Greedy Layer-Wise Training of Deep Networks

Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle
NIPS 2007

Presented by
Ahmed Hefny
Story so far ...

- Deep neural nets are more expressive: Can learn wider classes of functions with less **hidden units (parameters)** and **training examples**.

- Unfortunately they are not easy to train with **randomly initialized** gradient-based methods.
Hinton et. al. (2006) proposed greedy unsupervised layer-wise training:
- Greedy layer-wise: Train layers sequentially starting from bottom (input) layer.
- Unsupervised: Each layer learns a higher-level representation of the layer below. The training criterion does not depend on the labels.

- Each layer is trained as a Restricted Boltzman Machine. (RBM is the building block of Deep Belief Networks).

- The trained model can be fine tuned using a supervised method.
This paper

- Extends the concept to:
  - Continuous variables
  - *Uncooperative* input distributions
  - Simultaneous Layer Training

- Explores variations to better understand the training method:
  - What if we use greedy *supervised* layer-wise training?
  - What if we replace RBMs with auto-encoders?
Outline

• Review
  • Restricted Boltzman Machines
  • Deep Belief Networks
  • Greedy layer-wise Training

• Supervised Fine-tuning

• Extensions
  • Continuous Inputs
  • Uncooperative Input Distributions
  • Simultaneous Training

• Analysis Experiments
Outline

• Review
  • Restricted Boltzman Machines
  • Deep Belief Networks
  • Greedy layer-wise Training

• Supervised Fine-tuning

• Extensions
  • Continuous Inputs
  • Uncooperative Input Distributions
  • Simultaneous Training

• Analysis Experiments
Restricted Boltzmann Machine

Undirected bipartite graphical model with connections between **visible** nodes and **hidden** nodes.

Corresponds to joint probability distribution

\[
P(v, h) = \frac{1}{Z} \exp(-\text{energy}(v, h)) = \frac{1}{Z} \exp(v'Wh + b'v + c'h)
\]
Restricted Boltzmann Machine

Undirected bipartite graphical model with connections between visible nodes and hidden nodes.

Corresponds to joint probability distribution

\[ P(v, h) = \frac{1}{Z} \exp(h'Wv + b'v + c'h) \]

**Factorized Conditionals**

\[
Q(h|v) = \prod_j P(h_j|v) \quad P(v|h) = \prod_k P(v_k|h)
\]

\[
Q(h_j = 1|v) = \text{sigmoid}(c_j + \sum_k W_{jk}v_k) \quad P(v_k = 1|h) = \text{sigmoid}(b_k + \sum_j W_{jk}h_j)
\]
Restricted Boltzmann Machine (Training)

• Given input vectors $V_0$, adjust $\theta = (W, b, c)$ to increase $\log P(V_0)$

$$\log P(v_0) = \log \sum_h P(v_0, h) = \log \sum_h \exp(-\text{energy}(v_0, h)) - \log \sum_{v,h} \exp(-\text{energy}(v, h))$$

$$\frac{\partial \log P(v_0)}{\partial \theta} = - \sum_h Q(h|v_0) \frac{\partial \text{energy}(v_0, h)}{\partial \theta} + \sum_{v,h} P(v, h) \frac{\partial \text{energy}(v, h)}{\partial \theta}$$

$$\frac{\partial \log P(v_0)}{\partial \theta_k} = - \sum_h Q(h|v_0) \frac{\partial \text{energy}(v_0, h)}{\partial \theta_k} + \sum_v P(v) \sum_{h_k} Q(h_k|v) \frac{\partial \text{energy}(v, h)}{\partial \theta_k}$$
Restricted Boltzmann Machine (Training)

- Given input vectors $V_0$, adjust $\theta = (W, b, c)$ to increase $\log P(V_0)$

$$
\log P(v_0) = \log \sum_h P(v_0, h) = \log \sum_h \exp(-\text{energy}(v_0, h)) - \log \sum_{v,h} \exp(-\text{energy}(v, h))
$$

$$
\frac{\partial \log P(v_0)}{\partial \theta} = - \sum_h Q(h|v_0) \frac{\partial \text{energy}(v_0, h)}{\partial \theta} + \sum_{v,h} P(v, h) \frac{\partial \text{energy}(v, h)}{\partial \theta}
$$

$$
\frac{\partial \log P(v_0)}{\partial \theta_k} = - \sum_h Q(h|v_0) \frac{\partial \text{energy}(v_0, h)}{\partial \theta_k} + \sum_v P(v) \sum_{h_k} Q(h_k|v) \frac{\partial \text{energy}(v, h)}{\partial \theta_k}
$$
Restricted Boltzman Machine (Training)

