Deep Learning & Neural Networks
Lecture 2

Kevin Duh

Graduate School of Information Science
Nara Institute of Science and Technology

Jan 16, 2014

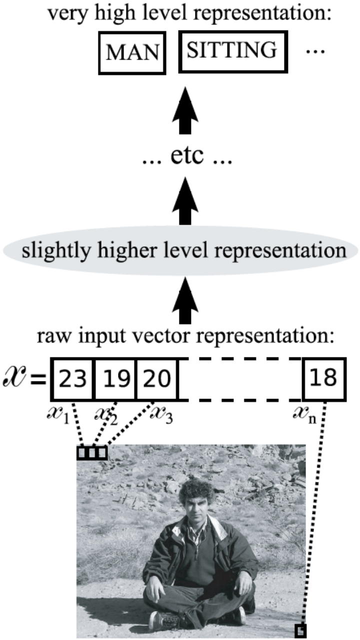
Today's Topics

1. General Ideas in Deep Learning
   - Motivation for Deep Architectures and why is it hard?
   - Main Breakthrough in 2006: Layer-wise Pre-Training
2. Approach 1: Deep Belief Nets [Hinton et al., 2006]
   - Restricted Boltzmann Machines (RBM)
   - Training RBMs with Contrastive Divergence
   - Stacking RBMs to form Deep Belief Nets
3. Approach 2: Stacked Auto-Encoders [Bengio et al., 2006]
   - Auto-Encoders
   - Denoising Auto-Encoders
4. Discussions
   - Why it works, when it works, and the bigger picture

Today's Topics

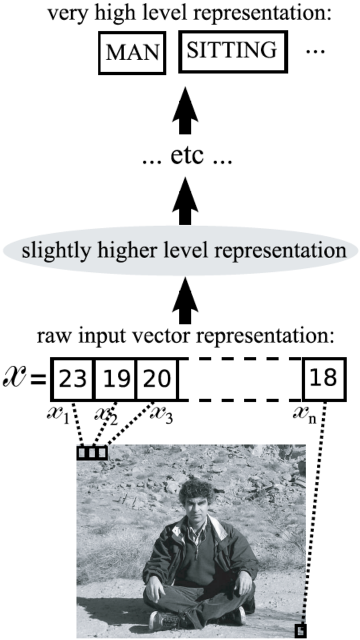
1. General Ideas in Deep Learning
   - Motivation for Deep Architectures and why is it hard?
   - Main Breakthrough in 2006: Layer-wise Pre-Training
2. Approach 1: Deep Belief Nets [Hinton et al., 2006]
   - Restricted Boltzmann Machines (RBM)
   - Training RBMs with Contrastive Divergence
   - Stacking RBMs to form Deep Belief Nets
3. Approach 2: Stacked Auto-Encoders [Bengio et al., 2006]
   - Auto-Encoders
   - Denoising Auto-Encoders
4. Discussions
   - Why it works, when it works, and the bigger picture

The Promise of Deep Architectures



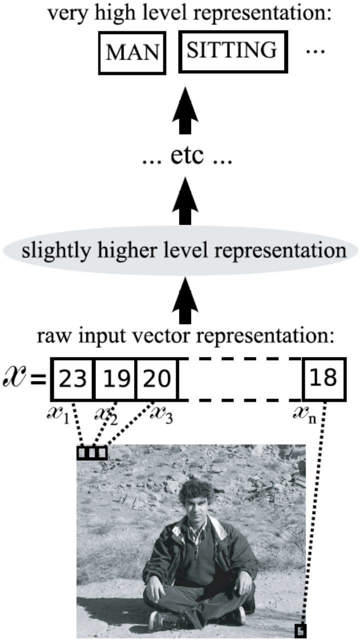
- *Understanding in AI* requires high-level abstractions, modeled by highly non-linear functions

The Promise of Deep Architectures



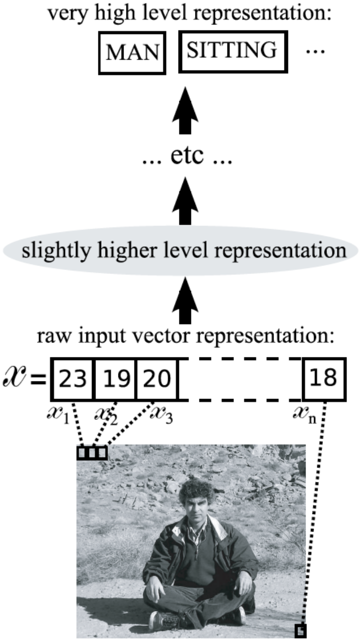
- *Understanding in AI* requires high-level abstractions, modeled by highly non-linear functions
- These abstractions must disentangle factors of variation in data (e.g. 3D pose, lighting)

