1

The next generation of neural networks

Geoffrey Hinton
Canadian Institute for Advanced Research
&
University of Toronto

2

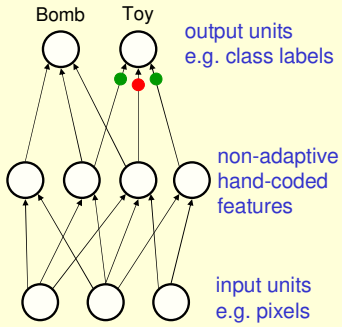
The main aim of neural networks

- People are much better than computers at recognizing patterns. How do they do it?
  - Neurons in the perceptual system represent features of the sensory input.
  - The brain learns to extract many layers of features. Features in one layer represent combinations of simpler features in the layer below.
- Can we train computers to extract many layers of features by mimicking the way the brain does it?
  - Nobody knows how the brain does it, so this requires both engineering insights and scientific discoveries.

3

First generation neural networks

- Perceptrons (~1960) used a layer of hand-coded features and tried to recognize objects by learning how to weight these features.
  - There was a neat learning algorithm for adjusting the weights.
  - But perceptrons are fundamentally limited in what they can learn to do.

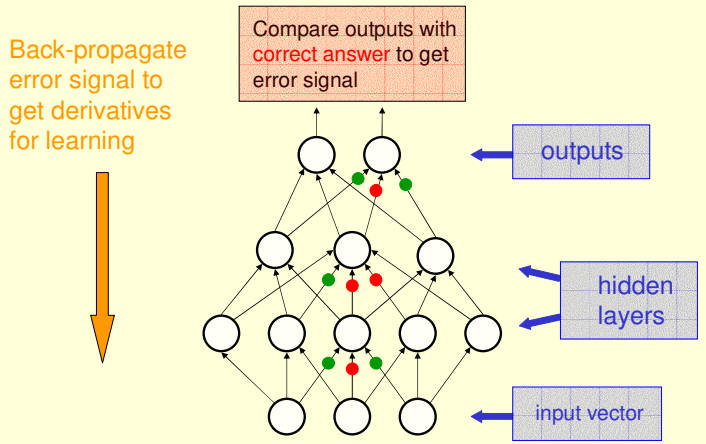


Sketch of a typical perceptron from the 1960's

4

Second generation neural networks (~1985)

Back-propagate error signal to get derivatives for learning



5

A temporary digression

- Vapnik and his co-workers developed a very clever type of perceptron called a Support Vector Machine.
  - Instead of hand-coding the layer of non-adaptive features, each training example is used to create a new feature using a fixed recipe.
    - The feature computes how similar a test example is to that training example.
  - Then a clever optimization technique is used to select the best subset of the features and to decide how to weight each feature when classifying a test case.
    - But its just a perceptron and has all the same limitations.
- In the 1990's, many researchers abandoned neural networks with multiple adaptive hidden layers because Support Vector Machines worked better.

6

What is wrong with back-propagation?

- It requires labeled training data.
  - Almost all data is unlabeled.
- The brain needs to fit about 10^14 connection weights in only about 10^9 seconds.
  - Unless the weights are highly redundant, labels cannot possibly provide enough information.
- The learning time does not scale well
  - It is very slow in networks with multiple hidden layers.
- The neurons need to send two different types of signal
  - Forward pass: signal = activity = y
  - Backward pass: signal = dE/dy

7

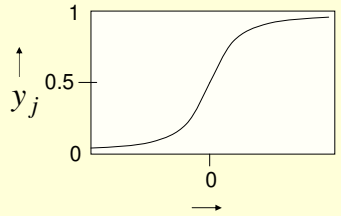
Overcoming the limitations of back-propagation

- We need to keep the efficiency of using a gradient method for adjusting the weights, but use it for modeling the structure of the sensory input.
  - Adjust the weights to maximize the probability that a generative model would have produced the sensory input. This is the only place to get 10^5 bits per second.
  - Learn p(image) not p(label | image)
- What kind of generative model could the brain be using?

