CSC 2535
Lecture 8
Products of Experts

Geoffrey Hinton

How to combine simple density models

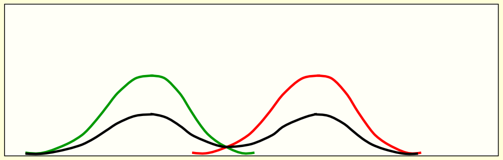
- Suppose we want to build a model of a complicated data distribution by combining several simple models. What combination rule should we use?
- Mixture models take a weighted sum of the distributions
  - Easy to learn
  - The combination is always vaguer than the individual distributions.
- Products of Experts multiply the distributions together and renormalize.
  - The product is much sharper than the individual distributions.
  - A nasty normalization term is needed to convert the product of the individual densities into a combined density.

mixing proportion

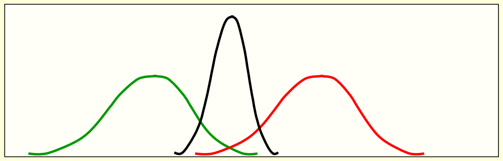
$$p(d) = \sum_m \pi_m p_m(d)$$

$$p(d) = \frac{\prod_m p_m(d)}{\sum_c \prod_m p_m(c)}$$

A picture of the two combination methods



Mixture model: Scale each distribution down and add them together



Product model: Multiply the two densities together at every point and then renormalize.

Products of Experts and energies

- Products of Experts multiply probabilities together. This is equivalent to adding log probabilities.
  - Mixture models add contributions in the probability domain.
  - Product models add contributions in the log probability domain. The contributions are energies.
- In a mixture model, the only way a new component can reduce the density at a point is by stealing mixing proportion.
- In a product model, any expert can veto any point by giving that point a density of zero (i.e. an infinite energy)
  - So its important not to have overconfident experts in a product model.
  - Luckily, vague experts work well because their product can be sharp.

How sharp are products of experts?

- If each of the M experts is a Gaussian with the same variance, the product is a Gaussian with a variance of 1/M on each dimension.
- But a product of lots of Gaussians is just a Gaussian
  - Adding Gaussians allows us to create arbitrarily complicated distributions.
  - Multiplying Gaussians doesn't.
  - So we need to multiply more complicated "experts".

"Uni-gauss" experts

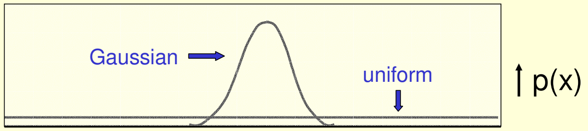
- Each expert is a mixture of a Gaussian and a uniform. This creates an energy dimple.

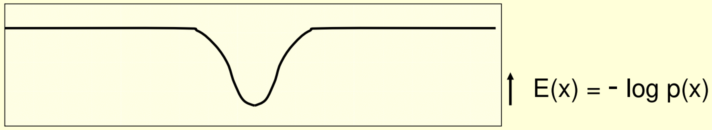
$$p_m(x) = \pi_m N(x \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) + \frac{1-\pi_m}{r}$$

Mixing proportion of Gaussian

Mean and variance of Gaussian

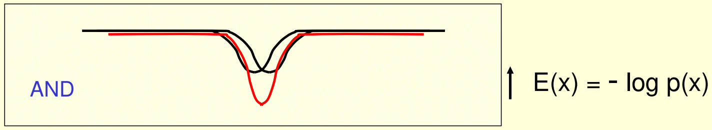
range of uniform

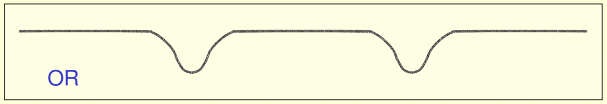




Combining energy dimples

- When we combine dimples, we get a sharper distribution if the dimples are close and a vaguer, multimodal distribution if they are further apart. We can get both multiplication and addition of probabilities.





