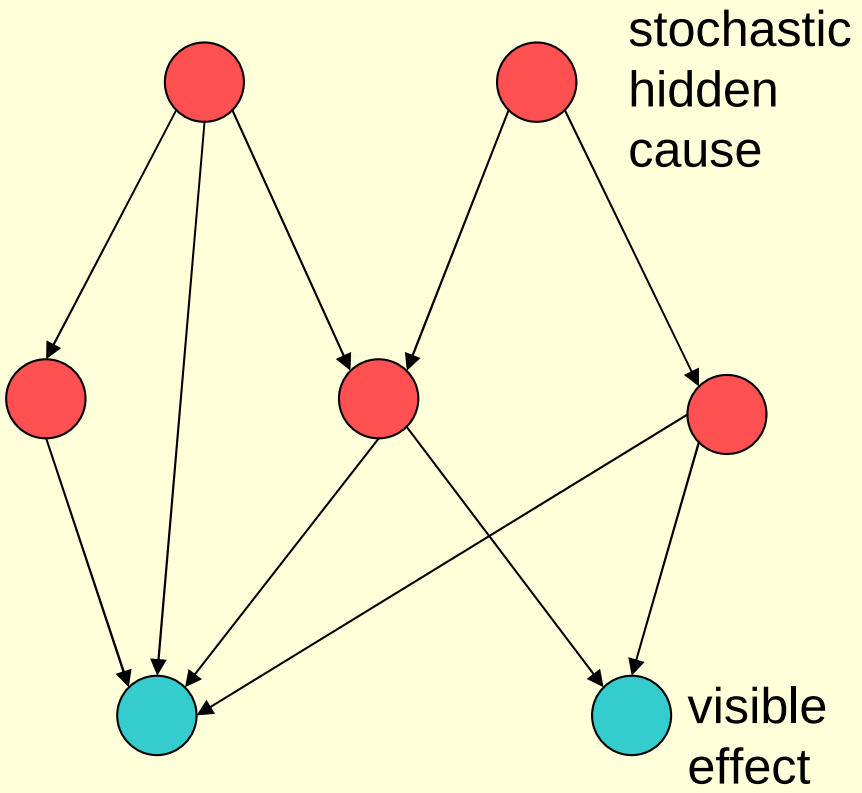
An efficient way to learn deep generative models

Geoffrey Hinton
Canadian Institute for Advanced Research
&
Department of Computer Science
University of Toronto

Joint work with: Ruslan Salakhutdinov,
Yee-Whye Teh, Simon Osindero, Ilya Sutskever,
Graham Taylor, Andriy Mnih

Belief Nets

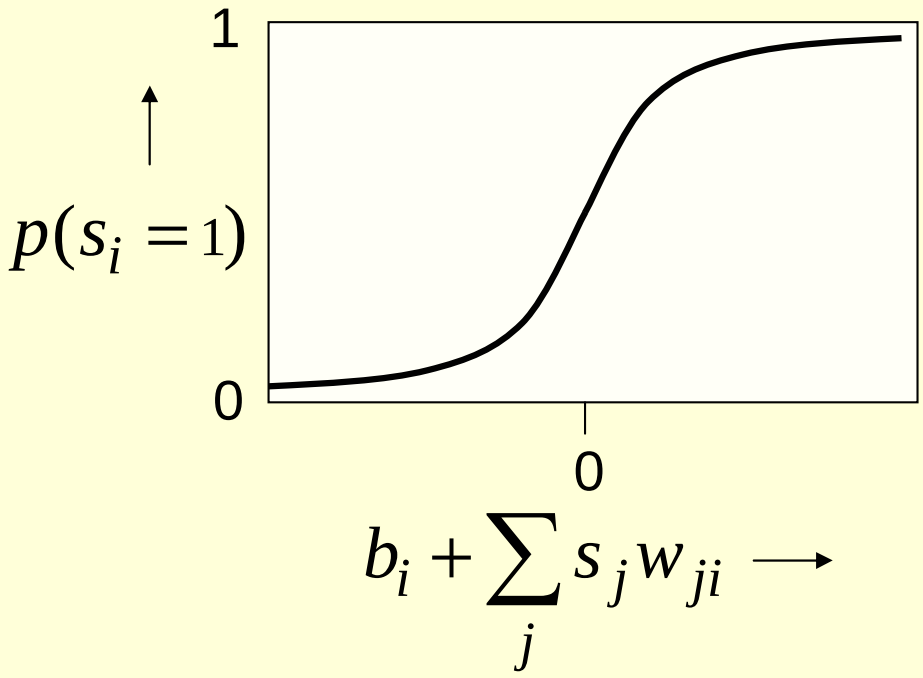
- A belief net is a directed acyclic graph composed of stochastic variables.
- We get to observe some of the variables and we would like to solve two problems:
- The inference problem: Infer the states of the unobserved variables.
- The learning problem: Adjust the interactions between variables to make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



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

Stochastic binary neurons

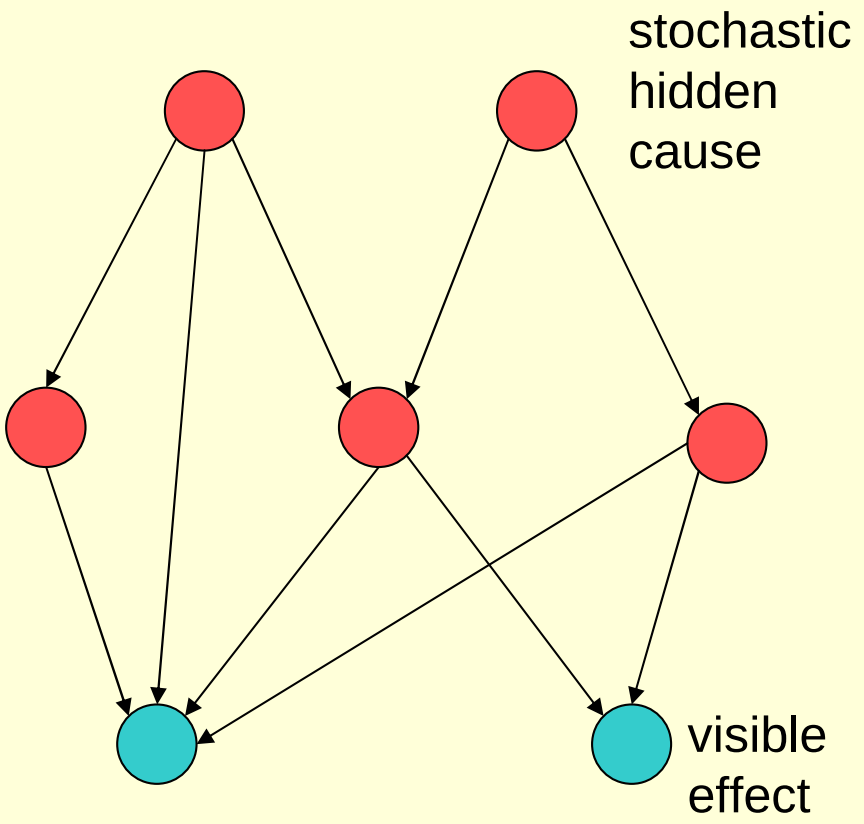
- These have a state of 1 or 0.
- The probability of turning on is determined by the weighted input from other neurons (plus a bias)



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

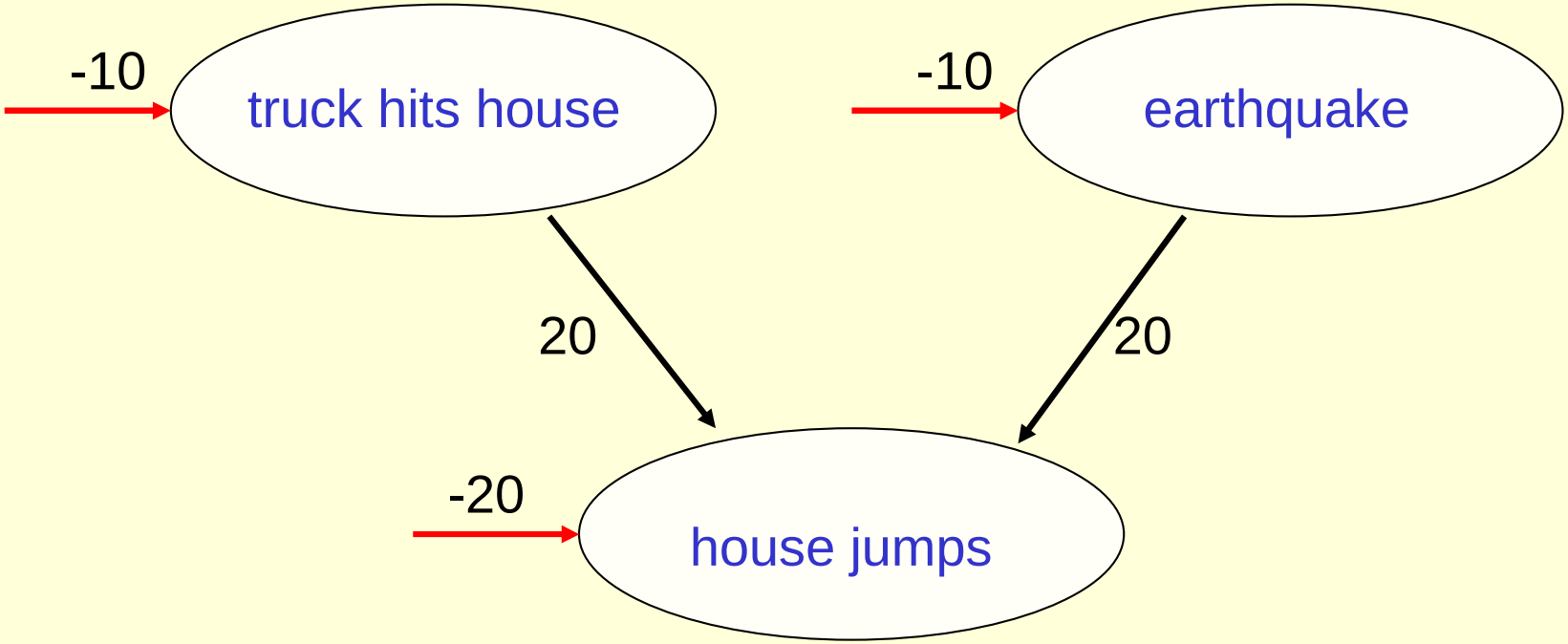
Learning Belief Nets

- 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?