8

The building blocks: Binary stochastic neurons

- y is the probability of producing a spike.



$$total\ input\ to\ neuron\ j = external\ input + \sum_i y_i w_{ij}$$

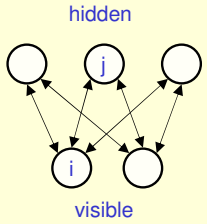
synaptic weight from i to j

output of neuron i

A simple learning module: A Restricted Boltzmann Machine

9

- We restrict the connectivity to make learning easier.
  - Only one layer of hidden units.
    - We will worry about multiple layers later
  - No connections between hidden units.
- In an RBM, the hidden units are independent given the visible states..
  - So we can quickly get an unbiased sample from the posterior distribution over hidden "causes" when given a data-vector :



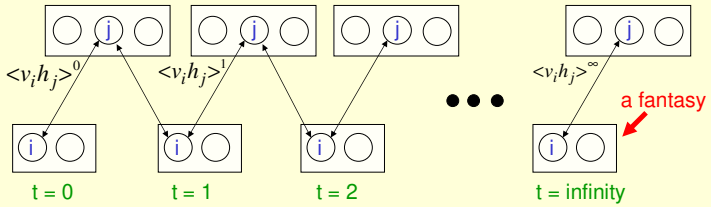
Weights → Energies → Probabilities

10

- Each possible joint configuration of the visible and hidden units has a Hopfield "energy"
  - The energy is determined by the weights and biases.
- The energy of a joint configuration of the visible and hidden units determines the probability that the network will choose that configuration.
- By manipulating the energies of joint configurations, we can manipulate the probabilities that the model assigns to visible vectors.
  - This gives a very simple and very effective learning algorithm.

A picture of "alternating Gibbs sampling" which can be used to learn the weights of an RBM

11



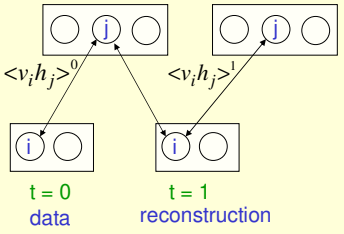
Start with a training vector on the visible units.

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

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

Contrastive divergence learning: A quick way to learn an RBM

12



Start with a training vector on the visible units.

Update all the hidden units in parallel

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

Update all the hidden units again.

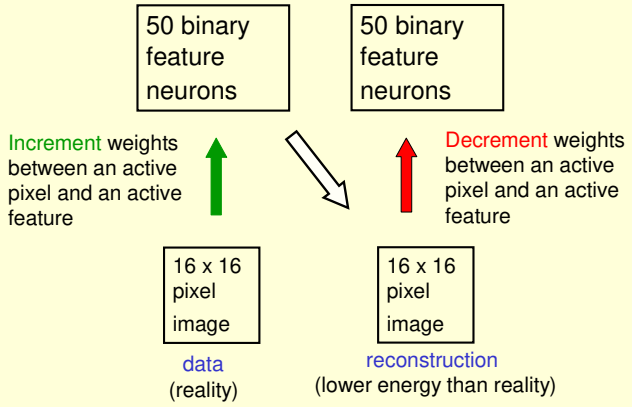
$$\Delta w_{ij} = \varepsilon \left(<v_i h_j>^0 - <v_i h_j>^1 \right)$$

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

It is approximately following the gradient of another objective function.

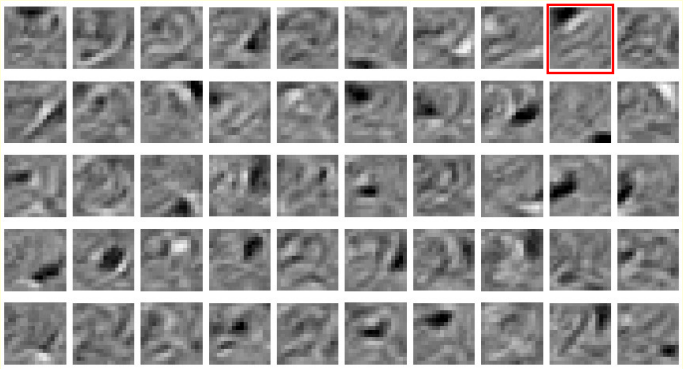
13

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



14

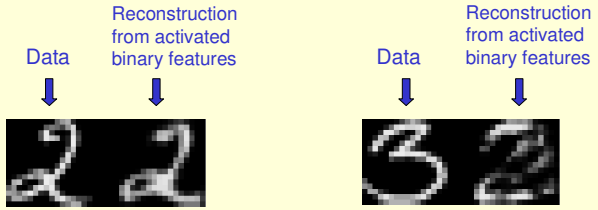
The final 50 x 256 weights



Each neuron grabs a different feature.

