Deep Belief Nets

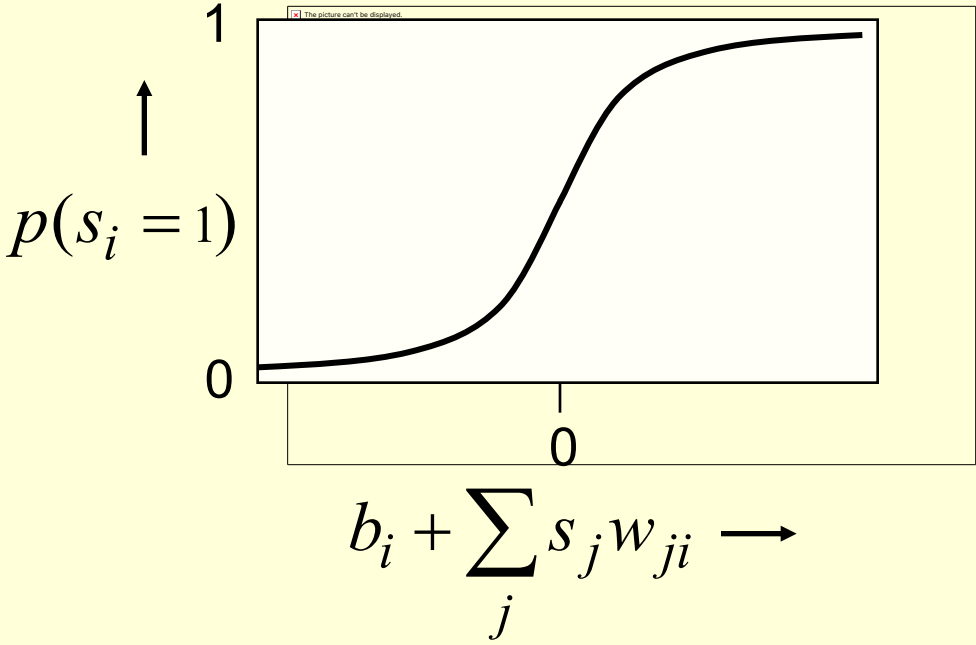
Geoffrey Hinton

Engineering Fellow,
Google Brain Team

Chief Scientific Adviser,
Vector Institute

Stochastic binary neurons

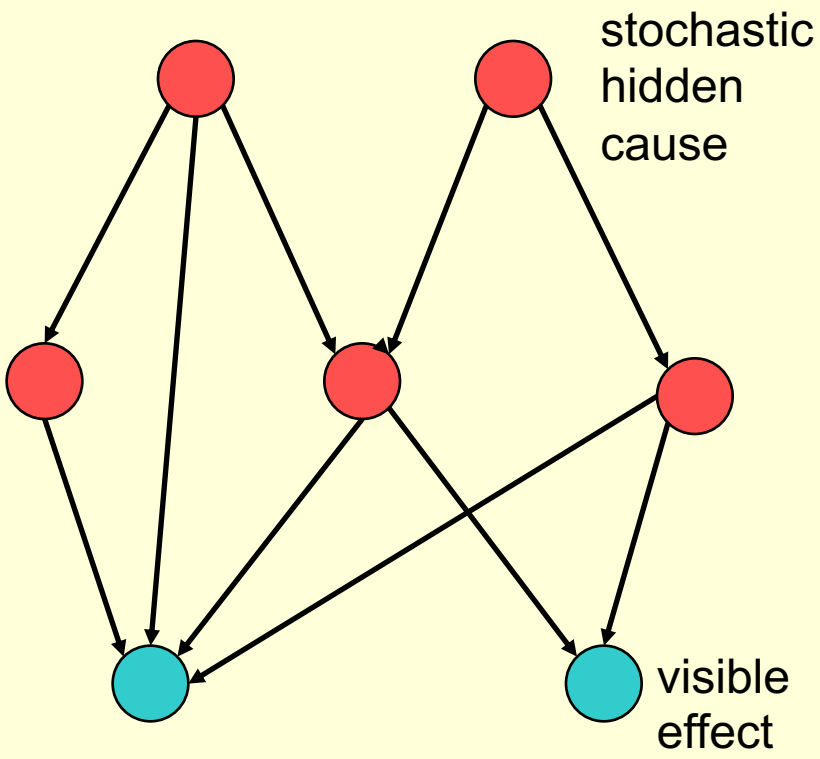
- These have a state of 1 or 0.
- The probability of turning on is determined by the weighted input from other neurons (plus a bias)



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

Belief Nets

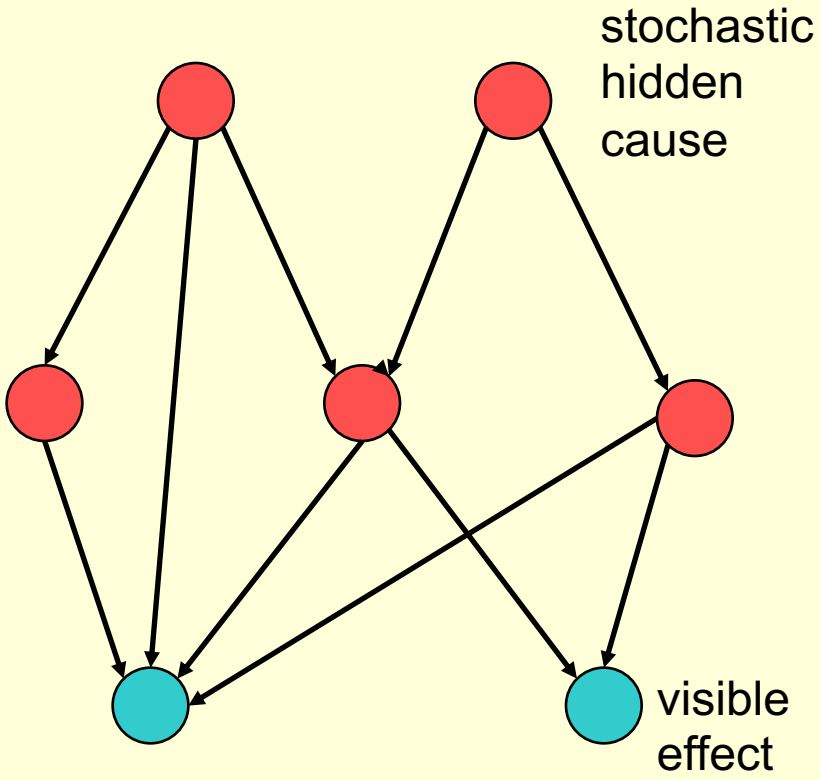
- A belief net is a directed acyclic graph composed of stochastic variables.
- We get to observe some of the variables and we would like to solve two problems:
- The inference problem: Infer the states of the unobserved variables.
- The learning problem: Adjust the interactions between variables to make the network more likely to generate the observed data.



