Neural Networks for Machine Learning

Lecture 12a
The Boltzmann Machine learning algorithm

Geoffrey Hinton
Nitish Srivastava,
Kevin Swersky
Tijmen Tieleman
Abdel-rahman Mohamed

The goal of learning

- We want to maximize the product of the probabilities that the Boltzmann machine assigns to the binary 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 cases 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
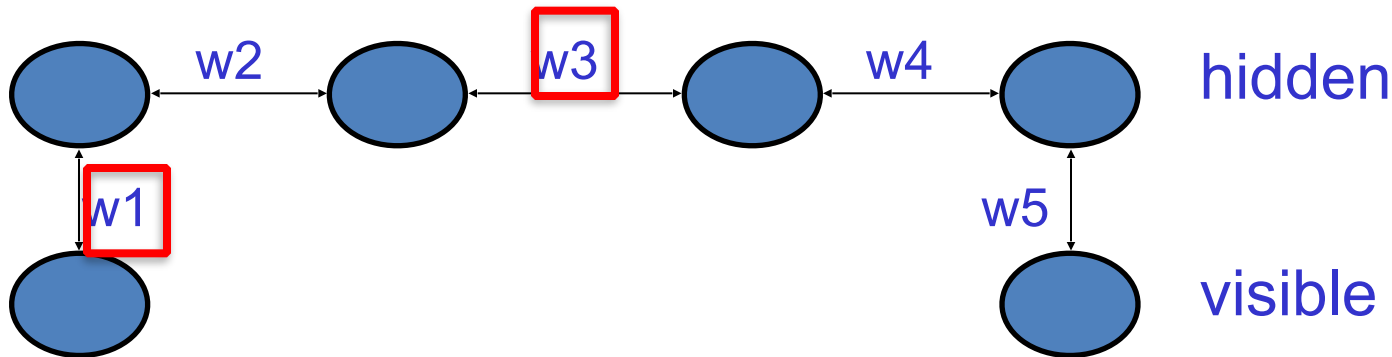
If the training set consists of (1,0) and (0,1) we want the product of all the weights to be negative.

So to know how to change w1 or w5 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

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

# 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.

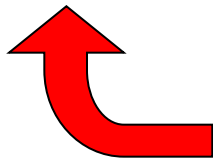- The energy is a linear function of the weights and states, so:

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

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

Why do we need the negative phase?
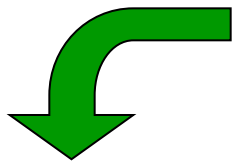
The positive phase finds hidden configurations that work well with v and lowers their energies.

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

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

An inefficient way to collect the statistics required 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.

Neural Networks for Machine Learning
Lecture 12b
More efficient ways to get the statistics
ADVANCED MATERIAL: NOT ON QUIZZES OR FINAL TEST

Geoffrey Hinton
Nitish Srivastava,
Kevin Swersky
Tijmen Tieleman
Abdel-rahman Mohamed

A better way of collecting the statistics

- If we start from a random state, it may take a long time to reach thermal equilibrium.
  - Also, its very hard to tell when we get there.
- Why not start from whatever state you ended up in last time you saw that datavector?
  - This stored state is called a "particle".

Using particles that persist to get a "warm start" has a big advantage:

- If we were at equilibrium last time and we only changed the weights a little, we should only need a few updates to get back to equilibrium.

Neal's method for collecting the statistics (Neal 1992)

- Positive phase: Keep a set of "data-specific particles", one per training case. Each particle has a current value that is a configuration of the hidden units.

  - Sequentially update all the hidden units a few times in each particle with the relevant datavector clamped.

  - For every connected pair of units, average $s_i s_j$ over all the data-specific particles.

- Negative phase: Keep a set of "fantasy particles". Each particle has a value that is a global configuration.

  - Sequentially update all the units in each fantasy particle a few times.

  - For every connected pair of units, average $s_i s_j$ over all the fantasy particles.

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

Adapting Neal's approach to handle mini-batches

- Neal's approach does not work well with mini-batches.
  - By the time we get back to the same datavector again, the weights will have been updated many times.
  - But the data-specific particle will not have been updated so it may be far from equilibrium.

- A strong assumption about how we understand the world:
  - When a datavector is clamped, we will assume that the set of good explanations (i.e. hidden unit states) is uni-modal.
  - i.e. we restrict ourselves to learning models in which one sensory input vector does not have multiple very different explanations.

The simple mean field approximation

- If we want to get the statistics right, we need to update the units stochastically and sequentially.

- But if we are in a hurry we can use probabilities instead of binary states and update the units in parallel.

- To avoid biphasic oscillations we can use damped mean field.

$$prob(s_i = 1) \quad = \quad \sigma\left(b_i + \sum_j s_j\, w_{ij}\right)$$

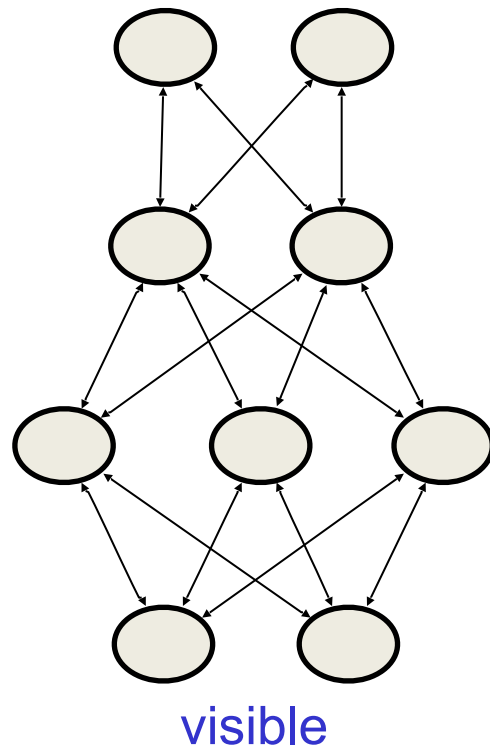$$p_i^{t+1} \quad = \quad \sigma\left(b_i + \sum_j p_j^t w_{ij}\right)$$

$$p_i^{t+1} \quad = \quad \lambda\, p_i^t \; + \; (1-\lambda)\,\sigma\left(b_i + \sum_j p_j^t w_{ij}\right)$$

An efficient mini-batch learning procedure for Boltzmann Machines (Salakhutdinov & Hinton 2012)

- **Positive phase:** Initialize all the hidden probabilities at 0.5.
  - Clamp a datavector on the visible units.
  - Update all the hidden units in parallel until convergence using mean field updates.
  - After the net has converged, record $p_i p_j$ for every connected pair of units and average this over all data in the mini-batch.

- **Negative phase:** Keep a set of "fantasy particles". Each particle has a value that is a global configuration.
  - Sequentially update all the units in each fantasy particle a few times.
  - For every connected pair of units, average $s_i s_j$ over all the fantasy particles.
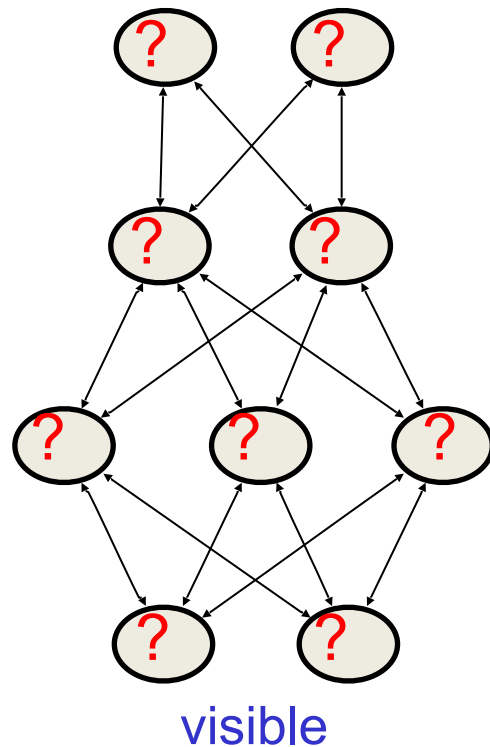
Making the updates more parallel

- In a general Boltzmann machine, the stochastic updates of units need to be sequential.

- There is a special architecture that allows alternating parallel updates which are much more efficient:
  - No connections within a layer.
  - No skip-layer connections.

