CIAR Second Summer School Tutorial
Lecture 1a

Sigmoid Belief Nets and Boltzmann Machines

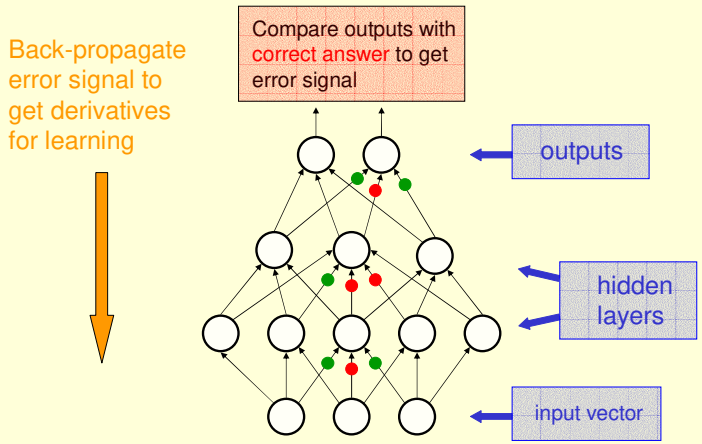
Geoffrey Hinton

A very old idea about how to build a perceptual system

- Start by learning some features of the raw sensory input. The features should capture interesting regularities in the input.
- Then learn another layer of features by treating the first layer of features as sensory data.
- Keep learning layers of features until the highest level "features" are so complex that they make it very easy to recognize objects, speech ….
- Fifty years later, we can finally make this work!

Good old-fashioned neural networks

Back-propagate error signal to get derivatives for learning



What is wrong with back-propagation?

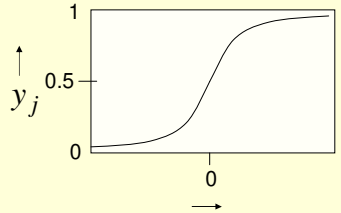
- It requires labeled training data.
  - Almost all data is unlabeled.
- We need to fit about 10^14 connection weights in only about 10^9 seconds.
  - Unless the weights are highly redundant, labels cannot possibly provide enough information.
- The learning time does not scale well
  - It is very slow in networks with more than two or three hidden layers.
- The neurons need to send two different types of signal
  - Forward pass: signal = activity = y
  - Backward pass: signal = dE/dy

Overcoming the limitations of back-propagation

- We need to keep the efficiency of using a gradient method for adjusting the weights, but use it for modeling the structure of the sensory input.
  - Adjust the weights to maximize the probability that a generative model would have produced the sensory input. This is the only place to get 10^5 bits per second.
  - Learn p(image) not p(label | image)
- What kind of generative model could the brain be using?

The building blocks: Binary stochastic neurons

- y is the probability of producing a spike.



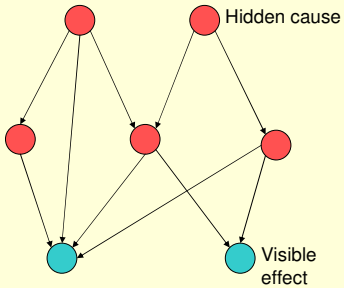
$$total\ input\ to\ neuron\ j = external\ input + \sum_i y_i w_{ij}$$

synaptic weight from i to j

output of neuron i

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.



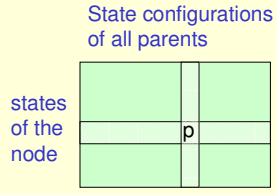
Ways to define the conditional probabilities

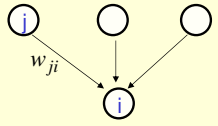
For nodes that have discrete values, we could use conditional probability tables.

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:

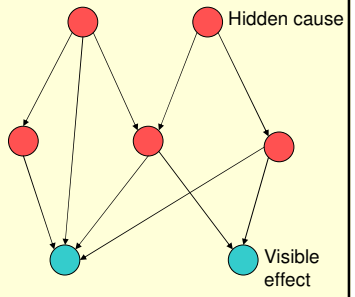
$$p(s_i = 1) = \frac{1}{1 + \exp(-\sum_j s_j w_{ji})}$$





What is easy and what is hard in a DAG?

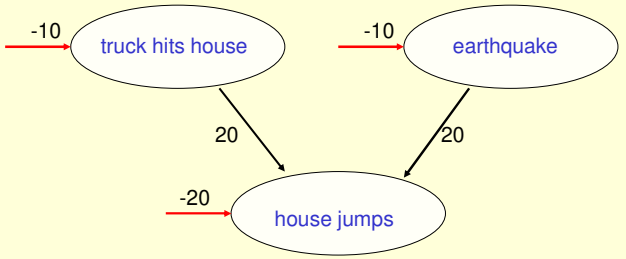
- It is easy to generate an unbiased example at the leaf nodes.
- It is typically hard to compute the posterior distribution over all possible configurations of hidden causes. It is also hard to compute the probability of an observed vector.
- Given samples from the posterior, it is easy to learn the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



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

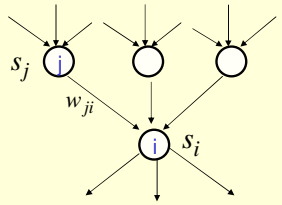
Explaining away

- 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 one example 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.
  - This minimizes the energy of the complete configuration.



