CSC321: Neural Networks

# Lecture 14: Clustering

Geoffrey Hinton

## 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?
  - What if the classes are hierarchical?
  - What if each datavector can be classified in many different ways? A one-out-of-N classification is not nearly as informative as a feature vector.

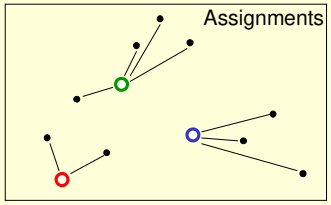
## The k-means algorithm

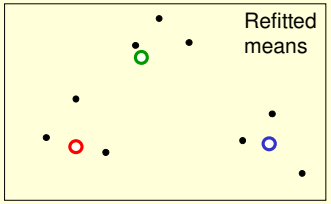
- Assume the data lives in a Euclidean space.
- Assume we want k classes.
- Assume we start with randomly located cluster centers

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.



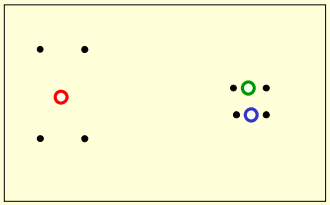

## Why K-means converges

- Whenever an assignment is changed, the sum squared distances of datapoints from their assigned cluster centers is reduced.
- Whenever a cluster center is moved the sum squared distances of the datapoints 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 datapoints to clusters, we can make soft assignments. One cluster may have a responsibility of .7 for a datapoint 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?

## 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.
  - Adjust the mixing proportions of the Gaussians.

## The E-step: Computing responsibilities

- In order to adjust the parameters, we must first solve the inference problem: Which Gaussian generated each datapoint?
  - 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

$$p(i \mid \mathbf{x}^c) = \frac{p(i)p(\mathbf{x}^c \mid i)}{p(\mathbf{x}^c)}$$ ← Bayes theorem

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

$$p(i) = \pi_i$$ ← Mixing proportion

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

↑ Product over all data dimensions

## The M-step: Computing 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

## More M-step: 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

$$\boldsymbol{\mu}_i^{new} = \frac{\sum_c p(i \mid \mathbf{x}^c)\, \mathbf{x}^c}{\sum_c p(i \mid \mathbf{x}^c)}$$

## More M-step: Computing the new variances

- We fit the variance of each Gaussian, i, on each dimension, d, 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.

## The next 5 slides are optional extra material that will not be in the final exam

- There are several different ways to show that EM converges.
- My favorite method is to show that there is a cost function that is reduced by both the E-step and the M-step.
- But the cost function is considerably more complicated than the one for K-Means.

## Why EM converges

- There is a cost function that is reduced by both the E-step and the M-step.

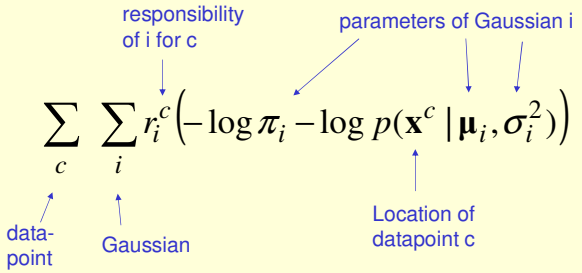
  Cost = expected energy – entropy

- The expected energy term measures how difficult it is to generate each datapoint from the Gaussians it is assigned to. It would be happiest giving all the responsibility for each datapoint to the most likely Gaussian (as in K-means).

- The entropy term encourages "soft" assignments. It would be happiest spreading the responsibility for each datapoint equally between all the Gaussians.

## The expected energy of a datapoint

- The expected energy of datapoint c is the average negative log probability of generating the datapoint
  - The average is taken using the responsibility that each Gaussian is assigned for that datapoint:




## The entropy term

- This term wants the responsibilities to be as uniform as possible.
- It fights the expected energy term.

$$entropy = -\sum_c \sum_i r_i^c \log r_i^c$$

log probabilities are always negative

## The E-step chooses the responsibilities that minimize the cost function

(with the parameters of the Gaussians held fixed)

- How do we find responsibility values for a datapoint that minimize the cost and sum to 1?
- The optimal solution to the trade-off between expected energy and entropy is to make the responsibilities be proportional to the exponentiated negative energies:

$$energy\ of\ assigning\ c\ to\ i = -\log \pi_i - \log p(\mathbf{x}^c \mid \boldsymbol{\mu}_i, \sigma_i^2)$$

$$optimal\ value\ of\ r_i^c \propto \exp(-energy)$$

$$\propto \pi_i\ p(\mathbf{x}^c \mid i)$$

- So using the posterior probabilities as responsibilities minimizes the cost function!

## The M-step chooses the parameters that minimize the cost function
(with the responsibilities held fixed)

- This is easy. We just fit each Gaussian to the data weighted by the responsibilities that the Gaussian has for the data.
  - When you fit a Gaussian to data you are maximizing the log probability of the data given the Gaussian. This is the same as minimizing the energies of the datapoints that the Gaussian is responsible for.
  - If a Gaussian has a responsibility of 0.7 for a datapoint the fitting treats it as 0.7 of an observation.
- Since both the E-step and the M-step decrease the same cost function, EM converges.