Generating from a product of experts

- Here is a correct but inefficient way to generate an unbiased sample from a product of experts:
  - Let each expert produce a datavector independently.
  - If all the experts agree, output the datavector.
  - If they do not all agree, start again.
- The experts generate independently, but because of the rejections, their hidden states are not independent in the ensemble of accepted cases.
  - The proportion of rejected attempts implements the normalization term.

Relationship to causal generative models

- Consider the relationship between the hidden variables of two different experts:

	Causal model	Product of experts
Hidden states unconditional on data	independent (generation is easy)	dependent (rejecting away)
Hidden states conditional on data	dependent (explaining away)	independent (inference is easy)

Learning a Product of Experts

datavector

$$p(d\,|\,\theta) = \frac{\prod_{m\in Models} p_m(d\,|\,\theta_m)}{\sum_c \prod_m p_m(c\,|\,\theta_m)}$$

Normalization term to make the probabilities of all possible datavectors sum to 1

$$\log p(d\,|\,\theta) = \sum_m \log p_m(d\,|\,\theta_m) - \log \sum_c \prod_m p_m(c\,|\,\theta_m)$$

$$\frac{\partial \log p(d\,|\,\theta)}{\partial \theta_m} = \frac{\partial \log p_m(d\,|\,\theta_m)}{\partial \theta_m} - \sum_c p(c\,|\,\theta) \frac{\partial \log p_m(c\,|\,\theta_m)}{\partial \theta_m}$$

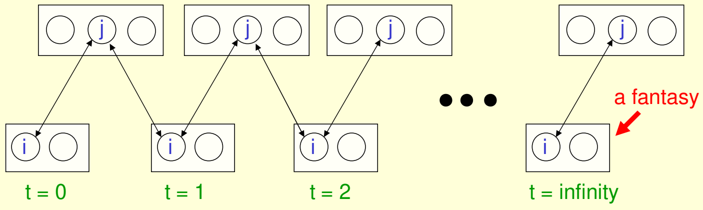
Sum over all possible datavectors

Probability of c under existing product model

Ways to deal with the intractable sum

- Set up a Markov Chain that samples from the existing model.
  - The samples can then be used to get a noisy estimate of the last term in the derivative
  - The chain may need to run for a long time before the fantasies it produces have the correct distribution.
- For uni-gauss experts we can set up a Markov chain by sampling the hidden state of each expert.
  - The hidden state is whether it used the Gaussian or the uniform.
  - The experts' hidden states can be sampled in parallel
    - This is a big advantage of products of experts.

The Markov chain for unigauss experts



Each hidden unit has a binary state which is 1 if the unigauss chose its Gaussian. Start with a training vector on the visible units. Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

Update the hidden states by picking from the posterior.

Update the visible states by picking from the Gaussian you get when you multiply together all the Gaussians for the active hidden units.

A shortcut

- Only run the Markov chain for a few time steps.
  - This gets negative samples very quickly.
  - It works well in practice.
- Why does it work?
  - If we start at the data, the Markov chain wanders away from them data and towards things that it likes more.
  - We can see what direction it is wandering in after only a few steps. It’s a big waste of time to let it go all the way to equilibrium.
  - All we need to do is lower the probability of the “confabulations” it produces and raise the probability of the data. Then it will stop wandering away.
    - The learning cancels out once the confabulations and the data have the same distribution.

A naïve model for binary data

For each component, j, compute its probability, p_j, of being on in the training set. Model the probability of test vector alpha as the product of the probabilities of each of its components:

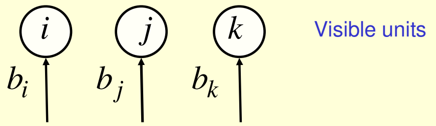
$$p(\mathbf{s}^{\alpha}) = \prod_j \left(\boxed{s_j^{\alpha}}\, p_j + \boxed{(1-s_j^{\alpha})}(1-p_j) \right)$$

Binary vector alpha

If component j of vector alpha is on

If component j of vector alpha is off

A neural network for the naïve model



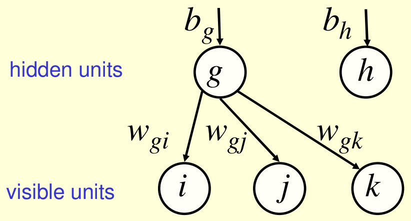
Each visible unit has a bias which determines its probability of being on or off using the logistic function.