We will use nets composed of layers of stochastic binary variables with weighted connections

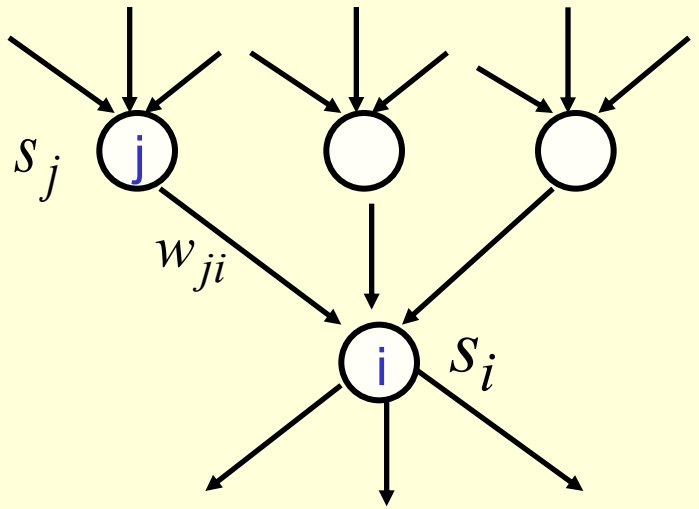
Learning Belief Nets

- It is easy to generate an unbiased example at the leaf nodes, so we can see what kinds of data the network believes in.
- It is hard to infer the posterior distribution over all possible configurations of hidden causes.
- It is hard to even get a sample from the posterior.
- So how can we learn deep belief nets that have millions of parameters?



The learning rule for sigmoid belief nets

- Learning is easy if we can get an unbiased sample from the posterior distribution over hidden states given the observed data.



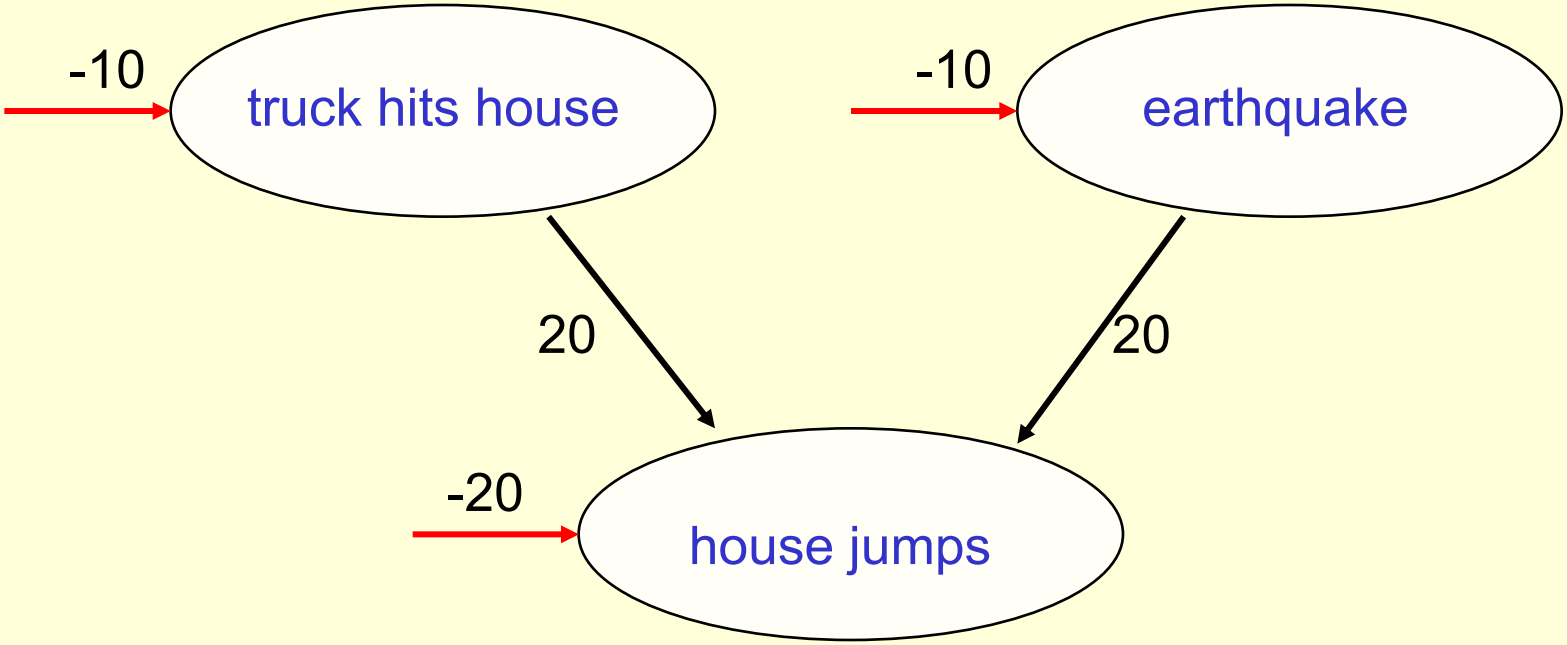
$$p_i \equiv p(s_i = 1) = \frac{1}{1 + \exp(-\sum_j s_j w_{ji})}$$

- For each unit, maximize the log probability that its binary state in the sample from the posterior would be generated by the sampled binary states of its parents.

