ACISS’09 tutorial on deep belief nets

Marcus Frean
Melbourne, 2009

1 December, 2009

Victoria University, Wellington, New Zealand

Outline of this tutorial

- Motivations
  - Deep autoencoders
  - Deep belief nets

- Sigmoid belief nets
  - Why are they hard to train?
  - Could layer-by-layer training work?

- Boltzmann machines
  - Why are they hard to train?
  - The restricted Boltzmann machine (RBM)

- Towers built from RBMs
  - How to do it
  - Why it works
  - Fine-tuning the result
  - 2 applications: a classifier and an autoencoder

Several of the diagrams used here are based on those in Geoff Hinton’s papers & lectures.

Back-propagation networks

Back-propagate error signal to get derivatives for learning

Compare outputs with correct answer to get error signal

Outputs

Hidden layers

Input vector

Autoencoder nets

- Unsupervised learners
- Map each pattern in a training set back to itself
- Dimensionality reduction, if there’s a "bottleneck"
- Could be trained by back-propagation

A nice way to do dimensionality reduction
why haven’t deep auto-encoders worked?
- all the hidden units interact
- the gradient gets tiny as you move away from the “output” layer

belief nets
- A belief net is a directed acyclic graph composed of stochastic variables
- We get to observe some of the variables and would like to solve two problems:
  - The inference problem: Infer the states of unobserved variables
  - The learning problem: Adjust the interactions between variables to make the network more likely to generate the observed data.

parameterized belief networks
- Large belief nets are still too powerful to learn with finite data.
- But we can parameterize the factors: e.g. sigmoid function...

what would a really interesting generative model for (say) images look like?
- stochastic
- lots of units
- several layers
- easy to sample from

sigmoid belief net
an interesting generative model
**Stochastic Neurons**

- Input to the $i^{th}$ neuron:
  \[ \phi_i = \sum_j w_{ji} x_j \]

- Probability of generating a 1:
  \[ p_i = \frac{1}{1 + \exp(\phi_i)} \]

- Learning rule for making $x$ more likely:
  \[ \Delta w_{ji} \propto (x_i - p_i) x_j \]

It's easy to make particular patterns more likely.

---

**Gibbs Sampling**

To draw samples from $p(x) = p(x_1, x_2, \ldots, x_n)$:

1. Choose $i$ at random
2. Choose $x_i$ from $p(x_i|x_{\bar{i}})$

This results in a Markov Chain.

Running the chain for long enough \(\rightarrow\) samples from $p(x)$.
Gibbs sampling in sigmoid belief nets

So what does \( p(x_i|x_i) \) look like, in a sigmoid belief net?

- two hidden "causes"
- visible node is observed...
- Gibbs sampling for \( h_1, h_2 \):

\[
p(h_1 = 1|v = 1) = \left[ 1 + \frac{(1 - f(b_1)) f(w_2)}{f(b_1) f(w_1 + w_2)} \right]^{-1} = \text{yuck!}
\]

Gibbs sampler in sigmoid belief net:
- ugly, slow
- reason is 'explaining away'

explaining away

Hidden states are
- independent in the \textit{prior}
- dependent in the \textit{posterior}

That dependence means sampling from one hidden unit has to cause a change in how all other hidden units update their states.

But we are interested in nets with \textit{lots} of hidden units.

an inconvenient truth:

there’s no quick way to draw a sample from \( p(\text{hidden}|\text{visible}) \)

sigmoidal belief nets are:

Easy to sample from as a generative model, but hard to learn

- sampling from the posterior is slow, due to explaining away. This also makes them hard to use for recognition.
- ‘deep’ layers learn nothing until the ‘shallow’ ones have settled, but shallow layers have to learn while being driven by the deep layers (chicken and egg...)

Let’s ignore the quibbles about slowness for a moment, and consider this idea: one way around the second difficulty might be to train the first layer as a simple BN first, and then the second layer, and so on.

building a deep BN layer-by-layer

Here’s a way to train a multilayer sigmoid belief net:

1. start with a single layer only. The hidden units are driven by bias inputs only. Train to maximize the likelihood of generating the training data.
2. freeze the weights in that layer, and replace the hidden units’ bias inputs by a second layer of weights.
3. train the second layer of weights to maximize the likelihood.
4. and so on...