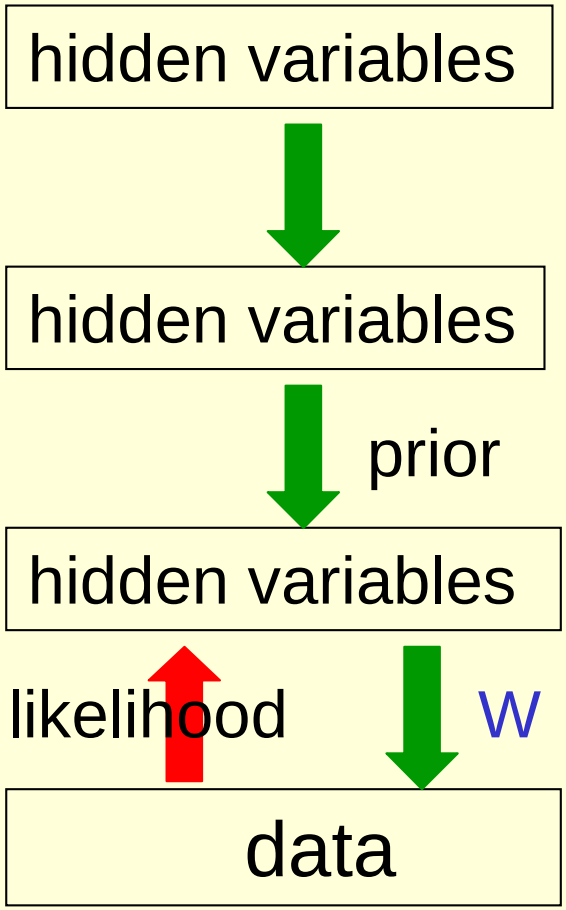
Explaining away (Judea Pearl)

- 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 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!



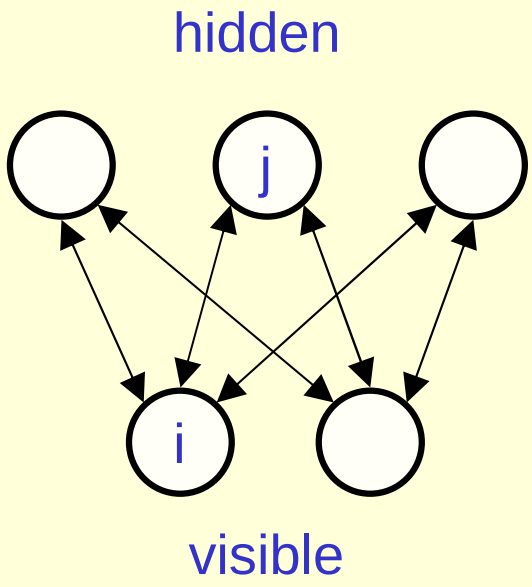
Two types of generative neural network

- 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).
  - If we restrict the connectivity in a special way, it is easy to learn a Boltzmann machine.

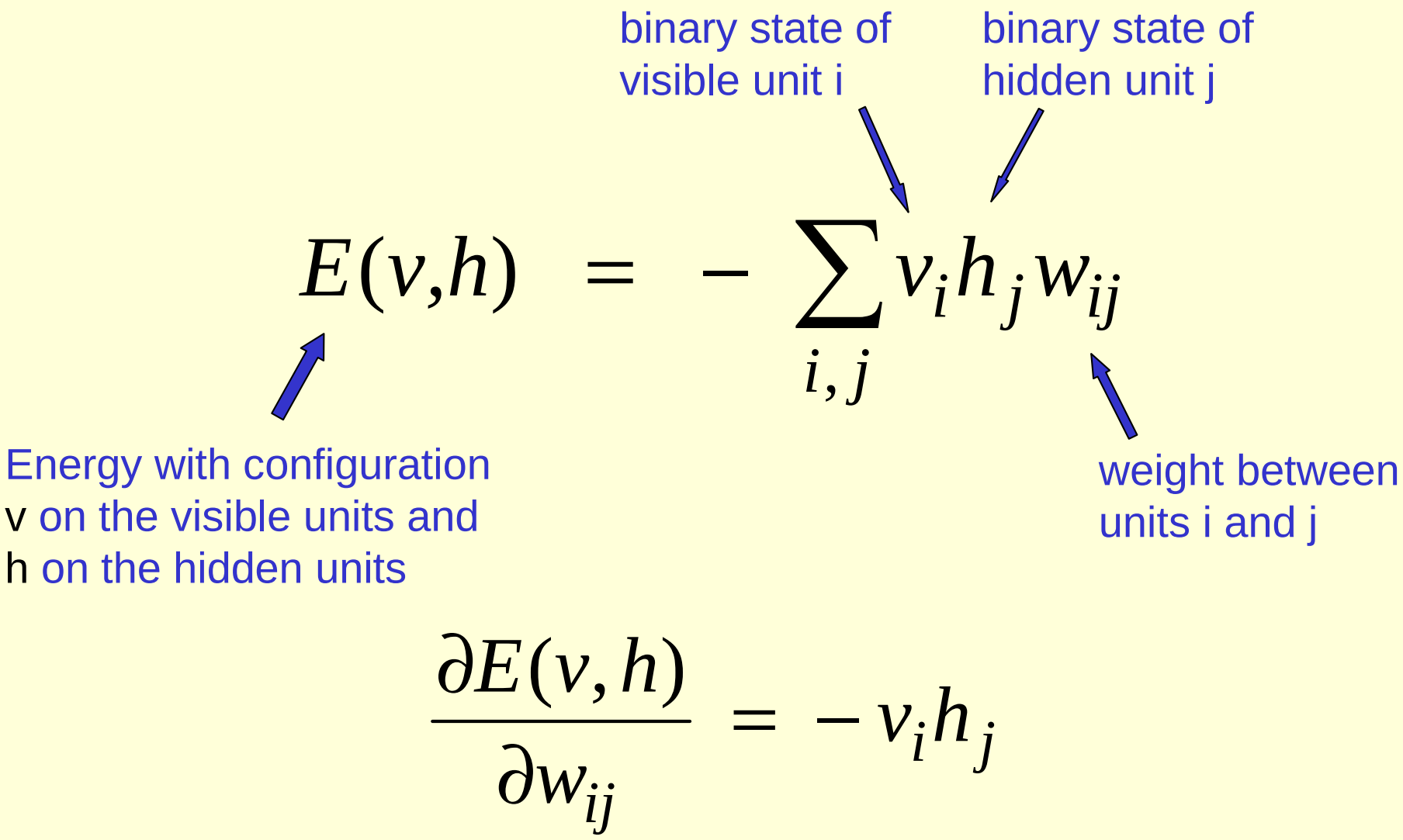
Restricted Boltzmann Machines

- 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



The Energy of a joint configuration

(ignoring terms to do with biases)



$$\frac{\partial E(v,h)}{\partial w_{ij}} = -v_i h_j$$

Weights → Energies → Probabilities

- Each possible joint configuration of the visible and hidden units has an energy
  - The energy is determined by the weights and biases (as in a Hopfield net).
- The energy of a joint configuration of the visible and hidden units determines its probability:

$$p(v,h) \propto e^{-E(v,h)}$$

- The probability of a configuration over the visible units is found by summing the probabilities of all the joint configurations that contain it.

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.

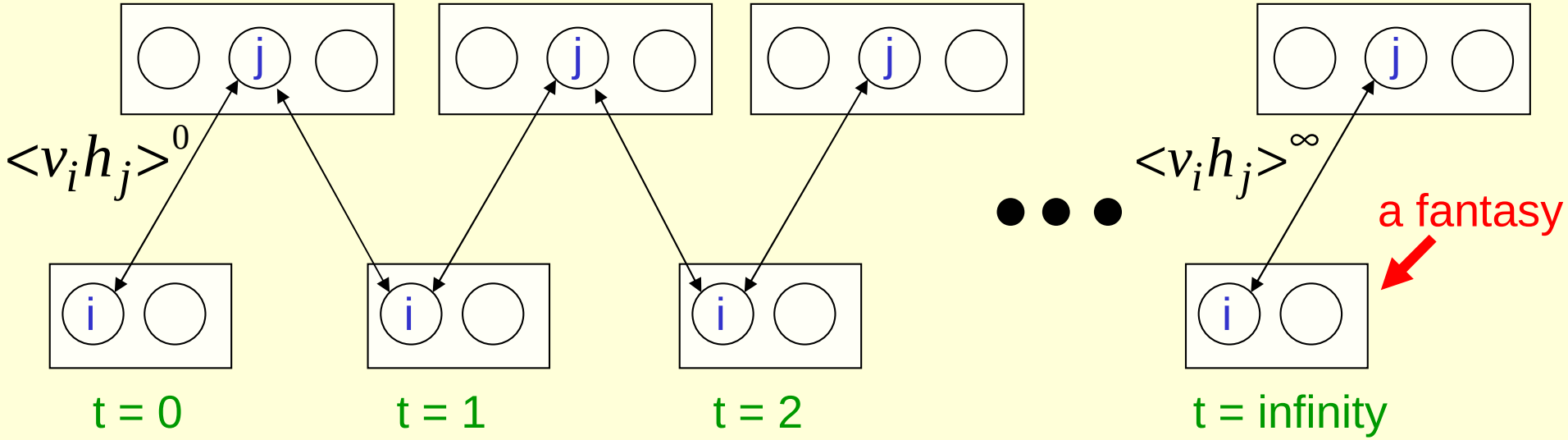
$$p(v,h) = \frac{e^{-E(v,h)}}{\sum_{u,g} e^{-E(u,g)}}$$

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.