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

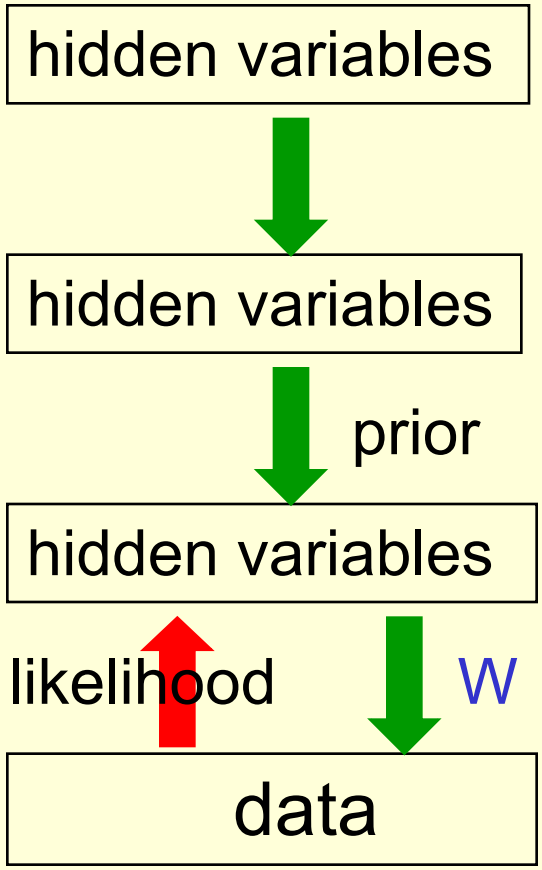
Explaining away (Judea Pearl)

- 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.



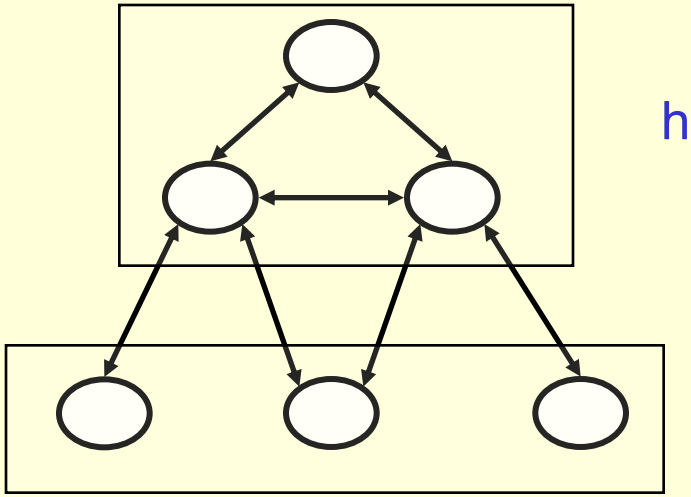
Why it is usually very hard to learn sigmoid belief nets one layer at a time

- To learn W, we need the posterior distribution in the first hidden layer.
- Problem 1: The posterior is typically very complicated 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!



Two types of generative neural network

- If we connect binary stochastic neurons in a directed acyclic graph we get a Sigmoid Belief Net (Radford Neal 1992).
- If we connect binary stochastic neurons using symmetric connections we get a Boltzmann Machine (Hinton & Sejnowski, 1983).



The energy function of a Boltzmann Machine

- The global energy is the sum of many contributions. Each contribution depends on one connection weight and the binary states of two neurons:

$$E = -\sum_i s_i b_i - \sum_{i<j} s_i s_j w_{ij}$$

- This simple quadratic energy function makes it possible for each unit to compute **locally** how it's state affects the global energy:

$$\text{Energy gap} = \Delta E_i = E(s_i = 0) - E(s_i = 1) = b_i + \sum_j s_j w_{ij}$$

How to generate samples from a Boltzmann machine

- Repeatedly update the states of the stochastic binary units using the update:

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

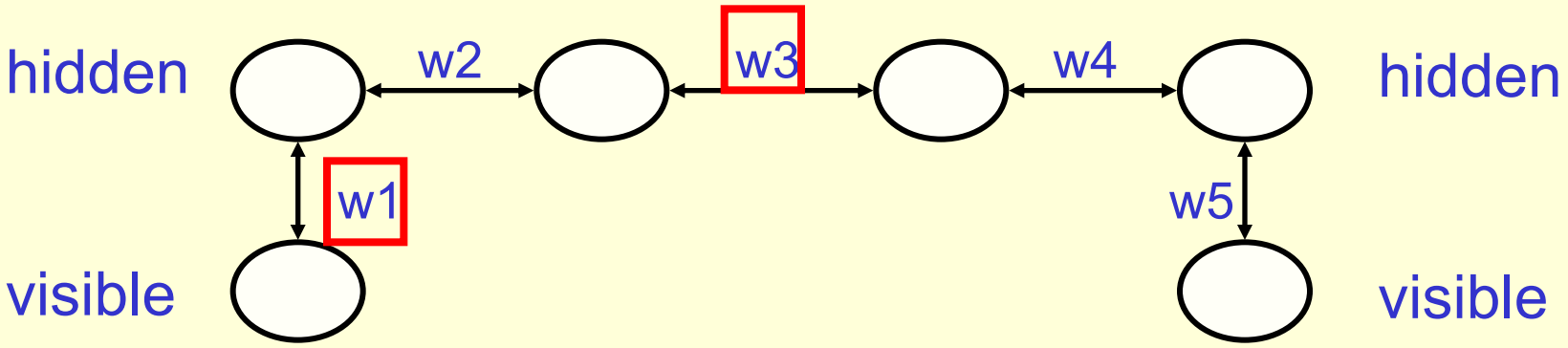
- Eventually, we will sample each global state with a probability proportional to exp(-E).

