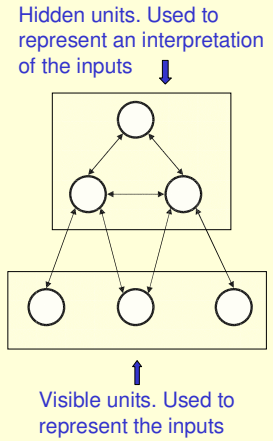
CSC2535 Lecture 4

Boltzmann Machines, Sigmoid Belief Nets and Gibbs sampling

Geoffrey Hinton

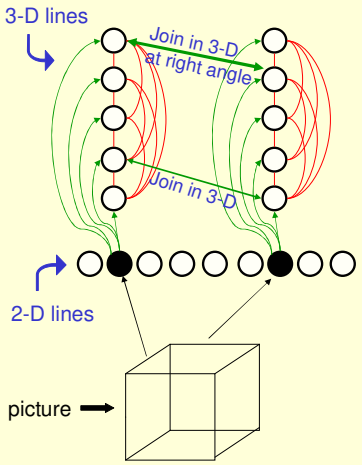
Another computational role for Hopfield nets

- Instead of using the net to store memories, use it to construct interpretations of sensory input.
  - The input is represented by the visible units.
  - The interpretation is represented by the states of the hidden units.
  - The badness of the interpretation is represented by the energy
- This raises two difficult issues:
  - How do we escape from poor local minima to get good interpretations?
  - How do we learn the weights on connections to the hidden units?



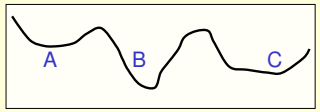
An example: Interpreting a line drawing

- Use one "2-D line" unit for each possible line in the picture.
  - Any particular picture will only activate a very small subset of the line units.
- Use one "3-D line" unit for each possible 3-D line in the scene.
  - Each 2-D line unit could be the projection of many possible 3-D lines. Make these 3-D lines compete.
- Make 3-D lines support each other if they join in 3-D. Make them **strongly support** each other if they join at right angles.



Noisy networks find better energy minima

- A Hopfield net always makes decisions that reduce the energy.
  - This makes it impossible to escape from local minima.
- We can use random noise to escape from poor minima.
  - Start with a lot of noise so its easy to cross energy barriers.
  - Slowly reduce the noise so that the system ends up in a deep minimum. This is "simulated annealing".



- We will come back to simulated annealing later. For now, we will keep the noise level fixed to avoid unneccessary complications in explaining the other good things that result from using stochastic units.

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

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

How a Boltzmann Machine models data

- It is not a causal generative model (like a sigmoid belief net) in which we first pick the hidden states and then pick the visible states given the hidden ones.
- Instead, everything is defined in terms of energies of joint configurations of the visible and hidden units.

The Energy of a joint configuration

binary state of unit i in joint configuration v, h

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i\in units} s_i^{\mathbf{vh}} b_i - \sum_{i<j} s_i^{\mathbf{vh}} s_j^{\mathbf{vh}} w_{ij}$$

Energy with configuration v on the visible units and h on the hidden units

bias of unit i

weight between units i and j

indexes every non-identical pair of i and j once

Using energies to define probabilities

- The probability of a joint configuration over both visible and hidden units depends on the energy of that joint configuration compared with the energy of all other joint configurations.

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

partition function

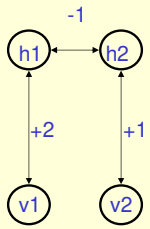
- The probability of a configuration of the visible units is the sum of the probabilities of all the joint configurations that contain it.