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

A picture of the maximum likelihood learning algorithm for an RBM

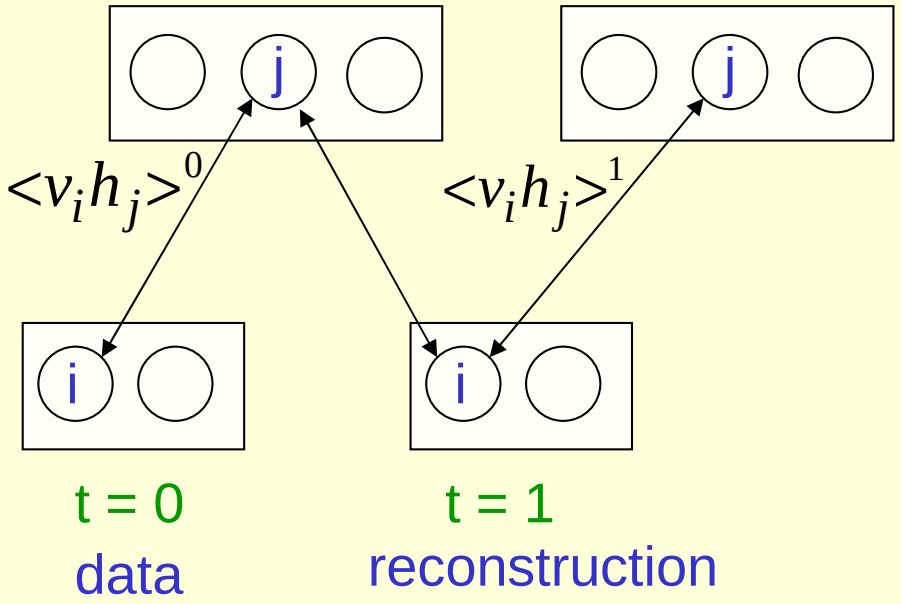


Start with a training vector on the visible units.

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”.

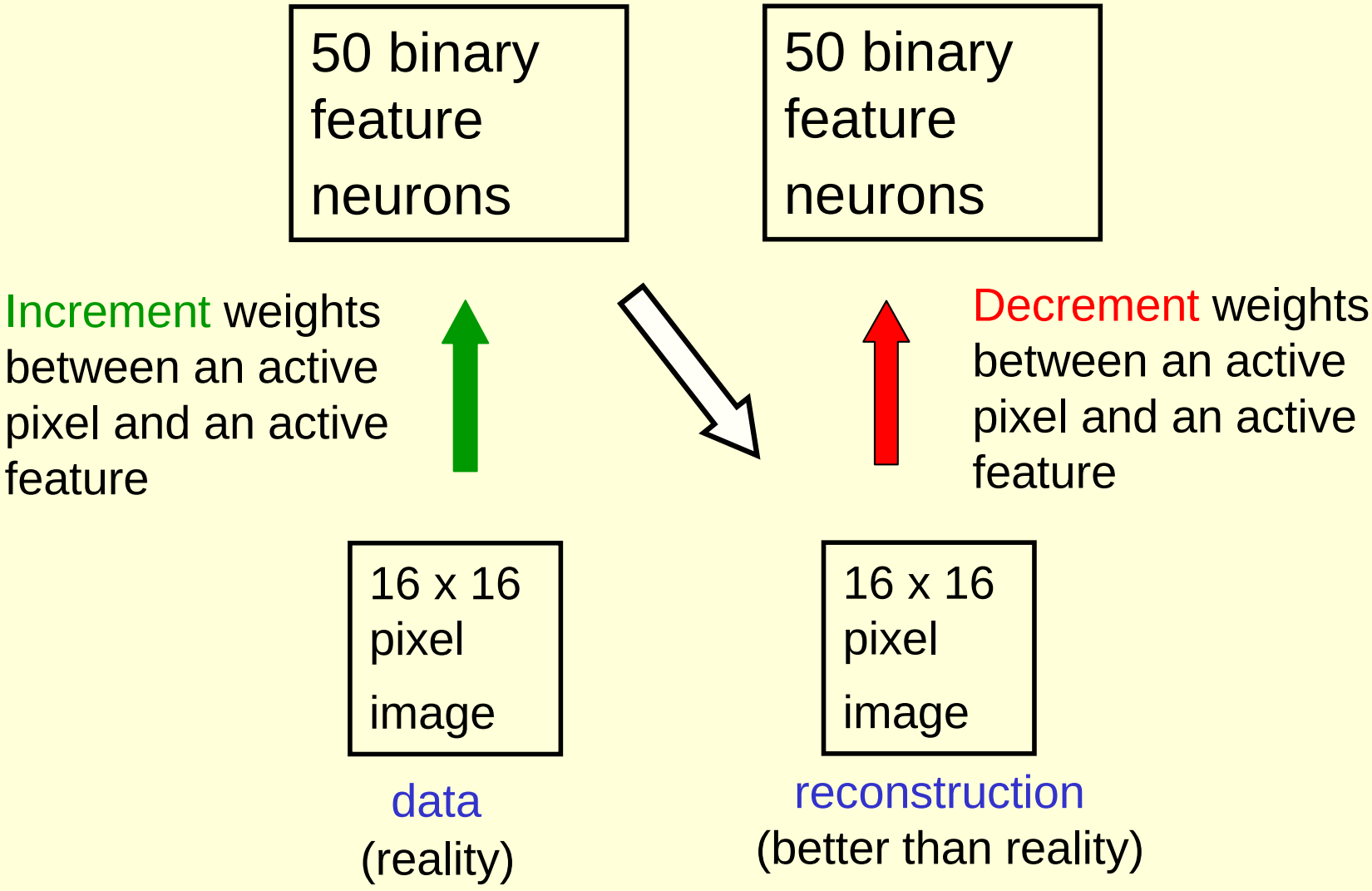
Update the hidden units again.

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

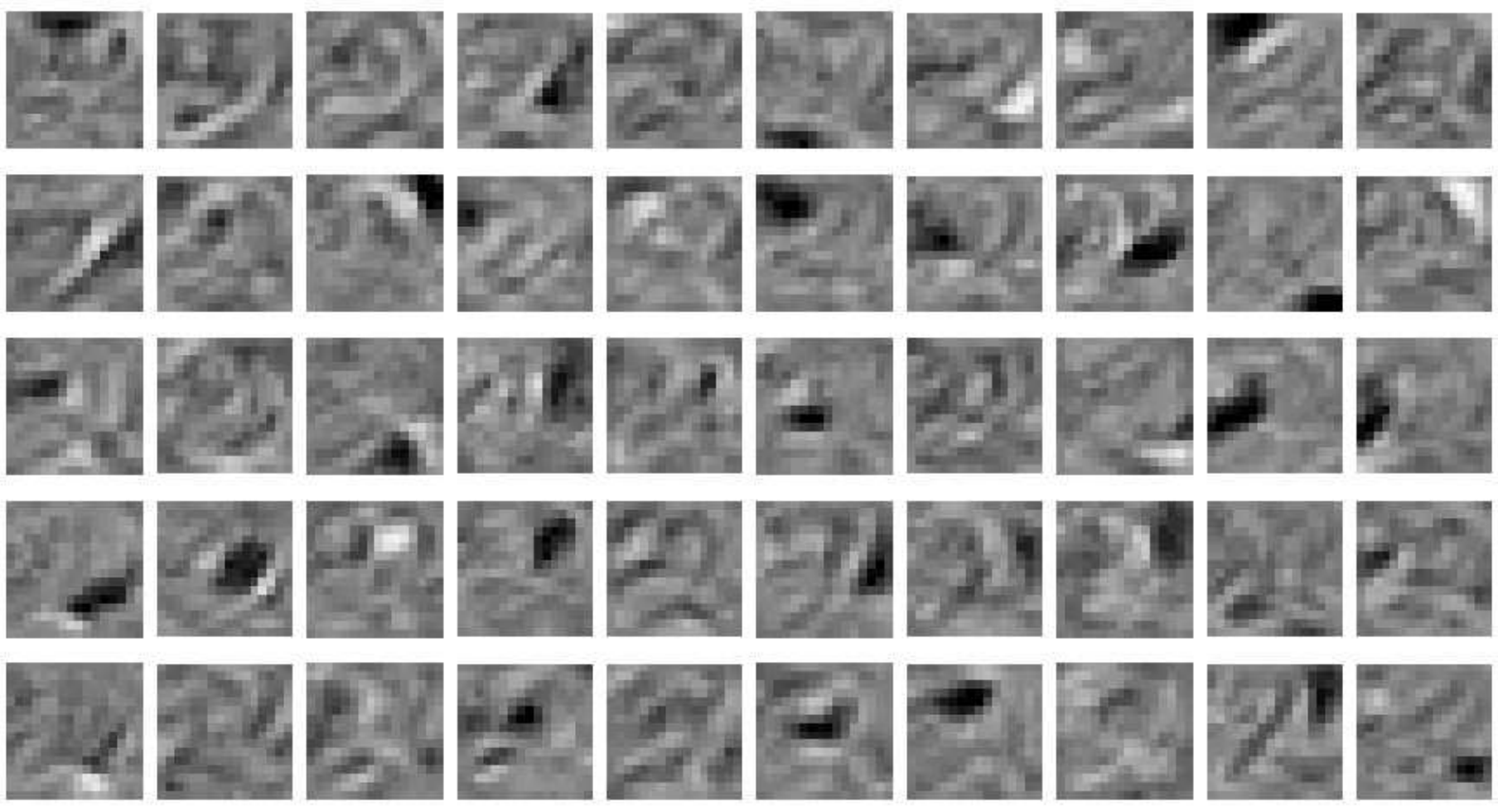
This is not following the gradient of the log likelihood. But it works well.

It is approximately following the gradient of another objective function.

How to learn a set of features that are good for reconstructing images of the digit 2



The final 50 x 256 weights

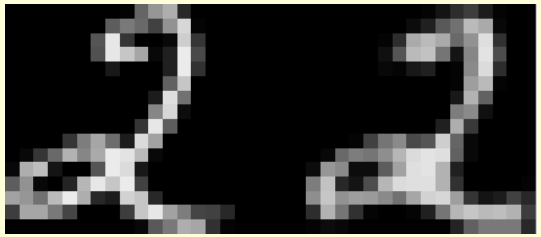


Each neuron grabs a different feature.

How well can we reconstruct the digit images from the binary feature activations?