- This is called a Deep Boltzmann Machine (DBM)
  - It's a general Boltzmann machine with a lot of missing connections.

visible

Making the updates more parallel

- In a general Boltzmann machine, the stochastic updates of units need to be sequential.

- There is a special architecture that allows alternating parallel updates which are much more efficient:

  - No connections within a layer.

  - No skip-layer connections.

- This is called a Deep Boltzmann Machine (DBM)

  - It's a general Boltzmann machine with a lot of missing connections.

visible

Can a DBM learn a good model of the MNIST digits?

## Do samples from the model look like real data?
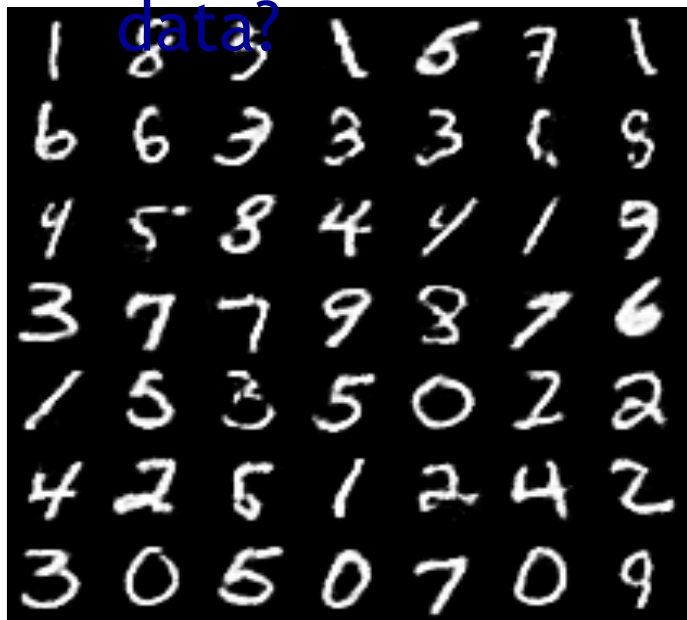
A puzzle

- Why can we estimate the "negative phase statistics" well with only 100 negative examples to characterize the whole space of possible configurations?

  – For all interesting problems the GLOBAL configuration space is highly multi-modal.

  – How does it manage to find and represent all the modes with only 100 particles?

The learning raises the effective mixing rate.

- The learning interacts with the Markov chain that is being used to gather the "negative statistics" (*i.e.* the data-independent statistics).
  - We cannot analyse the learning by viewing it as an outer loop and the gathering of statistics as an inner loop.

- Wherever the fantasy particles outnumber the positive data, the energy surface is raised.
  - This makes the fantasies rush around hyperactively.
  - They move around MUCH faster than the mixing rate of the Markov chain defined by the static current weights.

How fantasy particles move between the model's modes

- If a mode has more fantasy particles than data, the energy surface is raised until the fantasy particles escape.
  - This can overcome energy barriers that would be too high for the Markov chain to jump in a reasonable time.
- The energy surface is being changed to help mixing in addition to defining the model.
- Once the fantasy particles have filled in a hole, they rush off somewhere else to deal with the next problem.
  - They are like investigative journalists.
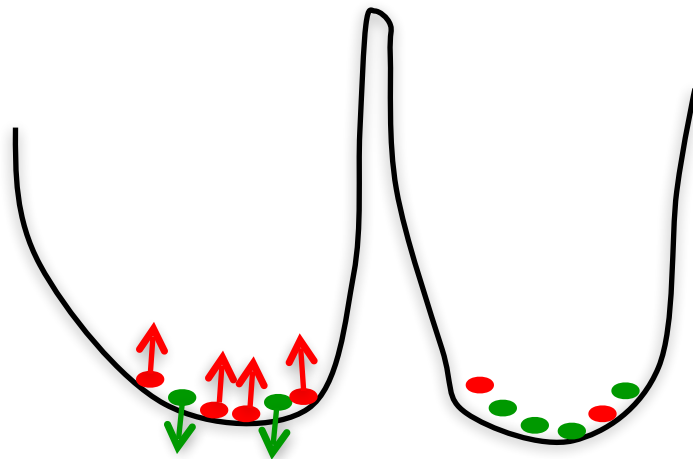
This minimum will get filled in by the learning until the fantasy particles escape.

Neural Networks for Machine Learning

Lecture 12c
Restricted Boltzmann Machines

Geoffrey Hinton

Nitish Srivastava,

Kevin Swersky

Tijmen Tieleman

Abdel-rahman Mohamed

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 it only takes one step to reach thermal equilibrium when the visible units are clamped.
 - So we can quickly get the exact value of :

$$< v_i h_j >_{\mathbf{v}}$$

hidden
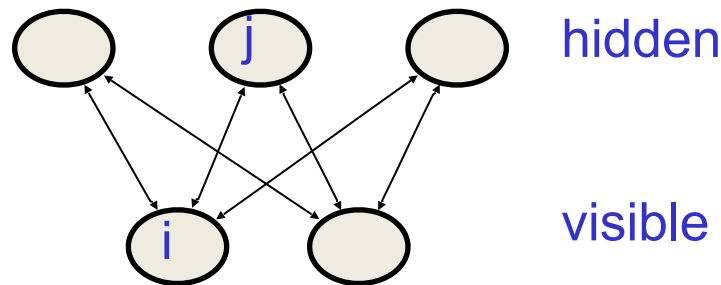
visible

$$p(h_j = 1) = \frac{1}{1 + e^{-\left(b_j + \sum_{i \in vis} v_i w_{ij}\right)}}$$

PCD: An efficient mini-batch learning procedure for Restricted Boltzmann Machines (Tieleman, 2008)

- **Positive phase:** Clamp a datavector on the visible units.
  - Compute the exact value of $< v_i h_j >$ for all pairs of a visible and a hidden unit.
  - For every connected pair of units, average $< v_i h_j >$ over all data in the mini-batch.

- **Negative phase:** Keep a set of "fantasy particles". Each particle has a value that is a global configuration.
  - Update each fantasy particle a few times using alternating parallel updates.
 - For every connected pair of units, average $v_i h_j$ over all the fantasy particles.

A picture of an inefficient version of the Boltzmann machine learning algorithm for an RBM

$<v_i h_j>^0$

$<v_i h_j>^\infty$

a fantasy

t = 0   t = 1   t = 2   t = infinity
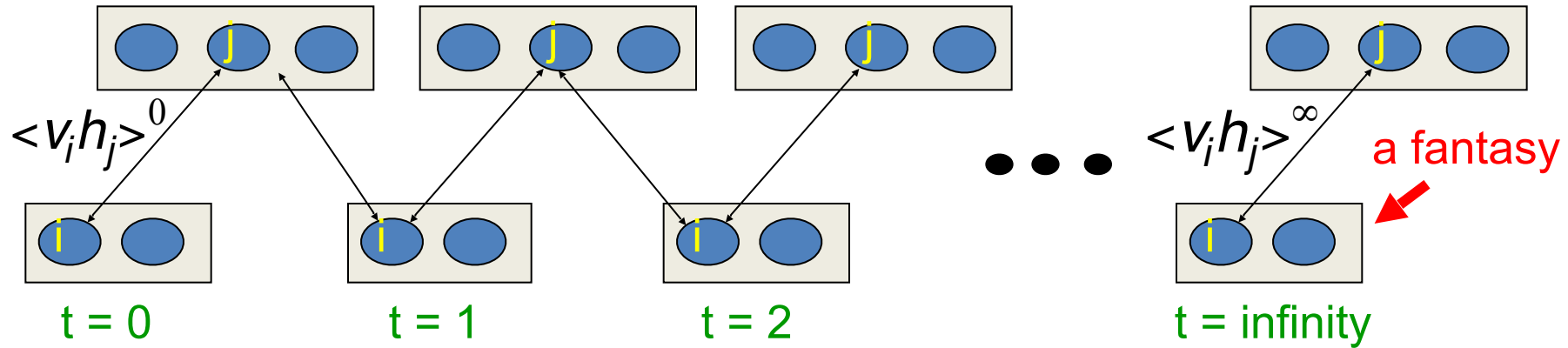
Start with a training vector on the visible units. Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

