

IS 6733: Deep Learning on Cloud Platforms




Deep Neural Network

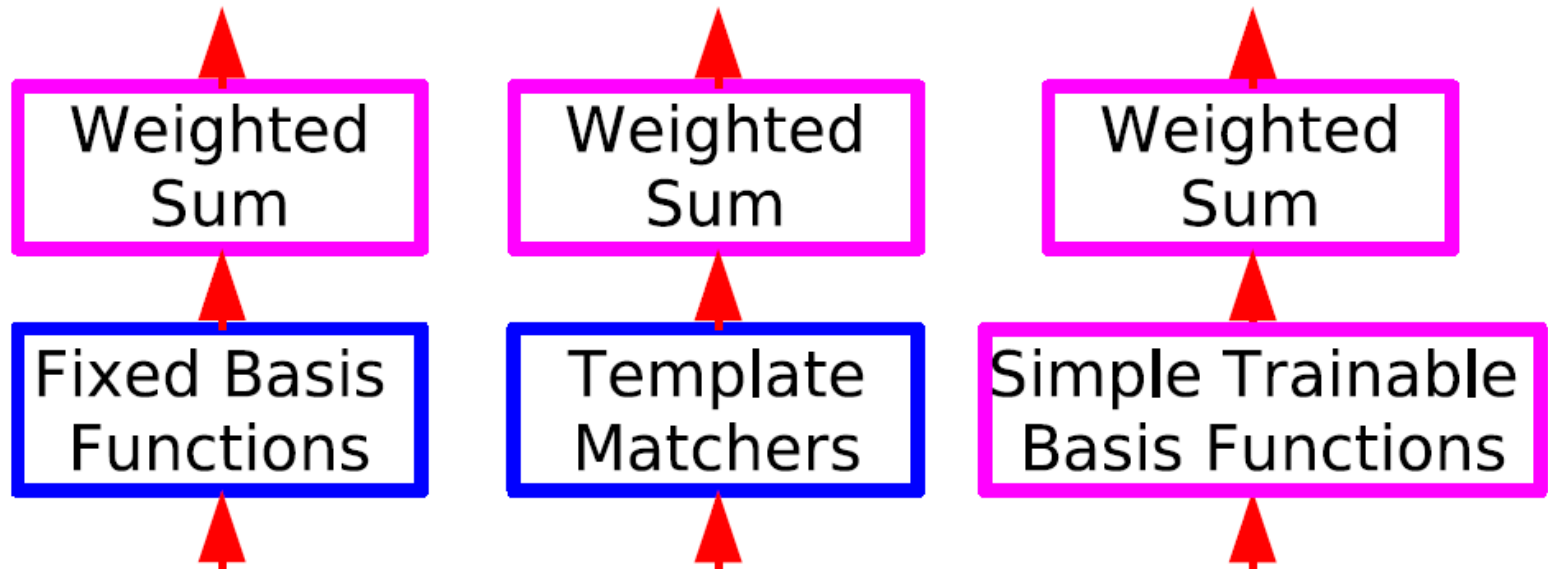
Copy Right Notice

 **Most slides in this presentation are adopted from slides of text book and various sources. The Copyright belong to the original authors. Thanks!**

Learning Architectures: Shallow and Deep

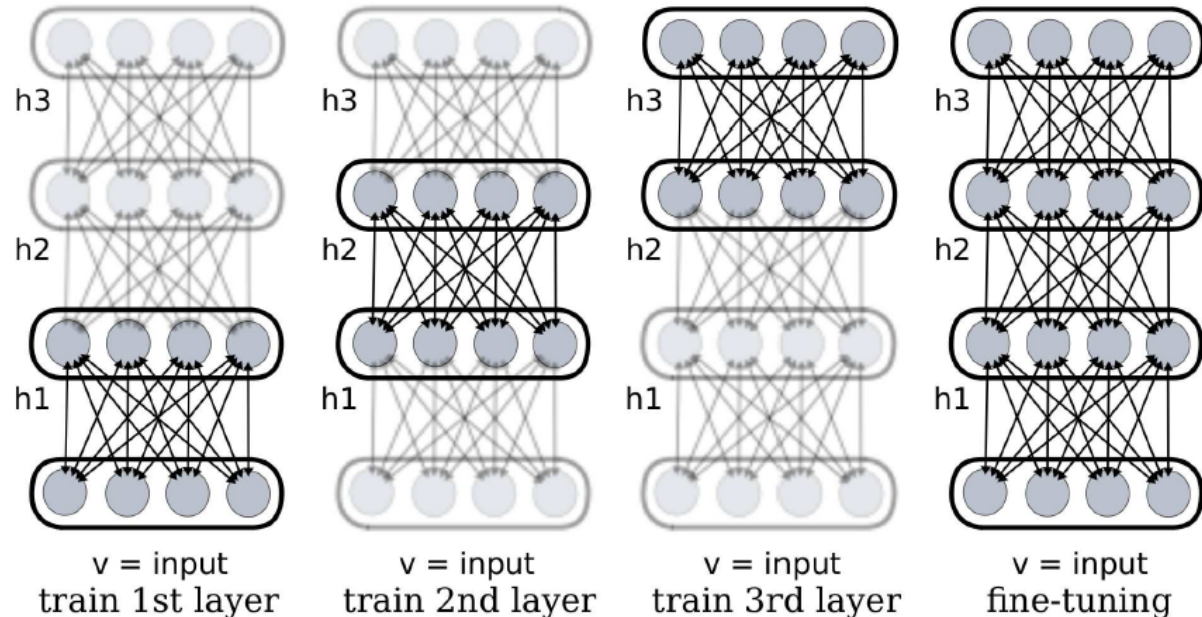
Different types of shallow architectures.

-  (a) Type-1: fixed preprocessing and linear predictor (logistic regression, Perceptron);
-  (b) Type-2: template matchers and linear predictor (kernel machine);
-  (c) Type-3: simple trainable basis functions and linear predictor (neural net with one hidden layer, RBF network).



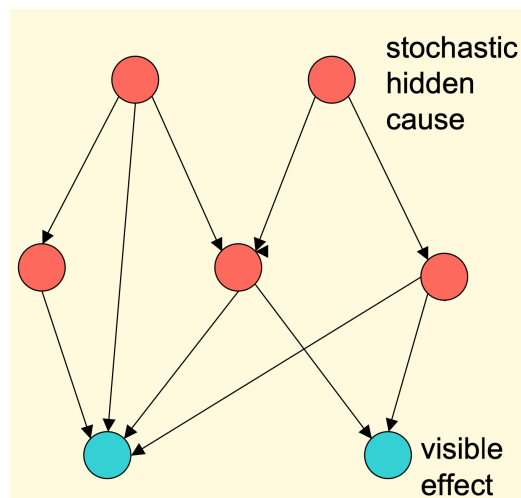
Learning Architectures: Shallow and Deep

- Deep architectures are *compositions of many layers of adaptive non-linear components*, in other words, they are cascades of parameterized non-linear modules that contain trainable parameters at all levels.
- Most functions that can be represented compactly by deep architectures cannot be represented by a compact shallow architecture.
- Problem with deep?
 - Many cases, deep nets are **hard to optimize**.