Data

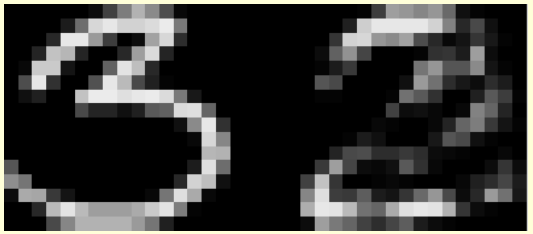
Reconstruction from activated binary features



New test images from the digit class that the model was trained on

Data

Reconstruction from activated binary features



Images from an unfamiliar digit class
(the network tries to see every image as a 2)

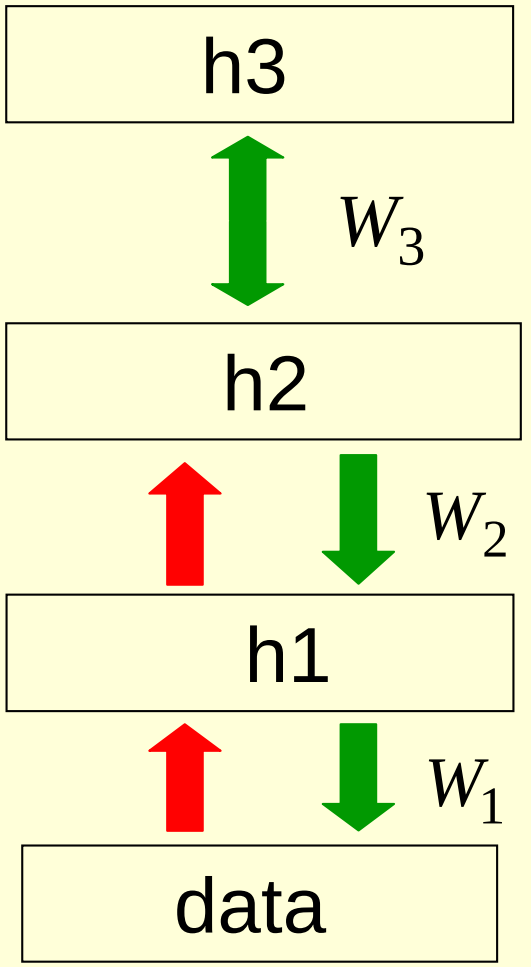
Training a deep network

- 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 get a better model of the set of training images.
  - The proof is complicated. It uses variational free energy, a method that physicists use for analyzing non-equilibrium systems.
  - But it is based on a neat equivalence (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

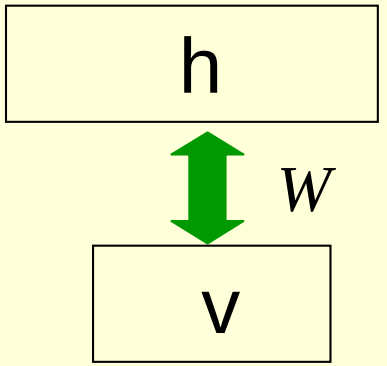
$$p(v) = \sum_h p(h)\ p(v \mid h)$$

If we leave p(v|h) alone and build a better model of p(h), we will improve p(v).

We need a better model of the aggregated posterior distribution over hidden vectors produced by applying W to the data.

What does each RBM achieve?

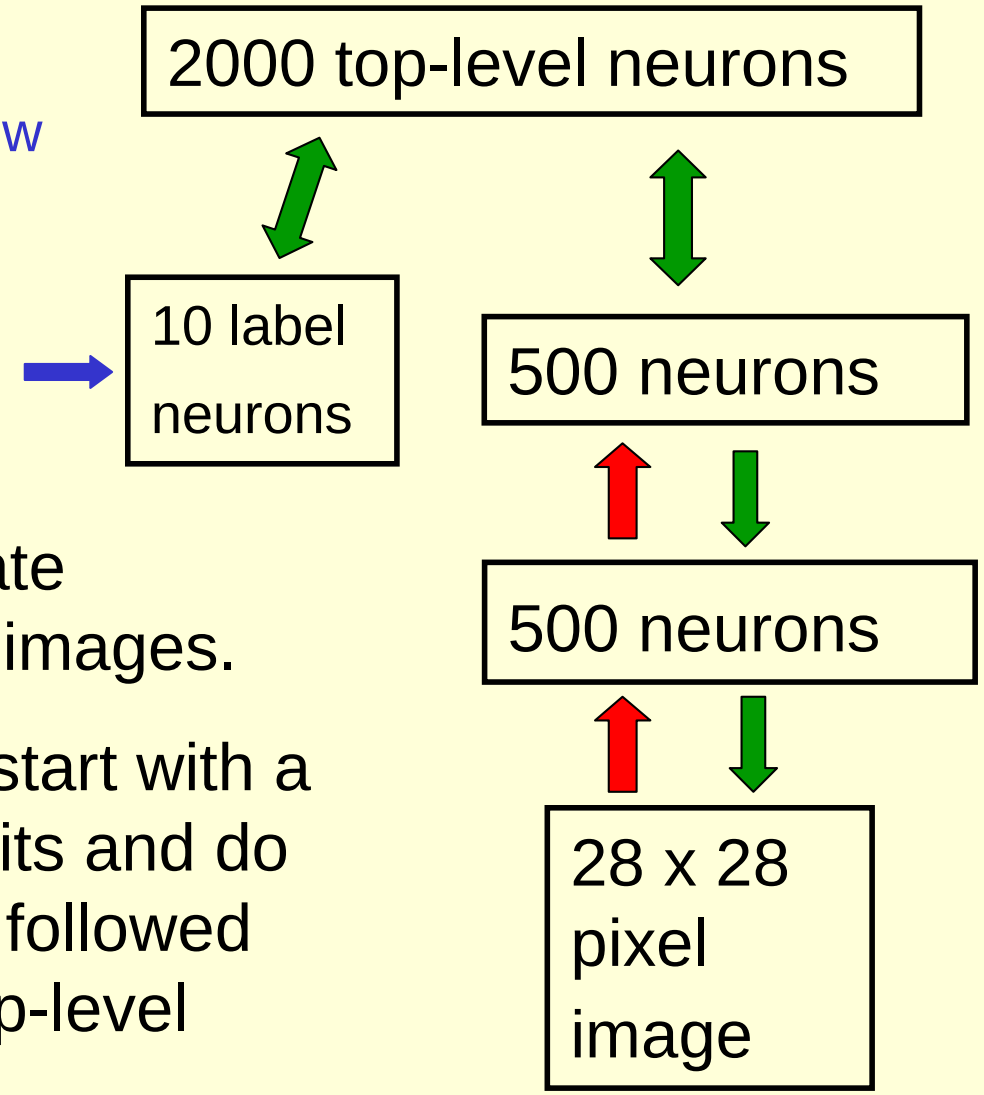
- It divides the task of modeling the data into two tasks and leaves the second task to the next RBM
  - Task 1: Learn generative weights that can convert the posterior distribution over the hidden units into the data.
  - Task 2: Learn to model the posterior distribution over the hidden units that is 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.



A neural model of digit recognition

The top two layers form an associative memory whose energy landscape models the low dimensional manifolds of the digits.

The energy valleys have names



The model learns to generate combinations of labels and images.

To perform recognition we start with a neutral state of the label units and do an up-pass from the image followed by a few iterations of the top-level associative memory.

Fine-tuning with a contrastive divergence version of the wake-sleep algorithm

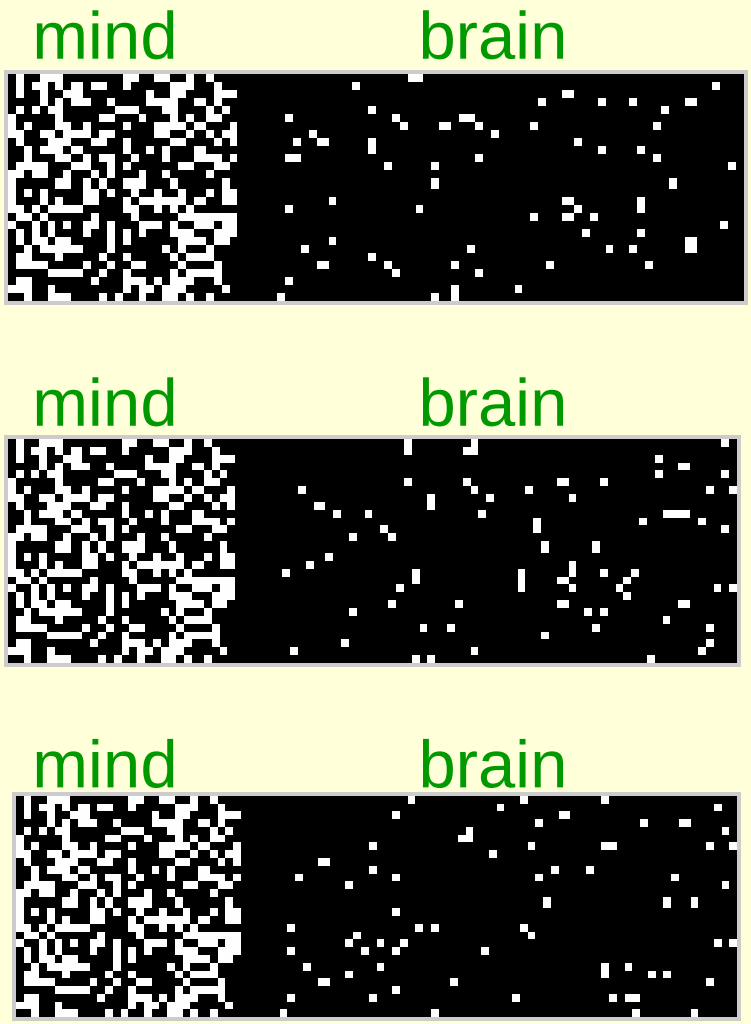
- After learning many layers of features, we can fine-tune the features to improve generation.
- 1. Do a stochastic bottom-up pass
  - Adjust the top-down weights to be good at reconstructing the feature activities in the layer below.
- 2. Do a few iterations of sampling in the top level RBM
  - Use CD learning to improve the RBM
- 3. Do a stochastic top-down pass
  - Adjust the bottom-up weights to be good at reconstructing the feature activities in the layer above.

