CIAR Summer School Tutorial

Lecture 1a: Mixtures of Gaussians, EM, and Variational Free Energy

Geoffrey Hinton

Two types of density model
(with hidden configurations h)

Stochastic generative model using directed acyclic graph (e.g. Bayes Net)

$$p(x) = \sum_h p(h)\ p(x \mid h)$$

Generation from model is easy
Inference can be hard
Learning is easy after inference

Energy-based models that associate an energy with each data vector + hidden configuration

$$p(x) = \frac{\sum_h e^{-E(x,h)}}{\sum_{x',h} e^{-E(x',h)}}$$

Generation from model is hard
Inference can be easy
Is learning hard?

Clustering

- We assume that the data was generated from a number of different classes. The aim is to cluster data from the same class together.
  - How do we decide the number of classes?
    - Why not put each datapoint into a separate class?
    - What is the payoff for clustering things together?
  - Clustering is not a very powerful way to model data, especially if each data-vector can be classified in many different ways? A one-out-of-N classification is not nearly as informative as a feature vector.
    - We will see how to learn feature vectors later.

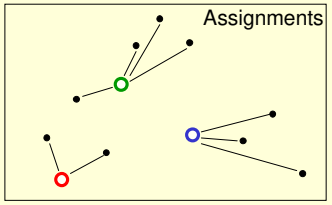
The k-means algorithm

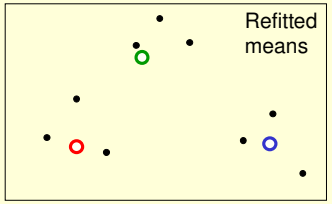
- Assume the data lives in a Euclidean space.
- Assume we want k classes.
- Assume we start with randomly located cluster centers

The algorithm alternates between two steps:

Assignment step: Assign each datapoint to the closest cluster.

Refitting step: Move each cluster center to the center of gravity of the data assigned to it.





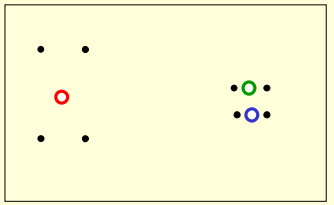
Why K-means converges

- Whenever an assignment is changed, the sum squared distances of data-points from their assigned cluster centers is reduced.
- Whenever a cluster center is moved the sum squared distances of the data-points from their currently assigned cluster centers is reduced.
- If the assignments do not change in the assignment step, we have converged.

Local minima

- There is nothing to prevent k-means getting stuck at local minima.
- We could try many random starting points
- We could try non-local split-and-merge moves: Simultaneously merge two nearby clusters and split a big cluster into two.

A bad local optimum



Soft k-means

- Instead of making hard assignments of data-points to clusters, we can make soft assignments. One cluster may have a responsibility of .7 for a data-point and another may have a responsibility of .3.
  - Allows a cluster to use more information about the data in the refitting step.
  - What happens to our convergence guarantee?
  - How do we decide on the soft assignments?
    - Maybe we can add a term that rewards softness to our sum squared distance cost function.

Rewarding softness

- If a datapoint is exactly halfway between two clusters, each cluster should obviously have the same responsibility for it.
- The responsibilities of all the clusters for one datapoint should add to 1.
- A sensible softness function is the entropy of the responsibilities.
  - Maximizing the entropy is like saying: be as uncertain as you can about which cluster has responsibility
  - We want high entropy responsibilities, but we also want to focus the responsibility for a data-point on the nearest cluster centers.