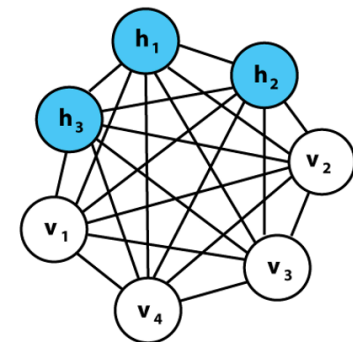


Two types of generative neural network

- 🐾 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.

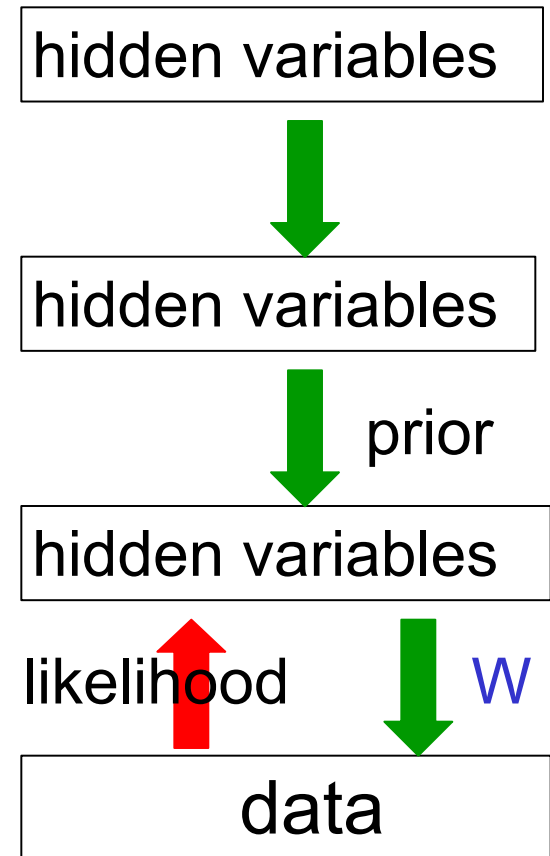


(Hinton and Sejnowski 1985)



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.



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.

$$p(s_i=1) = \frac{1}{1 + e^{-\sum_j s_j w_{ij} / T}} = \frac{1}{1 + e^{-\Delta E_i / T}}$$

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

Weights → Energies → Probabilities

 The energy of a joint configuration of the visible and hidden units determines its probability:

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

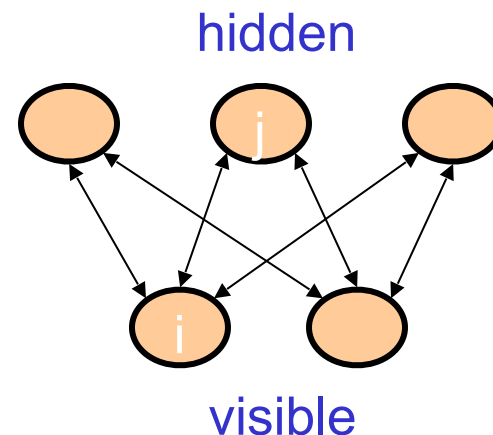
 The probability of a configuration over the visible units is found by summing the probabilities of all the joint configurations that contain it.

Restricted Boltzmann machines (RBM)

- 🐾 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.

$E(v, h) = -b'v - c'h - h'Wv$
 🐾 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
- 🐾 Approximation of the log-likelihood gradient:
- 🐾 Contrastive Divergence



binary state of visible unit i binary state of hidden unit j

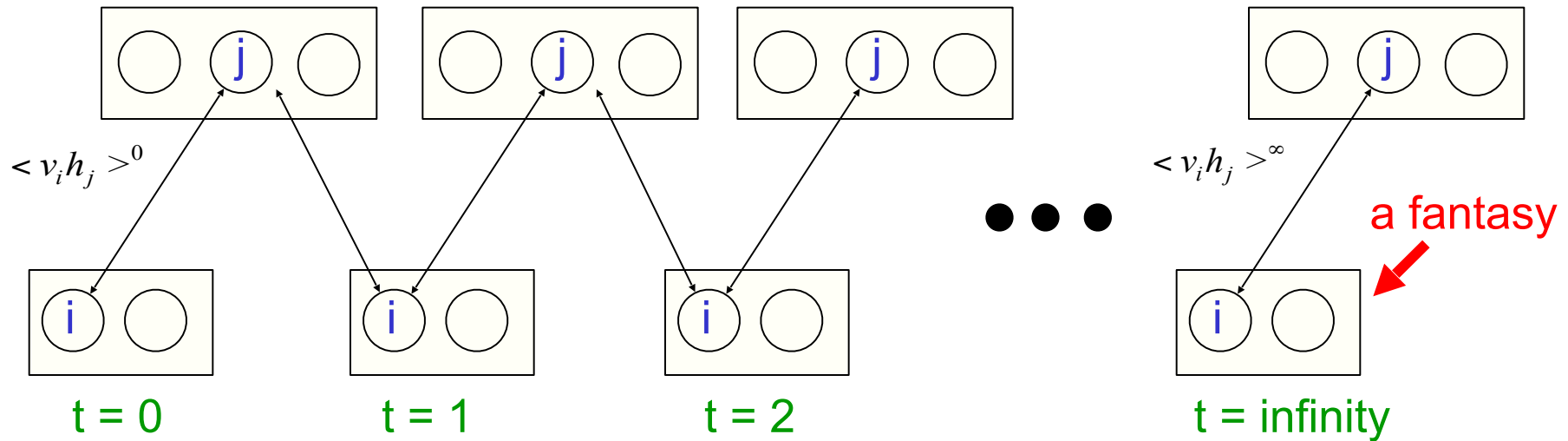
$$E(v, h) = - \sum_{i,j} v_i h_j w_{ij}$$

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

weight between units i and j

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

A picture of the maximum likelihood learning algorithm for an RBM

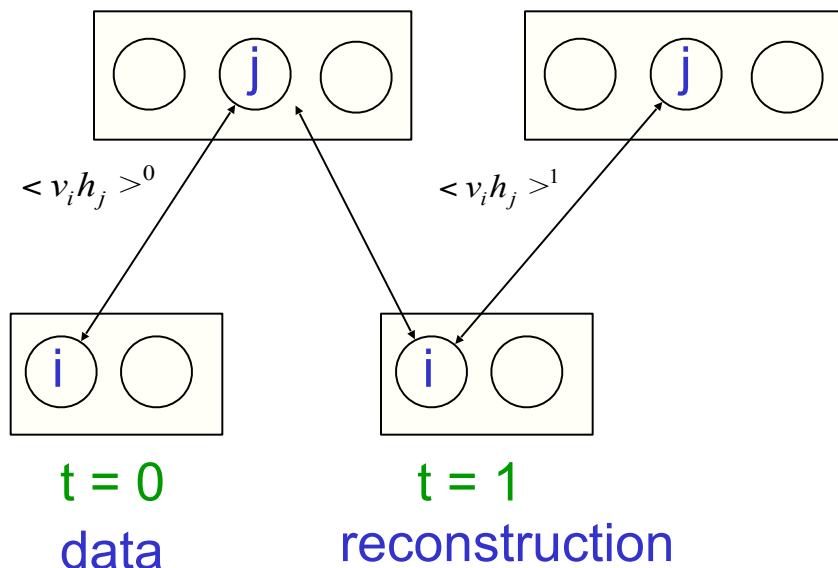


Start with a training vector on the visible units.

