What happens when we stack RBMs?
Restricted Boltzmann Machine (RBM)
Restricted Boltzmann Machine (RBM)

★ Models probability distribution of inputs.
★ Can be thought of as dimensionality reduction.
★ Learns to generate pictures of cats.
★ Easy to train.
Sigmoid Belief Nets

★ Would be better at generating pictures of cats (more layers).
★ Problems with training.
Stacking RBMs

Then train this RBM

Copy binary state for each $v$

Train this RBM first
Stacking RBMs

Then train this RBM

Compose the two RBM models to make a single DBN model

Train this RBM first

copy binary state for each v

It's not a Boltzmann machine!
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 pass to get states for all the other layers.
Why does greedy learning work?

The weights, $W$, in the bottom level RBM define many different distributions: $p(v|h)$; $p(h|v)$; $p(v,h)$; $p(h)$; $p(v)$. We can express the RBM model as

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

If we leave $p(v|h)$ alone and improve $p(h)$, we will improve $p(v)$. To improve $p(h)$, we need it to be a better model than $p(h;W)$ of the aggregated posterior distribution over hidden vectors produced by applying $W$ transpose to the data.
Fine-tuning

After learning many layers of features, we can fine-tune the features to improve generation.

1. **Do a stochastic bottom-up pass**
   Then adjust the top-down weights of lower layers to be good at reconstructing the feature activities in the layer below.

2. **Do a few iterations of sampling in the top level RBM**
   Then adjust the weights in the top-level RBM using CD.

3. **Do a stochastic top-down pass**
   Then adjust the bottom-up weights to be good at reconstructing the feature activities in the layer above.
The DBN used for modeling the joint distribution of MNIST digits and their labels

- The first two hidden layers are learned without using labels.
- The top layer is learned as an RBM for modeling the labels concatenated with the features in the second hidden layer.
- The weights are then fine-tuned to be a better generative model using contrastive wake-sleep.
Discriminative fine-tuning for DBNs
Fine-tuning for generation vs discrimination

★ Learn one layer at the time by stacking RBM-s
★ Treat as pre-training for initial set of weights
★ **Generation** - Contrastive wake-sleep
★ **Discrimination** - Backpropagation
  ○ Can overcome many of the limitations of standard backpropagation
  ○ Easier to learn deep nets
  ○ Nets generalize better
Why backpropagation works better with greedy pre-training - The optimization view

★ Greedily learning one layer at the time scales well to really big networks (especially with locality in each layer)
★ We already have sensible feature detectors before backpropagation
★ Backpropagation only need to perform local search from a sensible starting point
Why backpropagation works better with greedy pre-training - The overfitting view

★ Most of the information comes from input vectors (contains more information than the labels)
★ Labels are used only for fine-tuning  
  ○ Fine-tuning doesn’t need to discover new features  
  ○ Modifies features slightly to get the class boundaries right
★ Works well even for mostly unlabeled data (unlabeled data is still useful for discovering features)
Modelling distribution of digit images

- The top two layers form a RBM whose energy landscape should model the low dimensional manifolds of the digits
- Learns density model for unlabeled digit images
Generalization and discrimination of digits

★ Generating from the model, we get things that look like real digits of all classes

★ Does the hidden features really help with digit discrimination?
  ○ Add a 10-way softmax at the top
  ○ Do backpropagation
Results on the permutation-invariant MNIST task

Backprop net with one or two hidden layers (Platt; Hinton) 1.6%
## Results on the permutation-invariant MNIST task

<table>
<thead>
<tr>
<th>Method</th>
<th>Error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backprop net with one or two hidden layers (Platt; Hinton)</td>
<td>1.6%</td>
</tr>
<tr>
<td>Backprop with L2 constraints on incoming weights</td>
<td>1.5%</td>
</tr>
</tbody>
</table>
## Results on the permutation-invariant MNIST task

<table>
<thead>
<tr>
<th>Method</th>
<th>Error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backprop net with one or two hidden layers <em>(Platt; Hinton)</em></td>
<td>1.6%</td>
</tr>
<tr>
<td>Backprop with L2 constraints on incoming weights</td>
<td>1.5%</td>
</tr>
<tr>
<td>Support Vector Machines <em>(Decoste &amp; Schoelkopf, 2002)</em></td>
<td>1.4%</td>
</tr>
</tbody>
</table>
## Results on the permutation-invariant MNIST task

<table>
<thead>
<tr>
<th>Method</th>
<th>Error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backprop net with one or two hidden layers <em>(Platt; Hinton)</em></td>
<td>1.6%</td>
</tr>
<tr>
<td>Backprop with L2 constraints on incoming weights</td>
<td>1.5%</td>
</tr>
<tr>
<td>Support Vector Machines <em>(Decoste &amp; Schoelkopf, 2002)</em></td>
<td>1.4%</td>
</tr>
<tr>
<td>Generative model of joint density of images and labels (+ generative fine-tuning)</td>
<td>1.25%</td>
</tr>
</tbody>
</table>
## Results on the permutation-invariant MNIST task

