}_i \right) \|^2$
- wta = "winner takes all"
- Weights are "tied": Recognition weights are the transpose of generative weights.

The *K*-means energy function

- We can think of *K*-means as an instance of energy based learning, by defining the energy function

$$E(\boldsymbol{x}) = \| \boldsymbol{x} - W \mathrm{wta} \left(W^{\mathrm{T}} \boldsymbol{x} \right) \|^2 = \| \boldsymbol{x} - \boldsymbol{w}_{s(\boldsymbol{x})} \|^2$$

- Since far from the cluster-centers the energy goes to ∞, *K*-means has then low energy everywhere.
- In other words, *K*-means simply doesn't have the *capacity* to produce an arbitrary energy surface with low energy far away from data.

The K-means energy function

- If we define the unnormalized probability for a point as

$$q(\boldsymbol{x}_n) = \exp(-E(\boldsymbol{x}_n))$$

  we obtain a density model which is just the superposition of K bumps.
- So the probability goes to zero far from any cluster center.
- Unlike RBMs and many other probabilistic models, there is no need to lower density away from the data in this model. Instead, the density will be low "*by design*".
- (LeCun, 2006)

Winner-takes-all and lateral interactions

- The winner-takes-all function may be defined as

$$\mathrm{wta}(\boldsymbol{x}) = \mathrm{onehot}\big(\arg\min_k \|\boldsymbol{x} - \boldsymbol{w}_k\|^2\big)$$

  where $\boldsymbol{w}_k$ is a column of W.
- The squared distance can be written as

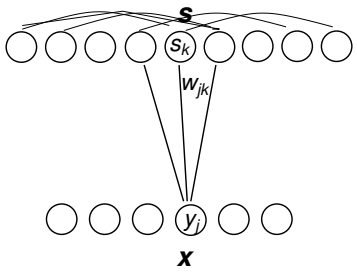
$$\|\boldsymbol{x} - \boldsymbol{w}_k\|^2 = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{w}_k - 2\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}$$

- If all $\boldsymbol{w}_k$ have the same norm, inference amounts to finding the hidden which maximizes

$$\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}$$

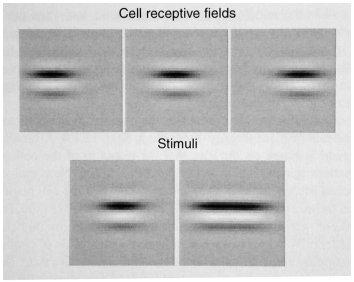
  which is the usual "simple cell" response.

Lateral interactions



- Since wta is a function of all the hiddens, inference requires the hiddens to talk to each other.
- This is commonly referred to as *lateral interactions*.
- Typically, the interactions take the form of *lateral inhibition*.

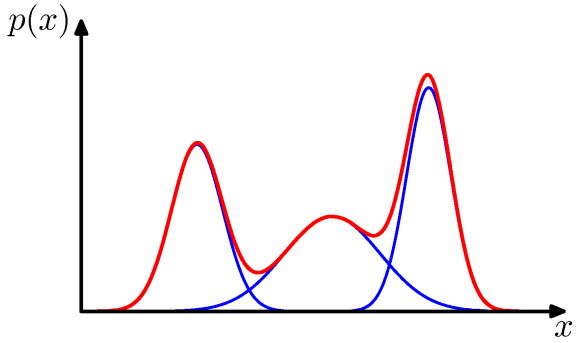
End-stopping



from: Natural Image Statistics (Hyvarinen, Hurri, Hoyer; 2009)

- For the simple cell in the top middle, a linear model would predict the bottom right stimulus to give at least as large a response as the bottom left stimulus.
- But for actual neural responses it may give a weaker response.
- This effect is known as *end-stopping*, and it may be counted as evidence for lateral inhibition.