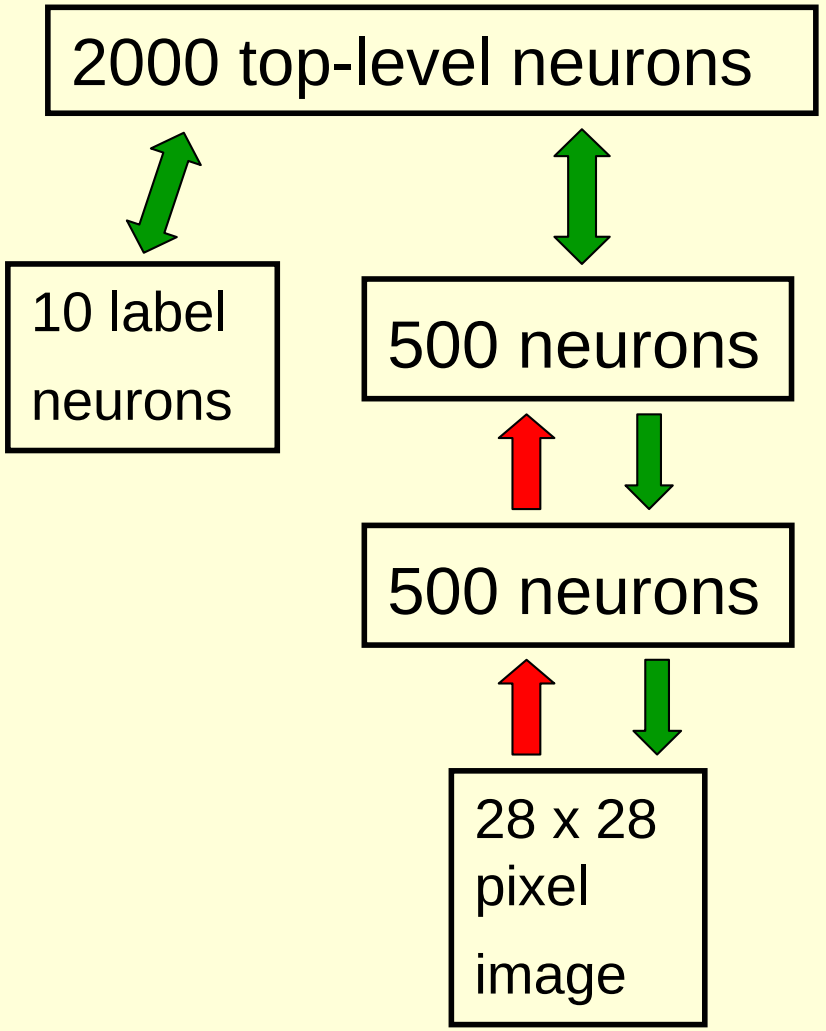
Show the movie of the network generating digits

(available at www.cs.toronto/~hinton)

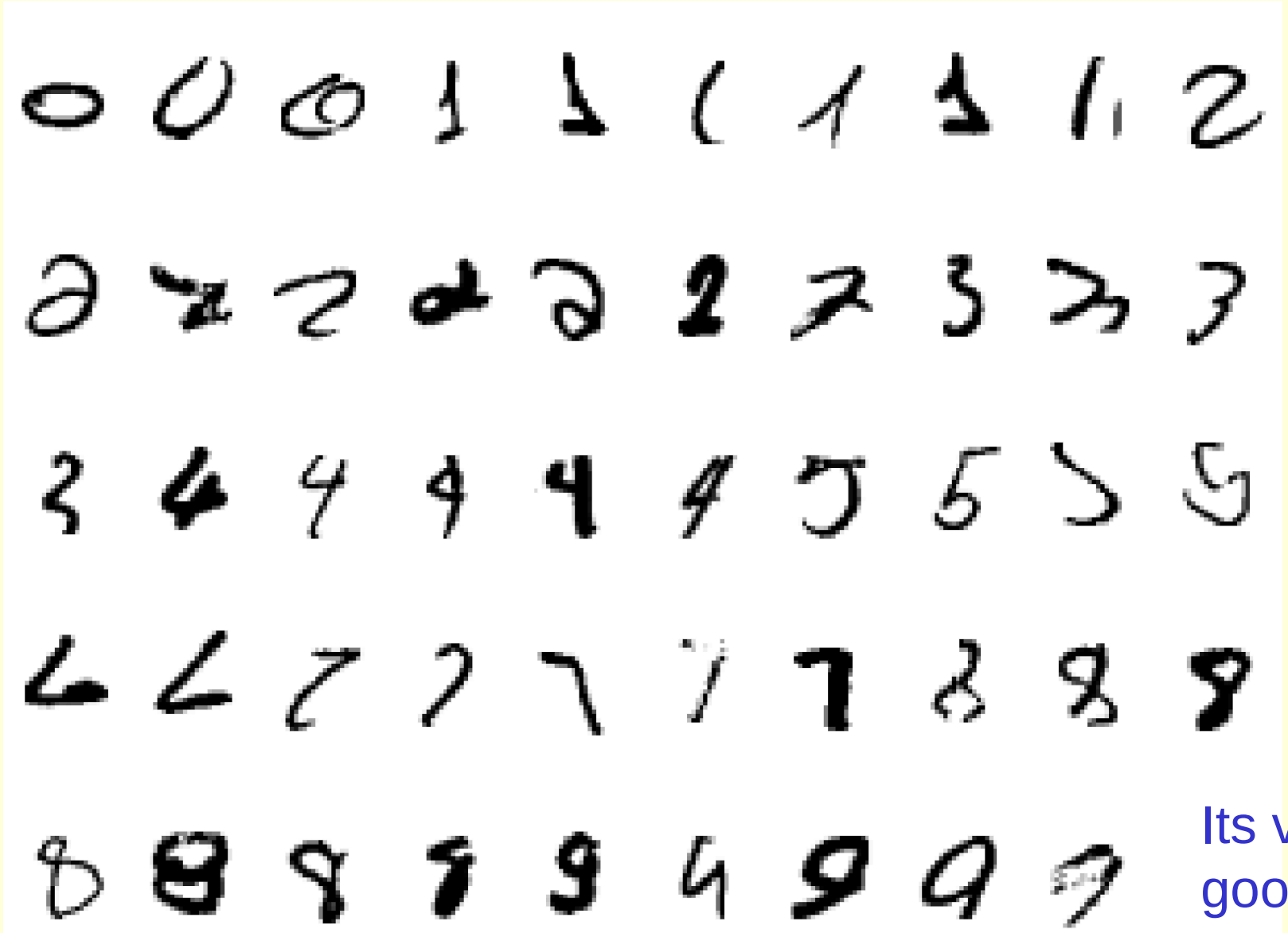
Samples generated by letting the associative memory run with one label clamped. There are 1000 iterations of alternating Gibbs sampling between samples.

What goes on in its mind if we show it an image composed of random pixels and ask it to fantasize from there?





Examples of correctly recognized handwritten digits that the neural network had never seen before



Its very good

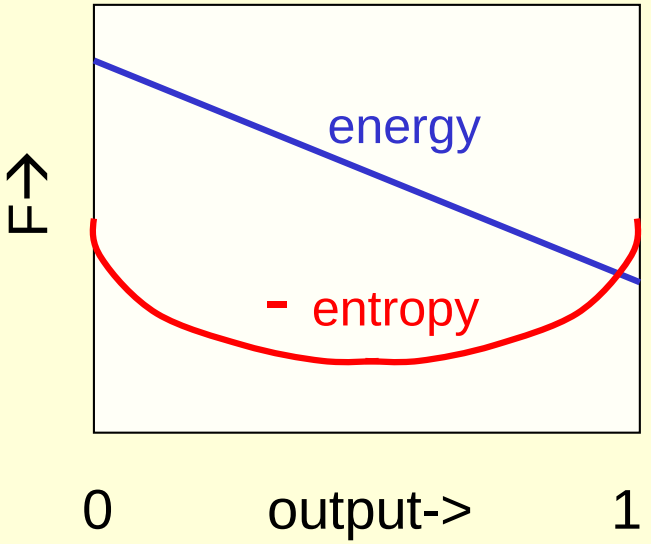
How well does it discriminate on MNIST test set with no extra information about geometric distortions?

- Generative model based on RBM's 1.25%
- Support Vector Machine (Decoste et. al.) 1.4%
- Backprop with 1000 hiddens (Platt) ~1.6%
- Backprop with 500 -->300 hiddens ~1.6%
- 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.

An RBM with real-valued visible units

- In a mean-field logistic unit, the total input provides a linear energy-gradient and the negative entropy provides a containment function with fixed curvature. So it is impossible for the value 0.7 to have much lower free energy than both 0.8 and 0.6. This is no good for modeling real-valued data.
- Using Gaussian visible units we can get much sharper predictions and alternating Gibbs sampling is still easy, though learning is slower.



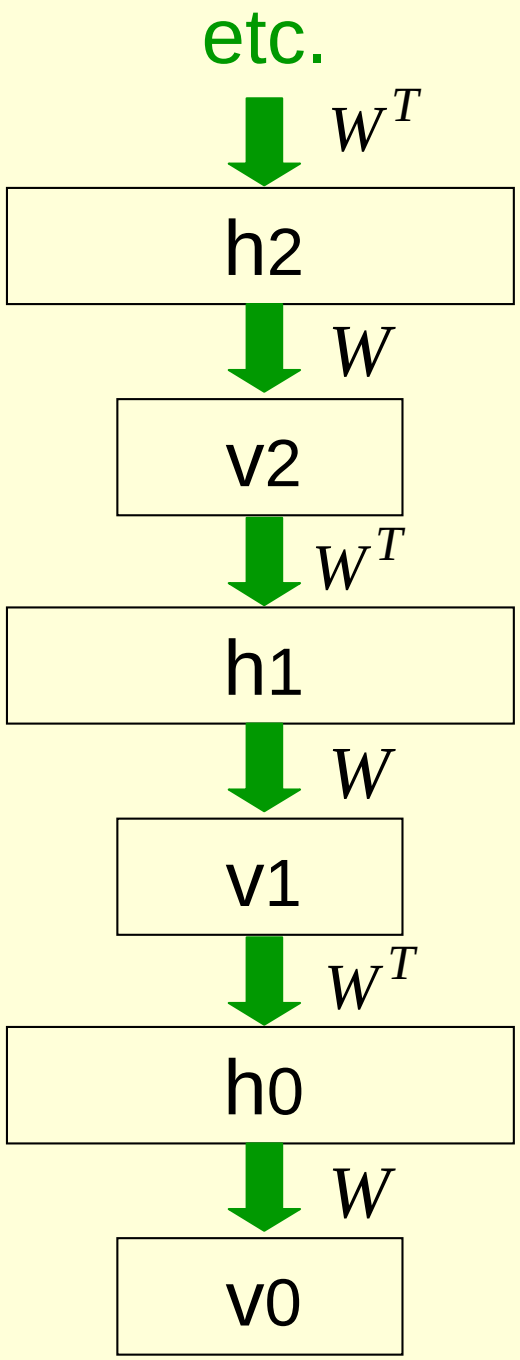
$$E(\mathbf{v},\mathbf{h}) = \sum_{i\ \varepsilon\ vis} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{j\ \varepsilon\ hid} b_j h_j - \sum_{i,j} \frac{v_i}{\sigma_i} h_j\ w_{ij}$$

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.