A mixture of naïve models

- Assume that the data was generated by first picking a particular naïve model and then generating a binary vector from this naïve model.
  - This is just like the mixture of Gaussians, but for binary data.

$$p(\mathbf{s}^{\alpha}) = \sum_{m \in Models} \pi_m \prod_j \left(s_j^{\alpha} p_j^m + (1-s_j^{\alpha})(1-p_j^m) \right)$$

A neural network for a mixture of naïve models



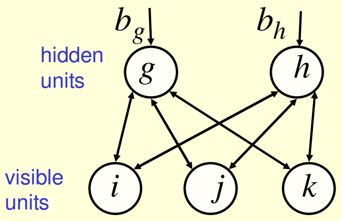
$$p(s_g=1) = \frac{e^{b_g}}{\sum_h e^{b_h}}$$

First activate exactly one hidden unit by picking from a softmax.

Then use the weights of this hidden unit to determine the probability of turning on each visible unit.

A neural network for a product of naïve models

- If you know which hidden units are active, use the weights from all of the active hidden units to determine the probability of turning on a visible unit.
- If you know which visible units are active, use the weights from all of the active visible units to determine the probability of turning on a hidden unit.
- If you do not know the states, start somewhere and alternate between picking hidden states given visible ones and picking visible states given hidden ones.



Alternating updates of the hidden and visible units will eventually sample from a product distribution

The distribution defined by one hidden unit

- If the hidden unit is off, assume the visible units have equal probability of being on and off. (This is the uniform distribution over visible vectors). If the unit is on, assume the visible units have probabilities defined by the hidden unit's weights.
  - So a single hidden unit can be viewed as defining a model that is a mixture of a uniform and a naïve model .
  - The binary state of the hidden unit indicates which component of the mixture we are using.
- Multiplying by a uniform distribution does not affect a normalized product, so we can ignore the hidden units that are off.
  - To sample a visible vector given the hidden states, we just need to multiply together the distributions defined by the hidden units that are on.

The logistic function computes a product of probabilities.

$$p(s=1) = \frac{1}{1+e^{-x}} \quad \textit{is equivalent to} \quad \frac{p(s=1)}{p(s=0)} = e^{x}$$

because p(s=0) = 1 - p(s=1)

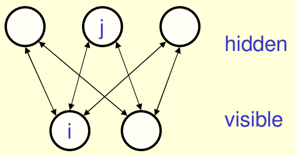
$$\frac{p(s_i{=}1 \mid s_g, s_h)}{p(s_i{=}0 \mid s_g, s_h)} = \frac{p(s_i{=}1 \mid g\ only)}{p(s_i{=}0 \mid g\ only)} \frac{p(s_i{=}1 \mid h\ only)}{p(s_i{=}0 \mid h\ only)}$$

$$= e^{s_g w_{gi}}\, e^{s_h w_{hi}} = e^{\sum_k s_k w_{ki}}$$

Restricted Boltzmann Machines

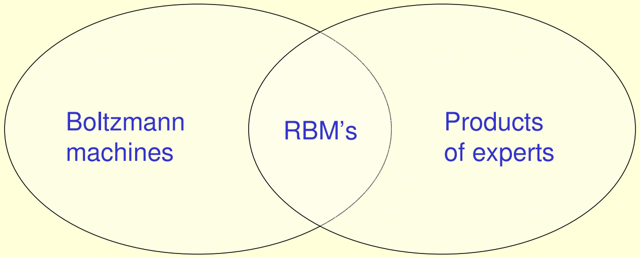
- We restrict the connectivity to make inference and learning easier.
  - Only one layer of hidden units.
  - No connections between hidden units.
- In an RBM it only takes one step to reach thermal equilibrium when the visible units are clamped.
  - So we can quickly get the exact value of :