15

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)

16

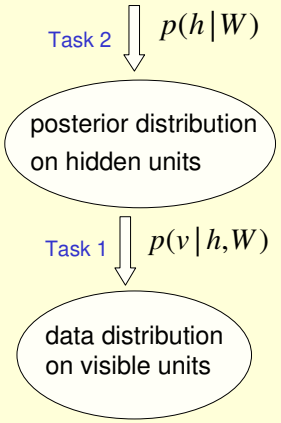
Training a deep network

- 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 complicated non-equilibrium systems.
  - But there is a simple intuitive explanation.

18

Why does greedy learning work?

- Each RBM converts its data distribution into a posterior distribution over its hidden units.
- This divides the task of modeling its data into two tasks:
  - Task 1: Learn generative weights that can convert the posterior distribution over the hidden units back into the data.
  - Task 2: Learn to model the posterior distribution over the hidden units.
  - The RBM does a good job of task 1 and a not so good job of task 2.
- Task 2 is easier (for the next RBM) than modeling the original data because the posterior distribution is closer to a distribution that an RBM can model perfectly.

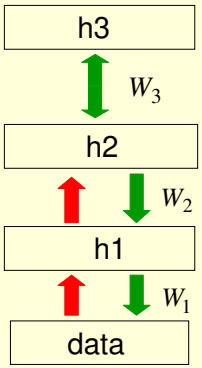


17

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

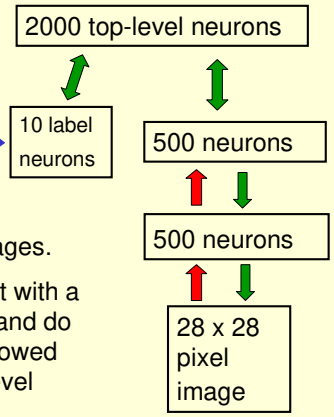


19

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.

20

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

- Replace the top layer of the causal network by an RBM
  - This eliminates explaining away at the top-level.
  - It is nice to have an associative memory at the top.
- Replace 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.

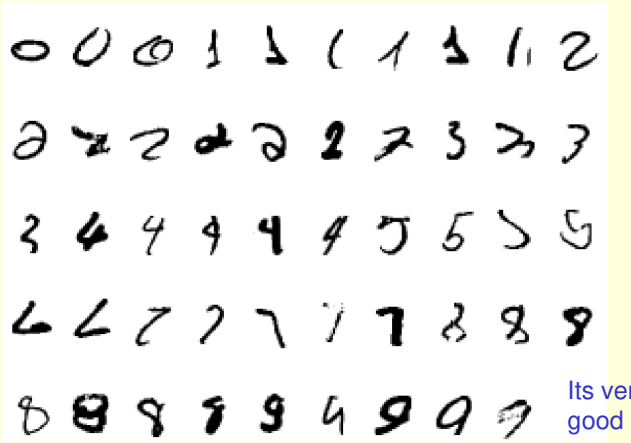
21

Show the movie of the network generating and recognizing digits

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

22

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



Its very good

23

How well does it discriminate on the 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.

24

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.

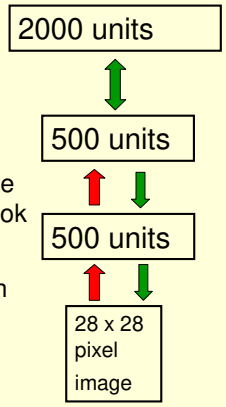
- We do not start backpropagation until we already have sensible weights that already do well at the task.
  - So the initial gradients are sensible and backpropagation only needs to perform a local search.

- 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.

25

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.

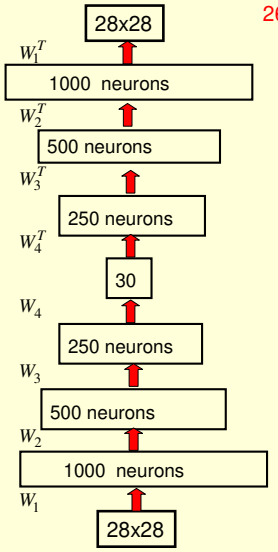
But do the hidden features really help with digit discrimination?