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

Contrastive divergence: A very surprising short-cut
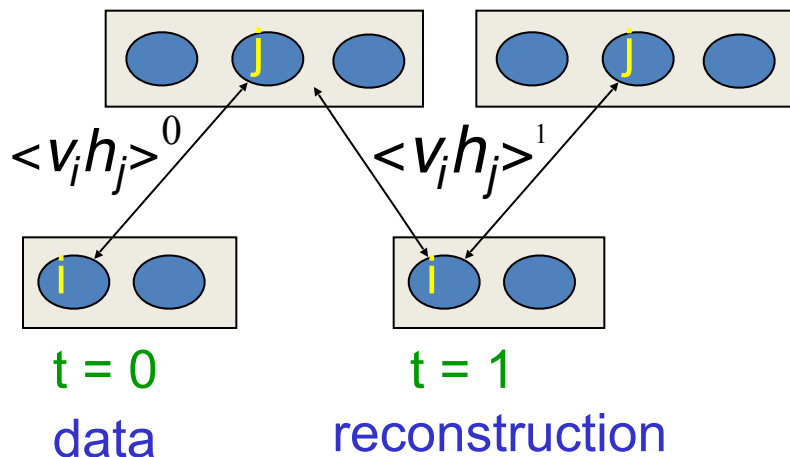
$<v_i h_j>^0$ $<v_i h_j>^1$

t = 0   t = 1

data   reconstruction

Start with a training vector on the visible units.

Update all the hidden units in parallel.

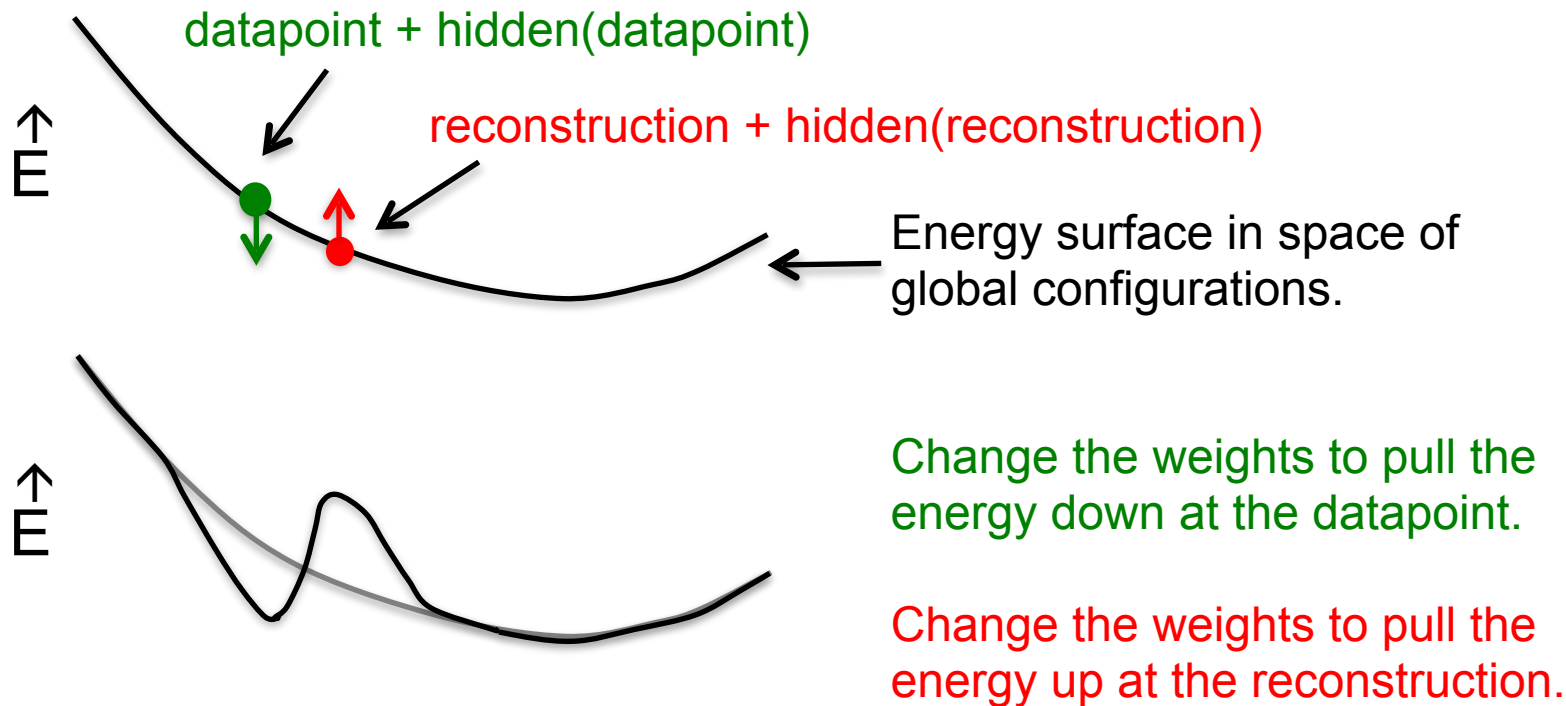Update the all the visible units in parallel to get a "reconstruction".

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.

Why does the shortcut work?

- If we start at the data, the Markov chain wanders away from the data and towards things that it likes more.
  - We can see what direction it is wandering in after only a few steps.
  - When we know the weights are bad, it is a waste of time to let it go all the way to equilibrium.

- All we need to do is lower the probability of the confabulations it produces after one full step and raise the probability of the data.
  - Then it will stop wandering away.
  - The learning cancels out once the confabulations and the data have the same distribution.

A picture of contrastive divergence learning

datapoint + hidden(datapoint)

reconstruction + hidden(reconstruction)

E ↑

Energy surface in space of global configurations.

E ↑

Change the weights to pull the energy down at the datapoint.

Change the weights to pull the energy up at the reconstruction.

When does the shortcut fail?

- We need to worry about regions of the data-space that the model likes but which are very far from any data.

  – These low energy holes cause the normalization term to be big and we cannot sense them if we use the shortcut.

  – Persistent particles would eventually fall into a hole, cause it to fill up then move on to another hole.

- A good compromise between speed and correctness is to start with small weights and use CD1 (*i.e.* use one full step to get the "negative data").

  – Once the weights grow, the Markov chain mixes more slowly so we use CD3.

  – Once the weights have grown more we use CD10.

Neural Networks for Machine Learning

Lecture 12d
An example of Contrastive Divergence Learning

Geoffrey Hinton
Nitish Srivastava,
Kevin Swersky
Tijmen Tieleman
Abdel-rahman Mohamed

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

| 50 binary neurons that learn features		50 binary neurons that learn features

Increment weights
between an active pixel
and an active feature

Decrement weights
between an active pixel
and an active feature

data
(reality)