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

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

A quick way to learn an RBM



Start with a training vector on the visible units.

Update all the hidden units in parallel

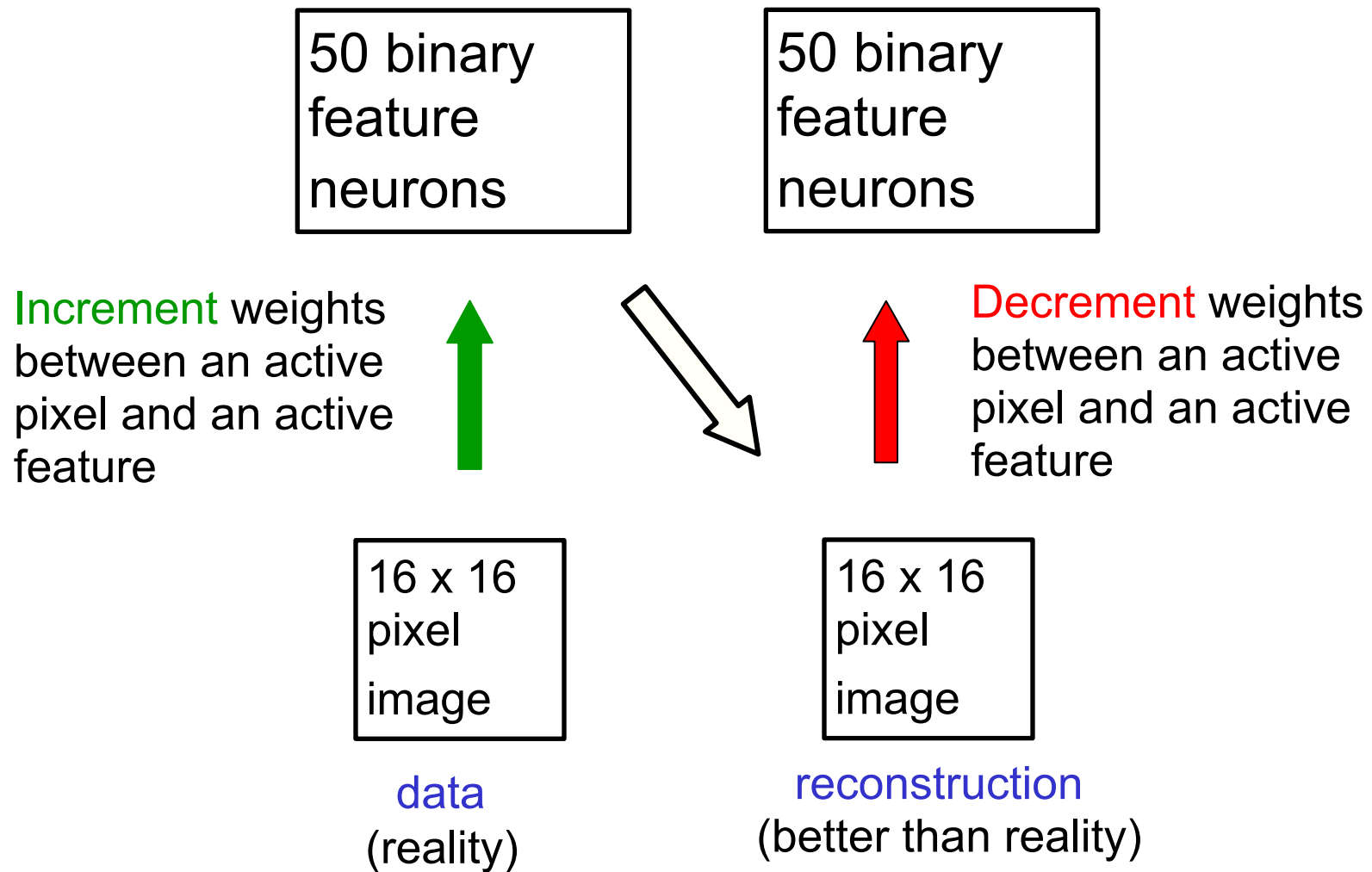
Update the all the visible units in parallel to get a “reconstruction”.

Update the hidden units again.

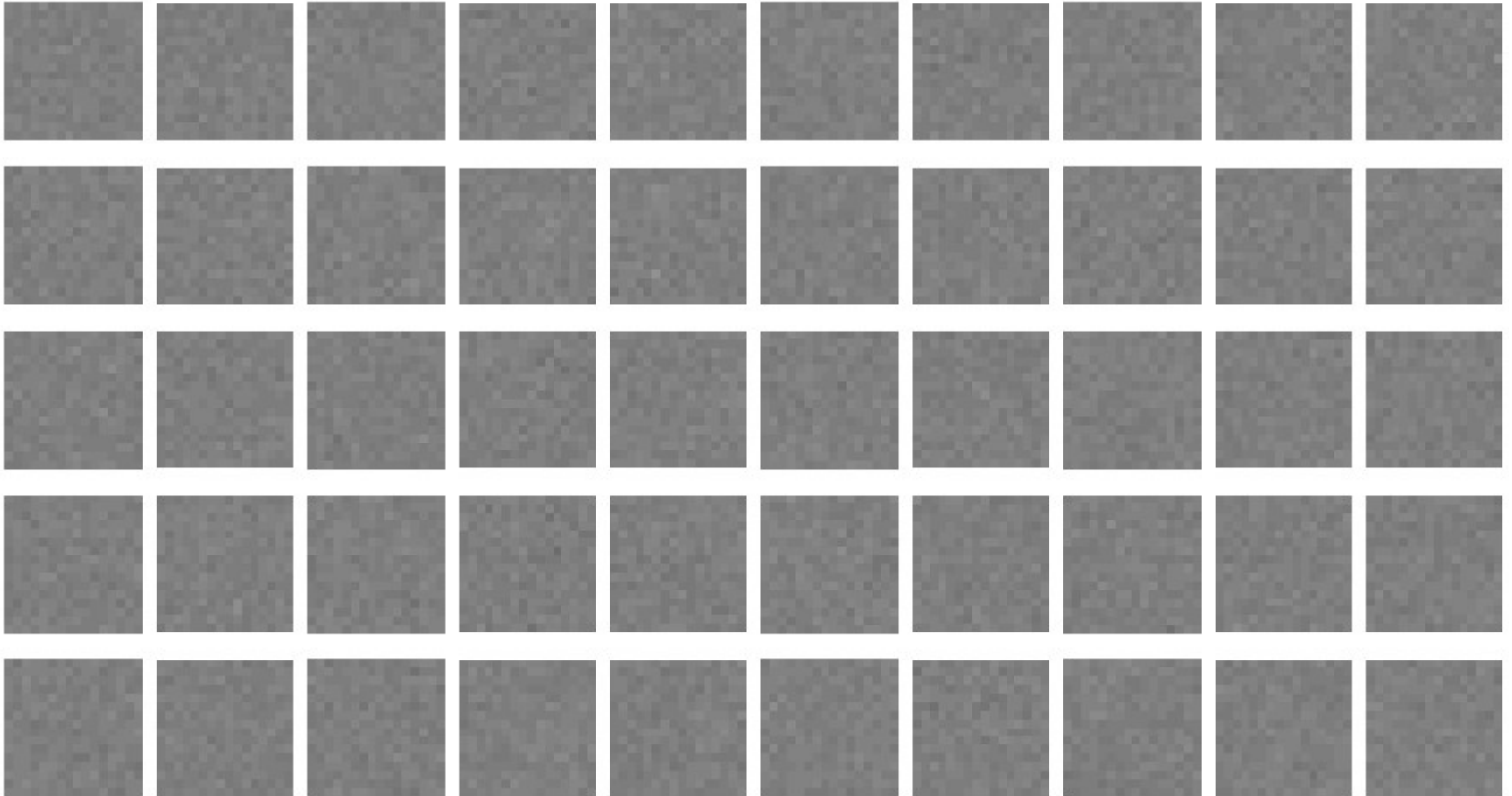
$$\Delta w_{ij} = \varepsilon (\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1)$$

