CSC 2535: Computation in Neural Networks

Lecture 9
Learning Multiple Layers of Features Greedily

Geoffrey Hinton

The story so far

- 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

The story so far (continued)

- Boltzmann Machines: Nice local learning rule that works in arbitrary networks.
  - Inference requires MCMC
  - Maximum likelihood learning requires unbiased samples from the model's distribution. These are very hard to get.
- Sigmoid Belief Nets: Its nice to have a generative model that we can generate from
  - Exact inference is intractable, but learning still works if we use approximate inference (e.g. factorial distributions).
- Restricted Boltzmann Machines: Exact inference is very easy because the posterior over hidden configurations is factorial.
  - Maximum likelihood learning is slow, but contrastive divergence learning is fast and often works well.
  - But we can only learn one layer of adaptive features!

Recursive 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.
  - Is this just a hack?
  - Can we treat all of the hidden layers as part of one big generative model rather than a hierarchy of separate models?
  - Can we prove that adding more layers will always help?

Learning by dividing and conquering

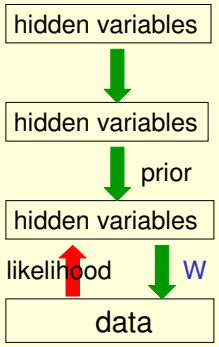
- Re-weighting the data: In boosting, we learn a sequence of simple models. After learning each model, we re-weight the data so that the next model learns to deal with the cases that the previous models found difficult.
  - There is a nice guarantee that the overall model gets better.
- Projecting the data: In PCA, we find the leading eigenvector and then project the data into the orthogonal subspace.
- Distorting the data: In projection pursuit, we find a non-Gaussian direction and then distort the data so that it is Gaussian along this direction.

Another way to divide and conquer

- Re-representing the data: Each time the base learner is called, it passes a transformed version of the data to the next learner.
  - Can we learn a deep, dense DAG one layer at a time, starting at the bottom, and still guarantee that learning each layer improves the overall model of the training data?
    - This seems very unlikely. Surely we need to know the weights in higher layers to learn lower layers?

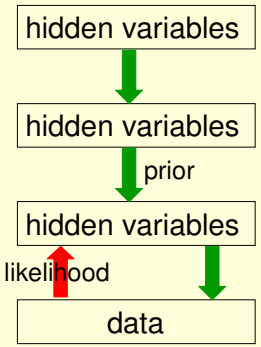
Why its hard to learn one layer at a time

- 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!



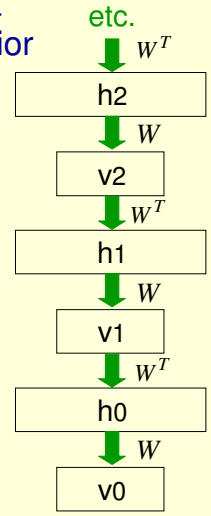
Using complementary priors to eliminate explaining away

- A "complementary" prior is defined as one that exactly cancels the correlations created by explaining away. So the posterior factors.
  - Under what conditions do complementary priors exist?
  - Complementary priors do not exist in general:
    - Parameter counting shows that complementary priors cannot exist if the relationship between the hidden variables and the data is defined by a separate conditional probability table for each hidden configuration.



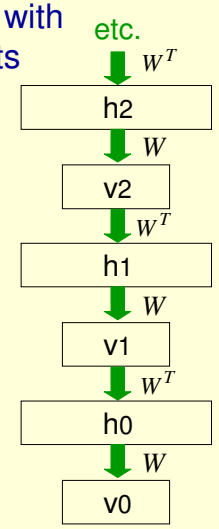
An example of a complementary prior

- The distribution generated by this infinite DAG with replicated weights is the equilibrium distribution for a compatible pair of conditional distributions: p(v|h) and p(h|v).
  - An ancestral pass of the DAG is exactly equivalent to letting a Restricted Boltzmann Machine settle to equilibrium.
  - So this infinite DAG defines the same distribution as an RBM.

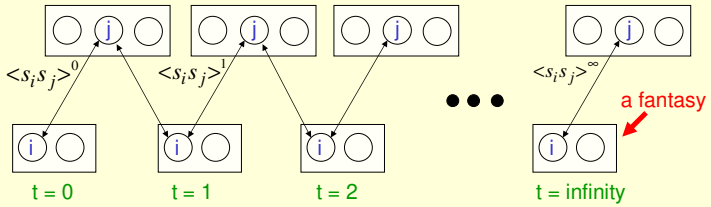


Inference in a DAG with replicated weights

- The variables in h0 are conditionally independent given v0.
  - Inference is trivial. We just multiply v0 by W^T
  - This is because the model above h0 implements a complementary prior.
- Inference in the DAG is exactly equivalent to letting a Restricted Boltzmann Machine settle to equilibrium starting at the data.



A picture of the Boltzmann machine 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.

