CSC2535: Computation in Neural Networks

Lecture 5: Free Energy and Variational Methods

Geoffrey Hinton

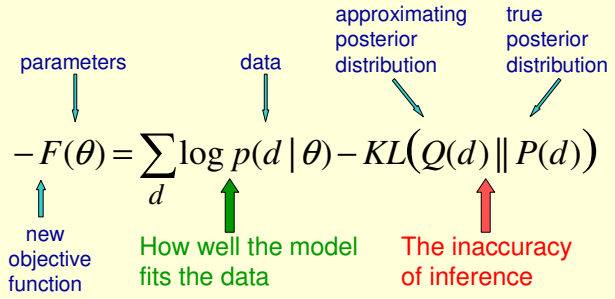
An apparently crazy idea

- Its hard to learn complicated models like Sigmoid Belief Nets because its hard to infer (or sample from) the posterior distribution over hidden configurations.
- Crazy idea: do inference wrong.
  - Maybe learning will still work
  - This turns out to be true for SBN's but not for Boltzmann machines.
    - One more nail in the coffin of Boltzmann Machines.

Approximate inference

- For models like sigmoid belief nets, it is intractable to compute the exact posterior distribution over hidden configurations. So what happens if we use a tractable approximation to the posterior?
  - e.g. assume the posterior over hidden configurations for each datavector factorizes into a product of distributions for each separate hidden cause.
- If we use the approximation for learning, there is no guarantee that learning will increase the probability that the model would generate the observed data.
- But maybe we can find a different and sensible objective function that is guaranteed to improve at each update of the parameters.

A trade-off between how well the model fits the data and the accuracy of inference



This makes it feasible to fit very complicated models, but the approximations that are tractable may be poor.

Two ways to derive F

- We can derive variational free energy as the objective function that is minimized by both steps of the Expectation and Maximization algorithm (EM).
- We can also derive it by using Minimum Description Length ideas.

Overview

- Clustering with K-means and a proof of convergence that uses energies.
- Clustering with a mixture of Gaussians and a proof of convergence that uses free energies
- The MDL view of clustering and the bits-back argument
- The MDL justification for incorrect inference.
- The MDL view of SBN's
- The wake-sleep algorithm.

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 objective function that is optimized by sensible clusterings?

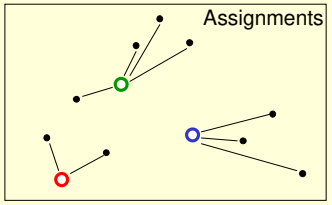
The k-means algorithm

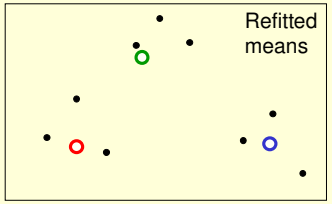
- Assume the data lives in a Euclidean space.
- Assume we want k classes.
- Assume we start with randomly located cluster centers

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.





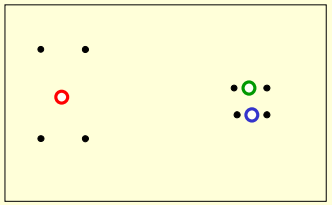
Why K-means converges

- 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.
- Test for convergence: 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 that it would produce exactly the data we observed.

The mixture of Gaussians generative model

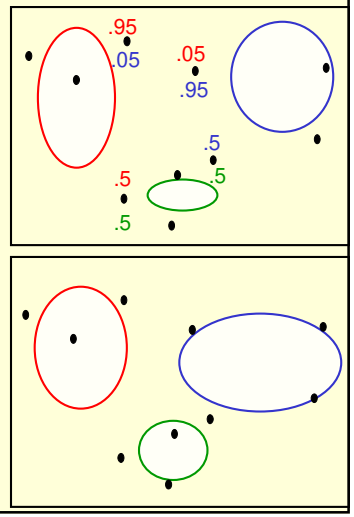
- First pick one of the k Gaussians with a probability that is called its "mixing proportion".
- Then generate a random point from the chosen Gaussian.
- The probability of generating the exact data we observed is zero, but we can still try to maximize the probability density.
  - Adjust the means of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

Fitting a mixture of Gaussians

The EM algorithm alternates between two steps:

E-step: Compute the posterior probability that each Gaussian generates each datapoint.

M-step: Assuming that the data really was generated this way, change the parameters of each Gaussian to maximize the probability that it would generate the data it is currently responsible for.



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 do we know that the updates improve things?

- Updating each Gaussian definitely improves the probability of generating the data if we generate it from the same Gaussians after the parameter updates.
  - But we know that the posterior will change after updating the parameters.
- A good way to show that this is OK is to show that there is a single function that is improved by both the E-step and the M-step.
  - The function we need is called Free Energy.