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

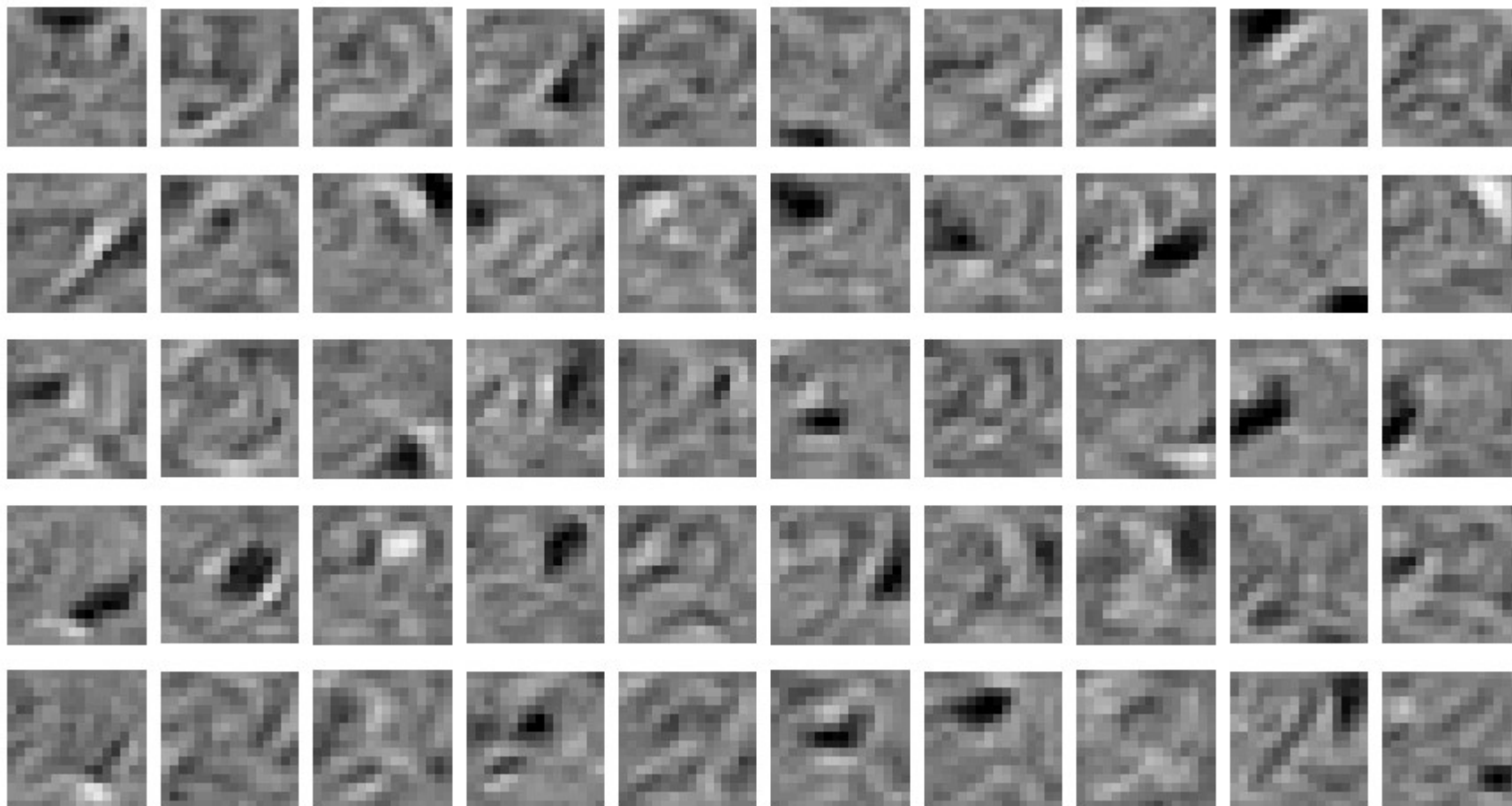


The weights of the 50 feature detectors



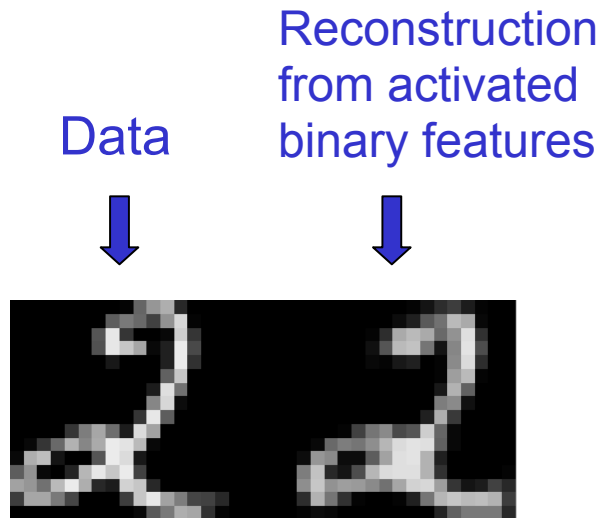
We start with small random weights to break symmetry

The final 50 x 256 weights

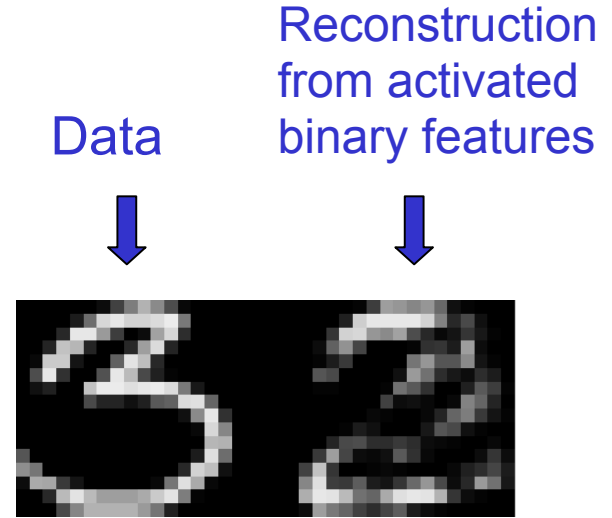


Each neuron grabs a different feature.