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

- The learning rule for a logistic DAG is:

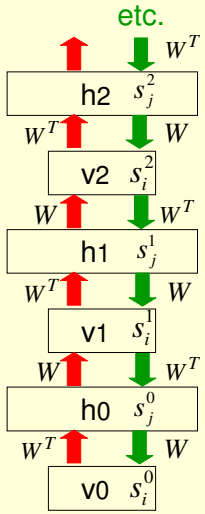
$$\Delta w_{ij} \propto s_j (s_i - \hat{s}_i)$$

- With replicated weights this becomes:

$$s_j^0 (s_i^0 - s_i^1) + s_i^1 (s_j^0 - s_j^1) + s_j^1 (s_i^1 - s_i^2) + \ldots$$

The derivatives for the recognition weights are zero.

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



Pro's and con's of replicating the weights

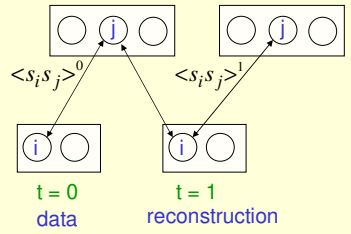
Advantages

- There are many less parameters.
- There is an efficient approximate learning procedure.
- After learning, inference of hidden states is fast and accurate.

Disadvantages

- The model is much less powerful than a deep network that has different weights in each layer.
- The brain clearly uses deep networks.

Contrastive divergence learning: 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.

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

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

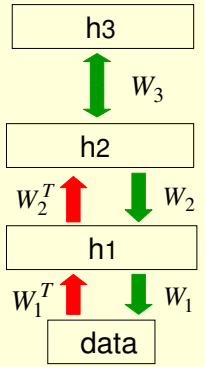
It is easy to understand what it does if we consider the infinite directed net.

Multilayer contrastive divergence

- Start by learning one hidden layer.
- Then re-present the data as the activities of the hidden units.
  - The same learning algorithm can now be applied to the re-presented data.
- Can we prove that each step of this greedy learning improves the log probability of the data under the overall model?
  - What is the overall model?

A simplified version with all hidden layers the same size

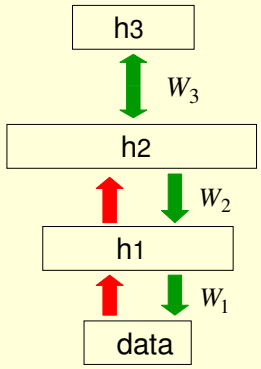
- The RBM at the top can be viewed as shorthand for an infinite directed net.
- When learning W_1 we can view the model in two quite different ways:
  - The model is an RBM composed of the data layer and h1.
  - The model is an infinite DAG with tied weights.
- After learning W_1 we untie it from the other weight matrices.
- We then learn W_2 which is still tied to all the matrices above it.



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 top-down ancestral 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 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 posterior hidden vectors produced by applying W to the data.

Why the hidden configurations should be treated as data when learning the next layer of weights

- After learning the first layer of weights:

$$\log p(x) \geq -\langle energy(x) \rangle + entropy(h1 \mid x)$$
$$\geq \sum_\alpha p(h1 = \alpha \mid x) \left[\log p(h1 = \alpha) + \log p(x \mid h1 = \alpha)\right] + entropy$$

- If we freeze the generative weights that define the likelihood term and the recognition weights that define the distribution over hidden configurations, we get:

$$\log p(x) \geq \sum_\alpha p(h1 = \alpha \mid x) \left[\log p(h1 = \alpha)\right] + constant$$

- Maximizing the RHS is equivalent to maximizing the log prob of "data" α that occurs with probability $p(h1 = \alpha \mid x)$

Why greedy learning works

- Each time we learn a new layer, the inference at the layer below becomes incorrect, but the variational bound on the log prob of the data improves.
- Since the bound starts as an equality, learning a new layer never decreases the log prob of the data, provided we start the learning from the tied weights that implement the complementary prior.
- Now that we have a guarantee we can loosen the restrictions and still feel confident.
  - Allow layers to vary in size.
  - Do not start the learning at each layer from the weights in the layer below.

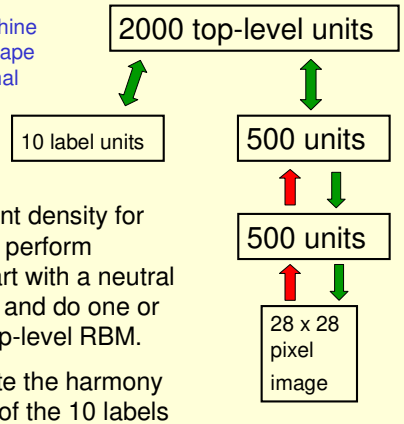
Back-fitting

- After we have learned all the layers greedily, the weights in the lower layers will no longer be optimal. We can improve them in two ways:
  - Untie the recognition weights from the generative weights and learn recognition weights that take into account the non-complementary prior implemented by the weights in higher layers.
  - Improve the generative weights to take into account the non-complementary priors implemented by the weights in higher layers.
- What algorithm should we use for improving on the weights that are learned greedily?