Add 10 softmaxed units to the top and do backpropagation. This gets 1.15% errors.

26

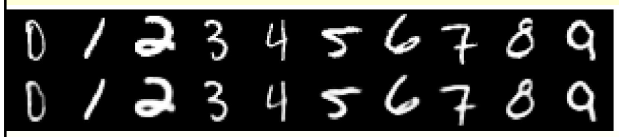
Deep Autoencoders
(Ruslan Salakhutdinov)

- 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.



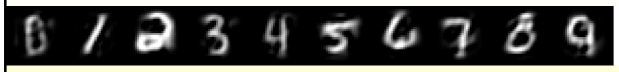
27

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



real data

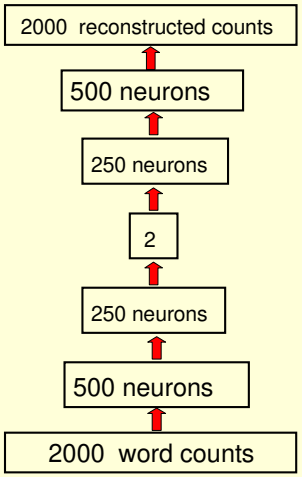
30-D deep auto



30-D PCA

28

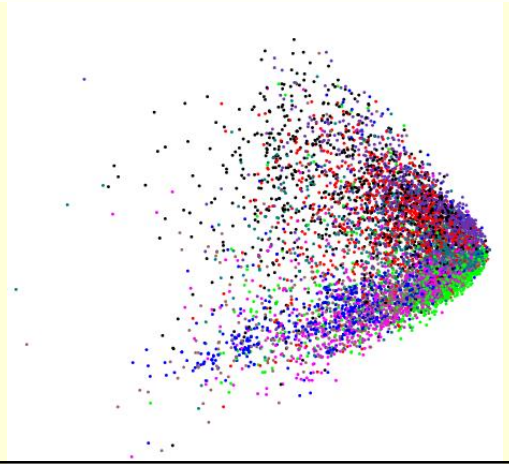
How to compress document count vectors



- We train the autoencoder to reproduce its input vector as its output
- This forces it to compress as much information as possible into the 2 real numbers in the central bottleneck.
- These 2 numbers are then a good way to visualize documents.

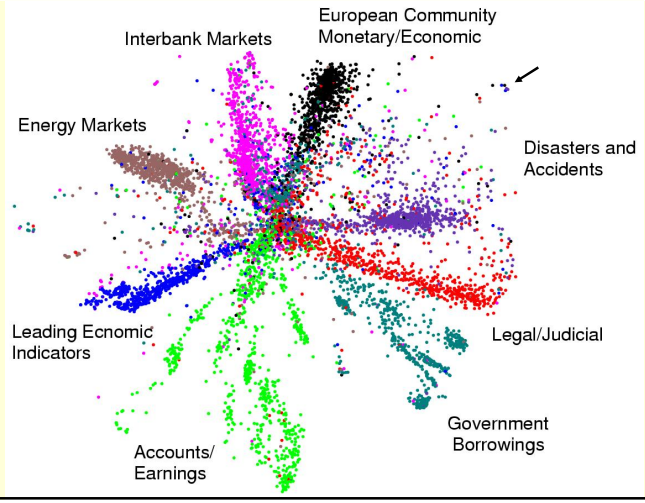
29

First compress all documents to 2 numbers using a type of PCA
Then use different colors for different document categories



30

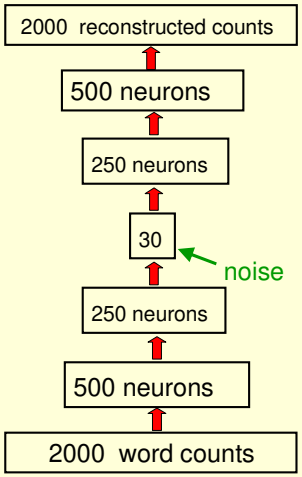
First compress all documents to 2 numbers.
Then use different colors for different document categories