16 x 16
pixel
image

16 x 16
pixel
image

reconstruction
(better than reality)
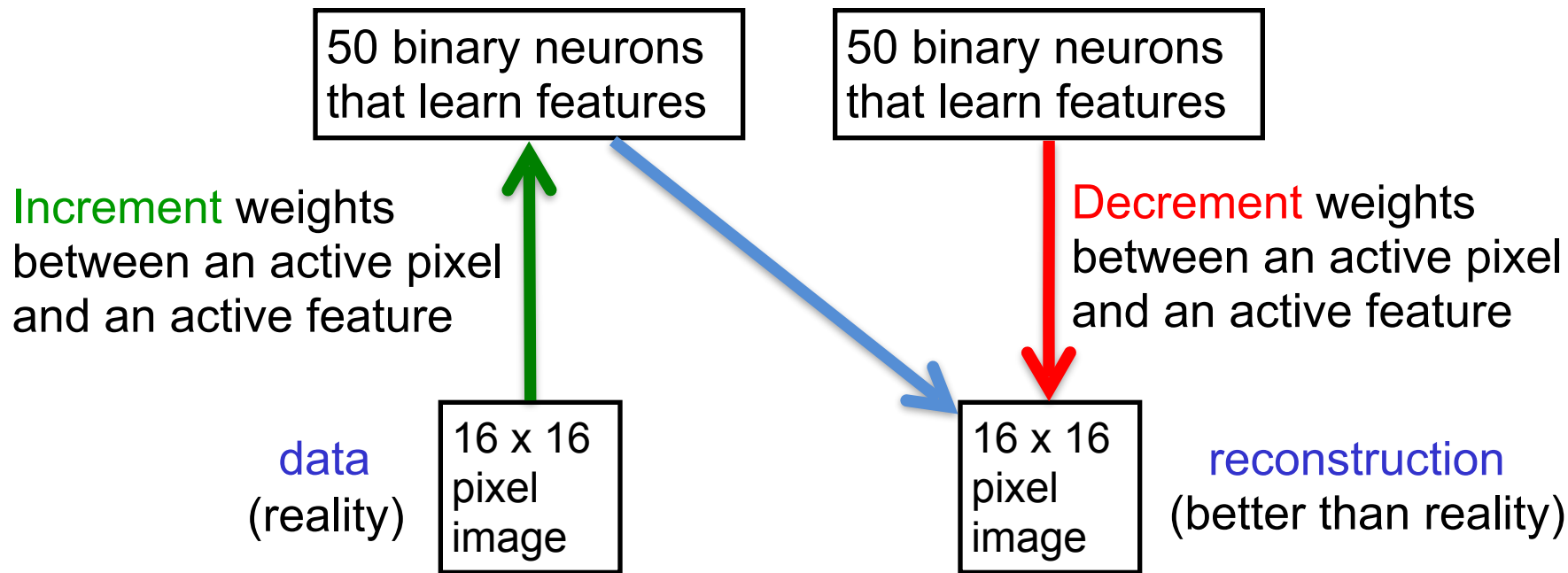
The weights of the 50 feature detectors
