Neural Networks

Lecture 20
Learning features one layer at a time

Learning multilayer networks

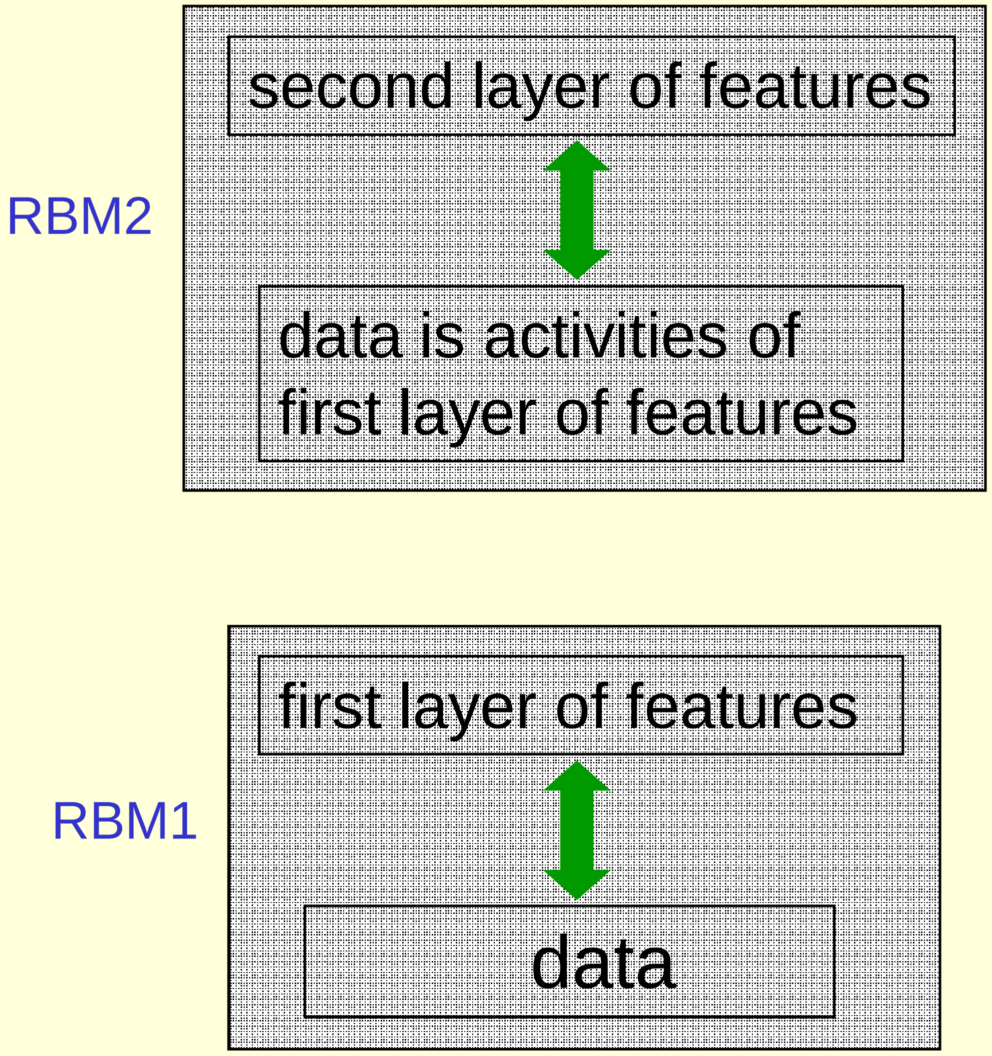
- We want to learn models with multiple layers of non-linear features.
- Perceptrons: Use a layer of hand-coded, non-adaptive features followed by a layer of adaptive decision units.
  - Needs supervision signal for each training case.
  - Only one layer of adaptive weights.
- Back-propagation: Use multiple layers of adaptive features and train by backpropagating error derivatives
  - Needs supervision signal for each training case.
  - Learning time scales poorly for deep networks.
- Support Vector Machines: Use a very large set of fixed features
  - Needs supervision signal for each training case.
  - Does not learn multiple layers of features