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




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

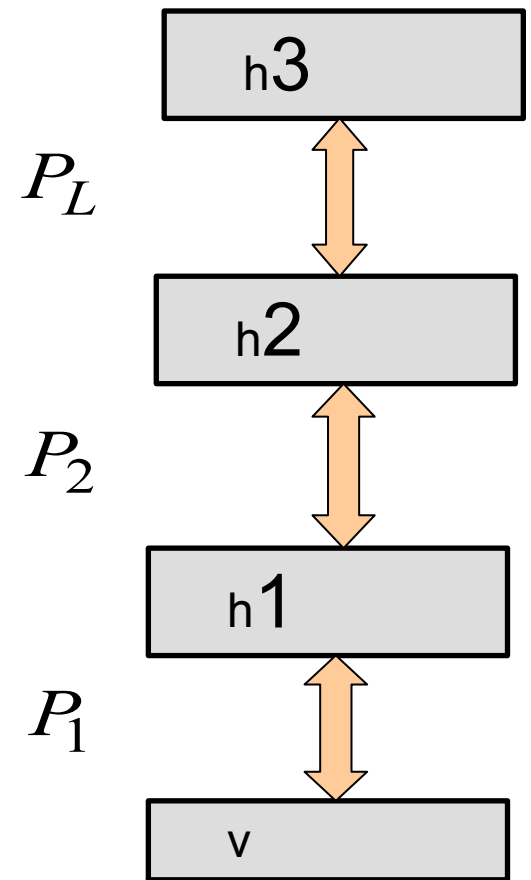


Images from an unfamiliar digit class
(the network tries to see every image as a 2)

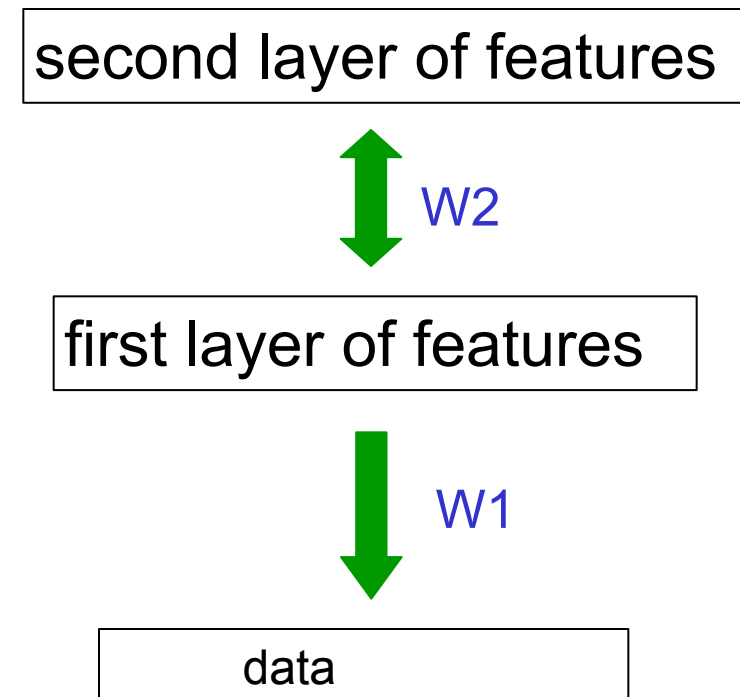
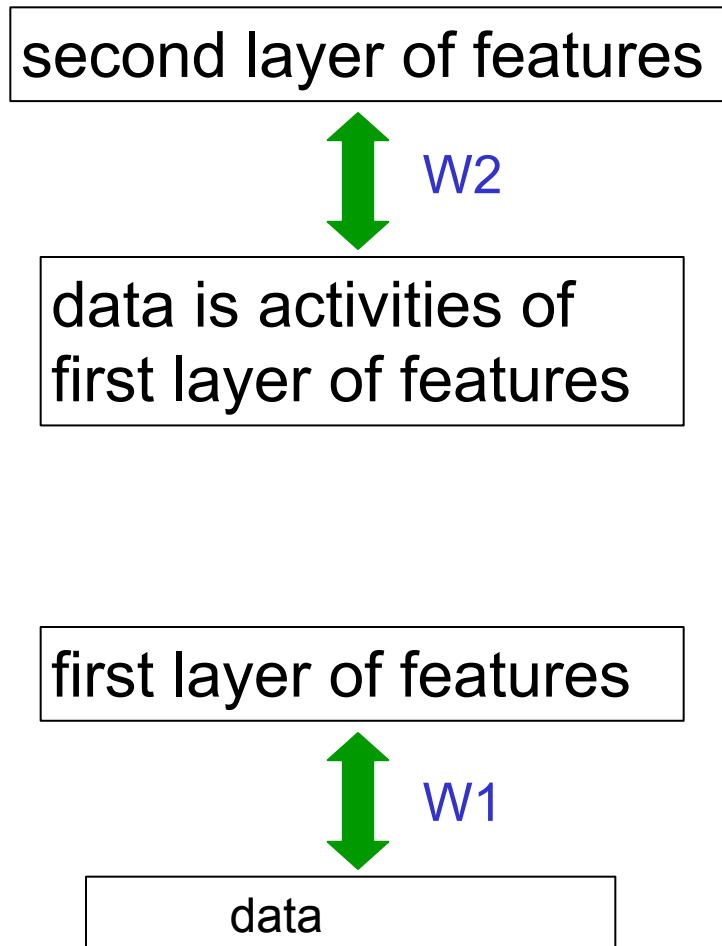
Deep Belief Networks

 Stacking RBMs to form Deep architecture

 DBN with L layers of models the joint distribution between observed vector x and L hidden layers h .



The overall model produced by composing two RBM's

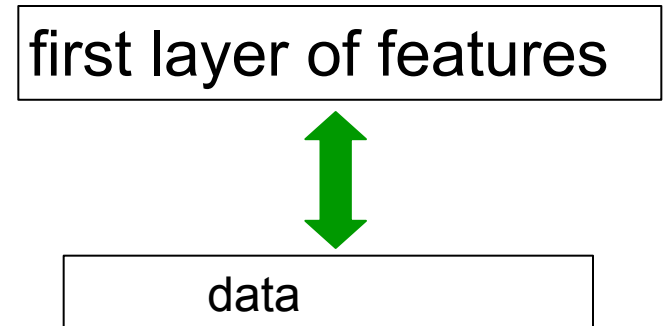
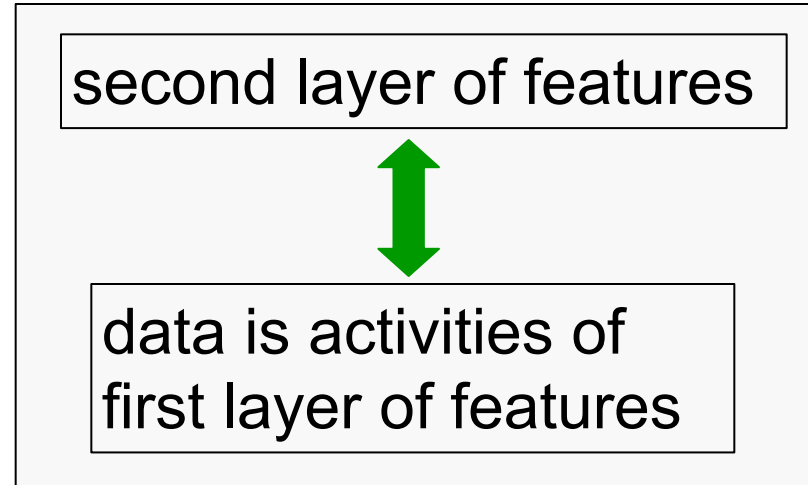


Training a deep network

- 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.
- And so on for as many hidden layers as we want.
- It can be proved that each time we add another layer of features we get a better model of the set of training images.