The goal of learning in a Boltzmann Machine

- We want to maximize the product of the probabilities that the Boltzmann machine assigns to the binary visible vectors in the training set.
  - This is equivalent to maximizing the sum of the log probabilities that the Boltzmann machine assigns to the training vectors.
- It is also equivalent to maximizing the probability that we would obtain exactly the N training vectors if we did the following
  - Let the network settle to its stationary distribution N different times with no external input.
  - Sample the visible vector once each time.

Why the learning could be difficult

Consider a chain of units with visible units at the ends



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

So to know how to change w1 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.

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

Derivative of log probability of one training vector, v under the model.

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

Expected value of product of states at thermal equilibrium with no clamping

$$\Delta w_{ij} \propto \left\langle s_i s_j \right\rangle_{data} - \left\langle s_i s_j \right\rangle_{model}$$

The obvious way to collect the statistics for learning

Hinton and Sejnowski (1983)

- Positive phase: Clamp a data vector on the visible units and set the hidden units to random binary states.
  - Update the hidden units one at a time until the network reaches thermal equilibrium at a temperature of 1.
  - Sample $< s_i s_j >$ for every connected pair of units.
  - Repeat for all data vectors in the training set and average.

- Negative phase: Set all the units to random binary states.
  - Update all the units one at a time until the network reaches thermal equilibrium at a temperature of 1.
  - Sample $< s_i s_j >$ for every connected pair of units.
  - Repeat many times (how many?) and average to get good estimates.

Why is the derivative so simple?

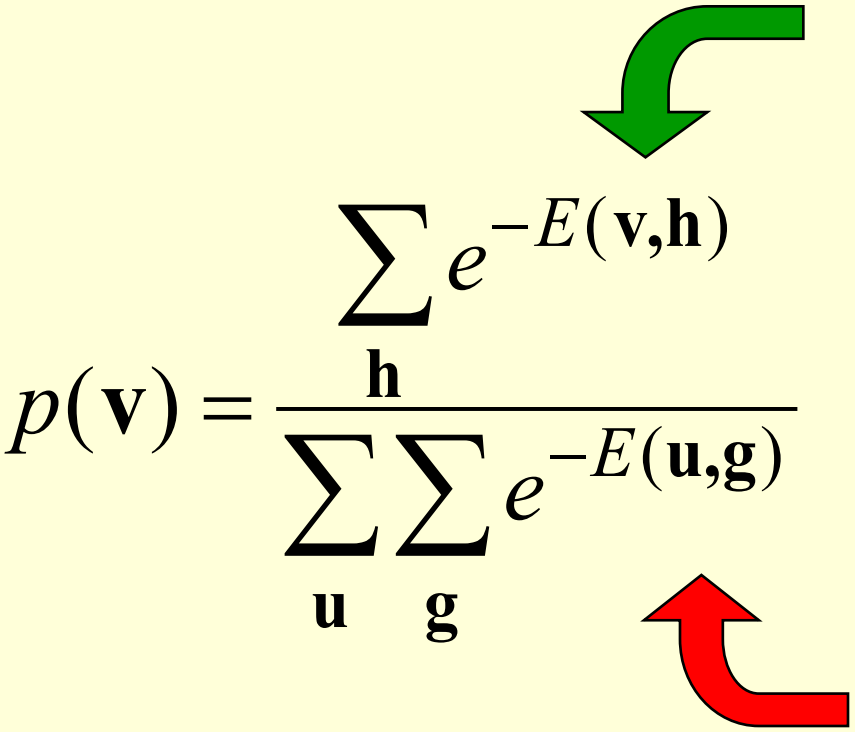
- The probability of a global configuration at thermal equilibrium is an exponential function of its energy.
  - So settling to equilibrium makes the log probability a linear function of the energy.
- The energy is a linear function of the weights and states, so:

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

- The process of settling to thermal equilibrium propagates information about the weights.
  - We don't need backpropagation.

Why do we need the negative phase?

(two ways to win a horse race)



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

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

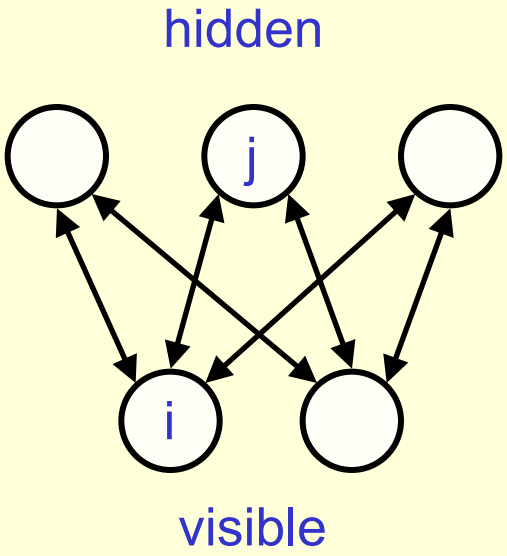
The negative phase finds the joint configurations that are the best competitors and raises their energies.