Why EM converges

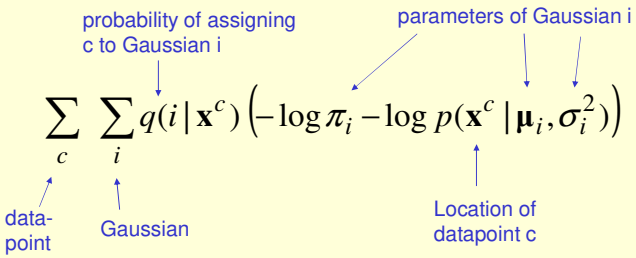
- There is a cost function that is reduced by both the E-step and the M-step.

  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 assigning each datapoint to the Gaussian that generates it most easily (as in K-means).
- The entropy term encourages "soft" assignments. It would be happiest spreading the assignment probabilities 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 probabilities of assigning the datapoint to each Gaussian. We can use any probabilities we like.



The entropy term

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

$$entropy = -\sum_c \sum_i q(i|\mathbf{x}^c) \log q(i|\mathbf{x}^c)$$

log probabilities are always negative

The E-step chooses the assignment probabilities that minimize the cost function
(with the parameters of the Gaussians held fixed)

- How do we find assignment probabilities 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 probabilities be proportional to the exponentiated negative energies:

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

$$optimal\ value\ of\ q(i|\mathbf{x}^c) \propto \exp(-energy)$$

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

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

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

- This is easy. We just fit each Gaussian to the data weighted by the assignment probabilities 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 is assigned a probability 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.

EM as coordinate descent in Free Energy

$$F(\mathbf{x}^c) = \sum_i q(i \mid \mathbf{x}^c)\left(-\log \pi_i - \log p(\mathbf{x}^c \mid i)\right) \; - \; \sum_i q(i \mid \mathbf{x}^c)\left(-\log q(i \mid \mathbf{x}^c)\right)$$

- Think of each different setting of the hidden and visible variables as a "configuration". The energy of the configuration has two terms:
  - The log prob of generating the hidden values
  - The 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.

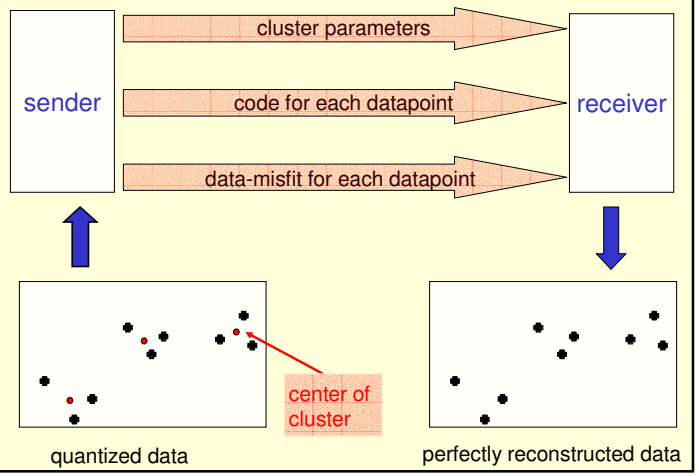
An incremental EM algorithm

- Partial 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.

We already have this sum but it includes the old term for d

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

An MDL approach to clustering



How many bits must we send?

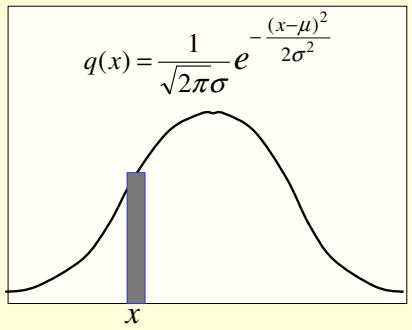
- Model parameters:
  - It depends on the priors and how accurately they are sent.
  - Lets ignore these details for now
- Codes:
  - If all n clusters are equiprobable, log n
    - This is extremely plausible, but wrong!
  - We can do it in less bits
    - This is extremely implausible but right.
- 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

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

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



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

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

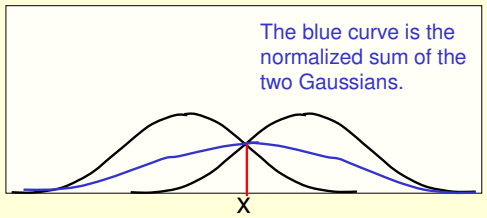
What is the best variance to use?

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

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

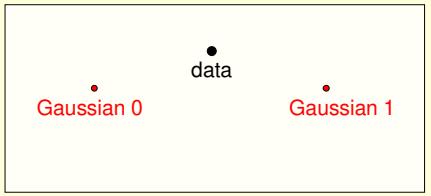
- It is obvious that this is minimized by setting the variance of the Gaussian to be the variance of the residuals.

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 the blue curve.
  - 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

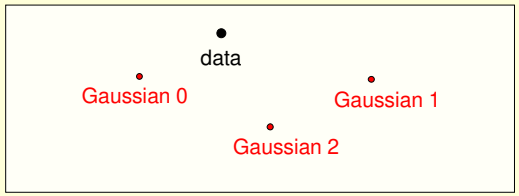


- Consider a datapoint that is equidistant from two cluster centers.
  - The sender could code it relative to cluster 0 or relative to cluster 1.
  - Either way, the sender has to send one bit to say which cluster is being used.
    - It seems like a waste to have to send a bit when you don't care which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



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

Probability of picking cluster i

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

Random bits required to pick which cluster

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).