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

An example of how weights define a distribution

v	h	$-E$	e^{-E}	$p(\mathbf{v},\mathbf{h})$	$p(\mathbf{v})$
1 1	1 1	2	7.39	.186	0.466
1 1	1 0	2	7.39	.186	
1 1	0 1	1	2.72	.069	
1 1	0 0	0	1	.025	
1 0	1 1	1	2.72	.069	0.305
1 0	1 0	2	7.39	.186	
1 0	0 1	0	1	.025	
1 0	0 0	0	1	.025	
0 1	1 1	0	1	.025	0.144
0 1	1 0	0	1	.025	
0 1	0 1	1	2.72	.069	
0 1	0 0	0	1	.025	
0 0	1 1	-1	0.37	.009	0.084
0 0	1 0	0	1	.025	
0 0	0 1	0	1	.025	
0 0	0 0	0	1	.025	

total =39.70



Getting a sample from the model

- If there are more than a few hidden units, we cannot compute the normalizing term (the partition function) because it has exponentially many terms.
- So use Markov Chain Monte Carlo to get samples from the model:
  - Start at a random global configuration
  - Keep picking units at random and allowing them to stochastically update their states based on their energy gaps.
- At thermal equilibrium, the probability of a global configuration is given by the Boltzmann distribution.

Thermal equilibrium

- Thermal equilibrium is a difficult concept!
  - It does not mean that the system has settled down into the lowest energy configuration.
  - The thing that settles down is the probability distribution over configurations.

Thermal equilibrium

- The best way to think about it is to imagine a huge ensemble of systems that all have exactly the same energy function.
  - The probability distribution is just the fraction of the systems that are in each possible configuration.
- We could start with all the systems in the same configuration, or with an equal number of systems in each possible configuration.
  - After running the systems stochastically in the right way, we eventually reach a situation where the number of systems in each configuration remains constant even though any given system keeps moving between configurations

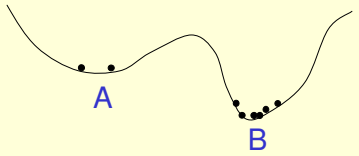
An analogy

- Imagine a casino in Las Vegas that is full of card dealers (we need many more than 52! of them).
- We start with all the card packs in standard order and then the dealers all start shuffling their packs.
  - After a few time steps, the king of spades still has a good chance of being next to queen of spades. The packs have not been fully randomized.
  - After prolonged shuffling, the packs will have forgotten where they started. There will be an equal number of packs in each of the 52! possible orders.
  - Once equilibrium has been reached, the number of packs that leave a configuration at each time step will be equal to the number that enter the configuration.
- The only thing wrong with this analogy is that all the configurations have equal energy, so they all end up with the same probability.

Detailed Balance

- When a Boltzmann machine reaches thermal equilibrium, the asymmetric transition probabilities between any pair of global configurations, A, B, are balanced by the relative probabilities of those configurations:

$$p(A)\ p(A \rightarrow B) \quad = \quad p(B)\ p(B \rightarrow A)$$



Getting a sample from the posterior distribution over distributed representations for a given data vector

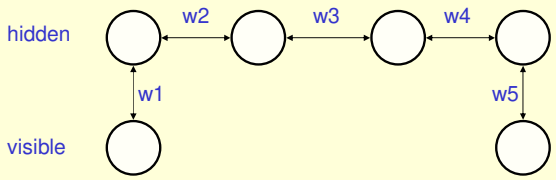
- The number of possible hidden configurations is exponential so we need MCMC to sample from the posterior.
  - It is just the same as getting a sample from the model, except that we keep the visible units clamped to the given data vector.
    - Only the hidden units are allowed to change states
- Samples from the posterior are required for learning the weights.

The goal of learning

- Maximize the product of the probabilities that the Boltzmann machine assigns to the vectors in the training set.
  - This is equivalent to maximizing the sum of the log probabilities of the training vectors.
  - It is also equivalent to maximizing the probabilities that we will observe those vectors on the visible units if we take random samples after the whole network has reached thermal equilibrium with no external input.

Why the learning could be difficult

Consider a chain of units with visible units at the ends



If the training set is (1,0) and (0,1) we want the product of all the weights to be negative.

So to know how to change w1 or w5 we must know w3.

A very surprising fact

- Everything that one weight needs to know about the other weights and the data is contained in the difference of two correlations.