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:



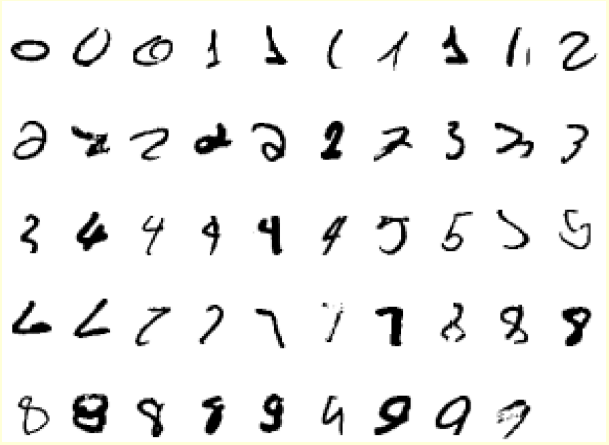
The model learns a joint density for labels and images. To perform recognition we can start with a neutral state of the label units and do one or two iterations of the top-level RBM.

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

Show the movie

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%

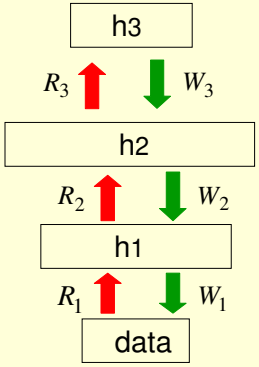
- Its better than backprop and much more neurally plausible because the neurons only need to send one kind of signal, and the teacher can be another sensory input.

All 125 errors

Samples generated by running top-level RBM with one label clamped. Initialized by an up-pass from a random binary image. 20 iterations between samples.

The wake-sleep algorithm

- Wake phase: Use the recognition weights to perform a bottom-up pass.
  - Train the generative weights to reconstruct activities in each layer from the layer above.
- Sleep phase: Use the generative weights to generate samples from the model.
  - Train the recognition weights to reconstruct activities in each layer from the layer below.

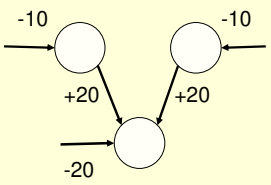


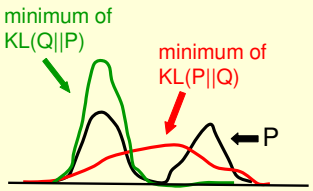
The flaws in the wake-sleep algorithm

- The recognition weights are trained to invert the generative model in parts of the space where there is no data.
  - This is wasteful.
- The recognition weights follow the gradient of the wrong divergence. They minimize KL(P||Q) but the variational bound requires minimization of KL(Q||P).
  - This leads to incorrect mode-averaging
- The posterior over the top hidden layer is very far from independent because the independent prior cannot eliminate explaining away effects.

Mode averaging

- If we generate from the model, half the instances of a 1 at the data layer will be caused by a (1,0) at the hidden layer and half will be caused by a (0,1).
  - So the recognition weights will learn to produce (0.5,0.5)
  - This represents a distribution that puts half its mass on very improbable hidden configurations.
- Its much better to just pick one mode and pay one bit.

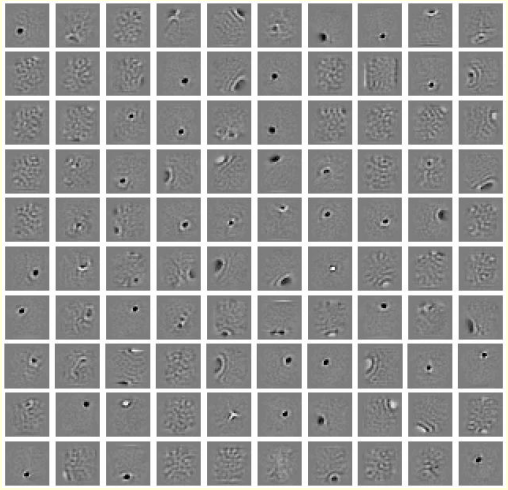




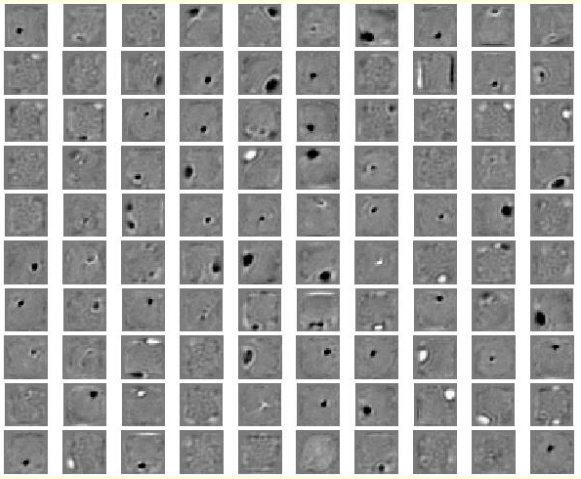
The up-down algorithm: A contrastive divergence version of wake-sleep

- Replace the top layer of the DAG by an RBM
  - This eliminates bad variational approximations caused by top-level units that are independent in the prior.
  - It is nice to have an associative memory at the top.
- Replace the ancestral pass in 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.

The receptive fields of the first hidden layer



The generative fields of the first hidden layer



THE END