<table>
<thead>
<tr>
<th>Model Description</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backprop net with one or two hidden layers (Platt; Hinton)</td>
<td>1.6%</td>
</tr>
<tr>
<td>Backprop with L2 constraints on incoming weights</td>
<td>1.5%</td>
</tr>
<tr>
<td>Support Vector Machines (Decoste &amp; Schoelkopf, 2002)</td>
<td>1.4%</td>
</tr>
<tr>
<td>Generative model of joint density of images and labels (+ generative fine-tuning)</td>
<td>1.25%</td>
</tr>
<tr>
<td>Generative model of unlabelled digits followed by gentle backpropagation</td>
<td>1.15%</td>
</tr>
</tbody>
</table>
Unsupervised “pre-training” also helps for models with more data and better priors

★ Used an additional 600,000 distorted digits
★ Also used convolutional multilayer neural networks

Ranzato et. al. (NIPS 2006)

<table>
<thead>
<tr>
<th>Error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back-propagation alone</td>
</tr>
<tr>
<td>Unsupervised layer-by-layer pre-training followed by backpropagation</td>
</tr>
</tbody>
</table>
Phone recognition on the TIMIT benchmark

- The best previous speaker-independent result on TIMIT (required averaging multiple models) - 24.4% error rate
- A deep net (after standard post-processing with a bi-phone model) - 20.7%
- Has changed speech recognition

183 HMM-state labels

2000 logistic hidden units

6 more layers of pre-trained weights

2000 logistic hidden units

15 frames of 40 filterbank outputs + their temporal derivatives

not pre-trained
What happens during discriminative fine-tuning?
Learning Dynamics of Deep Nets

The next 4 slides describe work by Yoshua Bengio’s group

Before fine-tuning

After fine-tuning
Effect of Unsupervised Pre-training

Erhan et. al. AISTATS’ 2009
Effect of Unsupervised Pre-training

Erhan et. al. AISTATS’ 2009
Effect of Depth

Without pre-training
Effect of Depth

Without pre-training

With pre-training
Trajectories of the learning in function space
(a 2-D visualization produced with t-SNE)

★ Each point is a model in function space.
★ Color = epoch
★ Top: trajectories without pre-training. Each trajectory converges to a different local minimum.
★ Bottom: Trajectories with pre-training.
★ No overlap!

Erhan et. al. AISTATS’ 2009
Why unsupervised pre-training makes sense

If image-label pairs were generated this way, it would make sense to try to go straight from images to labels. For example, do the pixels have even parity?
Why unsupervised pre-training makes sense

If image-label pairs were generated this way, it would make sense to try to go straight from images to labels. For example, do the pixels have even parity?

If image-label pairs are generated this way, it makes sense to first learn to recover the stuff that caused the image by inverting the high bandwidth pathway.
Modeling real-valued data with an RBM
Modeling real-valued data

★ For images of digits:
  ○ intermediate intensities can be represented as probabilities by using “mean-field” logistic units
  ○ We treat intermediate values as the probability that the pixel is inked

★ This doesn’t work for real images:
  ○ intensity of a pixel is almost always, almost exactly the average of the neighboring pixels
  ○ Mean-field logistic units cannot represent precise intermediate values
A standard type of real-valued visible unit

★ Model pixels as Gaussian variables
★ Alternating Gibbs sampling is still easy
★ Learning needs to be much slower

\[ E(v,h) = \sum_{i \in \text{vis}} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{j \in \text{hid}} b_j h_j - \sum_{i,j} \frac{v_i}{\sigma_i} h_j w_{ij} \]

- parabolic containment function
- energy-gradient produced by the total input to a visible unit
Gaussian-Binary RBM’s

★ Its extremely hard to learn tight variances for the visible units

★ When sigma is small, we need many more hidden units than visible units

When sigma is much smaller than 1:

★ Bottom-up effects are too big

★ Dop-down effects are too small
Stepped sigmoid units: implementing integer values

★ Make many copies of a stochastic binary unit
  ○ Same weights
  ○ Same adaptive bias $b$
  ○ Different fixed offsets to bias:
    $b - 0.5, b - 1.5, b - 2.5, b - 3.5, ...$
Fast approximations

★ Contrastive divergence learning works well for the sum of stochastic logistic units with offset biases.
  ○ The noise variance is $\sigma(y)$

★ Also works for rectified linear units
  ○ These are much faster to compute

$$\langle y \rangle = \sum_{n=1}^{n=\infty} \sigma(x + 0.5 - n) \approx \log(1 + e^x) \approx \max(0, x + \text{noise})$$
A nice property of rectified linear units

★ If a rectified linear unit has a bias of zero, it exhibits scale equivariance
  ○ A very nice property to have for images
    \[ R(a \mathbf{x}) = a R(\mathbf{x}) \quad \text{but} \quad R(a + b) \neq R(a) + R(b) \]
  ○ It is like the equivariance to translation exhibited by convolutional nets.
    \[ R(\text{shift}(\mathbf{x})) = \text{shift}(R(\mathbf{x})) \]
THANK YOU!

Thank god it's over!