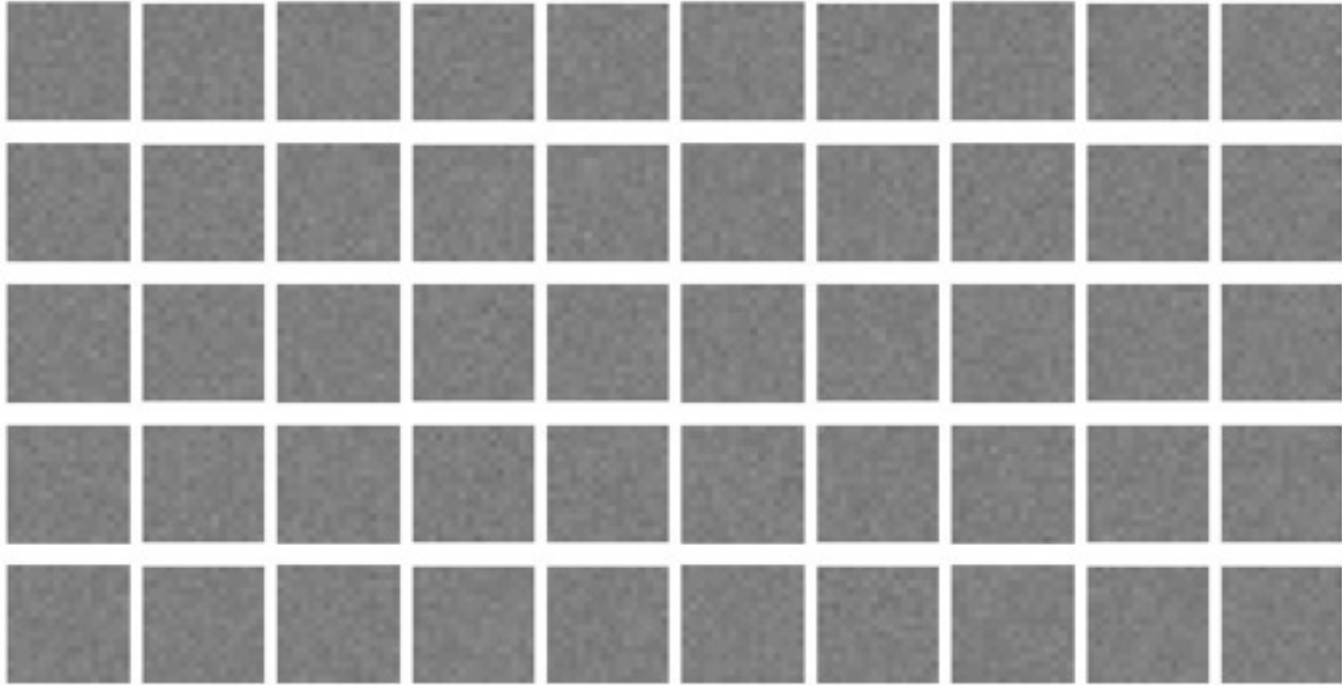
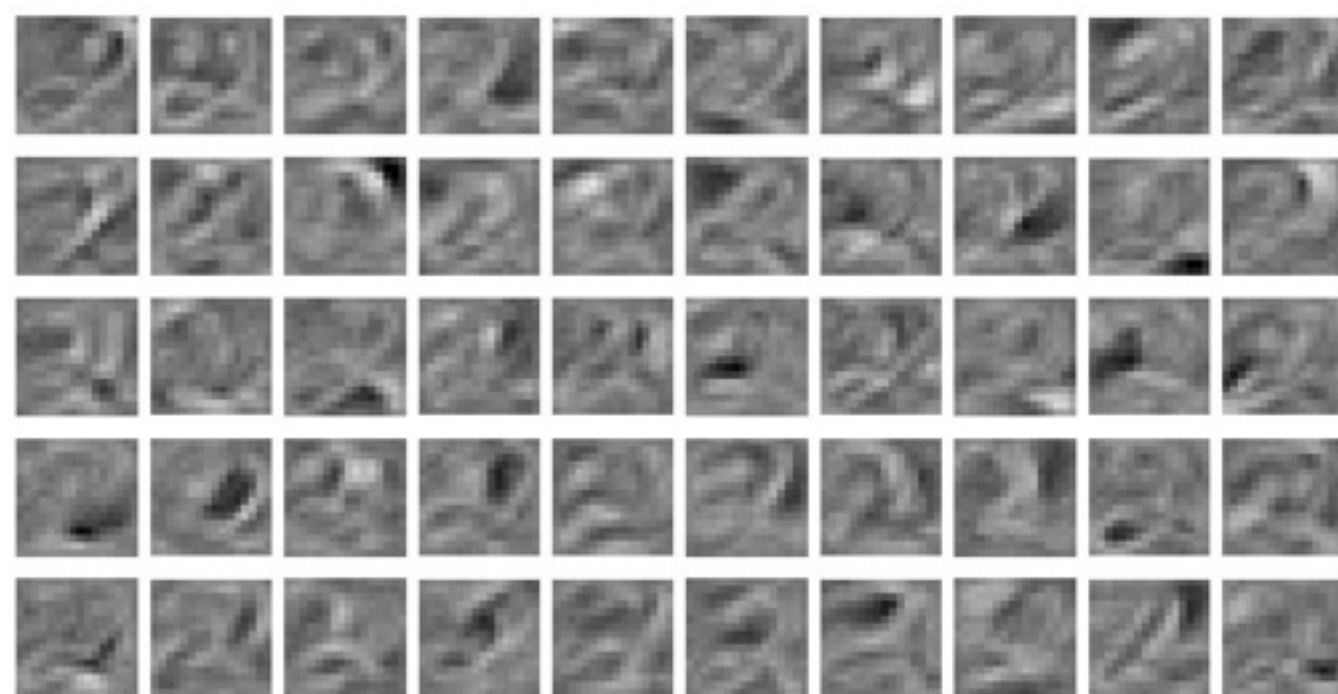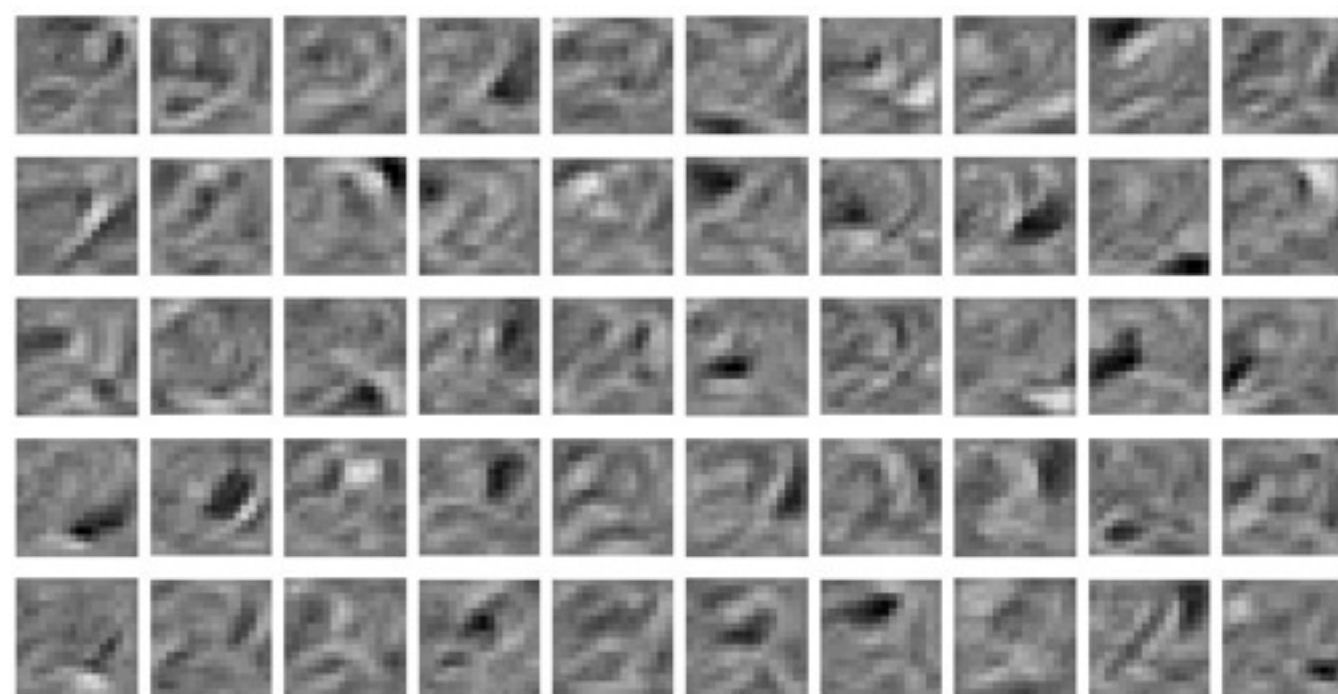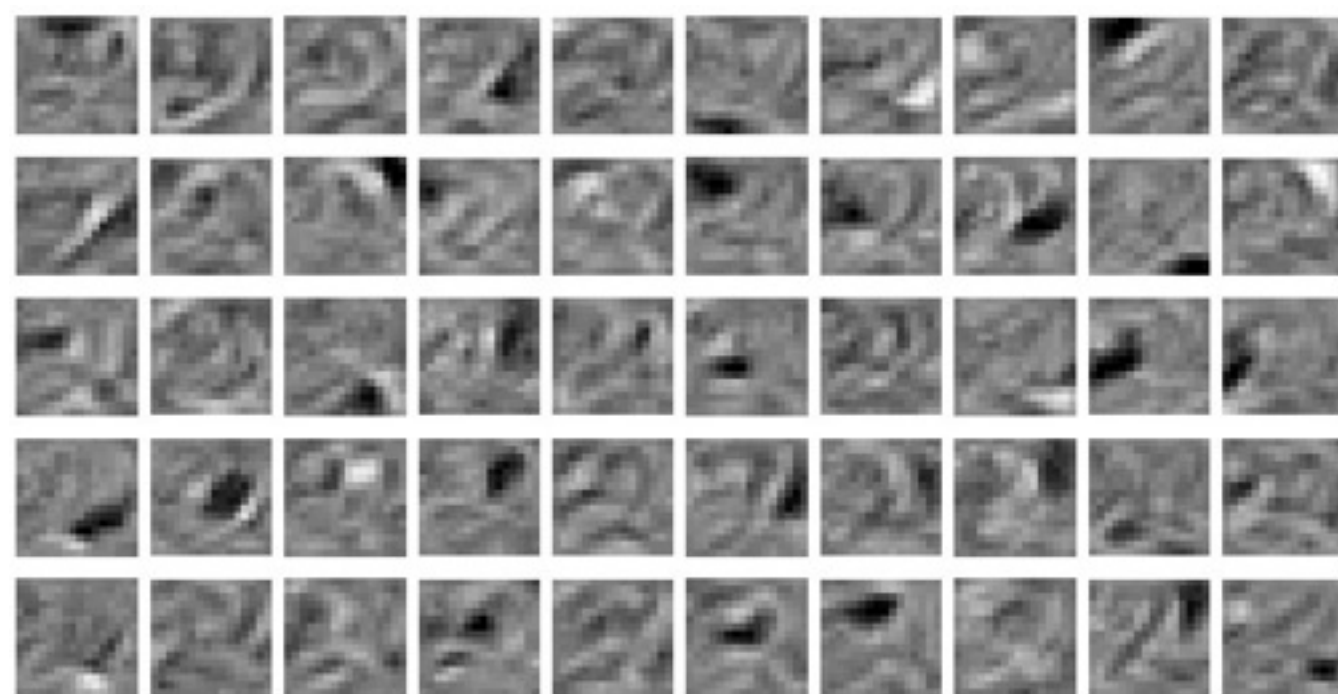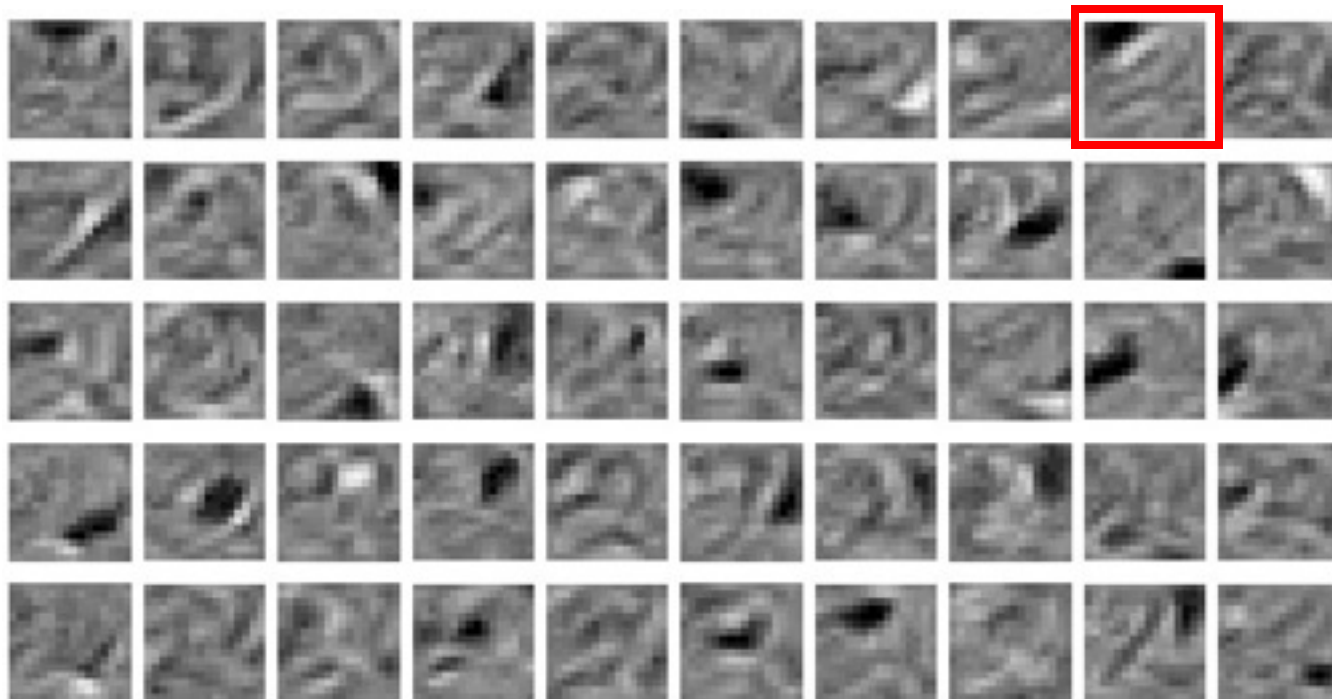We