Why Boltzmann machines are hard to learn

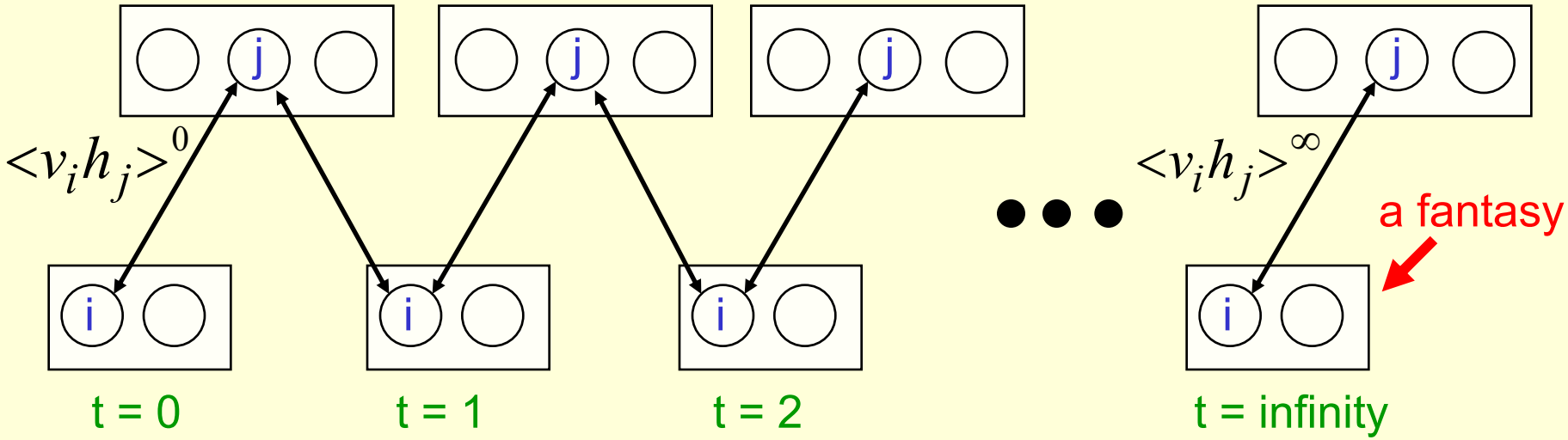
- We need to settle to the stationary distribution with each training vector clamped on the visible units.
- We need to settle to the stationary distribution with the visible units unclamped.
  - This is a highly multimodal distribution.
- The learning signal is the difference between two noisy statistics.
  - The difference is very noisy.

Restricted Boltzmann Machines

- We restrict the connectivity to make learning easier.
  - Only one layer of hidden units.
    - We will deal with more layers later
  - No connections between hidden units.
- In an RBM, the hidden units are conditionally independent given the visible states.
  - So we can quickly get an unbiased sample from the posterior distribution when given a data-vector.
  - This is a big advantage over directed belief nets



A picture of the maximum likelihood learning algorithm for an RBM

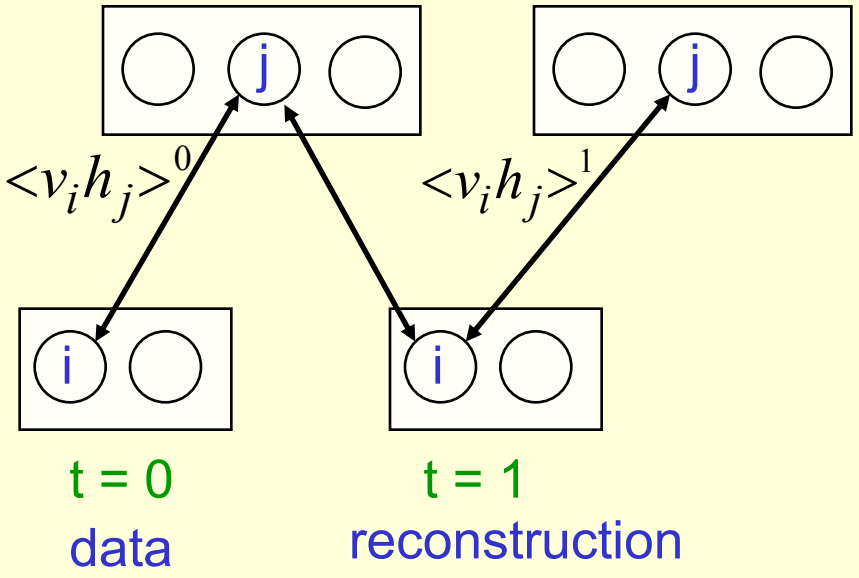


Start with a training vector on the visible units.

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

$$\frac{\partial \log p(v)}{\partial w_{ij}} = <v_i h_j>^0 - <v_i h_j>^\infty$$

A quick way to learn an RBM



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.

$$\Delta w_{ij} = \varepsilon \left(<v_i h_j>^0 - <v_i h_j>^1 \right)$$

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

It is approximately following the gradient of another objective function.

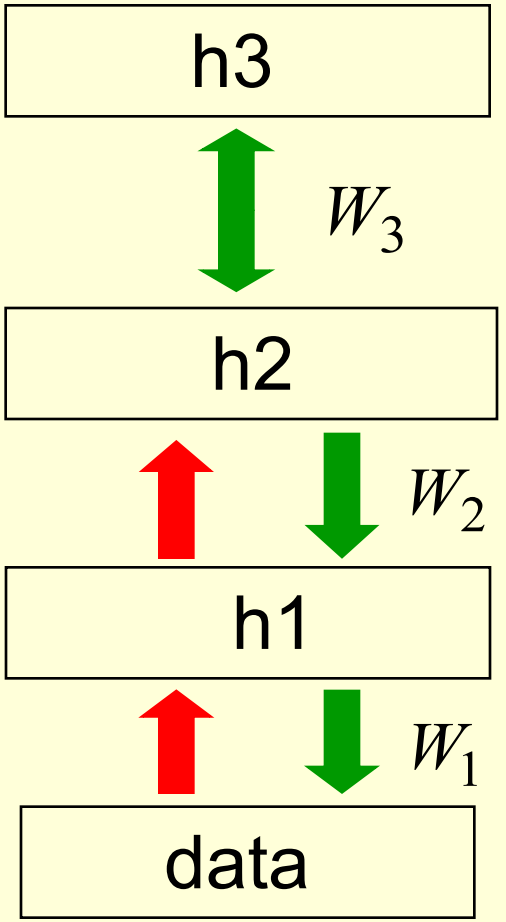
Training a deep network

- First train a layer of features that receive input directly from the pixels.
- Then treat the activations of the trained features as if they were pixels and learn features of features in a second hidden layer.
- It can be proved that each time we add another layer of features we improve a variational lower bound on the log probability of the training data.
  - The proof is slightly complicated.
  - But it is based on a neat equivalence between an RBM and a deep directed model (described later)

The generative model after learning 3 layers

- To generate data:
  1. Get an equilibrium sample from the top-level RBM by performing alternating Gibbs sampling.
  2. Perform a top-down pass to get states for all the other layers.