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

$$F_{ice} = < E_{ice} > -T\ H_{ice}$$

- 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.

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

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

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

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

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.

Why doesn't variational learning work for Boltzmann machines?

- The learning rule involves the difference of two terms.
- If we use approximations to the posterior distributions we get a lower bound on each term.
- This does not give us a bound on the difference.

Why do we need the negative phase?

$$p(\mathbf{v}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{u}}\sum_{\mathbf{g}} e^{-E(\mathbf{u},\mathbf{g})}}$$

The positive phase finds hidden configurations that work well with v and lowers their energies.

The negative phase finds the joint configurations that are the best competitors and raises their energies.

$$\log p(\mathbf{v}) = \log \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} - \log \sum_{\mathbf{u}}\sum_{\mathbf{g}} e^{-E(\mathbf{u},\mathbf{g})}$$

Stochastic units

- Replace the binary threshold units by binary stochastic units that make biased random decisions.
  - The "temperature" controls the amount of noise
  - Decreasing all the energy gaps between configurations is equivalent to raising the noise level.

$$p(s_i{=}1) = \frac{1}{1+e^{-\sum_j s_j w_{ij}/T}} = \frac{1}{1+e^{-\Delta E_i/T}}$$

temperature

$$\text{Energy gap} = \Delta E_i = E(s_i{=}0) - E(s_i{=}1)$$

The annealing trade-off

- At high temperature the transition probabilities for uphill jumps are much greater.

$$p(\text{pick higher energy state}) = \frac{1}{1+e^{\Delta E/T}}$$

Energy increase

- At low temperature the equilibrium probabilities of good states are much better than the equilibrium probabilities of bad ones.

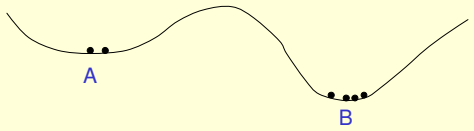
$$\frac{P_A}{P_B} = e^{-\frac{(E_A - E_B)}{T}}$$

How temperature affects transition probabilities

High temperature transition probabilities

$p(A \rightarrow B) = 0.2$

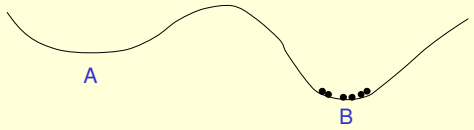
$p(A \leftarrow B) = 0.1$



Low temperature transition probabilities

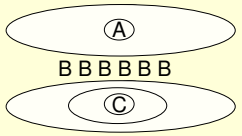
$p(A \rightarrow B) = 0.001$

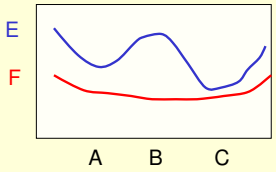
$p(A \leftarrow B) = 0.000001$



Why annealing works

- In high dimensions, energy barriers are typically much more degenerate than the minima that they separate.

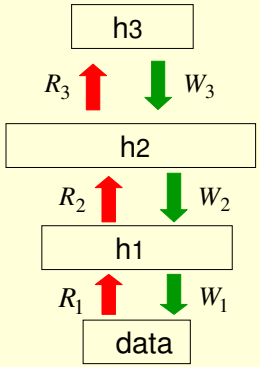




- At high temperature the free energy of the barrier is lower than the free energy of the minima.

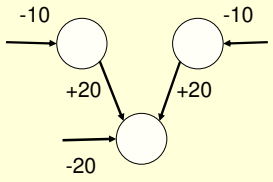
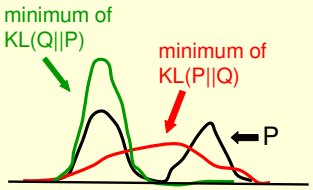
The wake-sleep algorithm

- Wake phase: Use the recognition weights to perform a bottom-up pass.
  - Train the generative weights to reconstruct activities in each layer from the layer above.
- Sleep phase: Use the generative weights to generate samples from the model.
  - Train the recognition weights to reconstruct activities in each layer from the layer below.



The flaws in the wake-sleep algorithm

- The recognition weights are trained to invert the generative model in parts of the space where there is no data.
  - This is wasteful.
- The recognition weights follow the gradient of the wrong divergence. They minimize KL(P||Q) but the variational bound requires minimization of KL(Q||P).
  - This leads to incorrect mode-averaging
- The posterior over the top hidden layer is very far from independent because the independent prior cannot eliminate explaining away effects.

Mode averaging

- If we generate from the model, half the instances of a 1 at the data layer will be caused by a (1,0) at the hidden layer and half will be caused by a (0,1).
  - So the recognition weights will learn to produce (0.5,0.5)
  - This represents a distribution that puts half its mass on very improbable hidden configurations.
- Its much better to just pick one mode and pay one bit.