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

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

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:

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

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

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

- 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.
  - The free energy, F, always decreases.
  - 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:

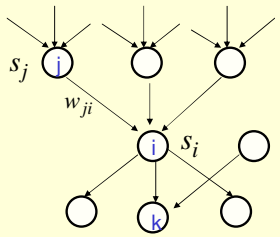
$$p(a)\,p(a \text{ to } b) \quad = \quad p(b)\,p(b \text{ to } a)$$

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 spouses 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

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

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

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

Approximate inference

- What if we use an approximation to the posterior distribution over hidden configurations?
  - e.g. assume the posterior 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.

The Free Energy

Free energy with data d clamped on visible units

$$F(d) = \sum_{c \in configs} p(c \mid d) E_c - \left[-\sum_{c \in configs} p(c \mid d) \log p(c \mid d) \right]$$

Expected energy

Entropy of distribution over configurations

Picking configurations with probability proportional to exp(-E) minimizes the free energy.

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

parameters, data, approximating posterior distribution, true posterior distribution

$$-F(\theta) = \sum_d \log p(d \mid \theta) - KL\big(Q(d) \,\|\, P(d)\big)$$

new objective function

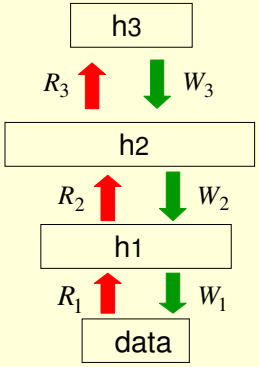
How well the model fits the data

The inaccuracy of inference

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

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.



What the wake phase achieves

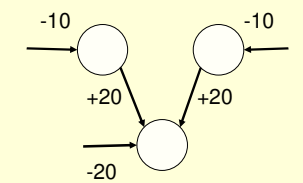
- The bottom-up recognition weights are used to compute a sample from the distribution Q over hidden configurations. Q approximates the true posterior, P.
  - In each layer Q assumes the states are independent given the states in the layer below. It ignores explaining away.
- The changes to the generative weights are designed to reduce the average cost (i.e. energy) of generating the data when the hidden configurations are sampled from the approximate posterior.
  - The updates to the generative weights follow the gradient of the variational bound with respect to the parameters of the model.

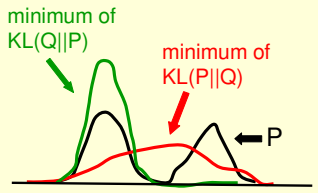
The flaws in the wake-sleep algorithm

- 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.

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.





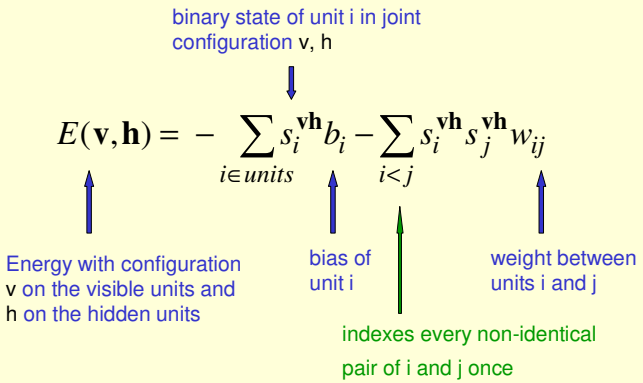
Summary

- By using the variational bound, we can learn sigmoid belief nets quickly.
- If we add bottom-up recognition connections to a generative sigmoid belief net, we get a nice neural network model that requires a wake phase and a sleep phase.
  - The activation rules and the learning rules are very simple in both phases. This makes neuroscientists happy.
- But there are problems:
  - The learning of the recognition weights in the sleep phase is not quite following the gradient of the variational bound.
  - Even if we could follow the right gradient, the variational approximation might be so crude that it severely limits what we can learn.
- Variational learning works because the learning tries to find regions of the parameter space in which the variational bound is fairly tight, even if this means getting a model that gives lower log probability to the data.

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



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

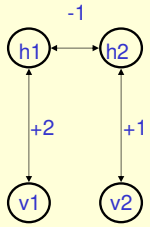
- The probability of a configuration of the visible units is the sum of the probabilities of all the joint configurations that contain it.

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

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

- 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

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

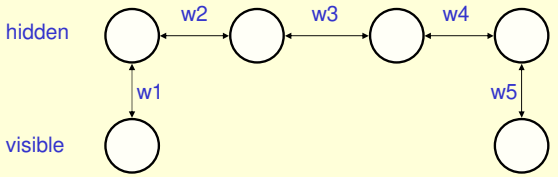
- The number of possible hidden configurations is exponential so we need MCMC to sample from the posterior.
  - It is just the same as getting a sample from the model, except that we keep the visible units clamped to the given data vector.
    - Only the hidden units are allowed to change states
- Samples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

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)

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

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