So the lower level bottom-up connections are not part of the generative model. They are just used for inference.



Why does greedy learning work?

The weights, W, in the bottom level RBM define p(v|h) and they also, indirectly, define p(h).

So we can express the RBM model as

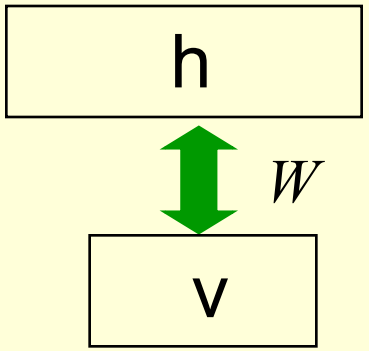
$$p(v) = \sum_h p(h)\ p(v \mid h)$$

If we leave p(v|h) alone and improve p(h), we will improve p(v).

To improve p(h), we need it to be a better model of the aggregated posterior distribution over hidden vectors produced by applying W to the data.

What does each RBM achieve?

- It divides the task of modeling the data into two tasks and leaves the second task to the next RBM
  - Task 1: Learn generative weights that can convert the posterior distribution over the hidden units into the data.
  - Task 2: Learn to model the posterior distribution over the hidden units that is produced by applying the transpose of the generative weights to the data
    - Task 2 is guaranteed to be easier (for the next RBM) than modeling the original data.



Fine-tuning for discrimination

- First learn one layer at a time greedily.
  - This does not require labeled data.

- Then add an output layer to the top and use backpropagation to fine-tune the model for better discrimination.
  - This overcomes many of the limitations of standard backpropagation.

Greedy pre-training makes backprop work better

- We do not start backpropagation until we already have sensible weights that already do well at the task.
  - So the initial gradients are sensible and backprop only needs to perform a local search.

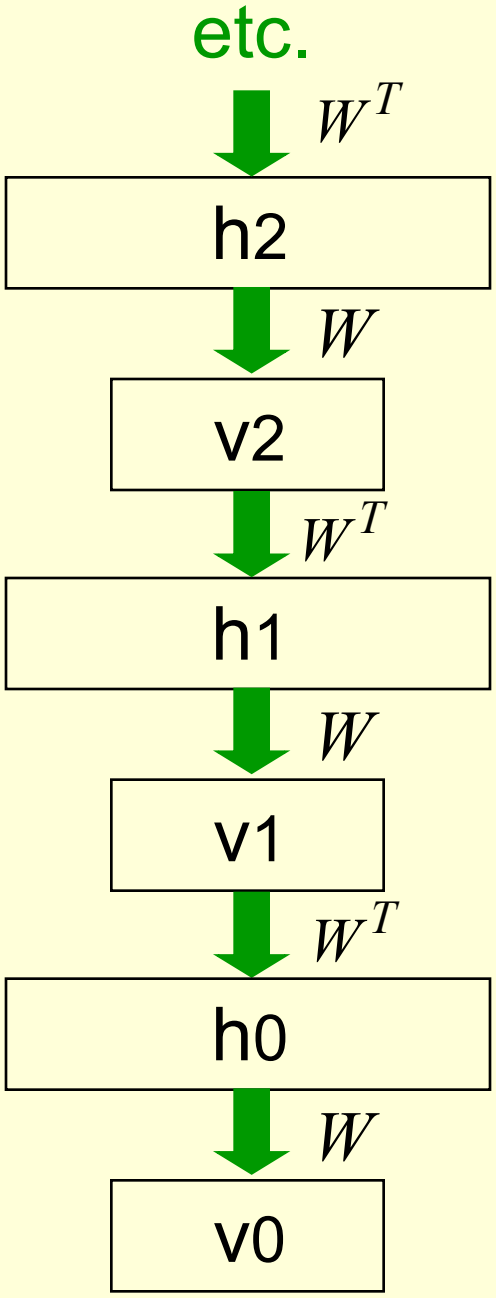
- Most of the information in the final weights comes from modeling the distribution of input vectors.
  - The precious information in the labels is only used for the final fine-tuning. It slightly modifies the features. It does not need to discover features.
  - This type of backpropagation works well even if most of the training data is unlabeled. The unlabeled data is still very useful for discovering good features.

Another view of why layer-by-layer learning works

- There is an unexpected equivalence between RBM's and directed networks with many layers that all use the same weights.
  - This equivalence also gives insight into why contrastive divergence learning works.

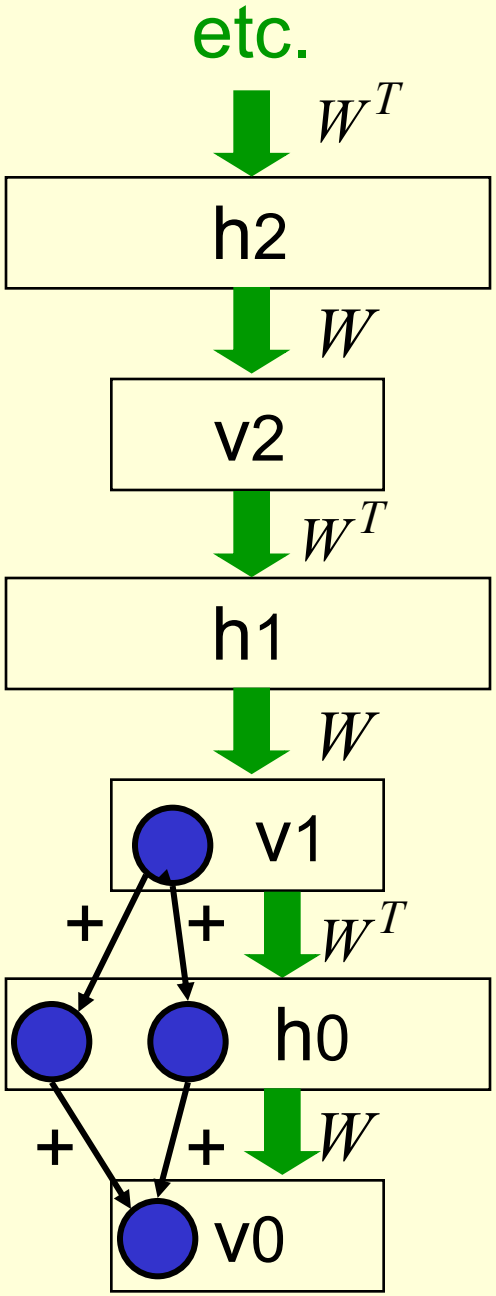
An infinite sigmoid belief net that is equivalent to an RBM

