1

What kind of a Graphical Model is the Brain?

Geoffrey Hinton

in collaboration with

Simon Osindero and Yee-Whye Teh

2

Overview

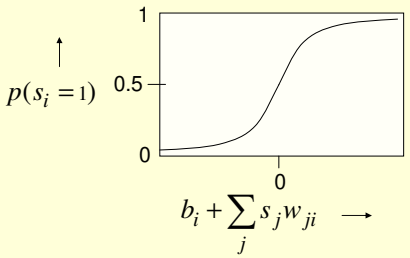
- We will combine two types of unsupervised neural net:
  - Undirected model = Boltzmann Machine
  - Directed model = Sigmoid Belief Net
- Boltzmann Machine learning is made efficient by restricting the connectivity & using contrastive divergence.
- Restricted Boltzmann Machines are shown to be equivalent to infinite Sigmoid Belief Nets with tied weights.
  - This equivalence suggests a novel way to learn deep directed belief nets one layer at a time.
  - This new method is fast and learns very good models, provided we do some fine-tuning afterwards
- We can now learn a really good generative model of the joint distribution of handwritten digit images and their labels.
  - It is better at recognizing handwritten digits than discriminative methods like SVM's or backpropagation.

3

Stochastic binary neurons

- These have a state of 1 or 0 which is a stochastic function of the neuron's bias, b, and the input it receives from other neurons.

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



4

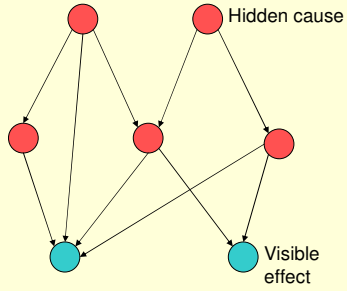
Two types of unsupervised neural network

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

5

Sigmoid Belief Nets

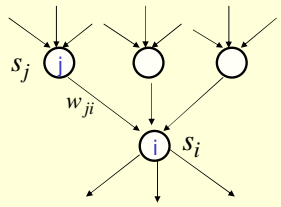
- It is easy to generate an unbiased example at the leaf nodes.
- It is typically hard to compute the posterior distribution over all possible configurations of hidden causes.
- Given samples from the posterior, it is easy to learn the local interactions



6

The learning rule for sigmoid belief nets

- Suppose we could observe the states of all the hidden units when the net was generating an observed data-vector.
  - This is equivalent to getting samples from the posterior distribution over hidden configurations given the observed datavactor.
- For each node, it is easy to maximize the log probability of its observed state given the observed states of its parents.



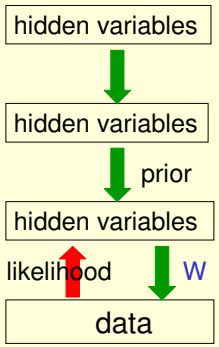
$$\Delta w_{ji} = \varepsilon\, s_j (s_i - p_i)$$

probability of i turning on given the states of its parents

7

Why learning is hard in a sigmoid belief net.

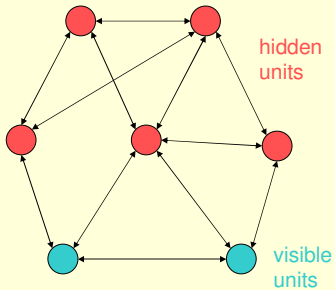
- To learn W, we need the posterior distribution in the first hidden layer.
- Problem 1: The posterior is typically intractable because of "explaining away".
- Problem 2: The posterior depends on the prior created by higher layers as well as the likelihood.
  - So to learn W, we need to know the weights 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!



8

How a Boltzmann Machine models data

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



9

The Energy of a joint configuration

binary state of unit i in joint configuration v,h

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

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

10

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.

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

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.

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

11

A very surprising fact

- Everything that one weight needs to know about the other weights and the data in order to do maximum likelihood learning 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

12

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.