Question: what should the training set be for the second layer?
The EM algorithm

The $W_1$ weights, and the biases into the hidden units, are trained so as to maximize the probability of generating the patterns on visible units in a data set.

The EM algorithm achieves this by repeating two steps, for each $v$ in the training data:

- E step: calculate $p(h|v)$
- M step: move the $W_1$ weights so as to make $v$ more likely, under $p(h|v)$. And move the hidden biases to better produce that distribution.

In practice we can’t work with the posterior $p(h|v)$ analytically: we have to sample from it instead.

For each $v$ in the training data:

- E step: draw a sample from $p(h|v)$
- M step: move the $W_1$ weights so as to make $v$ more likely, given that $h$.

Move the hidden biases to better produce that $h$.

Averaging over the training set, we have an aggregate posterior distribution:

$$p_{agg}(h) = \sum_{v \in D} p(h|v)$$

So the $W_1$ weights end up at values that maximize the likelihood of the data, given $h$ sampled from the aggregate posterior distribution.

The hidden bias weights end up at values that approximate this distribution.

Q: what is the best thing that the second layer could learn to do?

A: accurately generate the aggregate posterior distribution over the layer 1 hidden units. It is the distribution that makes the training data most likely, given $W_1$.

Easy! For each visible pattern, we just collect one sample (or more) from $p(h|v)$.

This gives us a greedy, layer-wise procedure for training deep belief nets.

Marcus Frean (Melbourne, 2009) ACISS'09 tutorial on deep belief nets
a comment about factorial distributions

This greedy procedure doesn’t work at all well. Subsequent layers add very little to what the first layer achieves, in modelling the data set. Why not?

Consider some patterns \( v \) on a set of visible nodes.

If these came from a world where the components of each vector are independent, then

\[
p(v) = p(v_1, v_2 \ldots v_n) = \prod_i p(v_i)
\]

and we say \( p(v) \) is factorial.

If \( p(v) \) is factorial, there is no point in having hidden units in the generative model

... a model that included hidden units could do no better than a model with just bias inputs to the visibles.

why does greedy training fail for deep belief nets?

EM:

- for each \( v \), sample from the posterior
- make these \( v \) and \( h \) more likely under the joint

prior distribution is factorial

\( \rightarrow W_1 \) tries to learn features that are independent in the prior

but this is premature: we want to add another layer precisely because we DON’T BELIEVE THIS!

Boltzmann machines

If we connect stochastic sigmoid units via symmetric weights, and avoid self-weights, we get a Boltzmann machine.

A Boltzmann machine generates state \( x \) with probability

\[
p(x) \propto e^{-E(x)}
\]

where \( E \) is the energy

\[
E(x) = -\frac{1}{2} \sum_{i,j} x_i x_j w_{ij} - \sum_i x_i b_i
\]

In other words, Gibbs sampling for the above distribution is achieved by calculating the weighted sum into each unit, putting it through the sigmoid function, and choosing to output a 1 with the resulting probability.
Gibbs sampling in Boltzmann machines

- sampling from the joint: just use the sigmoid activation rule
- sampling from hiddens, given a visible pattern: just use the sigmoid activation rule
- i.e. just use the sigmoid activation rule

Contrast that with the sigmoid belief net, where sampling from the joint was easy and instant, but sampling from the posterior wasn’t (and didn’t use the sigmoid activation).

**Sampling from a Boltzmann machine**

- easy
- pretty
- (but requires waiting for a Markov chain to reach equilibrium)

**Learning in Boltzmann machines**

The log likelihood of generating the training set $D$ is

$$\log L = \sum_{v \in D} \log p(v)$$

where

$$p(v) = \sum_{h} p(v, h)$$

with

$$p(v, h) = \frac{e^{-E(v, h)}}{Z}$$

$Z$ is normalisation, summed over all possible states.

**Learning by gradient ascent**

$$\Delta w_{ij} \propto \frac{\partial}{\partial w_{ij}} \log L$$

Piece of cake, surely...

**The problem**

The gradient estimate is itself the difference between two noisy estimates, each of which requires sampling from a long MCMC chain, after waiting for it to reach equilibrium.

This learning algorithm is beautiful but glacially slow in practice, to the point of being unusable. Despite their intuitive appeal as generative models, and their tempting similarities with biological neural nets, Boltzmann machines seemed doomed to the scrap heap until recently.