• Given input vectors $V_0$, adjust $\theta = (W, b, c)$ to increase $\log P(V_0)$

\[
\log P(v_0) = \log \sum_h P(v_0, h) = \log \sum_h \exp(-\text{energy}(v_0, h)) - \log \sum_{v,h} \exp(-\text{energy}(v, h))
\]

\[
\frac{\partial \log P(v_0)}{\partial \theta} = - \sum_h Q(h|v_0) \frac{\partial \text{energy}(v_0, h)}{\partial \theta} + \sum_{v,h} P(v, h) \frac{\partial \text{energy}(v, h)}{\partial \theta}
\]

\[
\frac{\partial \log P(v_0)}{\partial \theta_k} = - \sum_h Q(h|v_0) \frac{\partial \text{energy}(v_0, h)}{\partial \theta_k} + \sum_v P(v) \sum_{h_k} Q(h_k|v) \frac{\partial \text{energy}(v, h)}{\partial \theta_k}
\]

Sample $h_0$ given $v_0$ 
Sample $v_1$ and $h_1$ using Gibbs sampling
Restricted Boltzman Machine (Training)

• Now we can perform stochastic gradient descent on data log-likelihood

• Stop based on some criterion
  (e.g. reconstruction error $- \log P(v_1 = x|v_0 = x)$)
Deep Belief Network

• A DBN is a model of the form

\[ P(x, g^1, g^2, ..., g^l) = P(x|g^1) \, P(g^1|g^2) \, ... \, P(g^{l-2}|g^{l-1}) \, P(g^{l-1}, g^l) \]

\[ x = g^0 \text{ denotes input variables} \]
\[ g \text{ denotes hidden layers of causal variables} \]
Deep Belief Network

• A DBN is a model of the form

\[ P(x, g^1, g^2, ..., g^l) = P(x|g^1) \ P(g^1|g^2) \ ... \ P(g^{L-2}|g^{L-1})P(g^{L-1}, g^L) \]

\[ x = g^0 \] denotes input variables

\[ g \] denotes hidden layers of causal variables
Deep Belief Network

• A DBN is a model of the form

\[ P(x, g^1, g^2, \ldots, g^l) = P(x|g^1) \ P(g^1|g^2) \ \ldots \ P(g^{l-2}|g^{l-1})P(g^{l-1}, g^l) \]

\( x = g^0 \) denotes input variables
\( g \) denotes hidden layers of causal variables

\( P(g^{l-1}, g^l) \) is an RBM
\( P(g^i|g^{i+1}) = \prod_j P(g^i_j|g^{i+1}_j) \)
\( P(g^i_j|g^{i+1}) = \text{sigmoid}(b^i_j + \sum_k^{n_{i+1}} W^i_{kj} g^i_{k+1}) \)

RBM = Infinitely Deep network with tied weights
Greedy layer-wise training

• $P(g^1|g^0)$ is intractable
• Approximate with $Q(g^1|g^0)$
  • Treat bottom two layers as an RBM
  • Fit parameters using contrastive divergence
Greedy layer-wise training

• $P(g^1|g^0)$ is intractable

• Approximate with $Q(g^1|g^0)$
  • Treat bottom two layers as an RBM
  • Fit parameters using contrastive divergence

• That gives an approximate $\hat{P}(g^1)$

• We need to match it with $P(g^1)$
Greedy layer-wise training

- Approximate $P(g^l|g^{l-1}) \approx Q(g^l|g^{l-1})$
  - Treat layers $l - 1, l$ as an RBM
  - Fit parameters using contrastive divergence
  - Sample $g_0^{l-1}$ recursively using $Q(g^i|g^{i-1})$ starting from $g^0$
Outline

• Review
  • Restricted Boltzman Machines
  • Deep Belief Networks
  • Greedy layer-wise Training

• Supervised Fine-tuning

• Extensions
  • Continuous Inputs
  • Uncooperative Input Distributions
  • Simultaneous Training

• Analysis Experiments
Supervised Fine Tuning (In this paper)

• Use greedy layer-wise training to initialize weights of all layers except output layer.

• For fine-tuning, use stochastic gradient descent of a cost function on the outputs where the conditional expected values of hidden nodes are approximated using mean-field.

\[ E(g^i | g^{i-1} = \mu^{i-1}) = \mu^i = \text{sigm}(b^i + W^i \mu^{i-1}) \]
Supervised Fine Tuning (In this paper)

• Use greedy layer-wise training to initialize weights of all layers except output layer.