Learning multi-layer networks (continued)

- Boltzmann Machines: Nice local learning rule that works in arbitrary networks.
  - Getting the pairwise statistics required for learning can be very slow if the hidden units are interconnected.
  - Maximum likelihood learning requires unbiased samples from the model's distribution. These are very hard to get.
- Restricted Boltzmann Machines: Exact inference is very easy when the states of the visible units are known because then the hidden states are independent.
  - Maximum likelihood learning is still slow because of the need to get samples from the model, but contrastive divergence learning is fast and often works well.
  - But we can only learn one layer of adaptive features!

Stacking Restricted Boltzmann Machines

- First learn a layer of hidden features.
- Then treat the feature activations as data and learn a second layer of hidden features.
- And so on for as many hidden layers as we want.

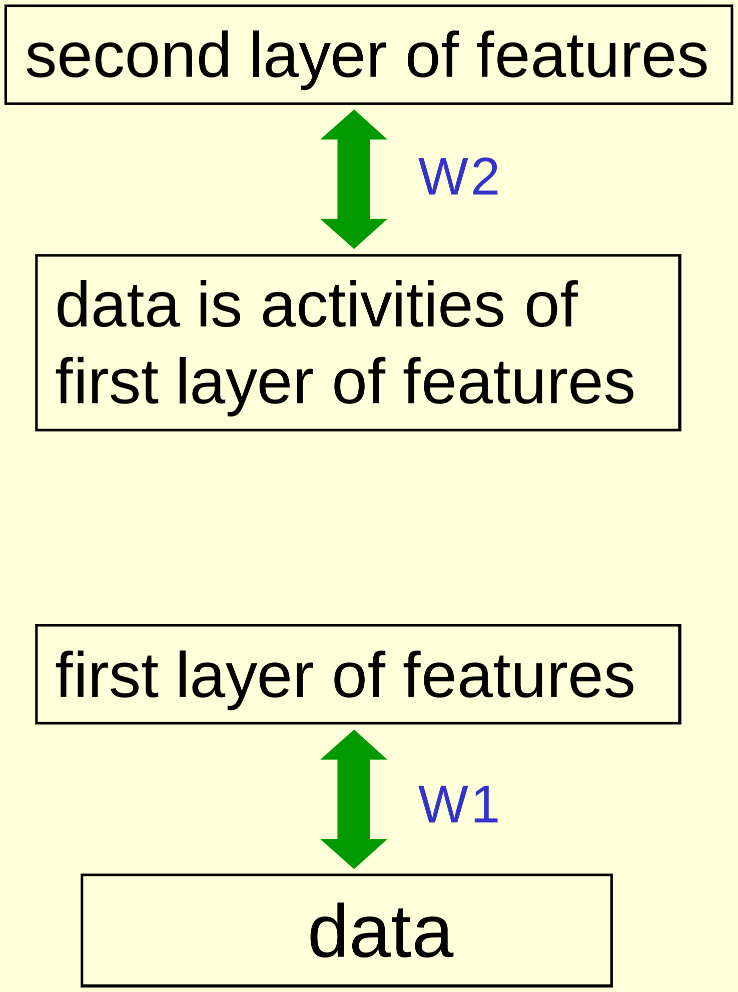


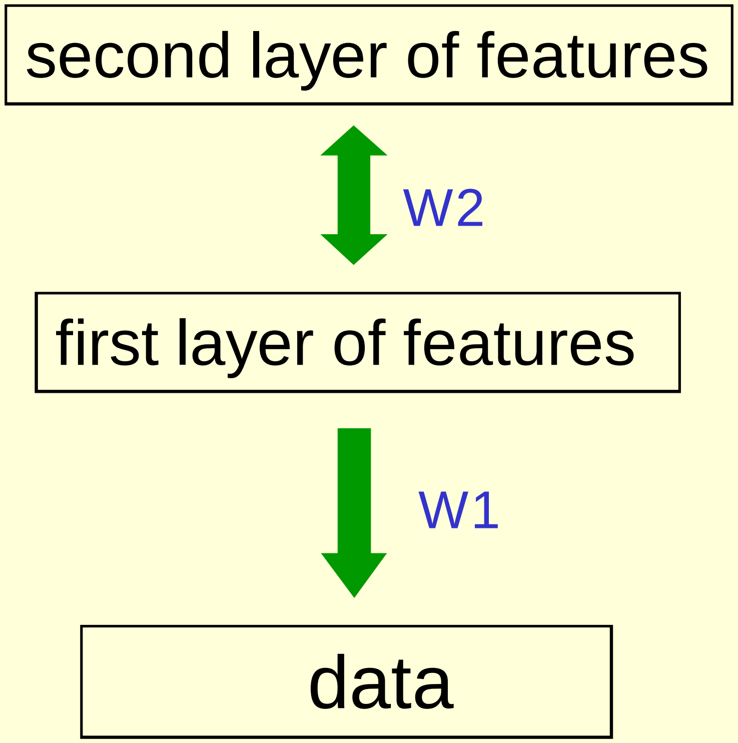
Stacking Restricted Boltzmann Machines

- 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 overall model produced by composing two RBM's



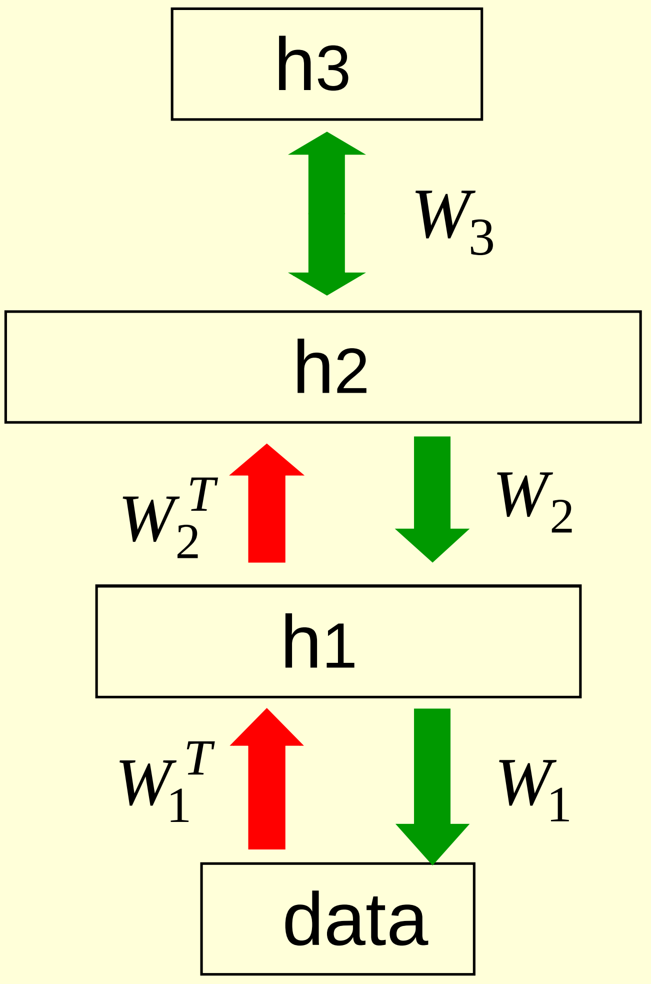


The generative model

- To generate data:

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

The lower-level, bottom-up connections are not part of the generative model. They are there to do fast approximate 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 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

$$p(v) = \sum_{h} \underset{\uparrow}{p(v,h)} = \sum_{h} p(h)\; \underset{\uparrow}{p(v\,|\,h)}$$

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 posterior hidden vectors produced by applying W to the data.

