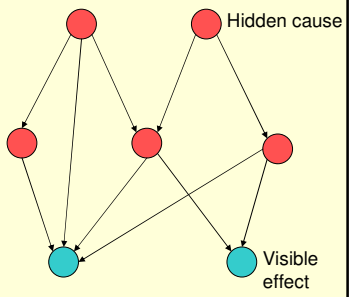
CSC321 2007

Lecture 23: Sigmoid Belief Nets and the wake-sleep algorithm

Geoffrey Hinton

Bayes Nets: Directed Acyclic Graphical models

- The model generates data by picking states for each node using a probability distribution that depends on the values of the node's parents.
- The model defines a probability distribution over all the nodes. This can be used to define a distribution over the leaf nodes.



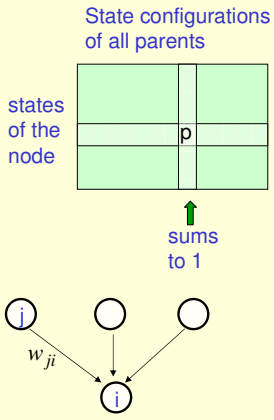
Ways to define the conditional probabilities

For nodes that have discrete values, we could use conditional probability tables.

For nodes that have real values we could let the parents define the parameters of a Gaussian

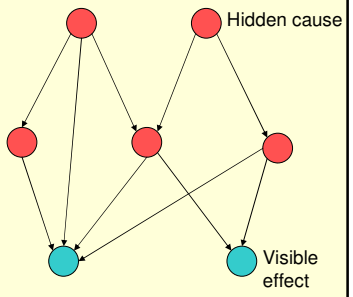
Alternatively we could use a parameterized function. If the nodes have binary states, we could use a sigmoid:

$$p(s_i = 1) = \frac{1}{1 + \exp(-\sum_j s_j w_{ji})}$$



What is easy and what is hard in a DAG?

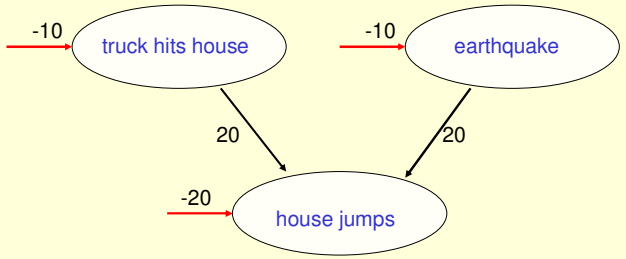
- It is easy to generate an unbiased example at the leaf nodes.
- It is typically hard to compute the posterior distribution over all possible configurations of hidden causes. It is also hard to compute the probability of an observed vector.
- Given samples from the posterior, it is easy to learn the conditional probabilities that define the model.



$$p(v) = \sum_h p(h)\, p(v \mid h)$$

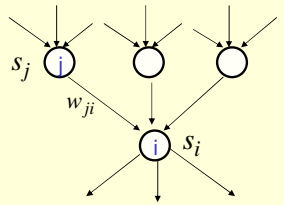
Explaining away

- Even if two hidden causes are independent, they can become dependent when we observe an effect that they can both influence.
  - If we learn that there was an earthquake it reduces the probability that the house jumped because of a truck.



The learning rule for sigmoid belief nets

- Suppose we could "observe" the states of all the hidden units when the net was generating the observed data.
  - E.g. Generate randomly from the net and ignore all the times when it does not generate data in the training set.
  - Keep n examples of the hidden states for each datavector in the training set.
- For each node, maximize the log probability of its "observed" state given the observed states of its parents.



$$p_i \equiv p(s_i = 1) = \frac{1}{1 + \exp(-\sum_j s_j w_{ji})}$$

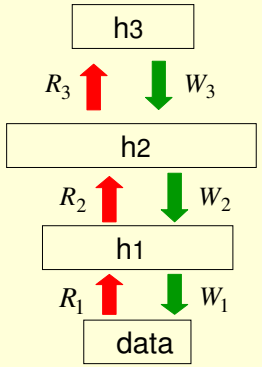
$$\Delta w_{ji} = \varepsilon \, s_j (s_i - p_i)$$

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.
- At each hidden layer, we assume the posterior over hidden configurations factorizes into a product of distributions for each separate hidden unit.

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.

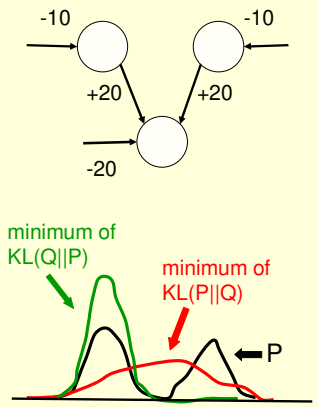


The flaws in the wake-sleep algorithm

- 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 do not follow the gradient of the log probability of the data. Nor do they follow the gradient of a bound on this probability.
  - 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.

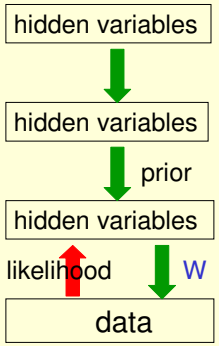
Mode averaging

- 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.