• Use backpropagation
Outline

• Review
  • Restricted Boltzmann Machines
  • Deep Belief Networks
  • Greedy layer-wise Training

• Supervised Fine-tuning

• Extensions
  • Continuous Inputs
  • Uncooperative Input Distributions
  • Simultaneous Training

• Analysis Experiments
Continuous Inputs

• Recall RBMs:
  \[ Q(h_j|v) \propto Q(h_j, v) \propto \exp(h_j w' v + b_j h_j) \propto \exp((w' v + b_j) h_j) = \exp(a(v) h_j) \]

• If we restrict \( h_j \in I = \{0,1\} \) then normalization gives us binomial with \( p \) given by sigmoid.

• Instead, if \( I = [0, \infty] \) we get exponential density

• If \( I \) is closed interval then we get truncated exponential
Continuous Inputs (Case for truncated exponential [0,1])

• Sampling
  
  For truncated exponential, inverse CDF can be used
  
  \[ h_j = F^{-1}(U) = \frac{\log(1-U \times (1-\exp(a(v))))}{a(v)} \]
  
  where \( U \) is sampled uniformly from [0,1]

• Conditional Expectation
  
  \[ E[h_j|v] = \frac{1}{1-\exp(-a(v))} - \frac{1}{a(v)} \]
Continuous Inputs

• To handle Gaussian inputs, we need to augment the energy function with a term quadratic in $h$.

• For a diagonal covariance matrix

$$P(h_j|v) = a(v)h_j + d_jh_j^2$$

Giving

$$E[h_j|z] = a(x)/2d^2$$
Continuous Hidden Nodes ?
Continuous Hidden Nodes?

• Truncated Exponential

\[ E[h_j | v] = \frac{1}{1 - \exp(-a(v))} - \frac{1}{a(v)} \]

• Gaussian

\[ E[h_j | v] = \frac{a(v)}{2d^2} \]
Uncooperative Input Distributions

• Setting
  
  \[ x \sim p(x) \]
  \[ y = f(x) + noise \]

• No particular relation between \( p \) and \( f \), (e.g. Gaussian and sinus)
Uncooperative Input Distributions

• Setting

\[ x \sim p(x) \]
\[ y = f(x) + noise \]

• No particular relation between \( p \) and \( f \), (e.g. Gaussian and sinus)

• Problem: Unsupervised pre-training may not help prediction
Outline

• Review
  • Restricted Boltzman Machines
  • Deep Belief Networks
  • Greedy layer-wise Training

• Supervised Fine-tuning

• Extensions

• Analysis Experiments
Uncooperative Input Distributions

• Proposal: Mix unsupervised and supervised training for each layer

Stochastic Gradient of input log likelihood by Contrastive Divergence

Stochastic Gradient of prediction error

Combined Update

Temp. Output Layer
Simultaneous Layer Training

• Greedy Layer-wise Training
• For each layer
  • Repeat Until Criterion Met
    • Sample layer input (by recursively applying trained layers to data)
    • Update parameters using contrastive divergence
Simultaneous Layer Training

- Simultaneous Training
- Repeat Until Criterion Met
  - Sample input to all layers
  - Update parameters of all layers using contrastive divergence

- Simpler: One criterion for the entire network
- Takes more time
Outline

• Review
  • Restricted Boltzman Machines
  • Deep Belief Networks
  • Greedy layer-wise Training

• Supervised Fine-tuning

• Extensions
  • Continuous Inputs
  • Uncooperative Input Distributions
  • Simultaneous Training

• Analysis Experiments
Experiments

• Does greedy unsupervised pre-training help?
• What if we replace RBM with auto-encoders?
• What if we do greedy *supervised* pre-training?

• Does continuous variable modeling help?
• Does partially supervised pre-training help?
Experiment 1

• Does greedy unsupervised pre-training help?
• What if we replace RBM with auto-encoders?
• What if we do greedy *supervised* pre-training?

• Does continuous variable modeling help?
• Does partially supervised pre-training help?
Experiment 1
## Experiment 1