An analogy

- In a mixture model, we define the probability of a datavector to be

$$p(v) = \underbrace{\sum_{h}}_{\text{index over all Gaussians}} \underbrace{p(h)}_{\text{mixing proportion of Gaussian}} \; \underbrace{p(v \mid h)}_{\text{probability of v given Gaussian h}}$$

- 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).
    - The derivation of this guarantee is quite complicated.

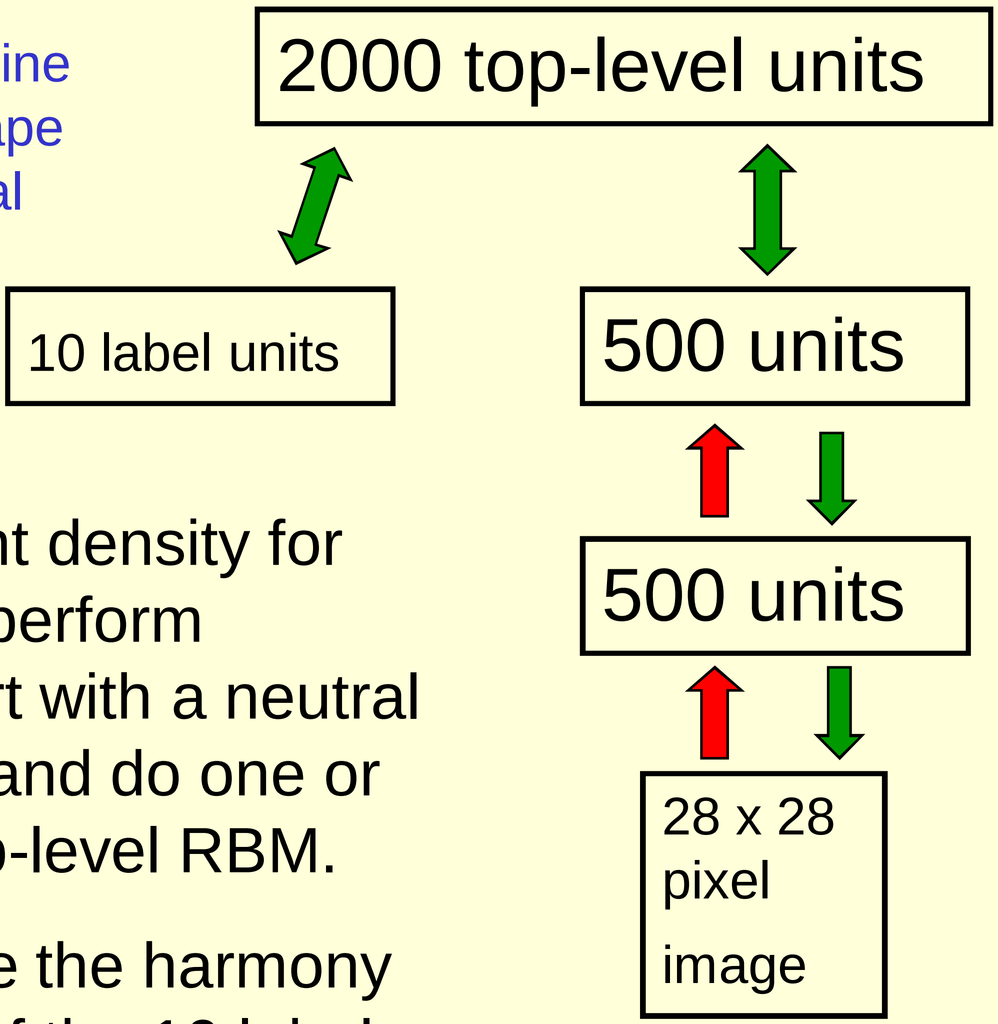
Back-fitting

- 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.