31

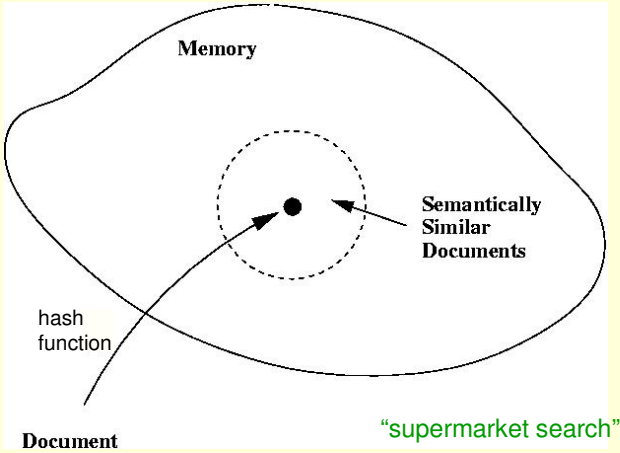
Finding binary codes for documents

- Train an auto-encoder using 30 logistic units for the code layer.
- During the fine-tuning stage, add noise to the inputs to the code units.
  - The "noise" vector for each training case is fixed. So we still get a deterministic gradient.
  - The noise forces their activities to become bimodal in order to resist the effects of the noise.
  - Then we simply round the activities of the 30 code units to 1 or 0.



32

Using a deep autoencoder as a hash-function for finding approximate matches



"supermarket search"

How good is a shortlist found this way?

33

- We have only implemented it for a million documents with 20-bit codes --- but what could possibly go wrong?
  - A 20-D hypercube allows us to capture enough of the similarity structure of our document set.
- The shortlist found using binary codes actually improves the precision-recall curves of TF-IDF.
  - Locality sensitive hashing (the fastest other method) is 50 times slower and has worse precision-recall curves.

Summary

34

- 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.
- 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 used for non-linear dimensionality reduction.
  - This leads to very effective ways of visualizing sets of documents or searching for similar documents.

THE END

Papers and demonstrations are available at
www.cs.toronto/~hinton

The extra slides explain some points in more detail and give additional examples.

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

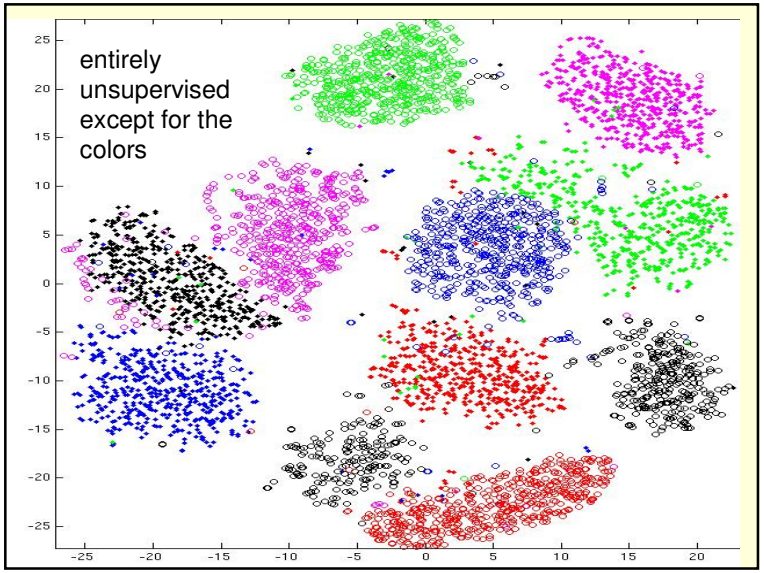
$$p(v) = \sum_h p(h)\ p(v \mid h)$$

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.

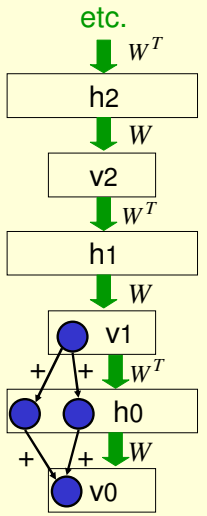
Do the 30-D codes found by the autoencoder preserve the class structure of the data?