RBM2

RBM1




Recursive Restricted Boltzmann Machines

 **Is learning a model of the hidden activities just a hack?**

 It does not seem as if we are learning a proper multilayer model because the lower weights do not depend on the higher ones.

 **Can we treat the hidden layers of the whole stack of RBM's as part of one big generative model rather than a model plus a model of a model etc.?**

 If it is one big model, it definitely is not a Boltzmann machine. The first hidden layer has two sets of weights (above and below) which would make the hidden activities very different from the activities of those units in either of the RBM's.

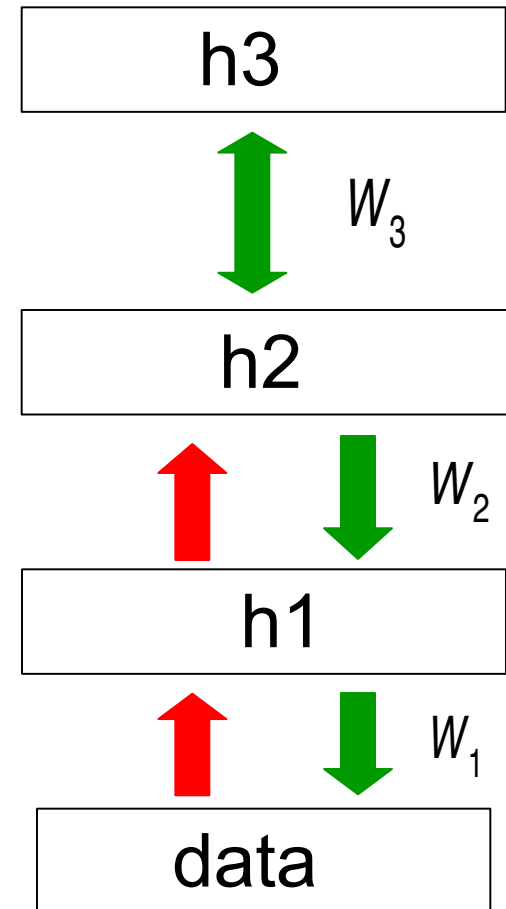
The generative model after learning 3 layers





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 stacking RBM's produce this kind of generative model?


-  It is not at all obvious that stacking RBM's produces a model in which the top two layers of features form an RBM, but the layers beneath that are not at all like a Boltzmann Machine.
-  To understand why this happens we need to ask how an RBM defines a probability distribution over visible vectors.

How an RBM defines the probabilities of hidden and visible vectors

 The weights in an RBM define $p(h|v)$ and $p(v|h)$ in a very straightforward way (lets ignore biases for now)

 To sample from $p(v|h)$, sample the binary state of each visible unit from a logistic that depends on its weight vector times h .

 To sample from $p(h|v)$, sample the binary state of each hidden unit from a logistic that depends on its weight vector times v .

 If we use these two conditional distributions to do alternating Gibbs sampling for a long time, we can get $p(v)$ or $p(h)$

 i.e. we can sample from the model's distribution over the visible or hidden units.

Why does layer-by-layer (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

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

index over all hidden vectors joint probability conditional probability

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.

An analogy

- 🐾 In a mixture model, we define the probability of a data vector to be


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

index over all Gaussians mixing proportion of Gaussian probability of v given Gaussian h

- 🐾 The learning rule for the mixing proportions is to make them match the posterior probability of using each Gaussian.
- 🐾 The weights of an RBM implicitly define a mixing proportion for each possible hidden vector.
- 🐾 To fit the data better, we can leave $p(v|h)$ the same and make the mixing proportion of each hidden vector more like the posterior over hidden vectors.


A guarantee

 **Can we prove that adding more layers will always help?**

 It would be very nice if we could learn a big model one layer at a time and guarantee that as we add each new hidden layer the model gets better.


 **We can actually guarantee the following:**


 There is a lower bound on the log probability of the data.


 Provided that the layers do not get smaller and the weights are initialized correctly (which is easy), every time we learn a new hidden layer this bound is improved (unless its already maximized).

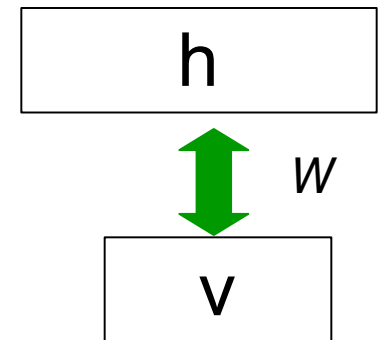
What does each RBM achieve?

 It divides the task of modelling 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 modelling the original data.



Back-fitting

-  **After we have learned all the layers greedily, the weights in the lower layers will no longer be optimal.**
-  **The weights in the lower layers can be fine-tuned in several ways.**
 -  For the generative model that comes next, the fine-tuning involves a complicated and slow stochastic learning procedure.
 -  If our ultimate goal is discrimination, we can use backpropagation for the fine-tuning.

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:

10 label units

2000 top-level units

500 units

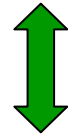
500 units

28 x 28
pixel
image

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 harmony of the RBM with each of the 10 labels