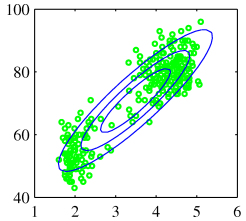
Gaussian mixture models



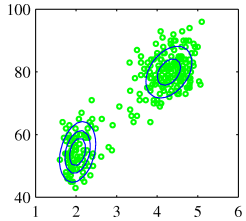
- If we add prior probabilities π_k for each cluster and interpret each cluster as a Gaussian with its own variance, we get the *Mixture of Gaussians*:

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

Gaussian mixture models



Gaussian fit to some data.



Gaussian mixture fit to the data.

Gaussian mixture models

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

- For $p(\boldsymbol{x})$ to be a probability distribution, we need $\sum_k \pi_k = 1$ and $\pi_k > 0 \quad \forall k$
- Thus, we may interpret the π_k as probabilities themselves.
- This motivates introducing latent variables $\boldsymbol{s}$ and re-writing the model, equivalently, in terms of two distributions $p(\boldsymbol{s})$ and $p(\boldsymbol{s}|\boldsymbol{x})$:

Gaussian mixture models

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{s}} p(\boldsymbol{s}) p(\boldsymbol{x}|\boldsymbol{s})$$

where

$$p(\boldsymbol{s}) = \prod_{k=1}^{K} \pi_k^{s_k}$$

is a discrete distribution ($\boldsymbol{s}$ is in one-hot encoding), and

$$p(\boldsymbol{x}|s_k = 1) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k)$$

is a conditional Gaussian distribution.

Gaussian mixture models

- The model is a generative model, where we first draw a mixture component from a discrete distribution, and then we draw the observation from a Gaussian, whose parameters depend on the component.
- To compute how likely a given observation $\boldsymbol{x}_n$ is to come from a particular mixture component, use Bayes' rule:

$$p(\boldsymbol{s}_n|\boldsymbol{x}_n) = \frac{p(\boldsymbol{x}_n|\boldsymbol{s}_n)p(\boldsymbol{s}_n)}{\sum_{\boldsymbol{s}_n} p(\boldsymbol{x}_n|\boldsymbol{s}_n)p(\boldsymbol{s}_n)}$$

- $p(s_{nk} = 1|\boldsymbol{x}_n)$ is called *responsibility* of mixture component k.
- The Gaussian mixture is like a "soft" version of K-means.

The EM algorithm

- Mixture models are usually trained using the EM algorithm (though one could use gradient descent).
- Given training data $\{\boldsymbol{x}_n\}$, we can write

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

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

The EM algorithm

- Instead of optimizing L, we will now optimize the lower bound $\mathcal{L}$ with respect to *both* the original parameters and the newly introduced auxiliary variables $q(\boldsymbol{s})$.
- To avoid clutter, it is convenient write this as

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

using the abbreviation $q_{nk} = q(\boldsymbol{s}_n = k)$

- The first term of $\mathcal{L}$ is the *expectation* of $\log p(\boldsymbol{x}_n, \boldsymbol{s}_n)$ with respect to $q(\boldsymbol{s}_n)$. It is commonly referred to as "expected complete log-likelihood".
- This is the only term that depends on the model parameters.
- For learning, set derivatives to zero:

The EM algorithm

- We have

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

$$\Leftrightarrow \boldsymbol{\mu}_k = \frac{\sum_n q_{nk} \boldsymbol{x}_n}{\sum_n q_{nk}}$$

- Similarly, one can derive

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

and

$$\pi_k = p(\boldsymbol{s}_k) = \frac{\sum_n q_{nk}}{N}$$

The EM algorithm

- But what about the auxiliary variables q_{nk}?
- We rewrite $\mathcal{L}$ once more in a different way:

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

The EM algorithm

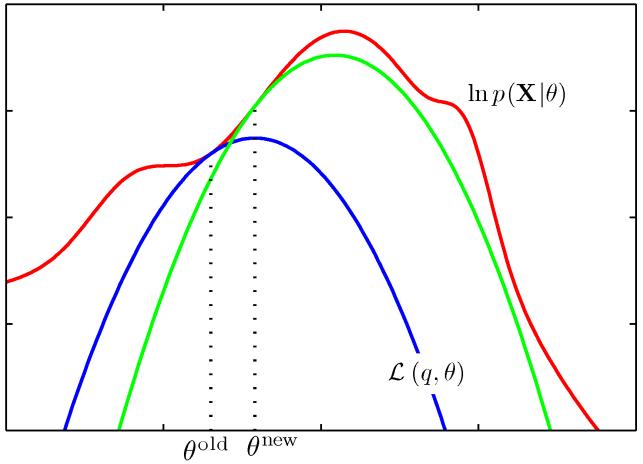
- Since the KL divergence is non-negative, setting $q(\boldsymbol{s}_n) = p(\boldsymbol{s}_n|\boldsymbol{x}_n)$ will make the bound $\mathcal{L}$ on L tight!
- But $p(\boldsymbol{s}_n|\boldsymbol{x}_n)$ is easy to compute, using Bayes' rule.
- We already know how to optimize $\mathcal{L}$ with respect to the model parameters. So we can repeatedly compute (by inferring q_{nk}), and then optimize, a tight lower bound on L.

EM algorithm summary

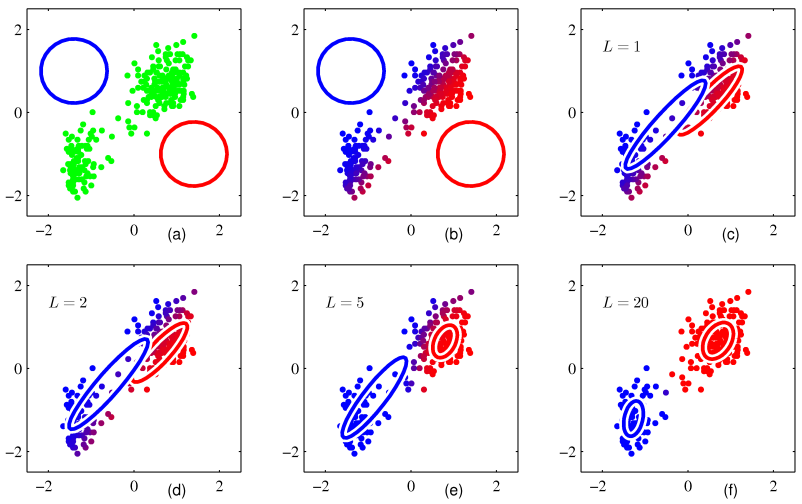
1. E-step: Evaluate the posteriors $p(\boldsymbol{s}_n|\boldsymbol{x}_n)$.
2. M-step: Optimize $\mathcal{L}$ with respect to the model parameters, keeping $q(\boldsymbol{s}_n) = p(\boldsymbol{s}_n|\boldsymbol{x}_n)$ fixed.

- The E-step computes the expected complete log-likelihood. It amounts to evaluating the responsibilities $p(\boldsymbol{s}_n|\boldsymbol{x}_n)$ for each point.
- The M-step maximizes the expected complete log-likelihood. In a Gaussian mixture, this amounts to setting parameters to responsibility-weighted sums.

EM optimizes a sequence of lower bounds



Example: Training a Gaussian mixture with EM



EM algorithm comments

- The EM algorithm can be applied to many latent variable models, not just mixtures of Gaussians.
- The E-step and the M-step have to be derived individually for each model, but the view from the lower bound $\mathcal{L}$ of the log-likelihood is always the same.
- One of the first models that deployed EM was the Hidden Markov Model.
- For many other models, computing $p(\boldsymbol{s}|\boldsymbol{x})$ is not tractable. In this case, it is still common to deploy a variation of EM, where we only improve the KL-divergence in the E-step rather than finding the exact posterior.