Disclaimer

A lot of the slides of this tutorial have been taken from presentations from:

- Honglak Lee
- Yann LeCun
- Geoff Hinton
- Andrew Ng
- Yoshua Bengio
Outline

1. Introduction
   - What is deep?
   - Training problems

2. Unsupervised pretraining
   - Pretraining
   - The denoising autoencoder
   - Predictive sparse coding
   - The deep belief network

3. Conclusion
What does deep mean?

\[ Y \]

\[ X_1 \]
\[ X_2 \]
\[ X_3 \]
\[ X_4 \]

\[ W \]
What does deep mean?
What does deep mean?
Properties of deep models

- Many non-linearities = One non-linearity
- More constrained space of transformations
Properties of deep models

- Many non-linearities = One non-linearity
- More constrained space of transformations
- Deep model = prior on the type of transformations
Properties of deep models

- Many non-linearities = One non-linearity
- More constrained space of transformations
- Deep model = prior on the type of transformations
- May need fewer computational units for the same function
Features for face recognition

Feature representation

3rd layer
"Objects"

2nd layer
"Object parts"

1st layer
"Edges"

Pixels

Image courtesy of Honglak Lee
Features for face recognition

Feature representation

- 1st layer: "Edges"
- 2nd layer: "Object parts"
- 3rd layer: "Objects"

- "First-level" features are not task-dependent
- "First-level" features are easy to learn
- "Higher-level" features are easier to learn given the low-level features

Image courtesy of Honglak Lee

Nicolas Le Roux (DL/UFL workshop)
A deep neural network

\[ H_1 = g(W_1 X) \]
\[ H_2 = g(W_2 H_1) \]
\[ Y = W_3 H_2 \]

\( g = \text{sigmoid or tanh} \)

Nicolas Le Roux (DL/UFL workshop)
Deep Learning tutorial
16/12/11 7 / 31
A deep neural network

\[ H^1 = g(W_1 X) \]
A deep neural network

\[ \begin{align*}
H^1 &= g(W_1 X) \\
H^2 &= g(W_2 H^1)
\end{align*} \]
A deep neural network

\[ H^1 = g(W_1 X) \]
\[ H^2 = g(W_2 H^1) \]
\[ Y = W_3 H^2 \]
\[ g = \text{sigmoid or tanh} \]
Training issues

$W_3$ is a lot easier to learn than $W_2$ and $W_1$

Compare

- Given set of features, find combination
- Given combination, find set of features
Training issues

$W_3$ is a lot easier to learn than $W_2$ and $W_1$

Given set of features, find combination
Given combination, find set of features

Feature representation

1st layer
“Edges”

2nd layer
“Object parts”

3rd layer
“Objects”
Practical results

- Parameters randomly initialized
- They do not change much (bad local minima)
- A random transformation is not very good
Practical results

- Parameters randomly initialized
- They do not change much (bad local minima)
- A random transformation is not very good

<table>
<thead>
<tr>
<th>Model</th>
<th>train.</th>
<th>valid.</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBN, unsupervised pre-training</td>
<td>0%</td>
<td>1.2%</td>
<td>1.2%</td>
</tr>
<tr>
<td>Deep net, auto-associator pre-training</td>
<td>0%</td>
<td>1.4%</td>
<td>1.4%</td>
</tr>
<tr>
<td>Deep net, supervised pre-training</td>
<td>0%</td>
<td>1.7%</td>
<td>2.0%</td>
</tr>
<tr>
<td>Deep net, no pre-training</td>
<td>0.004%</td>
<td>2.1%</td>
<td>2.4%</td>
</tr>
<tr>
<td>Shallow net, no pre-training</td>
<td>0.004%</td>
<td>1.8%</td>
<td>1.9%</td>
</tr>
</tbody>
</table>

Bengio et al., 2007
Practical results - TIMIT

Mohamed et al., **Deep Belief Networks for phone recognition**

Nicolas Le Roux (DL/UFL workshop)
The gist of pretraining

1. We want (potentially) deep networks

2. Training the lower layers is hard.
The gist of pretraining

1. We want (potentially) deep networks

2. Training the lower layers is hard.

Solution: optimize each layer without relying on the layers above.
The gist of pretraining
The gist of pretraining

1. Learn $W_1$ first

Given $W_1$ and $W_2$, learn $W_3$
The gist of pretraining

1. Learn $W_1$ first
2. Given $W_1$, learn $W_2$
The gist of pretraining

1. Learn $W_1$ first
2. Given $W_1$, learn $W_2$
3. Given $W_1$ and $W_2$, learn $W_3$
The gist of pretraining

Feature representation

1. Learn $W_1$ first
2. Given $W_1$, learn $W_2$
3. Given $W_1$ and $W_2$, learn $W_3$
Greedy learning

1. Train each layer in a supervised way
2. Use alternate cost hoping that it will be useful.
Greedy learning

1. Train each layer in a supervised way

2. Use alternate cost hoping that it will be useful.

Alternate cost: modeling the data well.
Denoising autoencoders

Vincent et al, 2008

- Corrupt the input (e.g. set 25% of inputs to 0)
- Reconstruct the uncorrupted input
- Use uncorrupted encoding as input to next level
Denoising autoencoders