13

Four reasons why learning is impractical in Boltzmann Machines

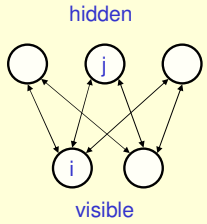
- If there are many hidden layers, it can take a long time to reach thermal equilibrium when a data-vector is clamped on the visible units.
- It takes even longer to reach thermal equilibrium in the "negative" phase when the visible units are unclamped.
  - The unconstrained energy surface needs to be highly multimodal to model the data.
- The learning signal is the difference of two sampled correlations which is very noisy.
- Many weight updates are required.

14

Restricted Boltzmann Machines

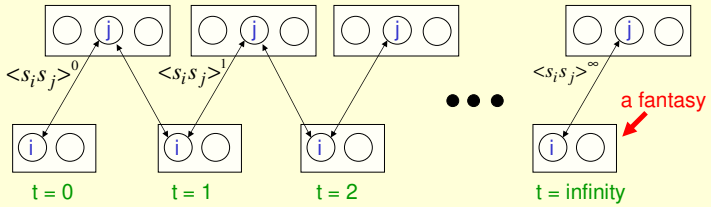
- We restrict the connectivity to make inference and learning easier.
  - Only one layer of hidden units.
  - No connections between hidden units.
- In an RBM, the hidden units are conditionally independent given the visible states. It only takes one step to reach thermal equilibrium when the visible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

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



15

A picture of the Boltzmann machine learning algorithm for an RBM



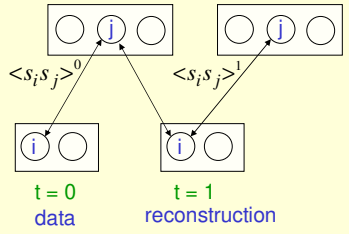
Start with a training vector on the visible units.

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

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

16

Contrastive divergence learning: 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.

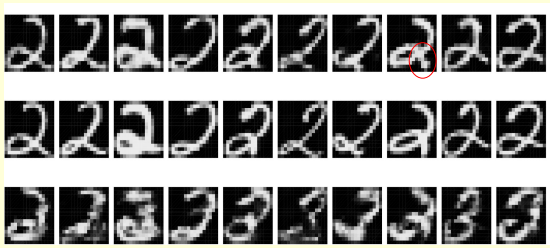
$$\Delta w_{ij} = \varepsilon \left(<s_i s_j>^0 - <s_i s_j>^1 \right)$$

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

When we consider infinite directed nets it will be easy to see why it works.

Using an RBM to learn a model of a digit class

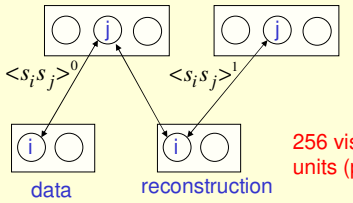
17



Reconstructions by model trained on 2's

Data

Reconstructions by model trained on 3's

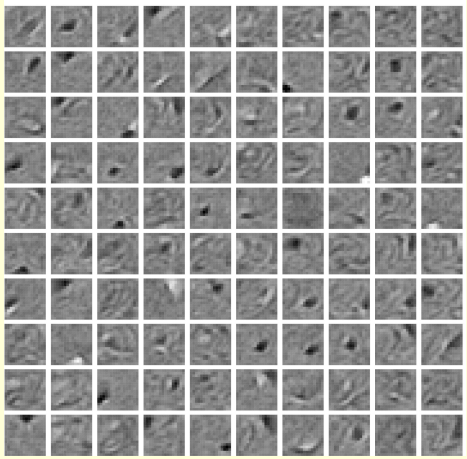


100 hidden units (features)

256 visible units (pixels)

The weights learned by the 100 hidden units

18



Each hidden unit is connected to all the pixels, but it learns mostly local features.