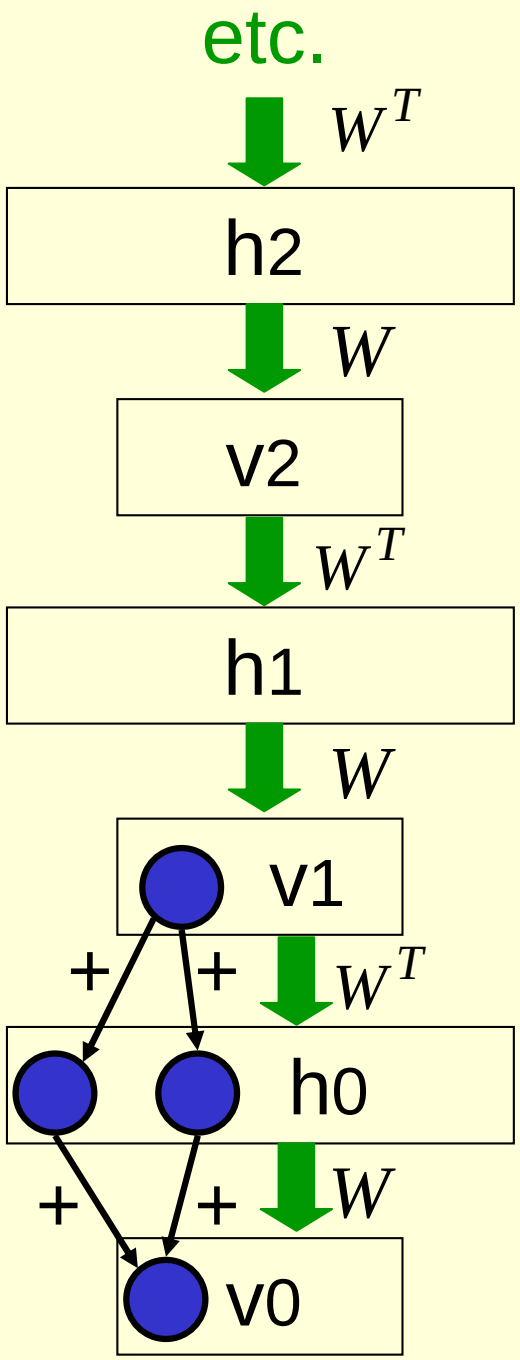
An infinite sigmoid belief net that is equivalent to an RBM

- 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 logistic DAG is:

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

- With replicated weights this becomes:

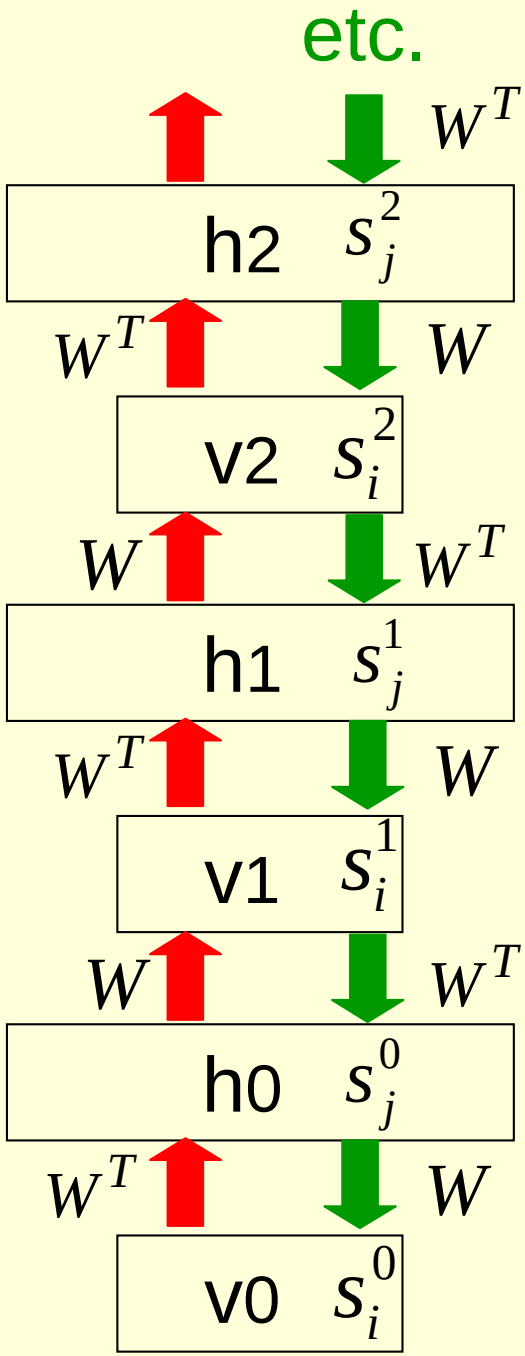
$$s_j^0 (s_i^0 - s_i^1) +$$

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

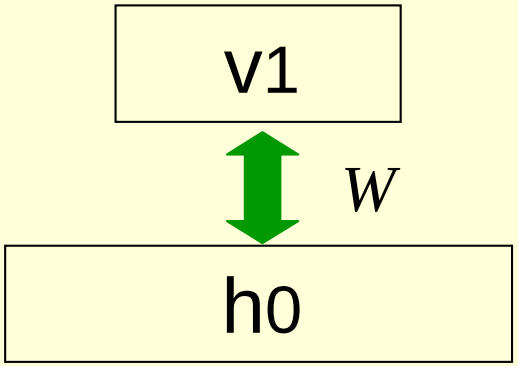
$$s_j^1 (s_i^1 - s_i^2) + \ldots$$

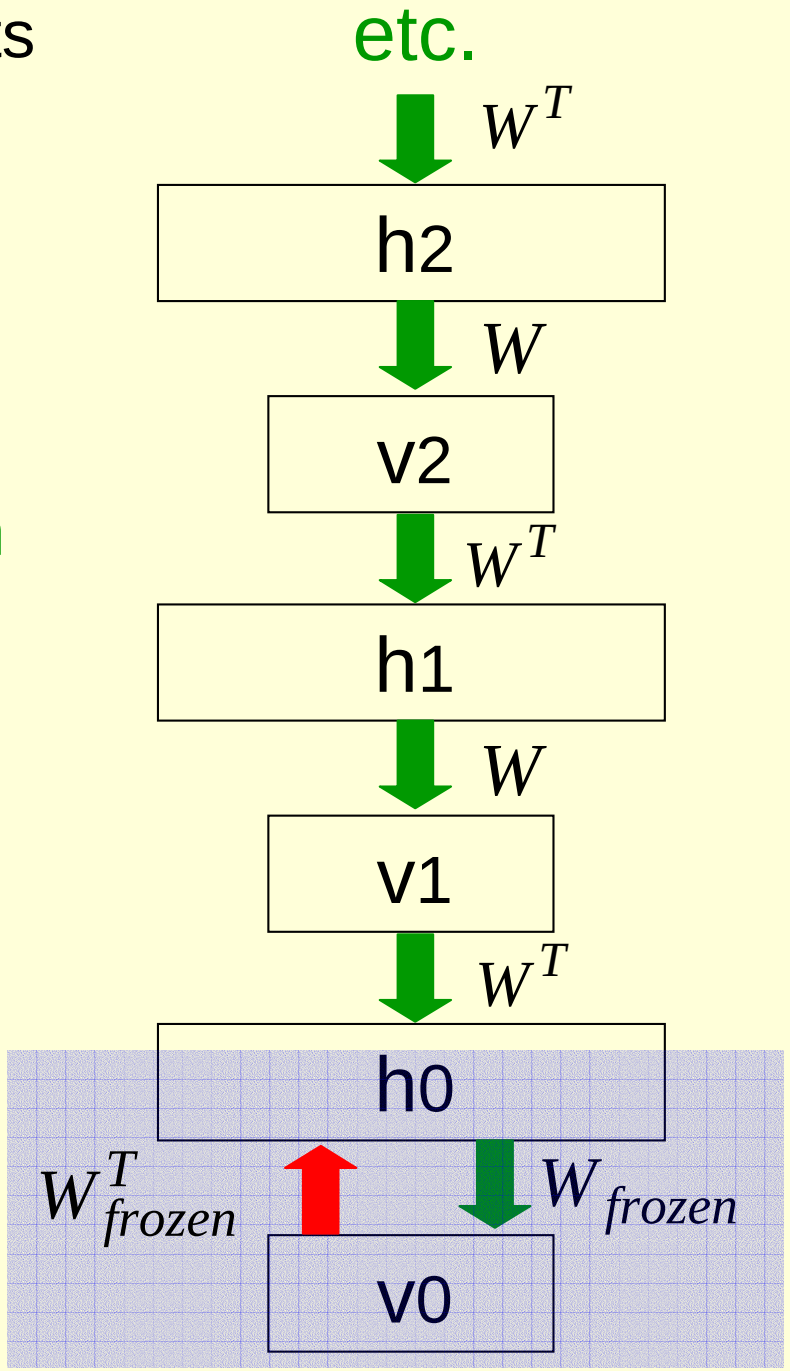
$$s_j^\infty s_i^\infty$$

The derivatives for the recognition weights are zero.