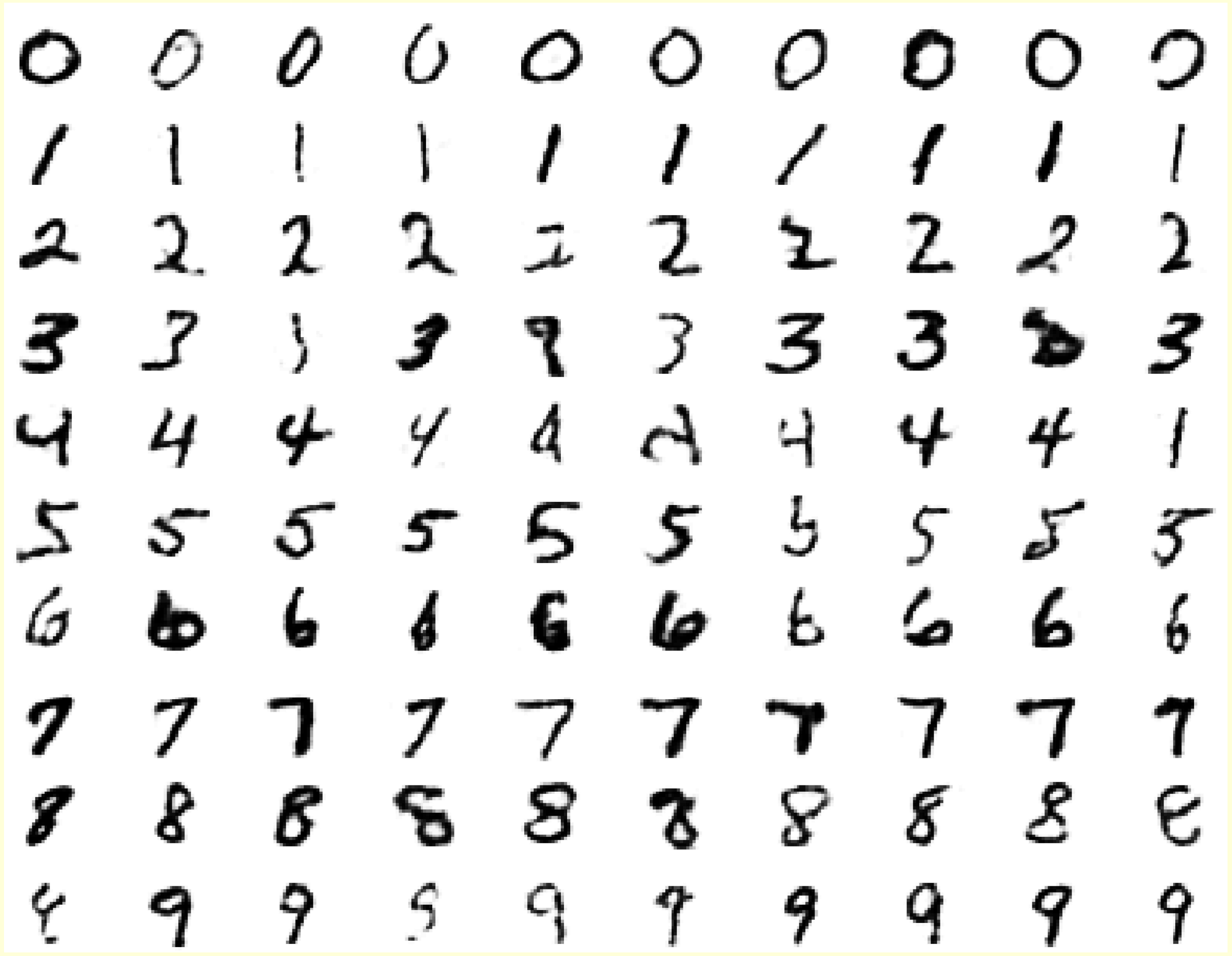
The valleys have names:



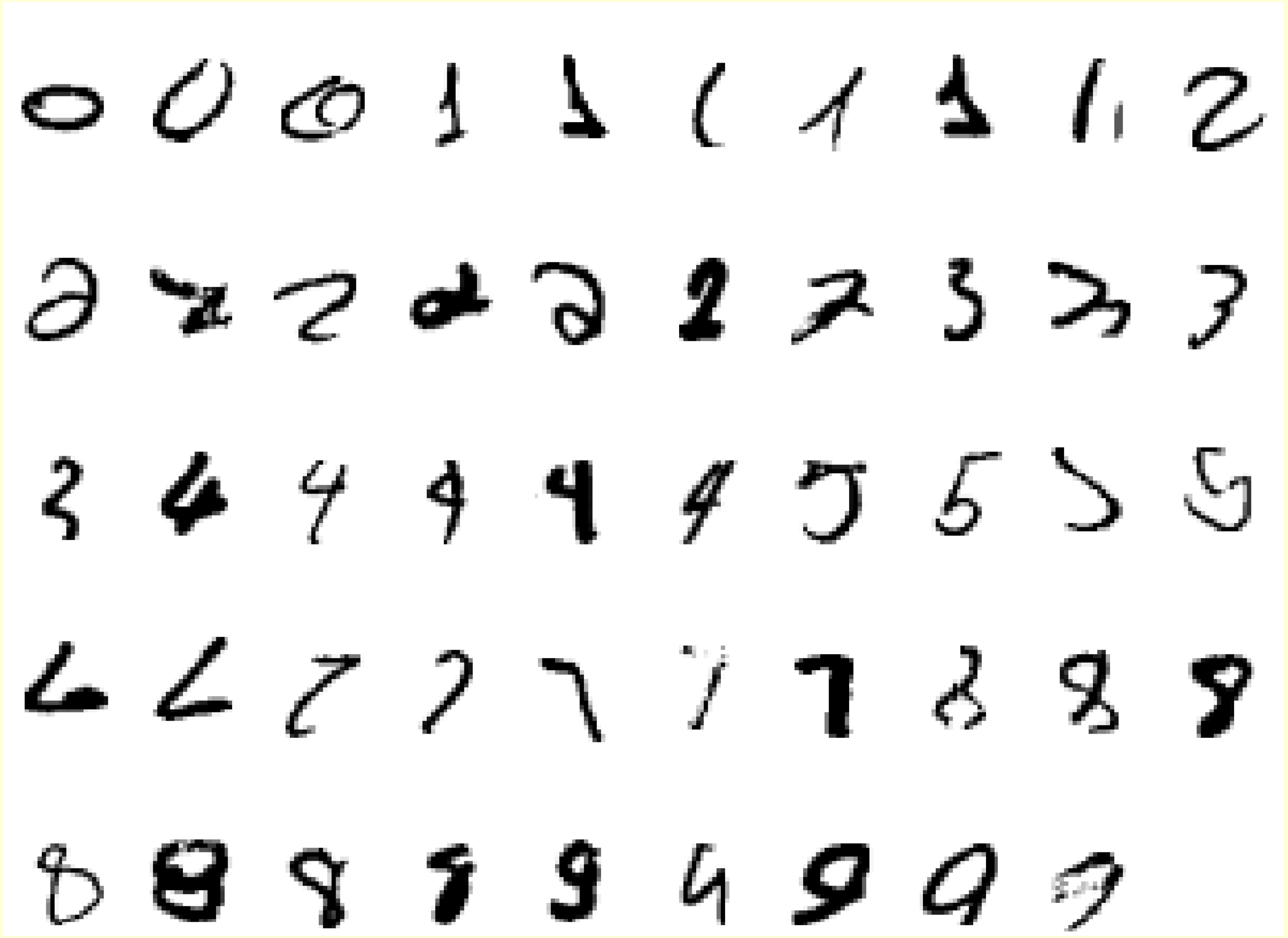
The model learns a joint density for labels and images. To perform recognition we can start with a neutral state of the label units and do one or two iterations of the top-level RBM.