By adding together a subset of these features we can reconstruct any 2 very accurately

White = positive weight

Black = negative weight

A surprising relationship between Boltzmann Machines and Sigmoid Belief Nets

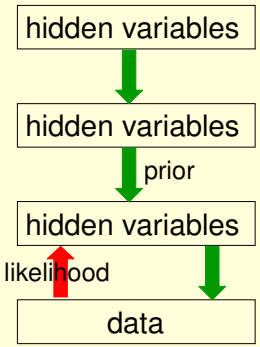
19

- Directed and undirected models seem very different.
- But there is a special type of multi-layer directed model in which it is easy to infer the posterior distribution over the hidden units because it has complementary priors.
- This special type of directed model is equivalent to an undirected model.
  - At first, this equivalence just seems like a neat trick
  - But it leads to a very effective new learning algorithm that allows multilayer directed nets to be learned one layer at a time.
    - The new learning algorithm resembles boosting with each layer being like a weak learner.

Using complementary priors to eliminate explaining away

20

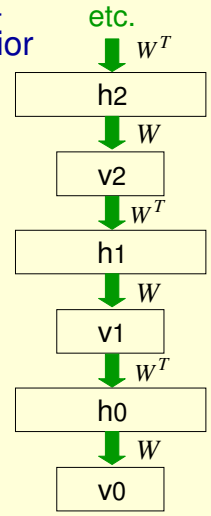
- A "complementary" prior is defined as one that exactly cancels the correlations created by explaining away. So the posterior factors.
  - Under what conditions do complementary priors exist?
  - Complementary priors do not exist in general:
    - Parameter counting shows that complementary priors cannot 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

21

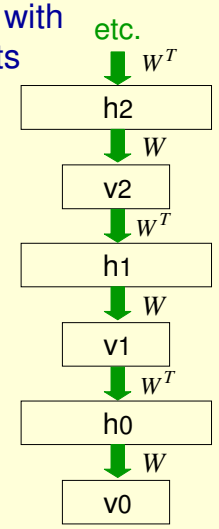
- The distribution generated by this infinite DAG with replicated weights is the equilibrium distribution for a compatible pair of conditional distributions: p(v|h) and p(h|v).
  - An ancestral pass of the DAG is exactly equivalent to letting a Restricted Boltzmann Machine settle to equilibrium.
  -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



Inference in a DAG with replicated weights

22

- 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.

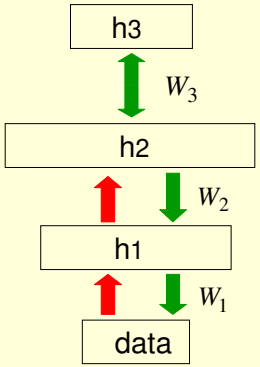


The generative model

23

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

  So the lower level bottom-up connections are not part of the generative model



Learning by dividing and conquering

24

- Re-weighting the data: In boosting, we learn a sequence of simple models. After learning each model, we re-weight the data so that the next model learns to deal with the cases that the previous models found difficult.
  - There is a nice guarantee that the overall model gets better.
- Projecting the data: In PCA, we find the leading eigenvector and then project the data into the orthogonal subspace.
- Distorting the data: In projection pursuit, we find a non-Gaussian direction and then distort the data so that it is Gaussian along this direction.

25

Another way to divide and conquer

- Re-representing the data: Each time the base learner is called, it passes a transformed version of the data to the next learner.
  - Can we learn a deep, dense DAG one layer at a time, starting at the bottom, and still guarantee that learning each layer improves the overall model of the training data?
    - This seems very unlikely. Surely we need to know the weights in higher layers to learn lower layers?

26

- The learning rule for a logistic DAG is:

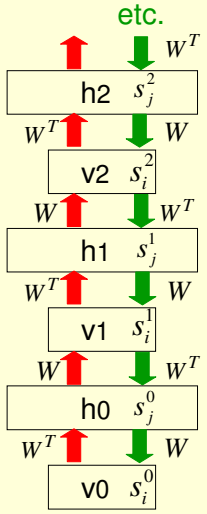
$$\Delta w_{ij} \propto s_j (s_i - \hat{s}_i)$$

- With replicated weights this becomes:

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

The derivatives for the recognition weights are zero.