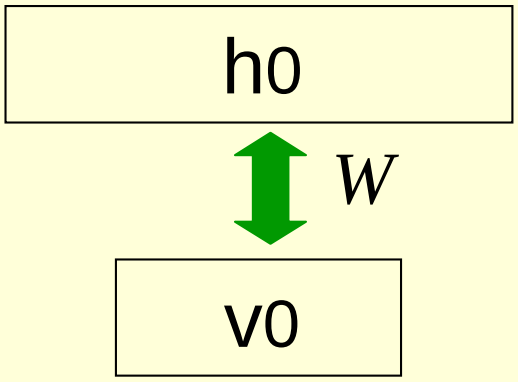
- Then freeze the first layer of weights in both directions and learn the remaining weights (still tied together).
  - This is equivalent to learning another RBM, using the aggregated posterior distribution of h0 as the data.

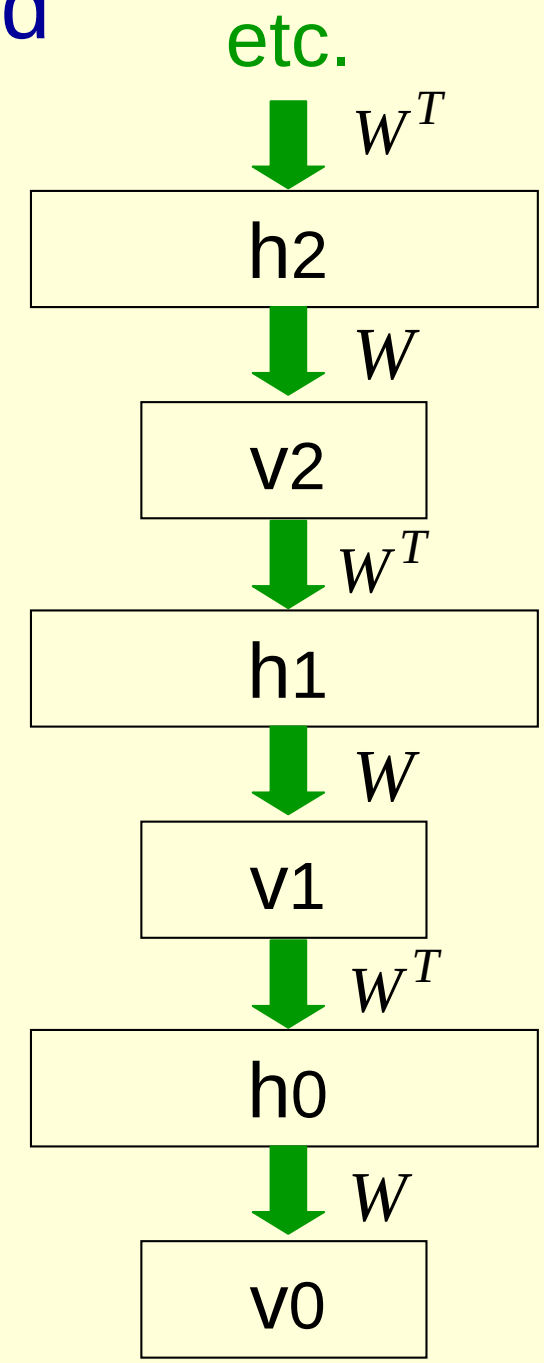




Learning a deep directed network

- 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.





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.

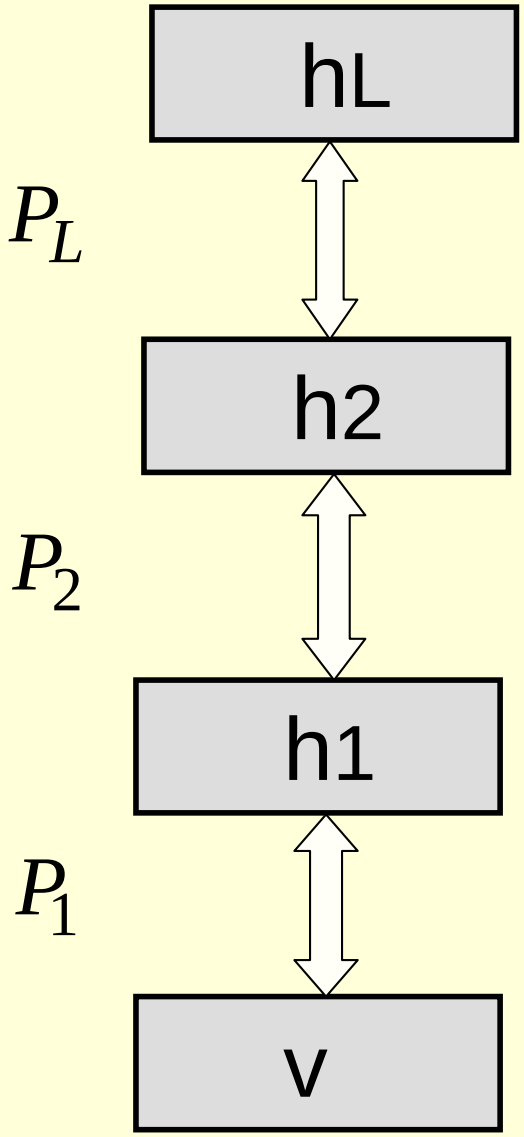
A stack of RBM's

- Each RBM has the same subscript as its hidden layer.
- Each RBM defines its own distribution over its visible vectors

$$P_l(h_{l-1}) = \frac{\sum_{h_l} \exp(-E(h_{l-1}, h_l))}{Z_l}$$

- Each RBM defines its own distribution over its hidden vectors

$$P_l(h_l) = \frac{\sum_{h_{l-1}} \exp(-E(h_{l-1}, h_l))}{Z_l}$$



The variational bound

Each time we replace the prior over the hidden units by a better prior, we win by the difference in the probability assigned

$$\log p(v) \geq \log P_1(v) + \sum_{l=1}^{l=L-1} \left\langle \log P_{l+1}(h_l) - \log P_l(h_l) \right\rangle_{Q(h_l|v)}$$

Now we cancel out all of the partition functions except the top one and replace log probabilities by goodnesses using the fact that:

$$\log P_l(x) = G_l(x) - \log Z_l$$

$$G(v) = \log \sum_h \exp(-E(v,h))$$

$$G(h) = \log \sum_v \exp(-E(v,h))$$

$$\log p(v) \geq G_1(v) + \sum_{l=1}^{l=L-1} \left\langle G_{l+1}(h_l) - G_l(h_l) \right\rangle_{Q(h_l|v)} - \log Z_L$$

This has simple derivatives that give a more justifiable fine-tuning algorithm than contrastive wake-sleep.

Using backpropagation for discriminative fine-tuning

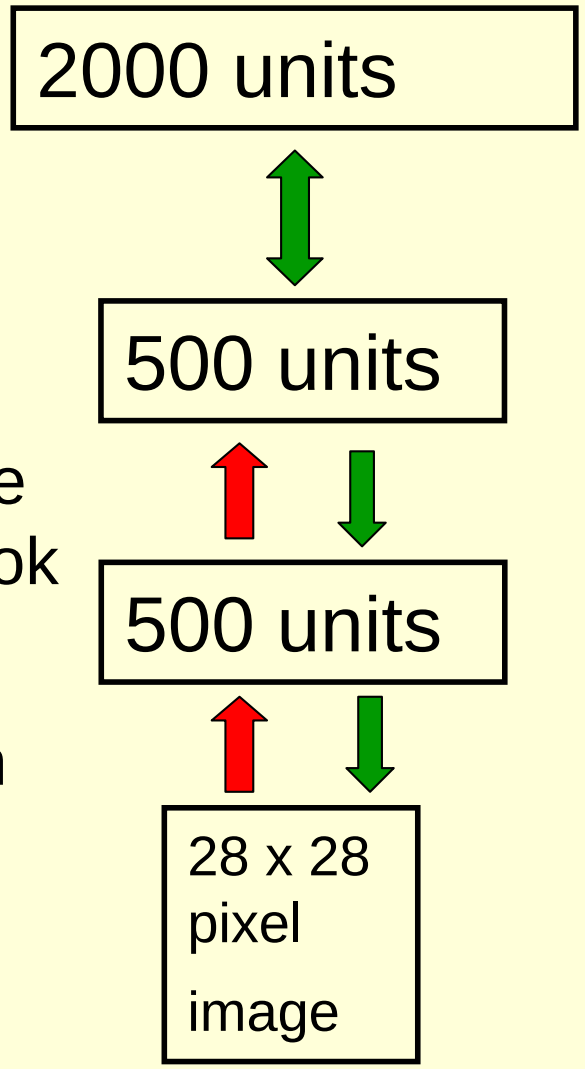
- Greedily learning one layer at a time scales well to really big networks, especially if we have locality in each layer.

- 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.

First, model the distribution of digit images

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



The network learns a density model for unlabeled digit images. When we generate from the model we often get things that look like real digits of all classes.

But do the hidden features really help with digit discrimination?

Add 10 softmaxed units to the top and do backpropagation.