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

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



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

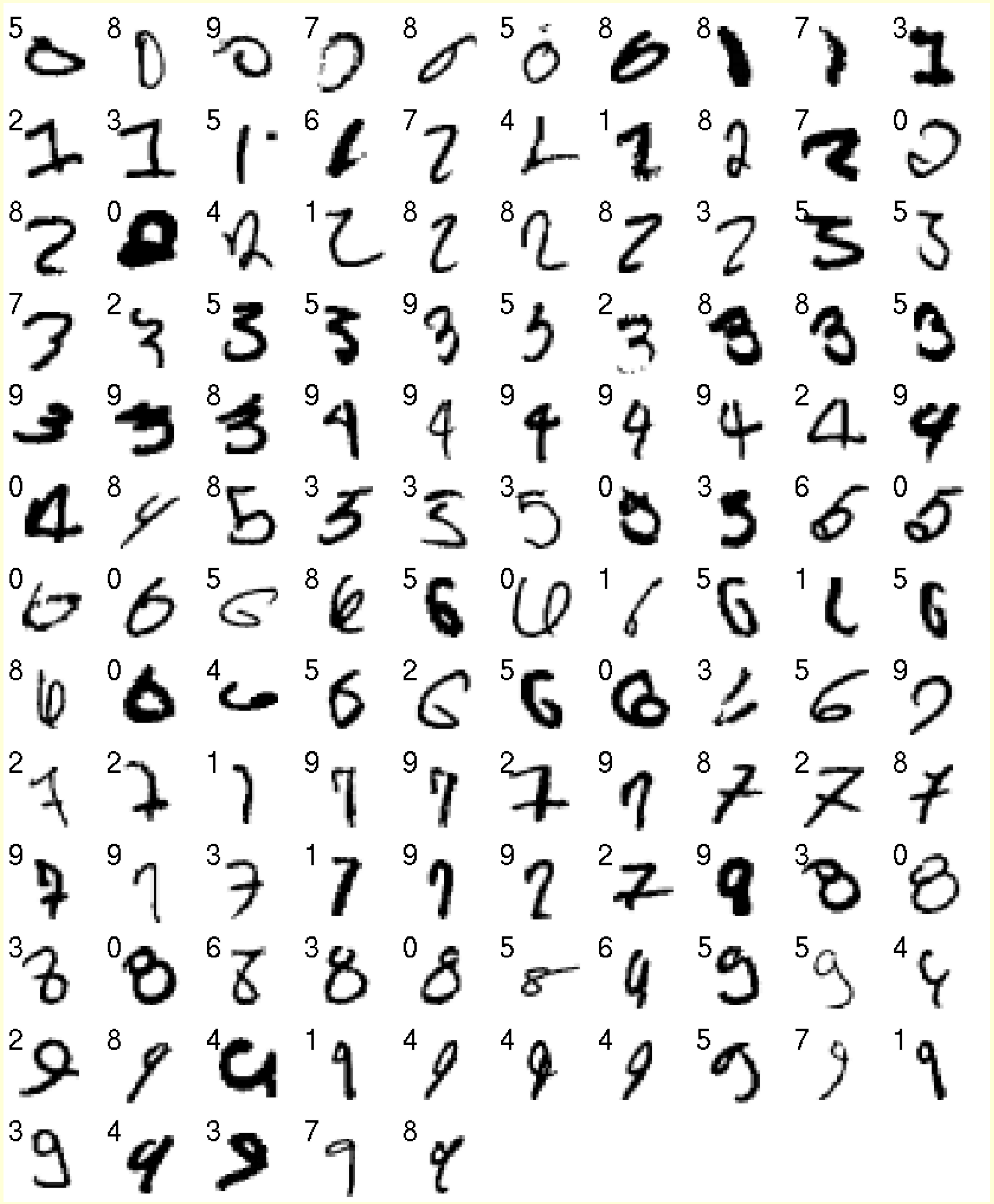


How well does it discriminate on MNIST test set with no extra information about geometric distortions?