Number of clusters, k

Responsibility of cluster i for datapoint j

$$H_j = \sum_{i=1}^{i=k} p_{ij} \log \frac{1}{p_{ij}}$$

Entropy of the responsibilities for datapoint j

The soft assignment step

Choose assignments to optimize the trade-off between two terms:

- minimize the squared distance of the datapoint to the cluster centers (weighted by responsibility)
- Maximize the entropy of the responsibilities

Cost of the assignments for datapoint j

Location of datapoint j

Location of cluster i

$$Cost_j = \sum_{i=1}^{i=k} p_{ij} \| \mathbf{x}_j - \boldsymbol{\mu}_i \|^2$$

$$- \sum_{i=1}^{i=k} p_{ij} \log \frac{1}{p_{ij}}$$

Responsibility of cluster i for datapoint j

The re-fitting step

- Weight each datapoint by the responsibility that the cluster has for it.
- Move the mean of the cluster to the center of gravity of the responsibility -weighted data.
- Notice that this is not a gradient step: There is no learning rate!

$$\boldsymbol{\mu}_i = \frac{\sum_{j=1}^{j=N} p_{ij} \mathbf{x}_j}{\sum_{j=1}^{j=N} p_{ij}}$$

Index over Gaussians

Index over datapoints

- How do we find the set of responsibility values that minimizes the cost and sums to 1?

$$Cost_j = \sum_{i=1}^{i=k} p_{ij} \| \mathbf{x}_j - \boldsymbol{\mu}_i \|^2$$

$$- \sum_{i=1}^{i=k} p_{ij} \log \frac{1}{p_{ij}}$$

- The optimal solution is to make the responsibilities be proportional to the exponentiated squared distances:

$$p_{ij} = \frac{e^{-\|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2}}{\sum_m e^{-\|\mathbf{x}_j - \boldsymbol{\mu}_m\|^2}}$$

Some difficulties with soft k-means

- If we measure distances in centimeters instead of inches we get different soft assignments.
  - It would be much better to have a method that is invariant under linear transformations of the data space (scaling, rotating ,elongating)
- Clusters are not always round.
  - It would be good to allow different shapes for different clusters.
- Sometimes its better to cluster by using low-density regions to define the boundaries between clusters rather than using high-density regions to define the centers of clusters.

A generative view of clustering

- We need a sensible measure of what it means to cluster the data well.
  - This makes it possible to judge different methods.
  - It may make it possible to decide on the number of clusters.
- An obvious approach is to imagine that the data was produced by a generative model.
  - Then we can adjust the parameters of the model to maximize the probability density that it would produce exactly the data we observed.

The mixture of Gaussians generative model

- First pick one of the k Gaussians with a probability that is called its "mixing proportion".
- Then generate a random point from the chosen Gaussian.
- The probability of generating the exact data we observed is zero, but we can still try to maximize the probability density.
  - Adjust the means of the Gaussians
  - Adjust the variances of the Gaussians on each dimension (or use a full covariance Gaussian).
  - Adjust the mixing proportions of the Gaussians.

Computing responsibilities

- In order to adjust the parameters, we must first solve the inference problem: Which Gaussian generated each datapoint, x?
  - We cannot be sure, so it's a distribution over all possibilities.
- Use Bayes theorem to get posterior probabilities

Posterior for Gaussian i

Prior for Gaussian i

$$p(i \mid \mathbf{x}) = \frac{p(i)p(\mathbf{x} \mid i)}{p(\mathbf{x})}$$

$$p(\mathbf{x}) = \sum_j p(j)p(\mathbf{x} \mid j)$$

$$p(i) = \pi_i$$ ← Mixing proportion

$$p(\mathbf{x} \mid i) = \prod_{d=1}^{d=k} \frac{1}{\sqrt{2\pi}\sigma_{i,d}} e^{-\frac{\| x_d - \mu_{i,d} \|^2}{2\sigma_{i,d}^2}}$$

↑ Product over all data dimensions

Computing the new mixing proportions

- Each Gaussian gets a certain amount of posterior probability for each datapoint.
- The optimal mixing proportion to use (given these posterior probabilities) is just the fraction of the data that the Gaussian gets responsibility for.