<table>
<thead>
<tr>
<th>Method</th>
<th>Abalone train</th>
<th>Abalone valid</th>
<th>Abalone test</th>
<th>Cotton train</th>
<th>Cotton valid</th>
<th>Cotton test</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Deep Network with no pre-training</td>
<td>4.23</td>
<td>4.43</td>
<td>4.2</td>
<td>45.2%</td>
<td>42.9%</td>
<td>43.0%</td>
</tr>
<tr>
<td>2. Logistic regression</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>44.0%</td>
<td>42.6%</td>
<td>45.0%</td>
</tr>
<tr>
<td>3. DBN, binomial inputs, unsupervised</td>
<td>4.59</td>
<td>4.60</td>
<td>4.47</td>
<td>44.0%</td>
<td>42.6%</td>
<td>45.0%</td>
</tr>
<tr>
<td>4. DBN, binomial inputs, partially supervised</td>
<td>4.39</td>
<td>4.45</td>
<td>4.28</td>
<td>43.3%</td>
<td>41.1%</td>
<td>43.7%</td>
</tr>
<tr>
<td>5. DBN, Gaussian inputs, unsupervised</td>
<td>4.25</td>
<td>4.42</td>
<td>4.19</td>
<td>35.7%</td>
<td>34.9%</td>
<td>35.8%</td>
</tr>
<tr>
<td>6. DBN, Gaussian inputs, partially supervised</td>
<td>4.23</td>
<td>4.43</td>
<td>4.18</td>
<td>27.5%</td>
<td>28.4%</td>
<td>31.4%</td>
</tr>
</tbody>
</table>
Experiment 1 (MSE and Training Errors)

<table>
<thead>
<tr>
<th>Method</th>
<th>Abalone</th>
<th></th>
<th></th>
<th>Cotton</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>train.</td>
<td>valid.</td>
<td>test.</td>
<td>train.</td>
<td>valid.</td>
<td>test.</td>
</tr>
<tr>
<td>1. Deep Network with no pre-training</td>
<td>4.23</td>
<td>4.43</td>
<td>4.2</td>
<td>45.2%</td>
<td>42.9%</td>
<td>43.0%</td>
</tr>
<tr>
<td>2. Logistic regression</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>44.0%</td>
<td>42.6%</td>
<td>45.0%</td>
</tr>
<tr>
<td>3. DBN, binomial inputs, unsupervised</td>
<td>4.59</td>
<td>4.60</td>
<td>4.47</td>
<td>44.0%</td>
<td>42.6%</td>
<td>45.0%</td>
</tr>
<tr>
<td>4. DBN, binomial inputs, partially supervised</td>
<td>4.39</td>
<td>4.45</td>
<td>4.28</td>
<td>43.3%</td>
<td>41.1%</td>
<td>43.7%</td>
</tr>
<tr>
<td>5. DBN, Gaussian inputs, unsupervised</td>
<td>4.25</td>
<td>4.42</td>
<td>4.19</td>
<td>35.7%</td>
<td>34.9%</td>
<td>35.8%</td>
</tr>
<tr>
<td>6. DBN, Gaussian inputs, partially supervised</td>
<td>4.23</td>
<td>4.43</td>
<td>4.18</td>
<td>27.5%</td>
<td>28.4%</td>
<td>31.4%</td>
</tr>
</tbody>
</table>

Partially Supervised < Unsupervised Pre-training < No Pre-training

Gaussian < Binomial
Experiment 2

• Does greedy unsupervised pre-training help?
• What if we replace RBM with auto-encoders?
• What if we do greedy *supervised* pre-training?

• Does continuous variable modeling help?
• Does partially supervised pre-training help?
Experiment 2

• Auto Encoders

• Learn a compact representation to reconstruct $X$

$$p(x) = \text{sigm}(c + W\text{sigm}(b + W' x))$$

• Trained to minimize reconstruction cross-entropy

$$R = -\sum_i x_i \log p(x_i) + (1 - x_i) \log p(1 - x_i)$$
Experiment 2

<table>
<thead>
<tr>
<th>Model</th>
<th>Experiment 2</th>
<th>Experiment 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>train.</td>
<td>valid.</td>
</tr>
<tr>
<td>DBN, unsupervised pre-training</td>
<td>0%</td>
<td>1.2%</td>
</tr>
<tr>
<td>Deep net, auto-associator pre-training</td>
<td>0%</td>
<td>1.4%</td>
</tr>
<tr>
<td>Deep net, supervised pre-training</td>
<td>0%</td>
<td>1.7%</td>
</tr>
<tr>
<td>Deep net, no pre-training</td>
<td>0.004%</td>
<td>2.1%</td>
</tr>
<tr>
<td>Shallow net, no pre-training</td>
<td>0.004%</td>
<td>1.8%</td>
</tr>
</tbody>
</table>

(500~1000) layer width  20 nodes in last two layers
Experiment 2

• Auto-encoder pre-training outperforms supervised pre-training but is still outperformed by RBM.

• Without pre-training, deep nets do not generalize well, but they can still fit the data if the output layers are wide enough.
Conclusions

• Unsupervised pre-training is important for deep networks.

• Partial supervision further enhances results, especially when input distribution and the function to be estimated are not closely related.

• Explicitly modeling conditional inputs is better than using binomial models.
Thanks