- 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?



Inference in a directed net with replicated weights

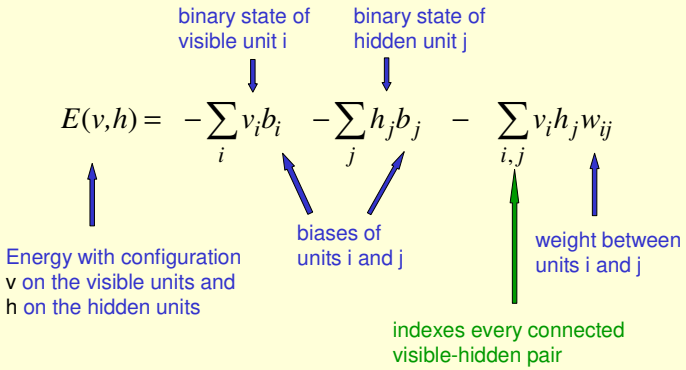
- The variables in h0 are conditionally independent given v0.
  - Inference is trivial. We just multiply v0 by W transpose.
  - The model above h0 implements a complementary prior.
  - Multiplying v0 by W transpose gives the product of the likelihood term and the prior term.
- Inference in the directed net is exactly equivalent to letting a Restricted Boltzmann Machine settle to equilibrium starting at the data.



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 W0 transpose 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 variational bound on the network's model of the data.
    - Hinton, Osindero and Teh (2006) prove that the improvement is always bigger than the loss.

The Energy of a joint configuration



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

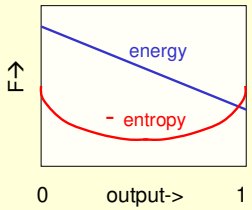
- The probability of a configuration of the visible units is the sum of the probabilities of all the joint configurations that contain it.

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

An RBM with real-valued visible units

(you don't have to understand this slide!)

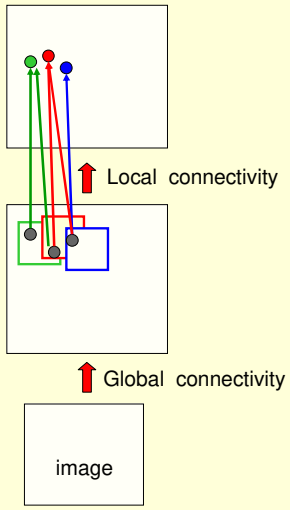
- In a mean-field logistic unit, the total input provides a linear energy-gradient and the negative entropy provides a containment function with fixed curvature. So it is impossible for the value 0.7 to have much lower free energy than both 0.8 and 0.6. This is no good for modeling real-valued data.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



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

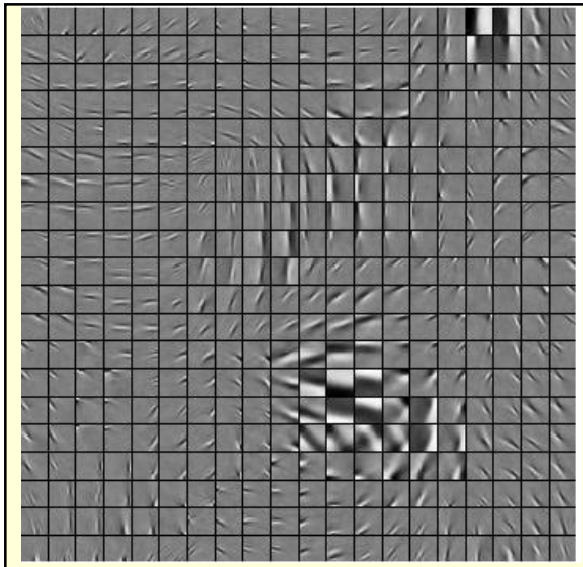
And now for something a bit more realistic

- Handwritten digits are convenient for research into shape recognition, but natural images of outdoor scenes are much more complicated.
  - If we train a network on patches from natural images, does it produce sets of features that look like the ones found in real brains?



A network with local connectivity

The local connectivity between the two hidden layers induces a topography on the hidden units.



Features learned by a net that sees 100,000 patches of natural images.

The feature neurons are locally connected to each other.

Osindero, Welling and Hinton (2006) Neural Computation