Posterior for Gaussian i

Data for training case c

$$\pi_i^{new} = \frac{\sum_{c=1}^{c=N} p(i \mid \mathbf{x}^c)}{N}$$

↑ Number of training cases

Computing the new means

- We just take the center-of gravity of the data that the Gaussian is responsible for.
  - Just like in K-means, except the data is weighted by the posterior probability of the Gaussian.
  - Guaranteed to lie in the convex hull of the data
    - Could be big initial jump

$$\mu_i^{new} = \frac{\sum_c p(i \mid \mathbf{x}_c)\, \mathbf{x}_c}{\sum_c p(i \mid \mathbf{x}_c)}$$

Computing the new variances

- For axis-aligned Gaussians, we just fit the variance of the Gaussian on each dimension to the posterior-weighted data
  - Its more complicated if we use a full-covariance Gaussian that is not aligned with the axes.

$$\sigma_{i,d}^2 = \frac{\sum_c p(i \mid \mathbf{x}^c)\ \| x_d^c - \mu_{i,d}^{new} \|^2}{\sum_c p(i \mid \mathbf{x}^c)}$$

How many Gaussians do we use?

- Hold back a validation set.
  - Try various numbers of Gaussians
  - Pick the number that gives the highest density to the validation set.
- Refinements:
  - We could make the validation set smaller by using several different validation sets and averaging the performance.
  - We should use all of the data for a final training of the parameters once we have decided on the best number of Gaussians.

Avoiding local optima

- EM can easily get stuck in local optima.
- It helps to start with very large Gaussians that are all very similar and to only reduce the variance gradually.
  - As the variance is reduced, the Gaussians spread out along the first principal component of the data.

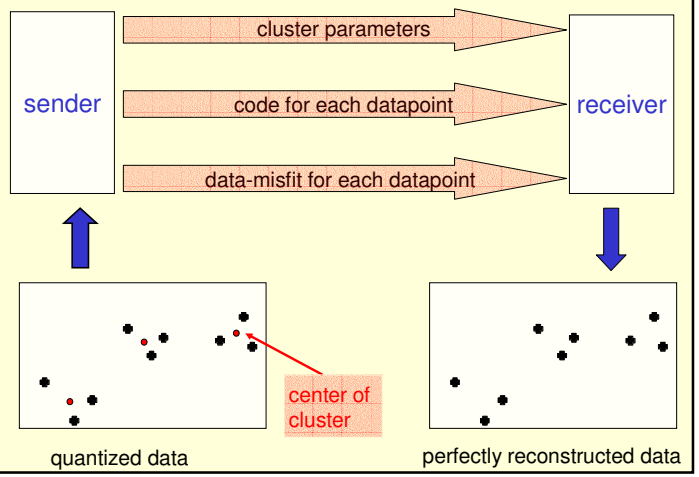
Speeding up the fitting

- Fitting a mixture of Gaussians is one of the main occupations of an intellectually shallow field called data-mining.
- If we have huge amounts of data, speed is very important. Some tricks are:
  - Initialize the Gaussians using k-means
    - Makes it easy to get trapped.
    - Initialize K-means using a subset of the datapoints so that the means lie on the low-dimensional manifold.
  - Find the Gaussians near a datapoint more efficiently.
    - Use a KD-tree to quickly eliminate distant Gaussians from consideration.
  - Fit Gaussians greedily
    - Steal some mixing proportion from the already fitted Gaussians and use it to fit poorly modeled datapoints better.

Proving that EM improves the log probability of the training data

- There are many ways to prove that EM improves the model.
- We will prove it by showing that there is a single function that is improved by both the E-step and the M-step.
  - This leads to efficient "variational" methods for fitting models that are too complicated to allow an exact E-step.
  - Brendan Frey will show how variational model-fitting can be used for some tough vision problems.

An MDL approach to clustering



How many bits must we send?

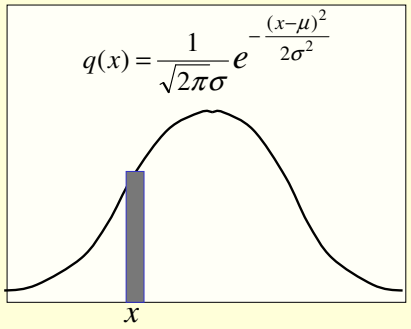
- Model parameters:
  - It depends on the priors and how accurately they are sent.
  - Lets ignore these details for now
- Codes:
  - If all n clusters are equiprobable, log n
    - This is extremely plausible, but wrong!
  - We can do it in less bits
    - This is extremely implausible but right.
- Data misfits:
  - If sender & receiver assume a Gaussian distribution within the cluster, -log[p(d)|cluster] which depends on the squared distance of d from the cluster center.