- The distribution generated by this infinite directed net with replicated weights is the equilibrium distribution for a compatible pair of conditional distributions: p(v|h) and p(h|v) that are both defined by W
  - A top-down pass of the directed net is exactly equivalent to letting a Restricted Boltzmann Machine settle to equilibrium.
  - So this infinite directed net defines the same distribution as an RBM.



Inference in a directed net with replicated weights

- The variables in h0 are conditionally independent given v0.
  - Inference is trivial. We just multiply v0 by W transpose.
  - The model above h0 implements a complementary prior.
  - Multiplying v0 by W transpose gives the product of the likelihood term and the prior term.
- Inference in the directed net is exactly equivalent to letting a Restricted Boltzmann Machine settle to equilibrium starting at the data.



- The learning rule for a sigmoid belief net is:

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

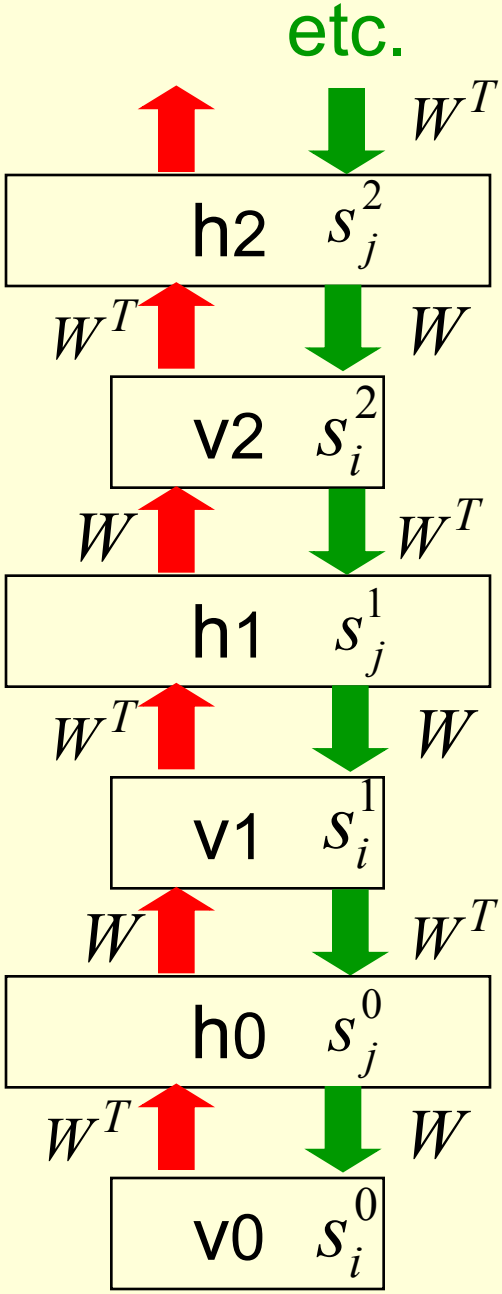
- With replicated weights this becomes:

$$s_j^0 (s_i^0 - s_i^1) +$$

$$s_i^1 (s_j^0 - s_j^1) +$$

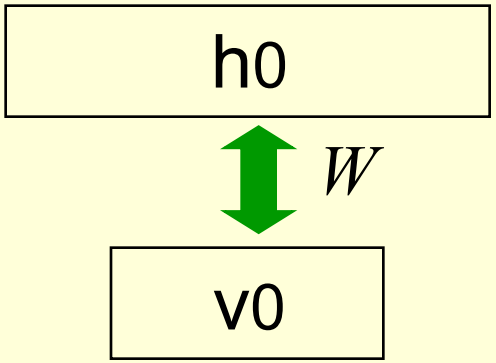
$$s_j^1 (s_i^1 - s_i^2) + \ldots$$

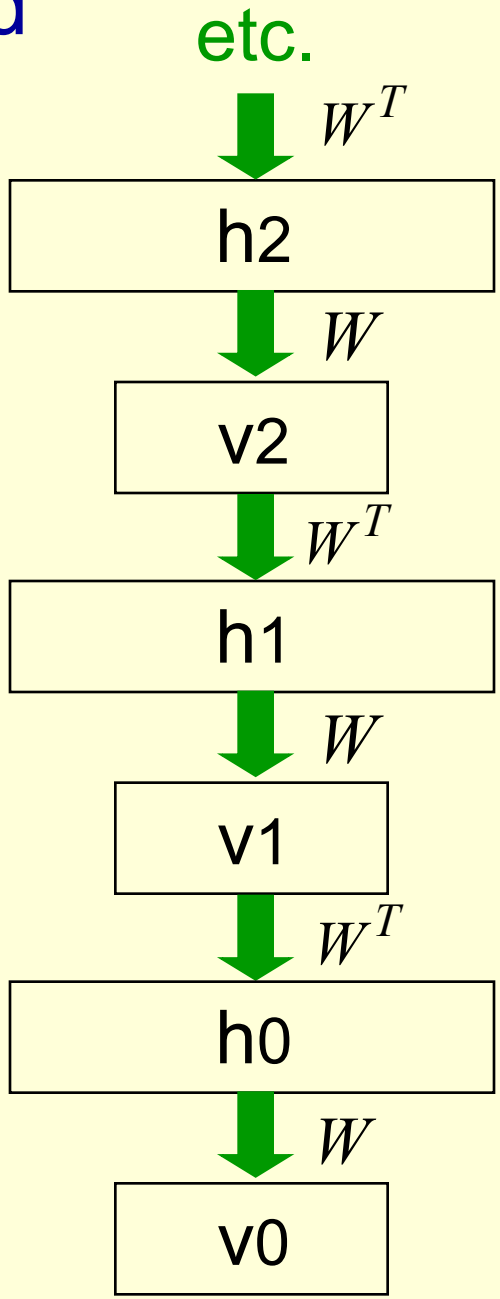
$$s_j^\infty s_i^\infty$$



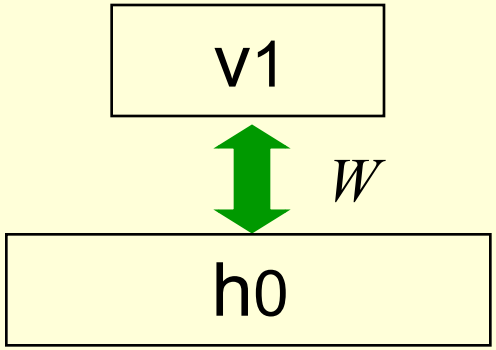
Learning a deep directed network

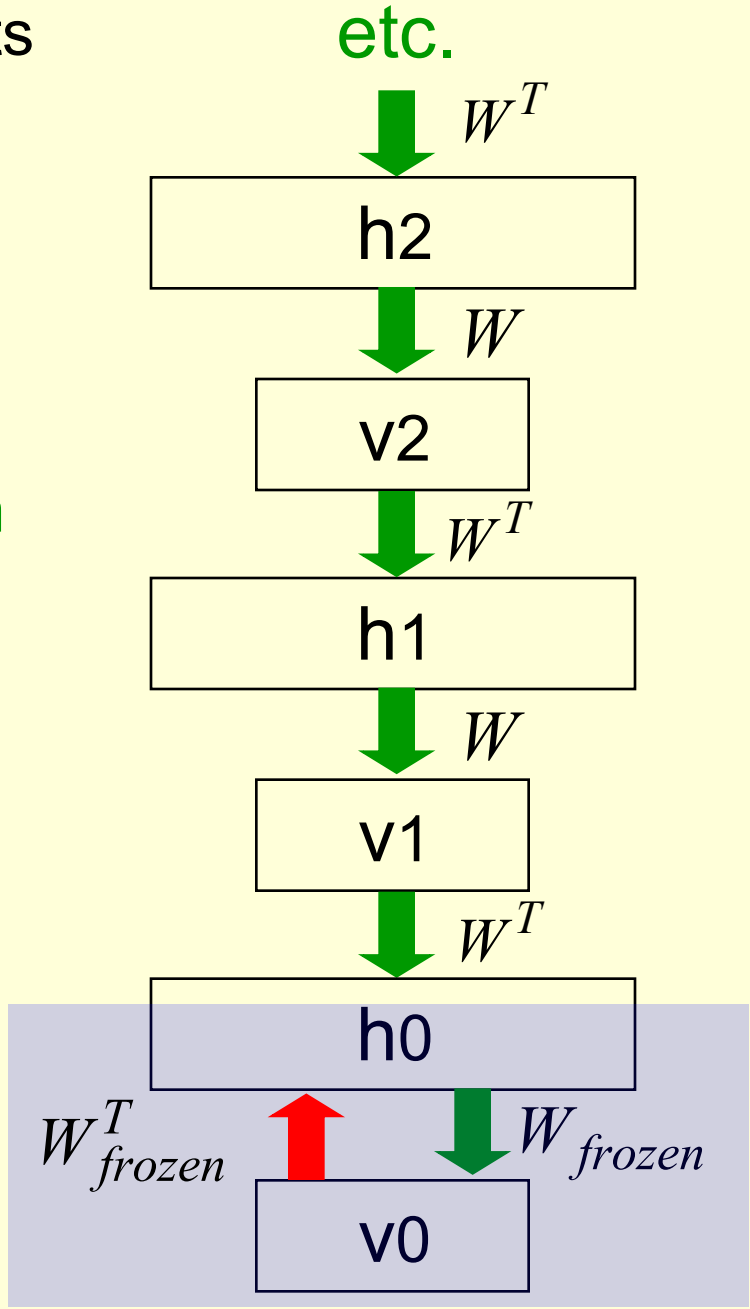
- First learn with all the weights tied
  - This is exactly equivalent to learning an RBM
  - Contrastive divergence learning is equivalent to ignoring the small derivatives contributed by the tied weights between deeper layers.





- Then freeze the first layer of weights in both directions and learn the remaining weights (still tied together).
  - This is equivalent to learning another RBM, using the aggregated posterior distribution of h0 as the data.





What happens when the weights in higher layers become different from the weights in the first layer?

- The higher layers no longer implement a complementary prior.
  - So performing inference using the frozen weights in the first layer is no longer correct.
  - Using this incorrect inference procedure gives a variational lower bound on the log probability of the data.
    - We lose by the slackness of the bound.
- The higher layers learn a prior that is closer to the aggregated posterior distribution of the first hidden layer.
  - This improves the network's model of the data.
    - Hinton, Osindero and Teh (2006) prove that this improvement is always bigger than the loss.

Summary

- Restricted Boltzmann Machines provide a simple way to learn a layer of features without any supervision.
  - Maximum likelihood learning is computationally expensive because of the normalization term, but contrastive divergence learning is fast and usually works well for learning good features.

- Many layers of representation can be learned by treating the hidden states of one RBM as the visible data for training the next RBM.

- This creates good generative models that can then be fine-tuned discriminatively.
  - In 2009, this led to a breakthrough in speech recognition

THE END