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



27

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.

28

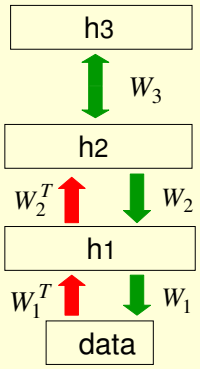
Multilayer contrastive divergence

- Start by learning one hidden layer.
- Then re-present the data as the activities of the hidden units.
  - The same learning algorithm can now be applied to the re-presented data.
- Can we prove that each step of this greedy learning improves the log probability of the data under the overall model?
  - What is the overall model?

29

A simplified version with all hidden layers the same size

- The RBM at the top can be viewed as shorthand for an infinite directed net.
- When learning W_1 we can view the model in two quite different ways:
  - The model is an RBM composed of the data layer and h1.
  - The model is an infinite DAG with tied weights.
- After learning W_1 we untie it from the other weight matrices.
- We then learn W_2 which is still tied to all the matrices above it.



30

Why the hidden configurations should be treated as data when learning the next layer of weights

- After learning the first layer of weights:

$$\log p(x) \geq -\langle energy(x) \rangle + entropy(h1|x)$$
$$\geq \sum_{\alpha} p(h1=\alpha \mid x)\left[\log p(h1=\alpha) + \log p(x \mid h1=\alpha)\right] + entropy$$

- If we freeze the generative weights that define the likelihood term and the recognition weights that define the distribution over hidden configurations, we get:

$$\log p(x) \geq \sum_{\alpha} p(h1=\alpha \mid x)\left[\log p(h1=\alpha)\right] + constant$$

- Maximizing the RHS is equivalent to maximizing the log prob of "data" α that occurs with probability $p(h1=\alpha \mid x)$

31

Why greedy learning works

- Each time we learn a new layer, the inference at the layer below becomes incorrect, but the variational bound on the log prob of the data improves.
- Since the bound starts as an equality, learning a new layer never decreases the log prob of the data, provided we start the learning from the tied weights that implement the complementary prior.
- Now that we have a guarantee we can loosen the restrictions and still feel confident.
  - Allow layers to vary in size.
  - Do not start the learning at each layer from the weights in the layer below.

32

Back-fitting

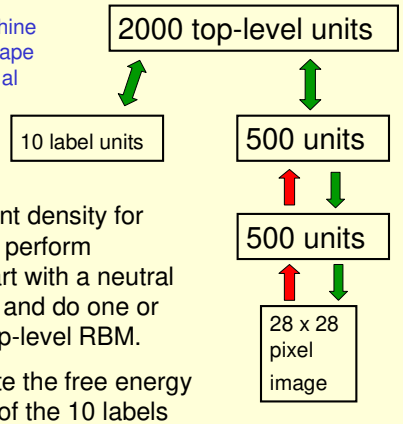
- After we have learned all the layers greedily, the weights in the lower layers will no longer be optimal. We can improve them in two ways:
  - Untie the recognition weights from the generative weights and learn recognition weights that take into account the non-complementary prior implemented by the weights in higher layers.
  - Improve the generative weights to take into account the non-complementary priors implemented by the weights in higher layers.
- What algorithm should we use for fine-tuning the weights that are learned greedily?
  - We use a contrastive version of the "wake-sleep" algorithm. This is explained in the written paper. It will not be described in the talk.

33

A neural network model of digit recognition

The top two layers form a restricted Boltzmann machine whose free energy landscape models the low dimensional manifolds of the digits.

The valleys have names:



The model learns a joint density for labels and images. To perform recognition we can start with a neutral state of the label units and do one or two iterations of the top-level RBM.