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle s_i s_j \rangle_{\mathbf{v}} - \langle s_i s_j \rangle_{free}$$

Derivative of log probability of one training vector

Expected value of product of states at thermal equilibrium when the training vector is clamped on the visible units

Expected value of product of states at thermal equilibrium when nothing is clamped

The batch learning algorithm

- Positive phase
  - Clamp a datavector on the visible units.
  - Let the hidden units reach thermal equilibrium at a temperature of 1 (may use annealing to speed this up)
  - Sample $s_i s_j$ for all pairs of units
  - Repeat for all datavectors in the training set.
- Negative phase
  - Do not clamp any of the units
  - Let the whole network reach thermal equilibrium at a temperature of 1 (where do we start?)
  - Sample $s_i s_j$ for all pairs of units
  - Repeat many times to get good estimates
- Weight updates
  - Update each weight by an amount proportional to the difference in $< s_i s_j >$ in the two phases.

Why is the derivative so simple?

- The probability of a global configuration at thermal equilibrium is an exponential function of its energy.
  - So settling to equilibrium makes the log probability a linear function of the energy
- The energy is a linear function of the weights and states

$$\frac{\partial E}{\partial w_{ij}} = s_i s_j$$

- The process of settling to thermal equilibrium propagates information about the weights.

Why do we need the negative phase?

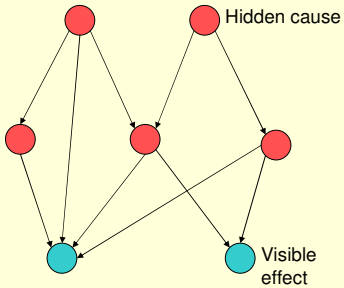
$$p(\mathbf{v}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{u}}\sum_{\mathbf{g}} e^{-E(\mathbf{u},\mathbf{g})}}$$

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.

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.



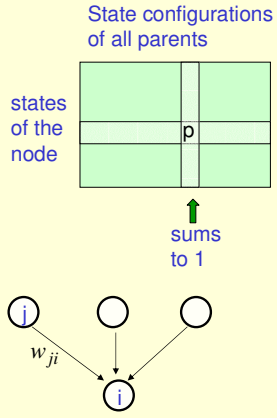
Ways to define the conditional probabilities

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

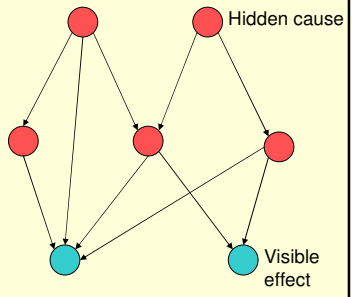
Alternatively we could use a parameterized function. If the nodes have binary states, we could use a sigmoid:

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



What is easy and what is hard in a DAG?

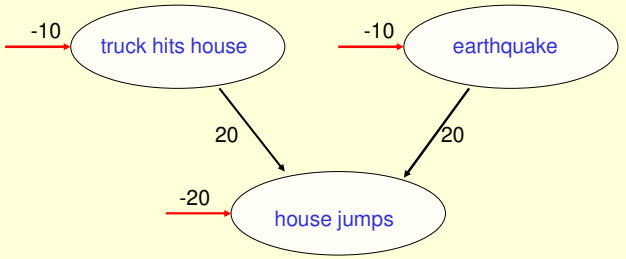
- It is easy to generate an unbiased example at the leaf nodes.
- It is typically hard to compute the posterior distribution over all possible configurations of hidden causes. It is also hard to compute the probability of an observed vector.
- Given samples from the posterior, it is easy to learn the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



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

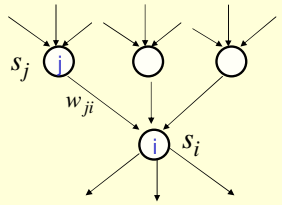
Explaining away

- 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.



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

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

The derivatives of the log prob

- If unit i is on:

$$\log p(s_i = 1) = -\log(1 + e^{-x_i})$$

$$\frac{\partial \log p(s_i = 1)}{\partial w_{ji}} = \frac{\partial x_i}{\partial w_{ji}} \frac{e^{-x_i}}{1 + e^{-x_i}}$$

$$= s_j (1 - p_i)$$

- If unit i is off:

$$\log p(s_i = 0) = \log\left(\frac{e^{-x_i}}{1 + e^{-x_i}}\right)$$

$$= -\log(1 + e^{-x_i}) - x_i$$

$$\frac{\partial \log p(s_i = 0)}{\partial w_{ji}} = s_j (1 - p_i) - s_j$$

- In both cases we get:

$$s_j (s_i - p_i)$$

Sampling from the posterior distribution

- In a densely connected sigmoid belief net with many hidden units it is intractable to compute the full posterior distribution over hidden configurations.
  - There are too many configurations to consider.
- But we can learn OK if we just get samples from the posterior.
  - So how can we get samples efficiently?
    - Generating at random and rejecting cases that do not produce data in the training set is hopeless.