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

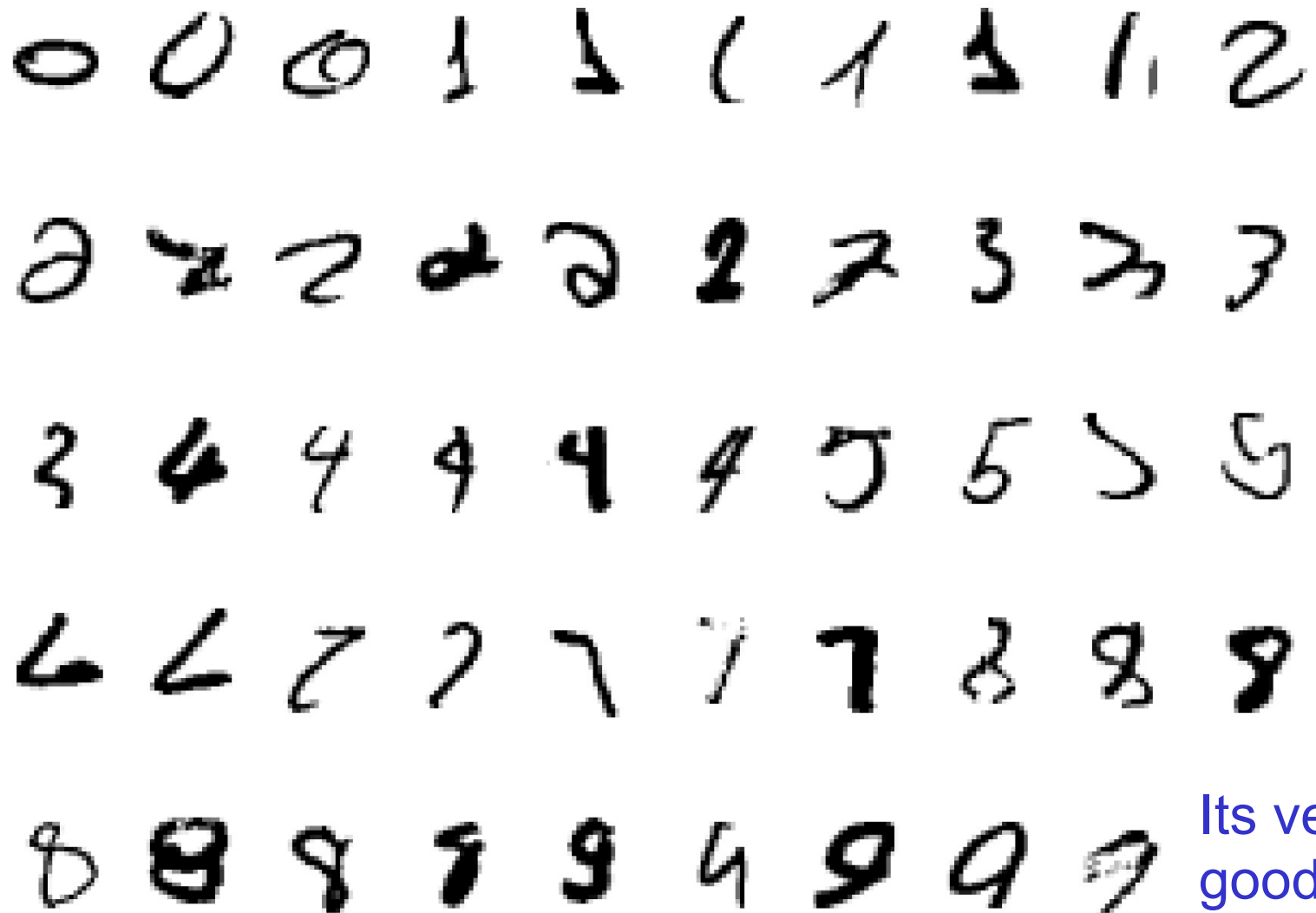
 3. Do a stochastic top-down pass

 Adjust the bottom-up weights to be good at reconstructing the feature activities in the layer above.

See the movie at






<http://www.cs.toronto.edu/~hinton/adi/index.htm>


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

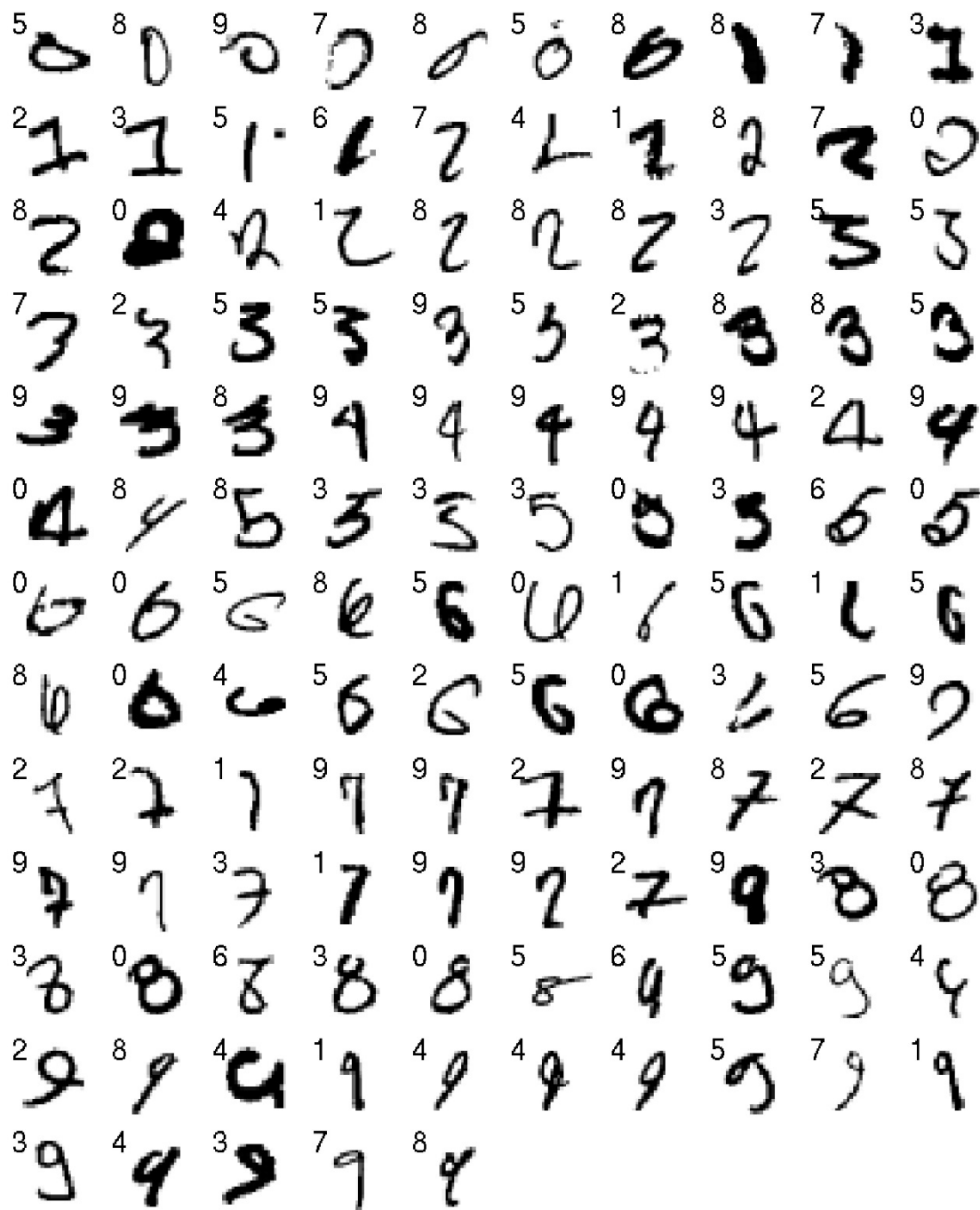


Its very
good

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%

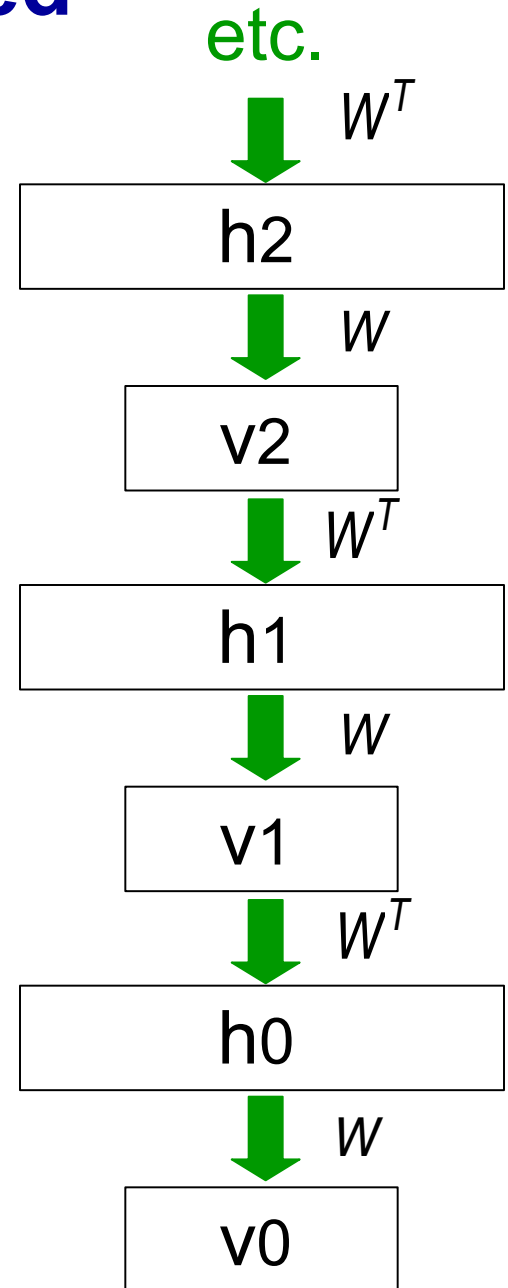
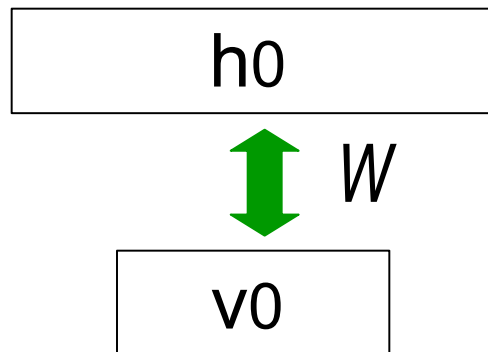
 **Its better than backprop and much more neurally plausible because the neurons only need to send one kind of signal, and the teacher can be another sensory input.**