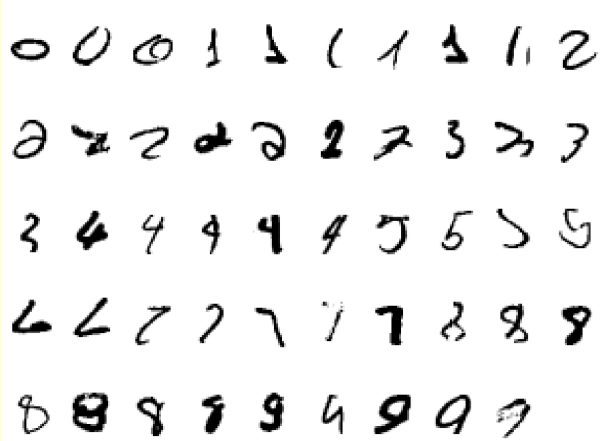
Or we can just compute the free energy of the RBM with each of the 10 labels

34

Samples generated by running the top-level RBM with one label clamped. There are 1000 iterations of alternating Gibbs sampling between samples.

35

Examples of correctly recognized MNIST test digits (the 49 closest calls)



36

How well does it discriminate on MNIST test set with no extra information about geometric distortions?

- Up-down net with RBM pre-training + CD10 1.25%
- SVM (Decoste & Scholkopf) 1.4%
- Backprop with 1000 hiddens (Platt) 1.5%
- Backprop with 500 -->300 hiddens 1.5%

- Separate hierarchy of RBM's per class 1.7%
- Learned motor program extraction ~1.8%
- K-Nearest Neighbor ~ 3.3%

- Its better than backprop and much more neurally plausible because the neurons only need to send one kind of signal, and the teacher can be another sensory input.

37

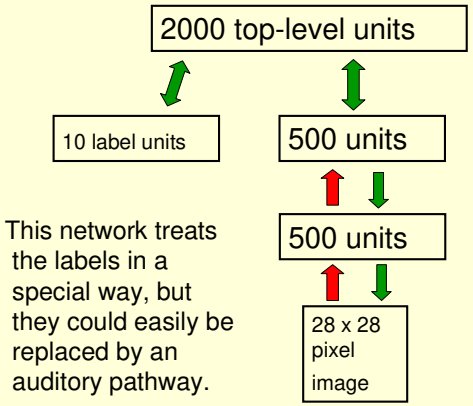
All 125 errors

38

Samples generated by running top-level RBM with one label clamped. Initialized by an up-pass from a random binary image. 20 iterations between samples.

39

Learning with realistic labels

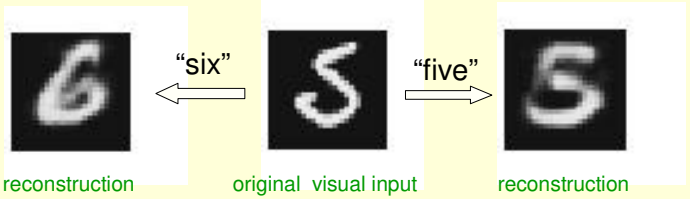


This network treats the labels in a special way, but they could easily be replaced by an auditory pathway.

40

Learning with auditory labels

- Alex Kaganov replaced the class labels by binarized cepstral spectrograms of many different male speakers saying digits.
- The auditory pathway then had multiple layers, just like the visual pathway. The auditory and visual inputs shared the top level layer.
- After learning, he showed it a visually ambiguous digit and then reconstructed the visual input from the representation that the top-level associative memory had settled on after 10 iterations.



41

A different way to capture low-dimensional manifolds

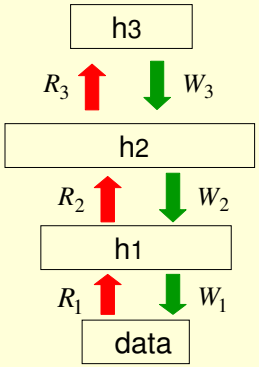
- Instead of trying to explicitly extract the coordinates of a datapoint on the manifold, map the datapoint to an energy valley in a high-dimensional space.
- The learned energy function in the high-dimensional space restricts the available configurations to a low-dimensional manifold.
  - We do not need to know the manifold dimensionality in advance and it can vary along the manifold.
  - We do not need to know the number of manifolds.
  - Different manifolds can share common structure.
- But we cannot create the right energy valleys by direct interactions between pixels.
  - So learn a multilayer non-linear mapping between the data and a high-dimensional latent space in which we can construct the right valleys.