start with small random weights to break symmetry
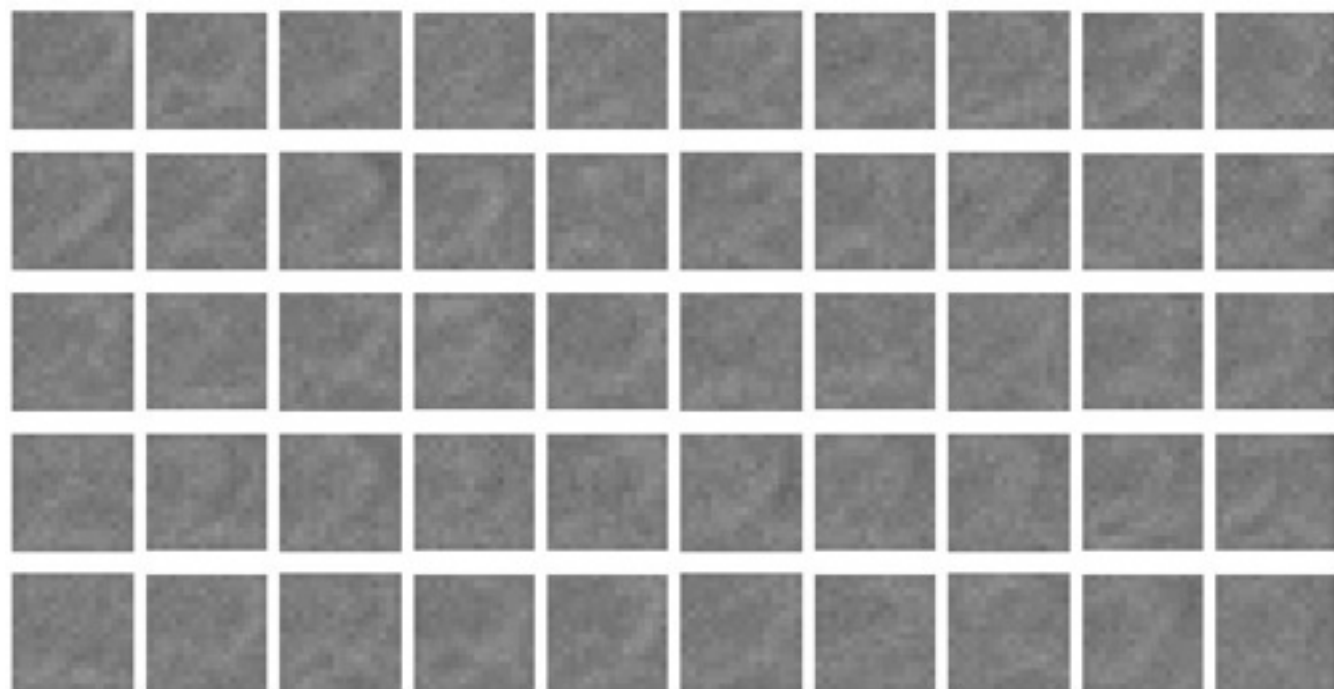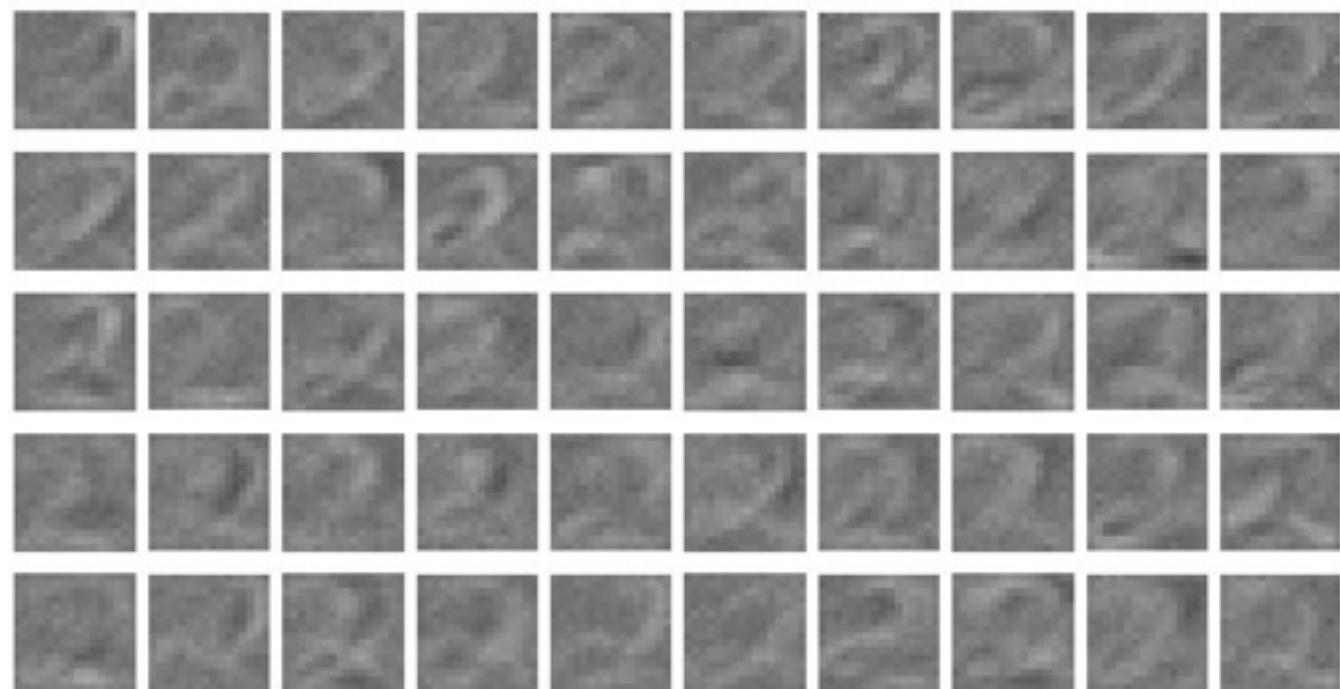