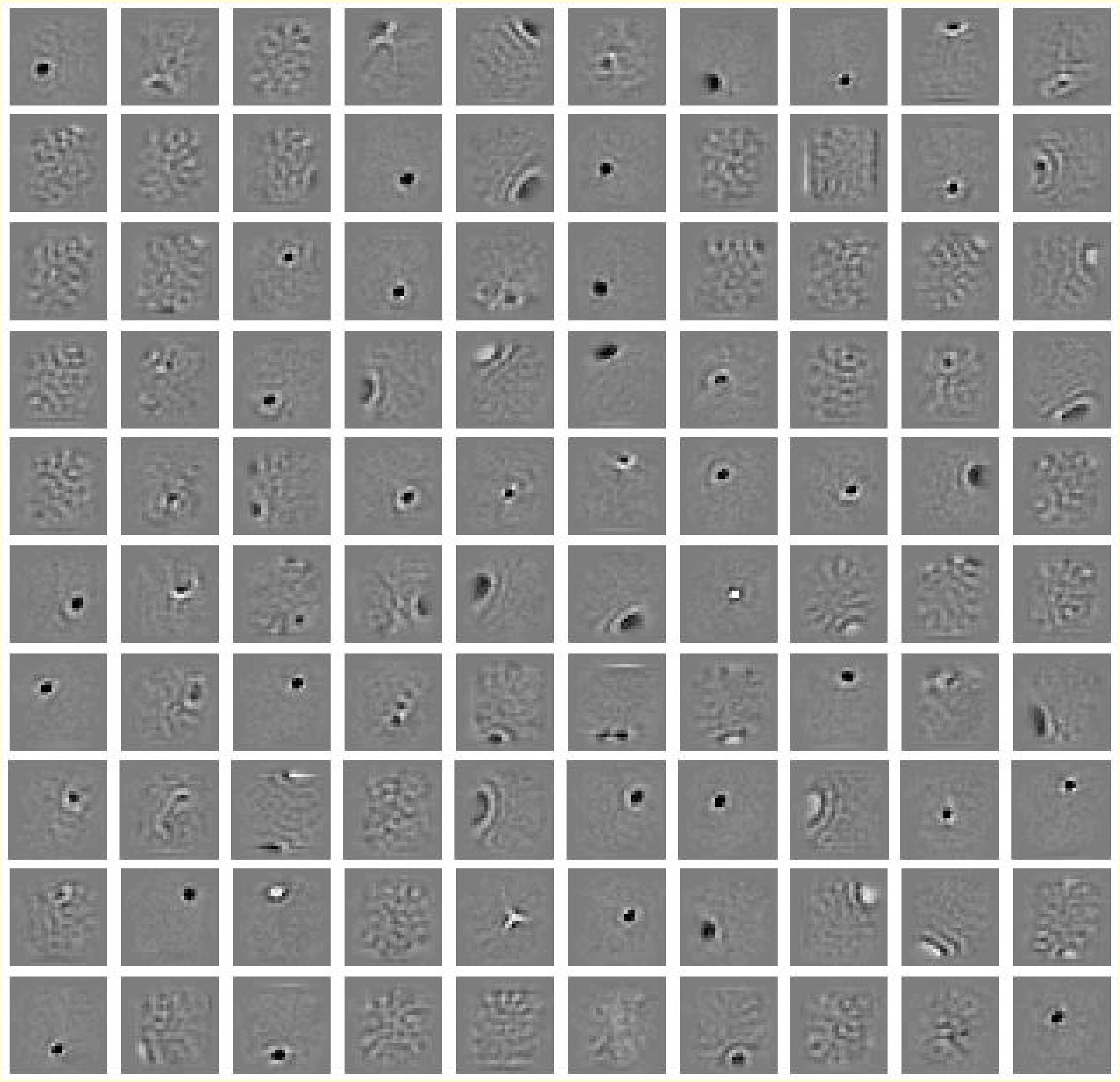
Up-down net with RBM pre-training + CD10	1.25%
SVM (Decoste & Scholkopf)	1.4%
Backprop with 1000 hiddens (Platt)	1.5%
Backprop with 500 -->300 hiddens	1.5%
Separate hierarchy of RBM’s per class	1.7%
Learned motor program extraction	~1.8%
K-Nearest Neighbor	~ 3.3%

- 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.

All 125 errors



The receptive fields of the first hidden layer



The generative fields of the first hidden layer