42

THE END

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

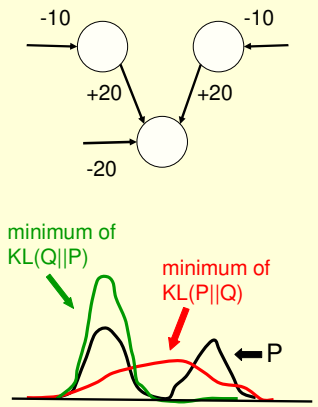
- The recognition weights are trained to invert the generative model in parts of the space where there is no data.
  - This is wasteful.
- The recognition weights follow the gradient of the wrong divergence. They minimize KL(P||Q) but the variational bound requires minimization of KL(Q||P).
  - This leads to incorrect mode-averaging
- The posterior over the top hidden layer is very far from independent because the independent prior cannot eliminate explaining away effects.

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

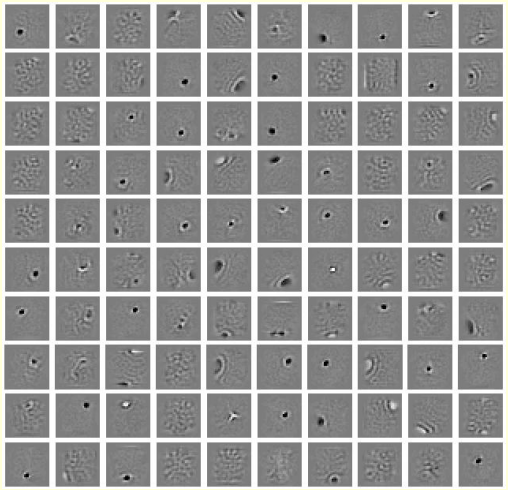
- Replace the top layer of the DAG by an RBM
  - This eliminates bad variational approximations caused by top-level units that are independent in the prior.
  - It is nice to have an associative memory at the top.
- Replace the ancestral pass in the sleep phase by a top-down pass starting with the state 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.

Mode averaging

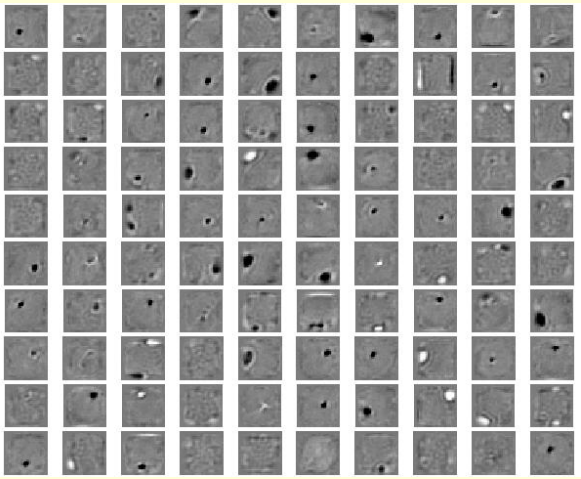
- If we generate from the model, half the instances of a 1 at the data layer will be caused by a (1,0) at the hidden layer and half will be caused by a (0,1).
  - So the recognition weights will learn to produce (0.5,0.5)
  - This represents a distribution that puts half its mass on very improbable hidden configurations.
- Its much better to just pick one mode and pay one bit.



The receptive fields of the first hidden layer



The generative fields of the first hidden layer



Independence relationships of hidden variables in three types of model

	Causal model	Product of experts	Square ICA
Hidden states unconditional on data	independent (generation is easy)	dependent (rejecting away)	independent (by definition)
Hidden states conditional on data	dependent (explaining away)	independent (inference is easy)	independent (the posterior collapses to a single point)

We now have a way to reduce this dependency so that variational inference works