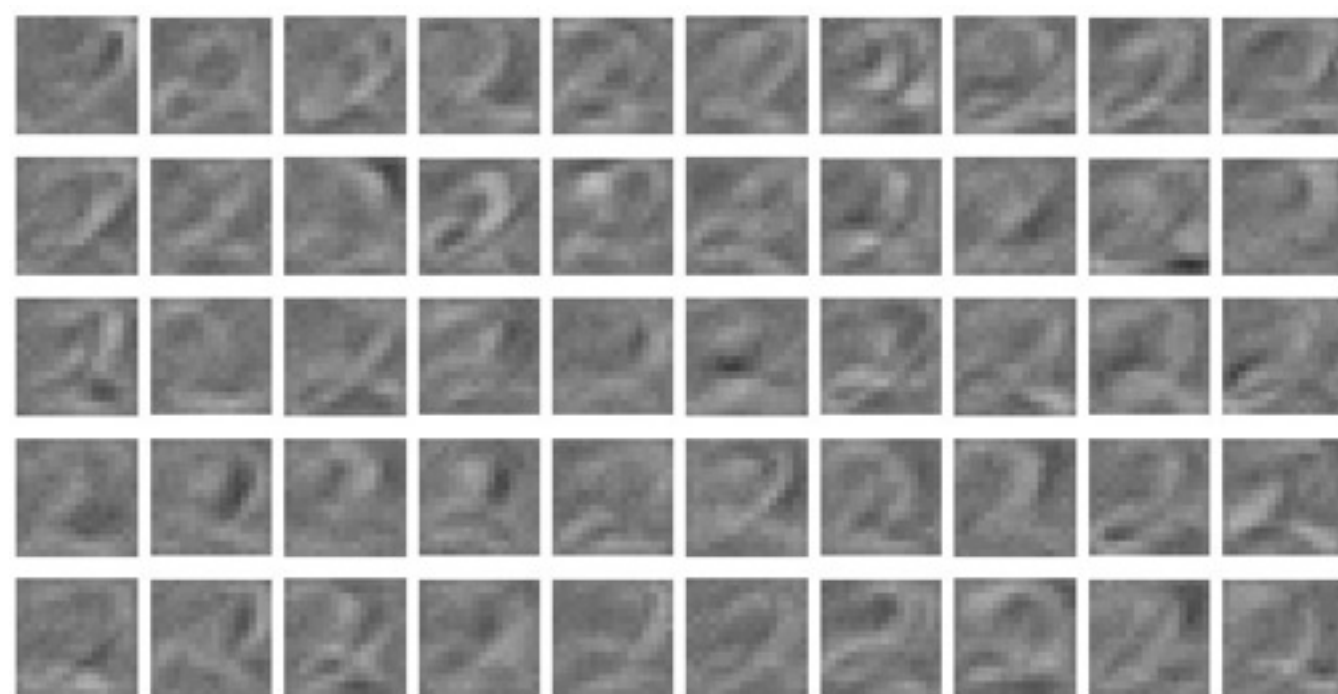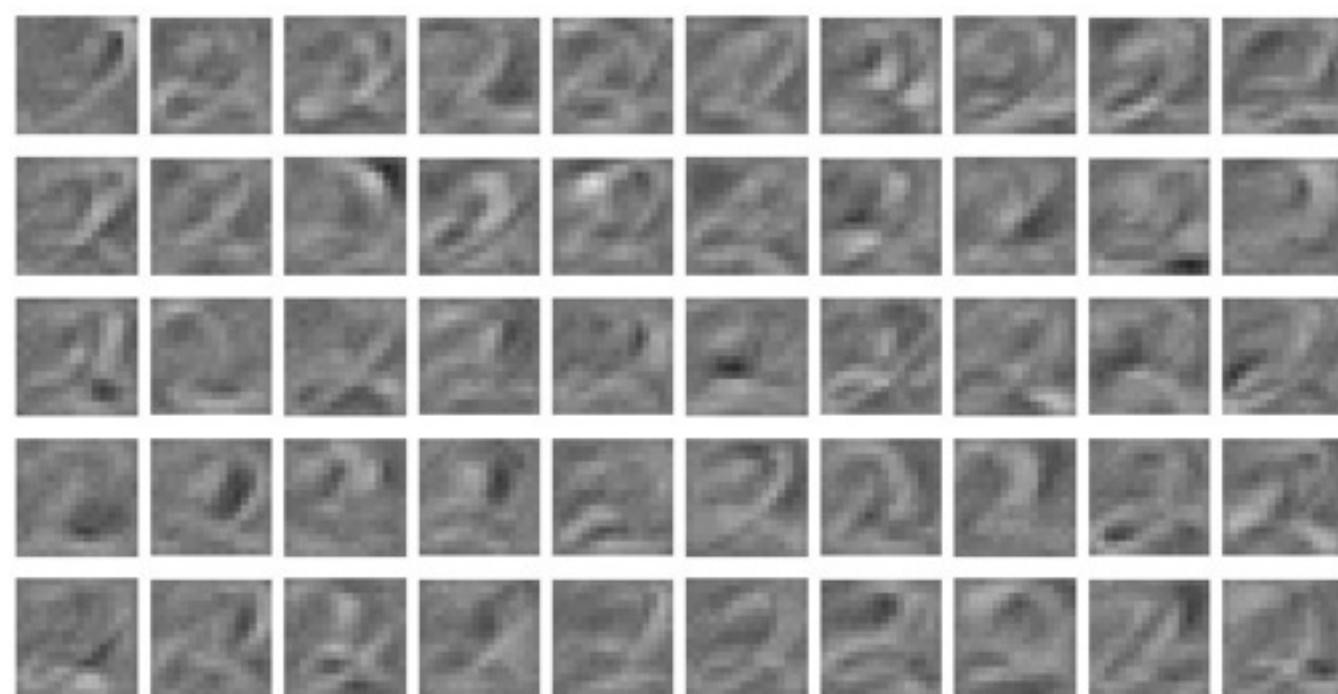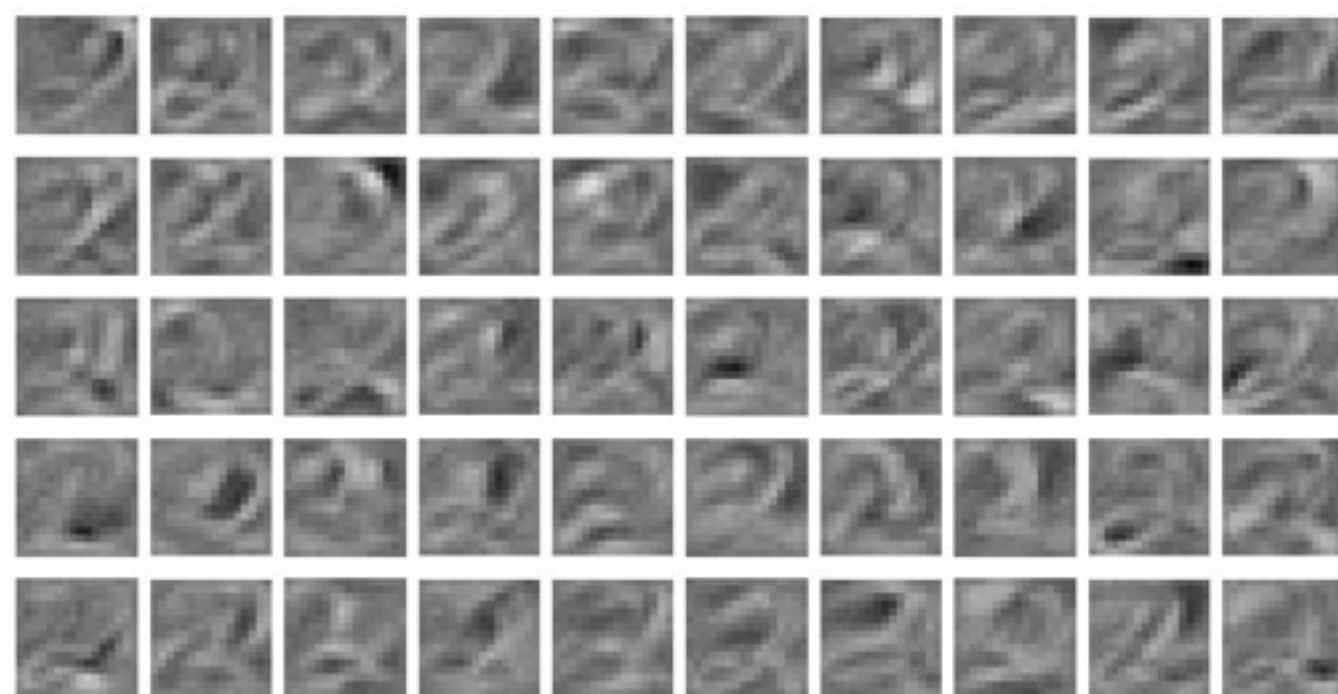
The final 50 x 256 weights: Each neuron grabs a different feature