Gibbs sampling

- First fix a datavector from the training set on the visible units.
- Then keep visiting hidden units and updating their binary states using information from their parents and descendants.
- If we do this in the right way, we will eventually get unbiased samples from the posterior distribution for that datavector.
- This is relatively efficient because almost all hidden configurations will have negligible probability and will probably not be visited.

The recipe for Gibbs sampling

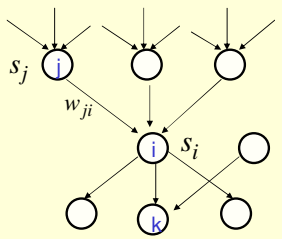
- Imagine a huge ensemble of networks.
  - The networks have identical parameters.
  - They have the same clamped datavector.
  - The fraction of the ensemble with each possible hidden configuration defines a distribution over hidden configurations.
- Each time we pick the state of a hidden unit from its posterior distribution given the states of the other units, the distribution represented by the ensemble gets closer to the equilibrium distribution.
  - A quantity called the "free energy" always decreases (see next lecture)
  - Eventually, we reach the stationary distribution in which the number of networks that change from configuration a to configuration b is exactly the same as the number that change from b to a:

Computing the posterior for i given the rest

- We need to compute the difference between the energy of the whole network when i is on and the energy when i is off.
  - Then the posterior probability for i is:

$$p(s_i = 1) = \frac{1}{1 + e^{-(E_{off} - E_{on})}}$$

- Changing the state of i changes two kinds of energy term:
  - how well the parents of i predict the state of i
  - How well i and its siblings predict the state of each descendant of i.



Terms in the global energy

- Compute for each descendant of i how the cost of predicting the state of that descendant changes

$$E_i^{below} = -\sum_{i \in pa(k)} \log p(s_k \mid pa(k))$$

- Compute for i itself how the cost of predicting the state of i changes

$$E_i^{above} = -\log p(s_i \mid pa(i))$$

parents of i

Ways to combine Gibbs sampling with learning

- The obvious method is to start with a random hidden configuration for each datavector and to do Gibbs sampling until we have reached equilibrium.
- Then use the equilibrium samples from the posterior distribution over hidden configurations to update the weights (online or batch or mini-batch)
- But how do we decide how much Gibbs sampling is required to reach equilibrium?
  - There is no simple test and if we don't do enough there is no guarantee that the learning will work, even if we use an infinitesimal learning rate.

A clever trick

- Instead of starting with a random hidden configuration, use the last hidden configuration for that training datavector before the weights were updated.
  - If the weight updates are small enough, the hidden configurations will start very close to the equilibrium distribution for each training datavector and the Gibbs sampling will make them even closer.
  - So we might as well update the weights after one round of Gibbs updating for each training datavector
- This method is even cleverer than it appears.
  - We will see in the next lecture that it works even if the hidden configurations are not close to equilibrium.

Comparison of sigmoid belief nets and Boltzmann machines

- SBN's can use a bigger learning rate because they do not have the negative phase (see Neal's paper).
- It is much easier to generate samples from an SBN so we can see what model we learned.
- It is easier to interpret the units as hidden causes.
- The Gibbs sampling procedure is much simpler in BM's.
- Gibbs sampling and learning only require communication of binary states in a BM, so its easier to fit into a brain.

Two types of density model with hidden units

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

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{h}) p(\mathbf{v} \mid \mathbf{h})$$

Generation from model is easy
Inference is generally hard
Learning is easy after inference

Energy-based models that associate an energy with each joint configuration

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

Generation from model is hard
Inference is generally hard
Learning requires a negative phase that is even harder than inference

This comparison looks bad for energy-based models