- Corrupt the input (e.g. set 25% of inputs to 0)
- Reconstruct the uncorrupted input
- Use uncorrupted encoding as input to next level

A representation is good if it works well with noisy data
Projects a “corrupted point” onto its position on the manifold

Two corrupted versions of the same point have the same representation
Stacking DAEs

1. Start with the lowest level and stack upwards
2. Train each layer on the intermediate code (features) from the layer below
3. Top layer can have a different output (e.g., softmax non-linearity) to provide an output for classification
Predictive sparse coding

Objective function for sparse coding:

\[ Z(X) = \arg\min_Z \frac{1}{2} \| X - DZ \|_2^2 + \lambda \| Z \|_1 \]

Z must:

- Be sparse
- Reconstruct \( X \) well
Predictive sparse coding

Objective function for sparse coding:

\[ Z(X) = \arg\min_Z \frac{1}{2} \| X - DZ \|_2^2 + \lambda \| Z \|_1 + \alpha \| Z - F(X, \theta) \|_2^2 \]

\[ F(X, \theta) = G \tanh(W_eX) \]

Z must:

- Be sparse
- Reconstruct X well
- Be well predicted by a feedforward model
Stacking PSDs

\[ Z(X) = \arg\min_{Z} \frac{1}{2} \| X - DZ \|_2^2 + \lambda \| Z \|_1 + \alpha \| Z - F_1(X, \theta) \|_2^2 \]

\[ F_1(X, \theta_1) = G \tanh(W_e^1 X) \]

1. Train the first layer using PSD on \( X \)
Stacking PSDs

\[ Z(X) = \arg\min_Z \frac{1}{2} \| X - DZ \|_2^2 + \lambda \| Z \|_1 + \alpha \| Z - F_1(X, \theta) \|_2^2 \]

\[ F_1(X, \theta_1) = G\tanh(W^1_e X) \]

1. Train the first layer using PSD on \( X \)

2. Use \( H^1 = |F_1(X, \theta)| \) as features
Stacking PSDs

\[ Z(H^1) = \arg\min_Z \frac{1}{2} \| H^1 - DZ \|_2^2 + \lambda \| Z \|_1 + \alpha \| Z - F_2(H^1, \theta) \|_2^2 \]

\[ F_2(H^1, \theta_2) = G \tanh(W^2_\theta H^1) \]

1. Train the first layer using PSD on \( X \)
2. Use \( H^1 = |F_1(X, \theta)| \) as features
3. Train the second layer using PSD on \( H^1 \)
Stacking PSDs

\[ Z(H^1) = \arg\min_Z \frac{1}{2} \| H^1 - DZ \|_2^2 + \lambda \| Z \|_1 + \alpha \| Z - F_2(H^1, \theta) \|_2^2 \]

\[ F_2(H^1, \theta_2) = G \tanh(W^2_\theta H^1) \]

1. Train the first layer using PSD on \( X \)
2. Use \( H^1 = |F_1(X, \theta)| \) as features
3. Train the second layer using PSD on \( H^1 \)
4. Use \( H^2 = |F_2(H^1, \theta)| \) as features
Stacking PSDs

\[ Z(H^1) = \arg\min_Z \frac{1}{2} \| H^1 - DZ \|_2^2 + \lambda \| Z \|_1 + \alpha \| Z - F_2(H^1, \theta) \|_2^2 \]

\[ F_2(H^1, \theta_2) = G \tanh(W \epsilon H^1) \]

1. Train the first layer using PSD on \( X \)

2. Use \( H^1 = |F_1(X, \theta)| \) as features

3. Train the second layer using PSD on \( H^1 \)

4. Use \( H^2 = |F_2(H^1, \theta)| \) as features

5. Keep going...
Stacking PSDs

\[ Z(H^1) = \arg\min_Z \frac{1}{2} \| H^1 - DZ \|_2^2 + \lambda \| Z \|_1 + \alpha \| Z - F_2(H^1, \theta) \|_2^2 \]

\[ F_2(H^1, \theta_2) = G\tanh(W_2^TH^1) \]

1. Train the first layer using PSD on \( X \)
2. Use \( H^1 = |F_1(X, \theta)| \) as features
3. Train the second layer using PSD on \( H^1 \)
4. Use \( H^2 = |F_2(H^1, \theta)| \) as features
5. Keep going...
6. Use \( H^{37} \) as input to your final classifier
The mothership: the DBN