$$< s_i s_j >_{\mathbf{v}}$$

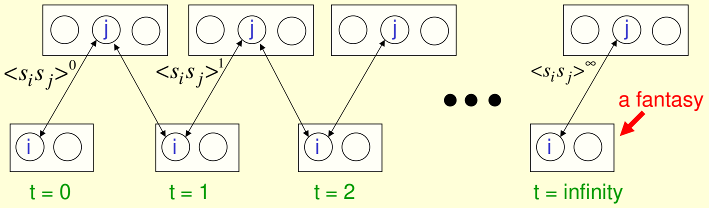

j
i
hidden
visible

$$p(s_j = 1) = \frac{1}{1 + e^{-(b_j + \sum_{i \in vis} s_i w_{ij})}}$$

Restricted Boltzmann Machines and products of experts


Boltzmann machines
RBM's
Products of experts

A picture of the Boltzmann machine learning algorithm for an RBM

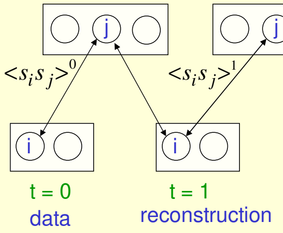

j
i
$<s_i s_j>^0$
$<s_i s_j>^1$
$<s_i s_j>^\infty$
a fantasy
t = 0
t = 1
t = 2
t = infinity

Start with a training vector on the visible units.

Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

$$\Delta w_{ij} = \varepsilon \, (<s_i s_j>^0 - <s_i s_j>^\infty)$$

A surprising short-cut


j
i
$<s_i s_j>^0$
$<s_i s_j>^1$
t = 0
data
t = 1
reconstruction

Start with a training vector on the visible units.

Update all the hidden units in parallel

Update the all the visible units in parallel to get a “reconstruction”.

Update the hidden units again.

$$\Delta w_{ij} = \varepsilon \, (<s_i s_j>^0 - <s_i s_j>^1)$$

This is not following the gradient of the log likelihood. But it works very well.

The shortcut

- Instead of taking the negative samples from the equilibrium distribution, use slight corruptions of the datavectors..
  - Much less variance because a datavector and its confabulation form a matched pair.
  - Seems to be very biased, but maybe it is optimizing a different objective function.
  - What about regions far from the data that have high density under the model?
- If the model is perfect and there is an infinite amount of data, the confabulations will be equilibrium samples. So the shortcut will not cause learning to mess up a perfect model.

Contrastive divergence

Aim is to minimize the amount by which a step toward equilibrium improves the data distribution.

data distribution

model's distribution

distribution after one step of Markov chain

$$CD = KL(P \| Q^{\infty}) - KL(Q^{1} \| Q^{\infty})$$

Minimize Contrastive Divergence

Minimize divergence between data distribution and model's distribution

Maximize the divergence between confabulations and model's distribution

Contrastive divergence

$$-\frac{\partial KL(Q^{0} \| Q^{\infty})}{\partial \theta} = -\left< \frac{\partial E}{\partial \theta} \right>_{Q^{0}} + \left< \frac{\partial E}{\partial \theta} \right>_{Q^{\infty}}$$

$$-\frac{\partial KL(Q^{1} \| Q^{\infty})}{\partial \theta} = -\left< \frac{\partial E}{\partial \theta} \right>_{Q^{1}} + \left< \frac{\partial E}{\partial \theta} \right>_{Q^{\infty}} - \frac{\partial Q^{1}}{\partial \theta} \frac{\partial KL(Q^{1} \| Q^{\infty})}{\partial Q^{1}}$$

Contrastive divergence makes the awkward terms cancel

changing the parameters changes the distribution of confabulations