We’re going to do two tricks to make Boltzmann machines practical devices:

- restrict the connectivity.
- use a new learning algorithm to train the weights.

and then we’re going to show how to use them to solve the towers problem of sigmoid belief nets...
trick # 1: restrict the connections
- Assume visible units are one layer, and hidden units are another.
- Throw out all the connections within each layer.

Restricted Boltzmann machine (RBM)

Gibbs sampling in an RBM
Since none of the units in a layer are connected, we can do Gibbs sampling by updating all of one layer at a time.

This is called “alternating” Gibbs sampling.

learning in an RBM
The very first upward pass gives us a sample from the equilibrium distribution over hidden units, given the visible ones. The clamped Hebbian phase of BM learning is done in the first step! This is a consequence of the fact that, in the RBM graphical model, hidden activities are conditionally independent of one another given the visible activities. FAST
- The anti-Hebbian phase involves unclamping the inputs, and waiting long enough to get a sample from the generative distribution. SLOW.

learning in an RBM
Repeat for all data:
- 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

\[ \Delta w_{ij} = \eta \left[ \langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^\infty \right] \]

restricted connectivity is trick #1:
it saves waiting for equilibrium in the clamped phase.
trick # 2: curtail the Markov chain during learning

1. start with a training vector on the visible units
2. update all the hidden units in parallel
3. update all the visible units in parallel to get a “reconstruction”
4. update the hidden units again

\[ \Delta w_{ij} = \eta \left[ \langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1 \right] \]

This is not following the correct gradient, but works well in practice. Hinton calls this “learning by contrastive divergence”.

why does this work?

- If we start at a data point, the Markov chain wanders off towards patterns that are more likely. We can see the direction it is wandering in just a few steps. It’s a big waste of time to let it go all the way to equilibrium.

NOTE: in an RBM the hidden units are conditionally independent given the visible ones, so there is no “explaining away” inference required.

trick #2: contrastive divergence

this saves waiting for equilibrium in the unclamped phase.

what kinds of distributions are RBMs well-suited for?

A single RBM and appropriate weights can generate any desired distribution over the visible units, if we give it enough hidden units (\( \approx 2^\# \text{visible} \)).

The probability of the joint state in an RBM is

\[ P(v, h|W) \propto \exp(h^T W v) \]

and so

\[ P(v|W) \propto \sum_h \exp(h^T W v) \]

Carefully tracking terms (!), in log space this leads to:

\[ \log P(v|W) = w_{0v} \cdot v + \sum_j \log(1 + e^{w_{jv} \cdot v}) + \text{constant} \]

where \( w_{jv} \) is the vector of weights between the \( j \)th hidden unit and the visible units.

- a linear trend across the input space
- a sum of functions of form \( f = \log(1 + e^\phi) \). This is zero for \( \phi < 0 \) and the identity function for \( \phi > 0 \), with a smooth transition.

Each hidden unit represents a “feature” characterised by the direction of its weight vector \( w_{jv} \).

The hidden unit lowers the energy of any states that are aligned with this vector, making them more likely.
natural distributions for RBMs
RBMs seem predisposed to capture distributions consisting of conjunctions of high probability features.

They should find it difficult to capture distributions that have 'probability holes' in them, since they can only add thresholded ramps together. To make a 'hole', they essentially have to add probability mass everywhere else.

But this needs to be taken with a grain of salt. David MacKay and I trained RBMs (using the exact gradient) to learn “parity” distribution problems: exponentially many probability holes.

Incredibly, an RBM with 6 hidden units can learn the parity distribution on 6 visible units perfectly! It does this by arranging a set of of 6 ramps in just the right way to get high probability mass on all 32 of the desired patterns, while staying low for the 32 undesirable ones.

RBM summary
Two tricks were used to make Boltzmann machines practical devices:

- **restrict the wiring**
  This saves waiting for equilibrium in the clamped phase.

- **truncate the Markov chain**
  This saves waiting for equilibrium in the unclamped phase.
(contrastive divergence)
e.g. samples from an RBM trained on a single class

![Samples from an RBM trained on a single class](image)

Marcus Frean (Melbourne, 2009) ACISS'09 tutorial on deep belief nets 41 / 60

---

e.g. weights after training on a single class

![Weights after training on a single class](image)

Marcus Frean (Melbourne, 2009) ACISS'09 tutorial on deep belief nets 42 / 60

---

**a suggestion**

**greedy layer-wise learning**

- take samples from the aggregate posterior and use them to train a another layer (and then another... and so on).