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

Data → Reconstruction from activated binary features

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

Data → Reconstruction from activated binary features

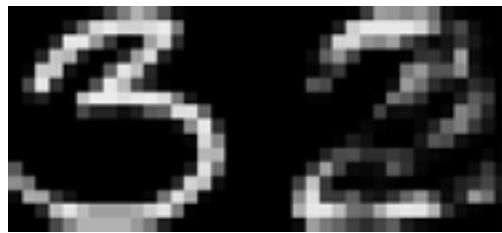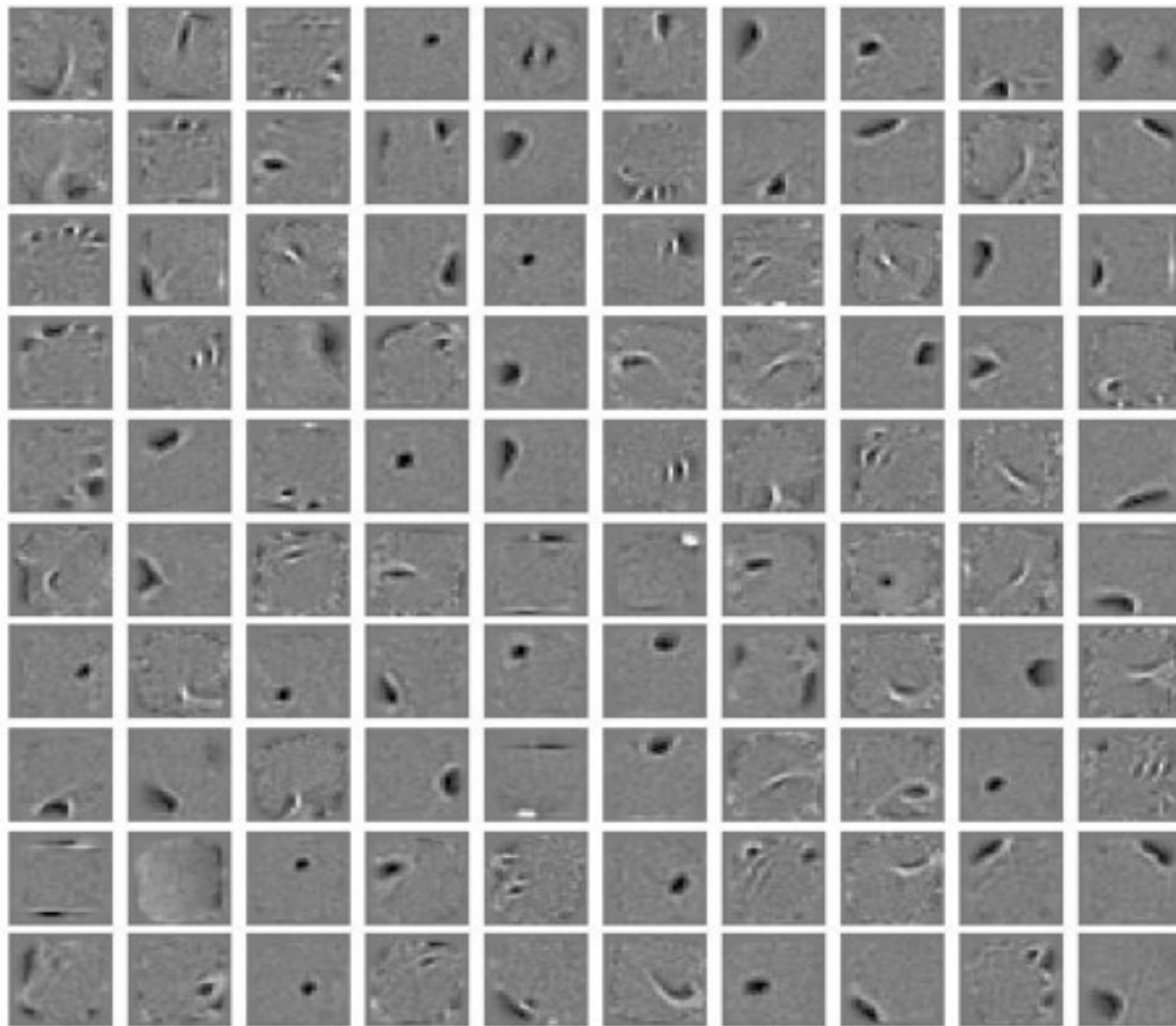Image from an unfamiliar digit class The network tries to see every image as a 2.

Some features learned in the first hidden layer of a model of all 10 digit classes using 500 hidden units.

Neural Networks for Machine Learning

Lecture 12e
RBMs for collaborative filtering

Geoffrey Hinton

Nitish Srivastava,

Kevin Swersky

Tijmen Tieleman

Abdel-rahman Mohamed

Collaborative filtering: The Netflix competition

- You are given most of the ratings that half a million Users gave to 18,000 Movies on a scale from 1 to 5.
  - Each user only rates a small fraction of the movies.
- You have to predict the ratings users gave to the held out movies.
  - If you win you get $1000,000

	M1	M2	M3	M4	M5	M6
U1				3		
U2	5		1			
U3		3	5			
U4	4		?			5
U5			4			
U6					2	

Lets use a "language model"

The data is strings of triples of the form: User, Movie, rating.

U2 M1 5

U2 M3 1

U4 M1 4

U4 M3 ?

All we have to do is to predict the next "word" well and we will get rich.

rating

scalar product

U4 feat

M3 feat

U4

M3

M3 feat

U4 feat

3.1

matrix factorization
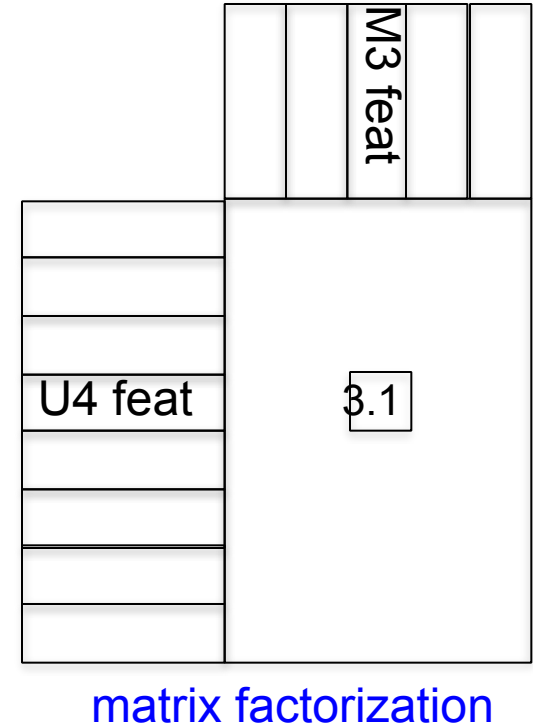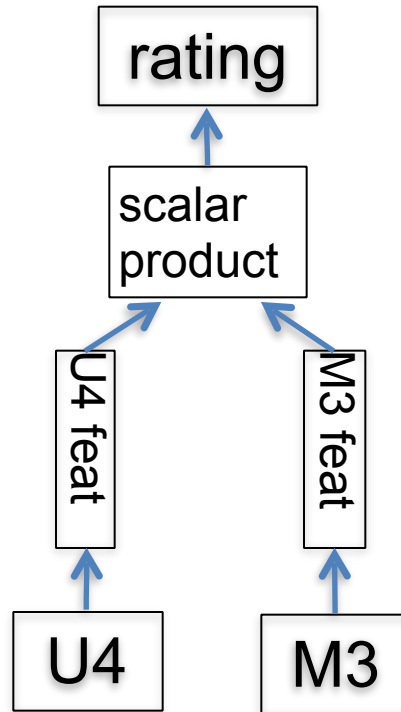
An RBM alternative to matrix factorization

- Suppose we treat each user as a training case.
  - A user is a vector of movie ratings.
  - There is one visible unit per movie and its a 5-way softmax.
  - The CD learning rule for a softmax is the same as for a binary unit.
  - There are ~100 hidden units.
- One of the visible values is unknown.
  - It needs to be filled in by the model.

about 100 binary hidden units

M1 M2 M3 M4 M5 M6 M7 M8
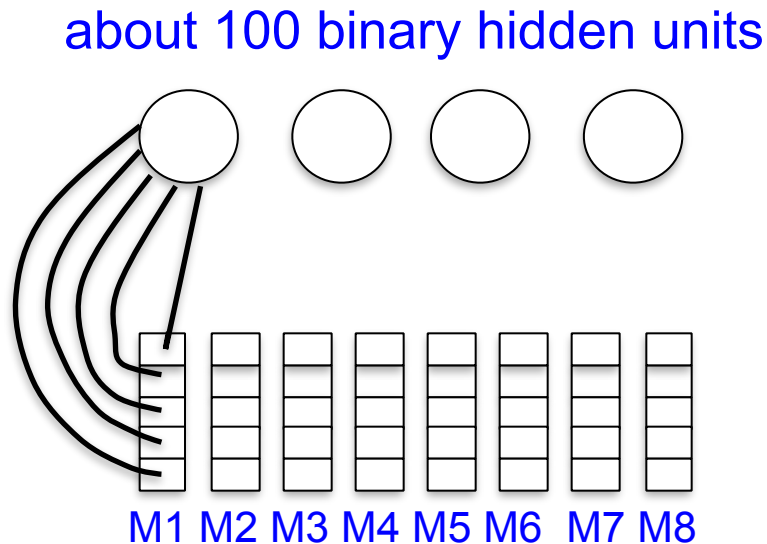
How to avoid dealing with all those missing ratings

- For each user, use an RBM that only has visible units for the movies the user rated.
- So instead of one RBM for all users, we have a different RBM for every user.
  - All these RBMs use the same hidden units.
  - The weights from each hidden unit to each movie are shared by all the users who rated that movie.

- Each user-specific RBM only gets one training case!
  - But the weight-sharing makes this OK.
- The models are trained with CD1 then CD3, CD5 & CD9.

How well does it work? (Salakhutdinov *et al.* 2007)

- RBMs work about as well as matrix factorization methods, but they give very different errors.

  – So averaging the predictions of RBMs with the predictions of matrix-factorization is a big win.

- The winning group used multiple different RBM models in their average of over a hundred models.

  – Their main models were matrix factorization and RBMs (I think).