Using a Gaussian agreed distribution

- Assume we need to send a value, x, with a quantization width of t
- This requires a number of bits that depends on

$$\frac{(x-\mu)^2}{2\sigma^2}$$

$$q(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



$$-\log(prob.\,mass) \approx -\log(t\ q(x))$$

$$\approx -\log(t) + \log(\sqrt{2\pi}\sigma) + \frac{(x-\mu)^2}{2\sigma^2}$$

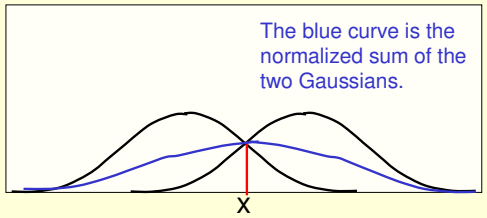
What is the best variance to use?

$$C \approx \sum_{c=1}^{N} -\log(t) + \log(\sqrt{2\pi}\sigma) + \frac{(x_c-\mu)^2}{2\sigma^2}$$

$$\frac{\partial C}{\partial \sigma} = \frac{N}{\sigma} - \frac{1}{\sigma^3}\sum_c (x_c-\mu)^2$$

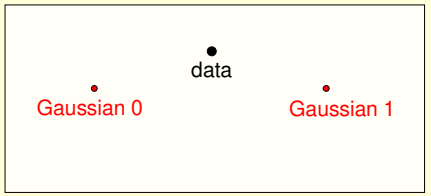
- It is obvious that this is minimized by setting the variance of the Gaussian to be the variance of the residuals.

Sending a value assuming a mixture of two equal Gaussians



- The point halfway between the two Gaussians should cost –log(p(x)) bits where p(x) is its density under one of the Gaussians.
  - But in the MDL story the cost should be –log(p(x)) plus one bit to say which Gaussian we are using.
  - How can we make the MDL story give the right answer?

The bits-back argument

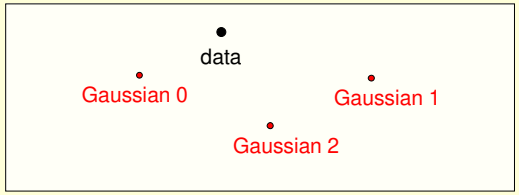


- Consider a datapoint that is equidistant from two cluster centers.
  - The sender could code it relative to cluster 0 or relative to cluster 1.
  - Either way, the sender has to send one bit to say which cluster is being used.
    - It seems like a waste to have to send a bit when you don't care which cluster you use.
    - It must be inefficient to have two different ways of encoding the same point.

Using another message to make random decisions

- Suppose the sender is also trying to communicate another message
  - The other message is completely independent.
  - It looks like a random bit stream.
- Whenever the sender has to choose between two equally good ways of encoding the data, he uses a bit from the other message to make the decision
- After the receiver has losslessly reconstructed the original data, the receiver can pretend to be the sender.
  - This enables the receiver to figure out the random bit in the other message.
- So the original message cost one bit less than we thought because we also communicated a bit from another message.

The general case



$$Expected\ Cost = \sum_i p_i E_i - \sum_i p_i \log \frac{1}{p_i}$$

Probability of picking cluster i

Bits required to send cluster identity plus data relative to cluster center

Random bits required to pick which cluster

Free Energy

$$Free\ Energy = \sum_i p_i E_i - T \sum_i p_i \log \frac{1}{p_i}$$

Probability of finding system in configuration i

Energy of configuration i

Temperature

Entropy of distribution over configurations

The equilibrium free energy of a set of configurations is the energy that a single configuration would have to have to have as much probability as that entire set.

$$e^{-\frac{F}{T}} = \sum_i e^{-\frac{E_i}{T}}$$

A Canadian example

$$F_{ice} = \langle E_{ice} \rangle - T\ H_{ice}$$

- Ice is a more regular and lower energy packing of water molecules than liquid water.
  - Lets assume all ice configurations have the same energy
- But there are vastly more configurations called water.

$$E_{ice} << E_{water}$$

$$H_{ice} << H_{water}$$

$$At\ T = 272,\quad F_{ice} << F_{water}$$

$$At\ T = 274,\quad F_{ice} >> F_{water}$$

What is the best distribution?