Results on permutation-invariant MNIST task

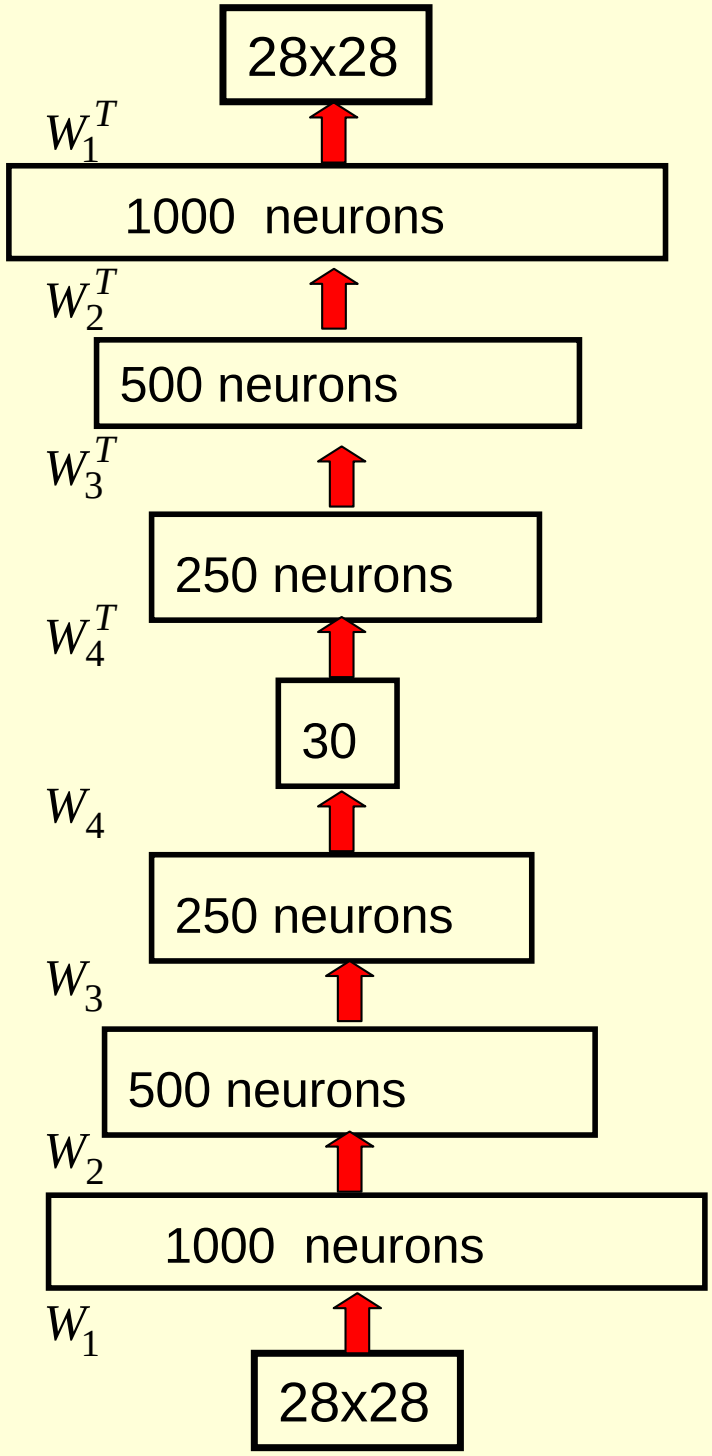
- Very carefully trained backprop net with one or two hidden layers (Platt; Hinton) 1.6%
- SVM (Decoste & Schoelkopf) 1.4%
- Generative model of joint density of images and labels (+ generative fine-tuning) 1.25%
- Generative model of unlabelled digits followed by gentle backpropagation 1.15%

Summary so far

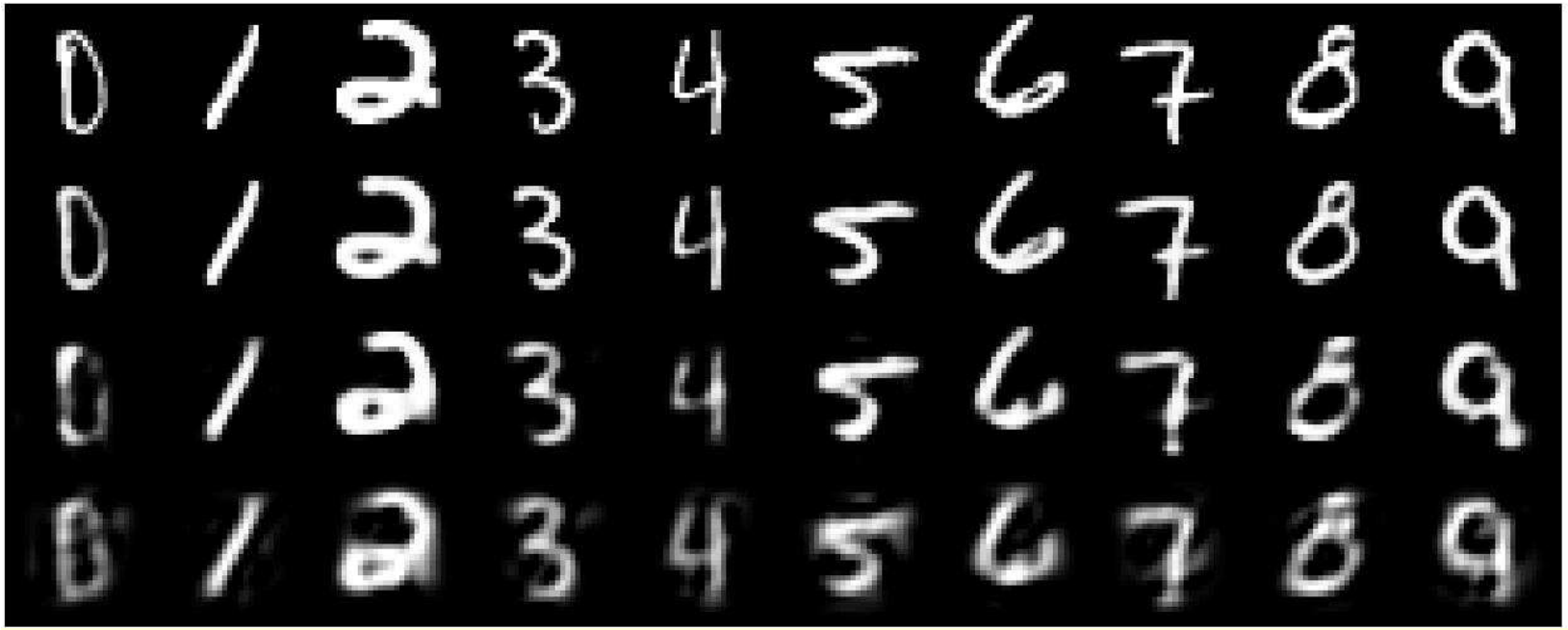
- Restricted Boltzmann Machines provide a simple way to learn a layer of features without any supervision.
- Many layers of representation can be learned by treating the hidden states of one RBM as the visible data for training the next RBM (a composition of experts).
- This creates good generative models that can then be fine-tuned.
  - Backpropagation can fine-tune discrimination.
  - Contrastive wake-sleep can fine-tune generation.
- The same ideas can be applied to high-dimensional sequential data.

Deep Autoencoders
(Ruslan Salakhutdinov)

- They always looked like a really nice way to do non-linear dimensionality reduction:
  - But it is very difficult to optimize deep autoencoders using backpropagation.
- We now have a much better way to optimize them:
  - First train a stack of 4 RBM's
  - Then "unroll" them.
  - Then fine-tune with backprop.



A comparison of methods for compressing digit images to 30 real numbers.



real data

30-D deep auto

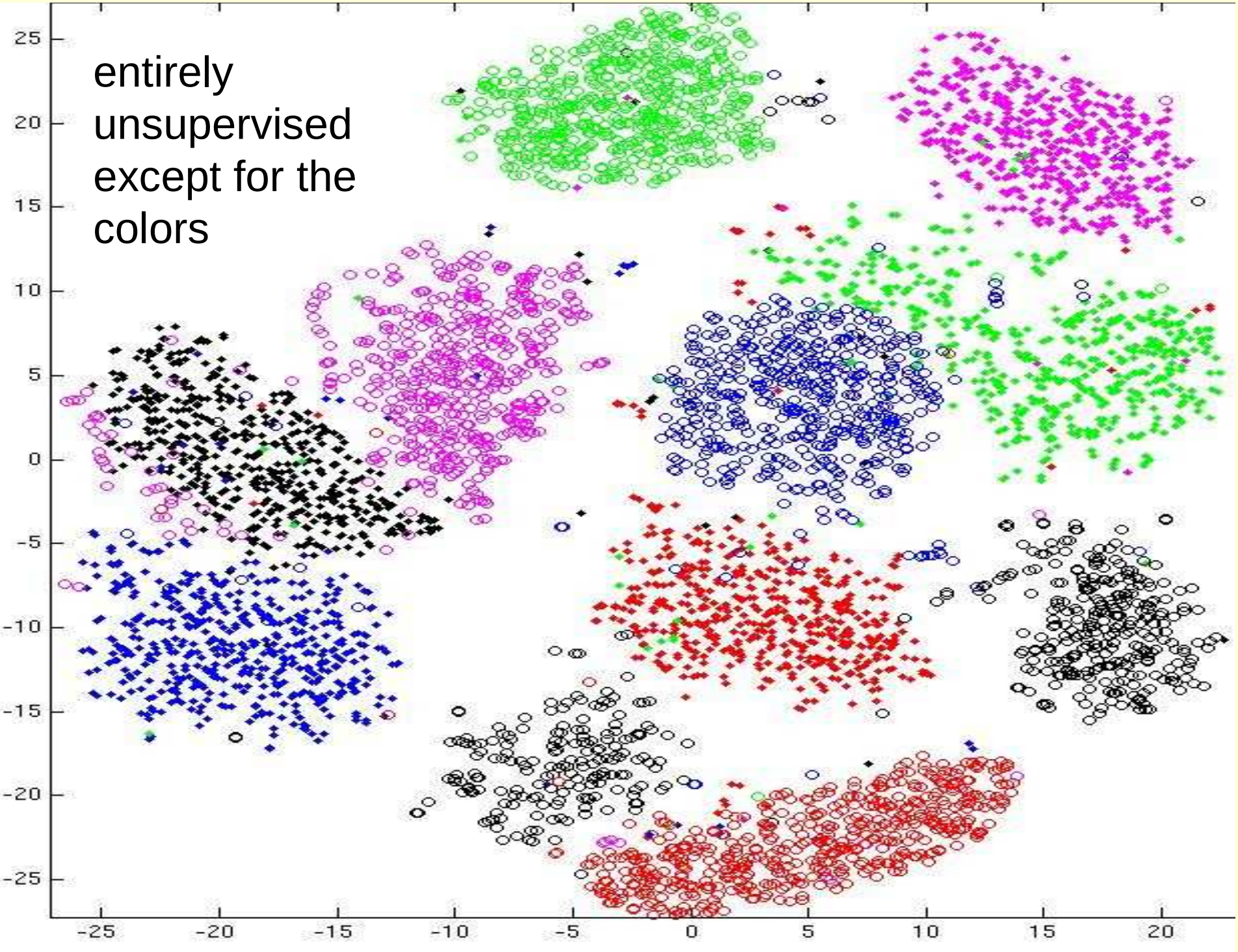
30-D logistic PCA

30-D PCA

Do the 30-D codes found by the autoencoder preserve the class structure of the data?

- Take the 30-D activity patterns in the code layer and display them in 2-D using a new form of non-linear multi-dimensional scaling (UNI-SNE)

- Will the learning find the natural classes?


entirely
unsupervised
except for the
colors
25
20
15
10
5
0
-5
-10
-15
-20
-25
-25
-20
-15
-10
-5
0
5
10
15
20