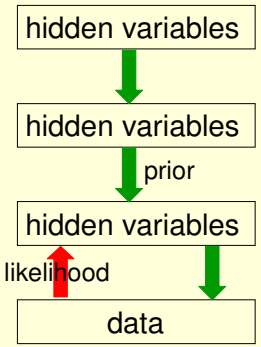
Why its hard to learn one layer at a time

- To learn W, we need the posterior distribution in the first hidden layer.
- Problem 1: The posterior is typically intractable because of "explaining away".
- Problem 2: The posterior depends on the prior as well as the likelihood.
  - So to learn W, we need to know the weights in higher layers, even if we are only approximating the posterior. All the weights interact.
- Problem 3: We need to integrate over all possible configurations of the higher variables to get the prior for first hidden layer. Yuk!



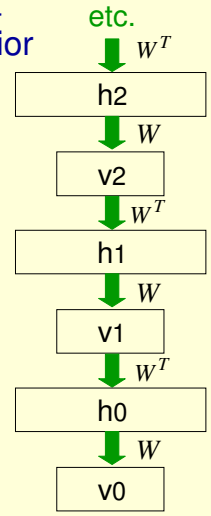
Using complementary priors to eliminate explaining away

- A "complementary" prior is defined as one that exactly cancels the correlations created by explaining away. So the posterior factors.
  - Under what conditions do complementary priors exist?
  - Complementary priors do not exist in general:
    - Parameter counting shows that complementary priors cannot exist if the relationship between the hidden variables and the data is defined by a separate conditional probability table for each hidden configuration.



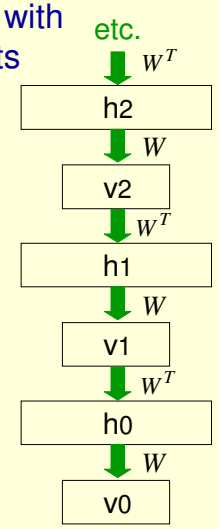
An example of a complementary prior

- The distribution generated by this infinite DAG with replicated weights is the equilibrium distribution for a compatible pair of conditional distributions: p(v|h) and p(h|v).
  - An ancestral pass of the DAG is exactly equivalent to letting a Restricted Boltzmann Machine settle to equilibrium.
  - So this infinite DAG defines the same distribution as an RBM.

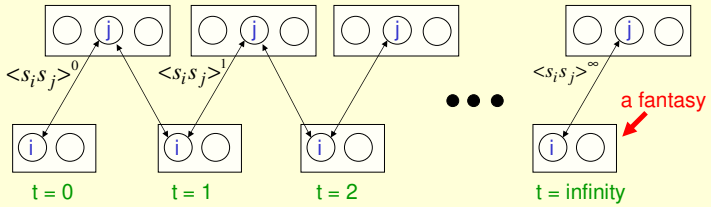


Inference in a DAG with replicated weights

- The variables in h0 are conditionally independent given v0.
  - Inference is trivial. We just multiply v0 by W^T
  - This is because the model above h0 implements a complementary prior.
- Inference in the DAG is exactly equivalent to letting a Restricted Boltzmann Machine settle to equilibrium starting at the data.



A picture of the Boltzmann machine learning algorithm for an RBM



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.

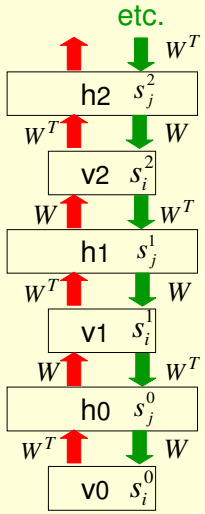
$$\Delta w_{ij} = \varepsilon \left(<s_i s_j>^0 - <s_i s_j>^\infty \right)$$

- The learning rule for a logistic DAG is:

$$\Delta w_{ij} \propto s_j (s_i - \hat{s}_i)$$

- With replicated weights this becomes:

$$s_j^0 (s_i^0 - s_i^1) + s_i^1 (s_j^0 - s_j^1) + s_j^1 (s_i^1 - s_i^2) + \ldots s_j^\infty s_i^\infty$$



Pro's and con's of replicating the weights

Advantages

- There are many less parameters.
- There is an efficient approximate learning procedure.
- After learning, inference of hidden states is fast and accurate.

Disadvantages

- The model is much less powerful than a deep network that has different weights in each layer.
- The brain clearly uses deep networks.

The up-down algorithm: A contrastive divergence version of wake-sleep

- Replace the top layer of the DAG by an RBM
  - This eliminates bad approximations caused by top-level units that are independent in the prior.
  - It is nice to have an associative memory at the top.
- Replace the ancestral pass in the sleep phase by a top-down pass starting with the state of the RBM produced by the wake phase.
  - This makes sure the recognition weights are trained in the vicinity of the data.
  - It also reduces mode averaging. If the recognition weights prefer one mode, they will stick with that mode even if the generative weights like some other mode just as much.