- The sender and receiver can use any distribution they like
  - But what distribution minimizes the expected message length
- The minimum occurs when we pick codes using a Boltzmann distribution:

$$p_i = \frac{e^{-E_i}}{\sum_j e^{-E_j}}$$

- This gives the best trade-off between entropy and expected energy.
  - It is how physics behaves when there is a system that has many alternative configurations each of which has a particular energy (at a temperature of 1).

EM as coordinate descent in Free Energy

$$F(d) = \sum_i p_i[-\log \pi_i - \log p(d \mid i)] \quad - \quad \sum_i p_i(-\log p_i)$$

- Think of each different setting of the hidden and visible variables as a "configuration". The energy of the configuration has two terms:
  - The negative log prob of generating the hidden values
  - The negative log prob of generating the visible values from the hidden ones
- The E-step minimizes F by finding the best distribution over hidden configurations for each data point.
- The M-step holds the distribution fixed and minimizes F by changing the parameters that determine the energy of a configuration.

The advantage of using F to understand EM

- There is clearly no need to use the optimal distribution over hidden configurations.
  - We can use any distribution that is convenient so long as:
    - we always update the distribution in a way that improves F
    - We change the parameters to improve F given the current distribution.
- This is very liberating. It allows us to justify all sorts of weird algorithms.

The indecisive means algorithm

Suppose that we want to cluster data in a way that guarantees that we still have a good model even if an adversary removes one of the cluster centers from our model.

- E-step: find the two cluster centers that are closest to each data point. Each of these cluster centers is given a responsibility of 0.5 for that datapoint.
- M-step: Re-estimate each cluster center to be the mean of the datapoints it is responsible for.

- "Proof" that it converges:
  - The E-step optimizes F subject to the constraint that the distribution contains 0.5 in two places.
  - The M-step optimizes F with the distribution fixed

An incremental EM algorithm

- E-step: Look at a single datapoint, d, and compute the posterior distribution for d.
- M-step: Compute the effect on the parameters of changing the posterior for d
  - Subtract the contribution that d was making with its previous posterior and add the effect it makes with the new posterior.

$$\boldsymbol{\mu}_i^{new(d)} = \frac{p^{new}(i \mid \mathbf{x}_d)\,\mathbf{x}_d + \sum_{c \neq d} p^{old}(i \mid \mathbf{x}_c)\,\mathbf{x}_c}{p^{new}(i \mid \mathbf{x}_d) + \sum_{c \neq d} p^{old}(i \mid \mathbf{x}_c)}$$

Stochastic MDL using the wrong distribution over codes

- If we want to communicate the code for a datavector, the most efficient method requires us to pick a code randomly from the posterior distribution over codes.
  - This is easy if there is only a small number of possible codes. It is also easy if the posterior distribution has a nice form (like a Gaussian or a factored distribution)
  - But what should we do if the posterior is intractable?
    - This is typical for non-linear distributed representations.
- We do not have to use the most efficient coding scheme!
  - If we use a suboptimal scheme we will get a bigger description length.
    - The bigger description length is a bound on the minimal description length.
    - Minimizing this bound is a sensible thing to do.
  - So replace the true posterior distribution by a simpler distribution.
    - This is typically a factored distribution.

A spectrum of representations

- PCA is powerful because it uses distributed representations but limited because its representations are linearly related to the data.
- Clustering is powerful because it uses very non-linear representations but limited because its representations are local (not componential).
- We need representations that are both distributed and non-linear
  - Unfortunately, these are typically very hard to learn.

	Local	Distributed
Linear		PCA
non-linear	clustering	What we need