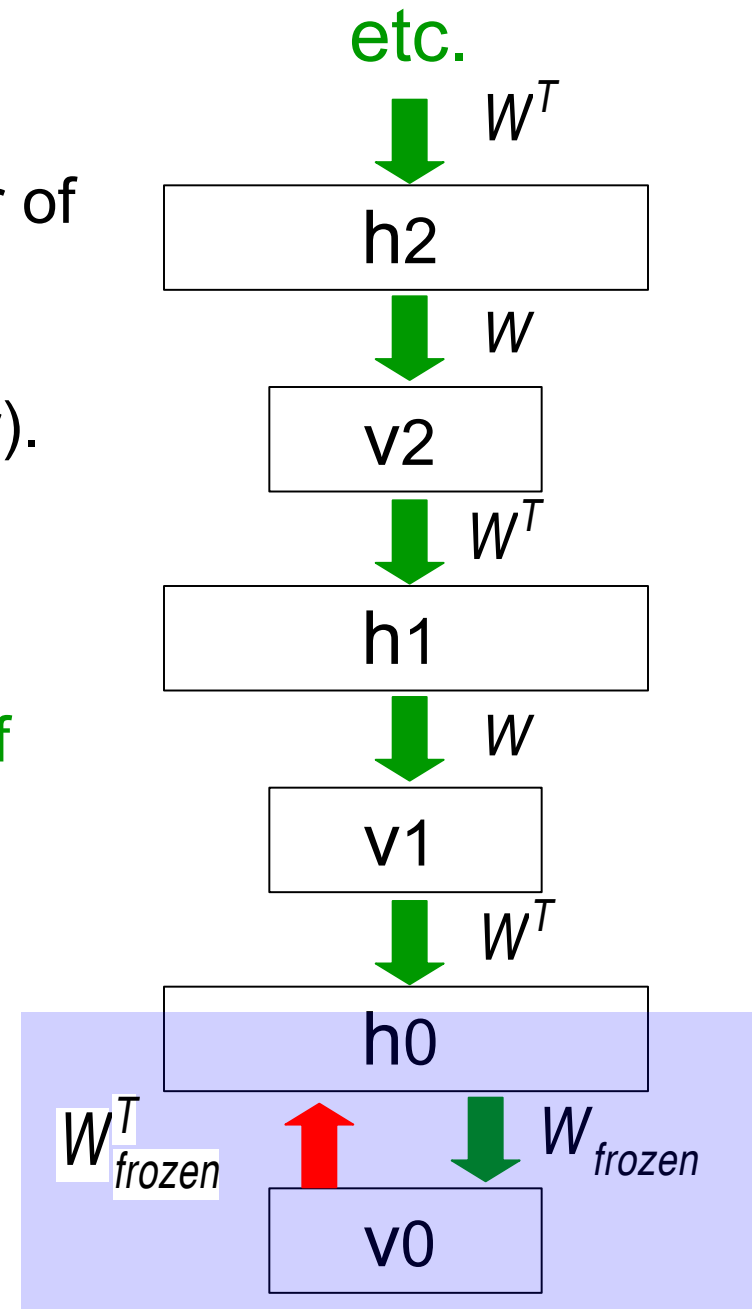
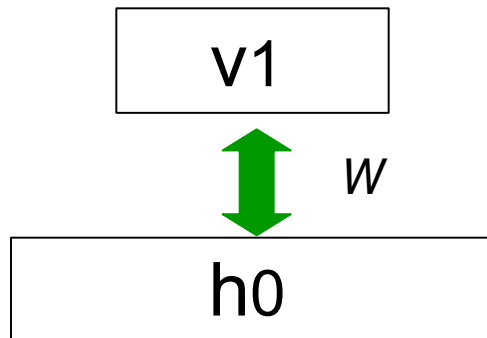
All 125 errors

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 h_0 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.

Using backpropagation for 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.

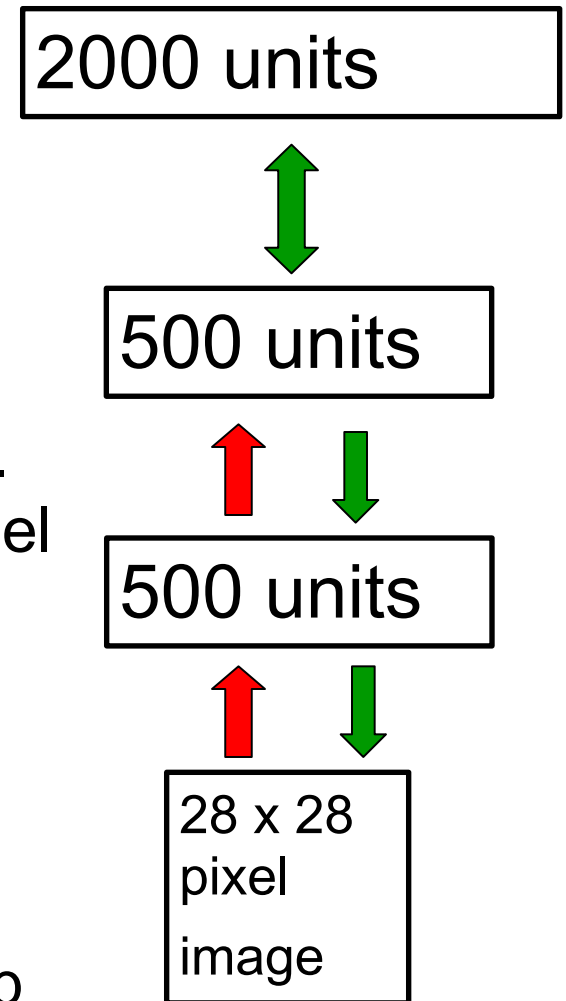
model the distribution of digit images

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
1.6% one or two hidden layers (Platt; Hinton)

 SVM (Decoste & Schoelkopf)
1.4%

 Generative model of joint density of
1.25% images and labels (+ generative fine-tuning)

 Generative model of unlabelled digits
1.15% followed by gentle backpropagation

Summary

-  **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.**