To generate data, we could

- run alternating Gibbs sampling for a while on the top layer, and then
- do a top-down pass to the visible layer.

This gives us a deep belief net (with an RBM as the top layer).

---

**a suggestion**

But why would we expect this to do any better than the more obvious procedure of using single-layer sigmoid belief nets in the same way?

Two ideas that help understanding:

- RBMs are *already* deep BNs...
- RBMs *don’t* try to force the aggregate posterior to be factorial...

---

![Diagram of greedy layer-wise learning](image)
RBMs are already belief nets

- an RBM with alternating Gibbs
  Sampling is equivalent to a sigmoid belief net with \( \infty \) many layers.
- So when we train an RBM, we’re really training an \( \infty \) deep belief net!
- It’s just that the weights of all layers are tied.

RBMs don’t make the aggregate posterior \( \sim \) factorial

in a 1-layer sigmoid belief net:

- The prior over hiddens is factorial, but the posterior isn’t.
- Tries to make the aggregate posterior factorial.
- ...leaves little for subsequent layers to do.

in a restricted Boltzmann machine:

- The posterior over hiddens is factorial, but the prior isn’t.
- DOESN’T try to force the aggregate posterior to be factorial.
- ...leaves more for the next layer to do.

1 layer: weights and samples after training on “4”

2 layer network
samples from RBM trained on multiple classes

application # 1: classification

Say there are 10 classes in the data. Since we have a generative model we can do semi-supervised learning:

- train the network almost entirely using unlabelled data, but include a “1-of-N” softmax unit in the top layer RBM.
- just use labels to identify which softmax state corresponds to which class label, at the end.

Notice how different this is to supervised learning!
### Application #1: Classification

Application #1: Classification

(All details, and movies of the above in generative and recognition mode, are available via Geoff Hinton's homepage)

Marcus Frean (Melbourne, 2009) ACISS'09 tutorial on deep belief nets

53 / 60

### Fine Tuning

So far, the up and down weights have been symmetric, as required by the Boltzmann machine learning algorithm. But once the tower is built they can be "untied":

- **Wake**: do a bottom-up pass, starting with a pattern from the training set. Use the delta rule to make this more likely under the generative model.
- **Sleep**: do a top-down pass, starting from an equilibrium sample from the top RBM. Use the delta rule to make this more likely under the recognition model.

[CD version: start top RBM at the sample from the wake phase, and don’t wait for equilibrium before doing the top-down pass].

**Wake-Sleep Learning Algorithm**

This unties the recognition weights from the generative ones.

### Application #2: Autoencoding

Application #2: Autoencoding

Basic training

Unrolling

Fine-tuning with back-prop

Marcus Frean (Melbourne, 2009) ACISS'09 tutorial on deep belief nets

54 / 60

### Example: Autoencoding Digits

Example: Autoencoding Digits

Notice: again, the fine tuning (by back-prop this time) achieves its effect by untying the generative and recognition weights.

Test image

30-d autoencoder

30-d logistic PCA

30-d PCA

Notice: again, the fine tuning (by back-prop this time) achieves its effect by untying the generative and recognition weights.
example: autoencoding digits
Codes for digits, produced by taking the first 2 principal components.
Codes from a 784-1000-500-250-2 autoencoder.

example: autoencoding documents
Hinton & Salakhutdinov, Science, 2007

semantic hashing
- Suppose the input is (say) word counts from documents
- Suppose the bottleneck is (say) 20 binary neurons
- Surprised? cf. 20 questions...
- That’s a 20-bit hashcode of the document
- It’s a “semantic” hash: similar documents will have similar codes
- We can navigate to documents that are “one question away” (without knowing what the question is..!)

Salakhutdinov & Hinton, 2007

summary
- Sigmoid belief nets
  - why are they hard to train?
  - could layer-by-layer training work?
- Boltzmann machines
  - why are they hard to train?
  - RBMs
- Deep belief nets built from RBMs
  - how to do it
  - why it works
  - fine-tuning the result [Key: untying the weights]
  - 2 applications: a classifier and an autoencoder

I hope that helps - thanks for listening!