- Unsupervised pretraining works by reconstructing the data
- What if we tried to have a full generative model of the data?
The mothership: the DBN

- Unsupervised pretraining works by reconstructing the data
- What if we tried to have a full generative model of the data?
Restricted Boltzmann Machine (RBM)

\[ E(x, h) = - \sum_{ij} W_{ij} x_i h_j = -x^\top Wh \]

\[ P(x, h) = \frac{\exp [x^\top Wh]}{\sum_{x_0, h_0} \exp [x_0^\top Wh_0]} \]
Type of variables in an RBM

\[ P(x, h) = \frac{\exp [x^\top Wh]}{\sum_{x_0, h_0} \exp [x_0^\top Wh_0]} \]

- This formulation is correct when \( x \) and \( h \) are binary variables.
- You have no choice over the type of \( x \) (this is your data).
- You can choose what type \( h \) is.
- There are other formulations when \( x \) is not binary.
Training an RBM

\[ P(x, h) = \frac{\exp [x^\top Wh]}{\sum_{x_0, h_0} \exp [x_0^\top Wh_0]} \]

- We have a joint probability of data and “features”
- Maximizing \( P(x) \) will give us meaningful features
- There are efficient ways to do that (check my webpage)
Stacking RBMs

- Learn the first layer by maximizing $P(x) = \sum_{h^1} P(x, h^1)$
- Once trained, the features of $x$ are $E_P[h^1|x]$
- Learn the second layer by maximizing $P(h^1) = \sum_{h^2} P(h^1, h^2)$
## Results - 1

<table>
<thead>
<tr>
<th>Problem</th>
<th>SVM(_{rbf})</th>
<th>DBN-1</th>
<th>DBN-3</th>
<th>SAA-3</th>
<th>SdA-3 ((\nu))</th>
<th>SVM(_{rbf}(\nu))</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>3.03±0.15</td>
<td>3.94±0.17</td>
<td>3.11±0.15</td>
<td>3.46±0.16</td>
<td>2.80±0.14 (10%)</td>
<td>3.07 (10%)</td>
</tr>
<tr>
<td>rot</td>
<td>11.11±0.28</td>
<td>14.69±0.31</td>
<td>10.30±0.27</td>
<td>10.30±0.27</td>
<td>10.29±0.27 (10%)</td>
<td>11.62 (10%)</td>
</tr>
<tr>
<td>bg-rand</td>
<td>14.58±0.31</td>
<td>9.80±0.26</td>
<td>6.73±0.22</td>
<td>11.28±0.28</td>
<td>10.38±0.27 (40%)</td>
<td>15.63 (25%)</td>
</tr>
<tr>
<td>bg-img</td>
<td>22.61±0.37</td>
<td>16.15±0.32</td>
<td>16.31±0.32</td>
<td>23.00±0.37</td>
<td>16.68±0.33 (25%)</td>
<td>23.15 (25%)</td>
</tr>
<tr>
<td>rot-bg-img</td>
<td>55.18±0.44</td>
<td>52.21±0.44</td>
<td>47.39±0.44</td>
<td>51.93±0.44</td>
<td>44.49±0.44 (25%)</td>
<td>54.16 (10%)</td>
</tr>
<tr>
<td>rect</td>
<td>2.15±0.13</td>
<td>4.71±0.19</td>
<td>2.60±0.14</td>
<td>2.41±0.13</td>
<td>1.99±0.12 (10%)</td>
<td>2.45 (25%)</td>
</tr>
<tr>
<td>rect-img</td>
<td>24.04±0.37</td>
<td>23.69±0.37</td>
<td>22.50±0.37</td>
<td>24.05±0.37</td>
<td>21.59±0.36 (25%)</td>
<td>23.00 (10%)</td>
</tr>
<tr>
<td>convex</td>
<td>19.13±0.34</td>
<td>19.92±0.35</td>
<td>18.63±0.34</td>
<td>18.41±0.34</td>
<td>19.06±0.34 (10%)</td>
<td>24.20 (10%)</td>
</tr>
</tbody>
</table>
Results - 2
Outline

1. Introduction
   - What is deep?
   - Training problems

2. Unsupervised pretraining
   - Pretraining
   - The denoising autoencoder
   - Predictive sparse coding
   - The deep belief network

3. Conclusion
Take-home messages - Deep learning

- Building a hierarchy of features can be beneficial
- Training all layers at once is very hard
- Pretraining alleviates the problem of bad local minima
- Regularizes the model
Take-home messages - Pretraining

- Pretraining requires building blocks
- **ALL** pretraining methods share the same idea: model the input while having somewhat invariant features
- By **no means** is this necessarily the best method
Take-home messages - Pretraining

- Pretraining requires building blocks
- **ALL** pretraining methods share the same idea: model the input while having somewhat invariant features
- By **no means** is this necessarily the best method
- And now for some applications...
This is the end...

Questions?