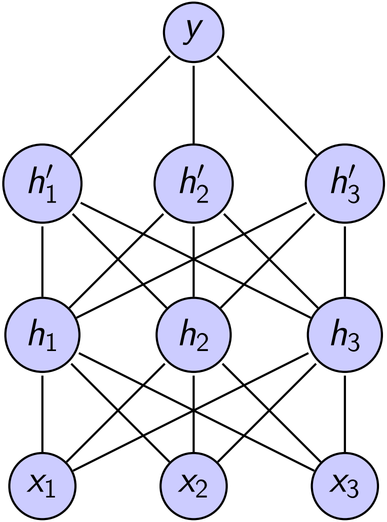
The Promise of Deep Architectures



- *Understanding in AI* requires high-level abstractions, modeled by highly non-linear functions
- These abstractions must disentangle factors of variation in data (e.g. 3D pose, lighting)
- Deep Architecture is one way to achieve this: each intermediate layer is a successively higher level abstraction

*(*Example from [Bengio, 2009])*

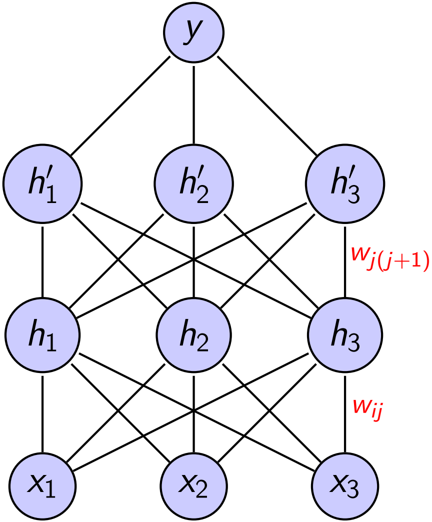

very high level representation:
MAN
SITTING
...
... etc ...
slightly higher level representation
raw input vector representation:
x =
23
19
20
18
x1
x2
x3
xn


y
h'1
h'2
h'3
h1
h2
h3
x1
x2
x3

Why are Deep Architectures hard to train?

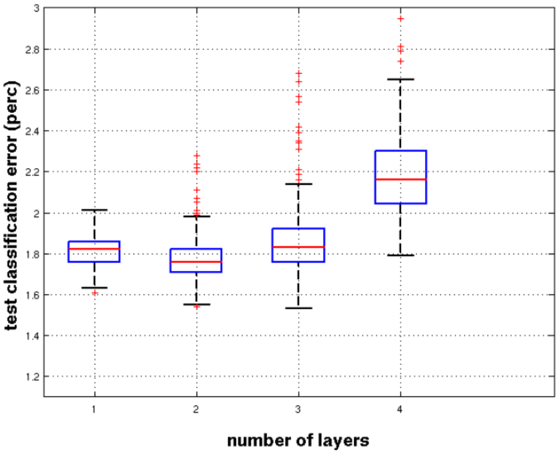
Vanishing gradient problem in Backpropagation

- $\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j} \frac{\partial in_j}{\partial w_{ij}} = \delta_j x_i$
- $\delta_j = \left[\sum_{j+1} \delta_{j+1} w_{j(j+1)} \right] \sigma'(in_j)$
- δ_j may vanish after repeated multiplication



Empirical Results: Poor performance of Backpropagation on Deep Neural Nets [Erhan et al., 2009]

- MNIST digit classification task; 400 trials (random seed)
- Each layer: initialize w_{ij} by uniform$[-1/\sqrt{(FanIn)}, 1/\sqrt{(FanIn)}]$
- Although $L+1$ layers is more expressive, worse error than L layers



Local Optimum Issue in Neural Nets

- For 2-Layer Net and more, the training objective is not convex, so different local optima may be achieved depending on initial point
- For Deep Architectures, Backpropagation is apparently getting a local optimum that does not generalize well

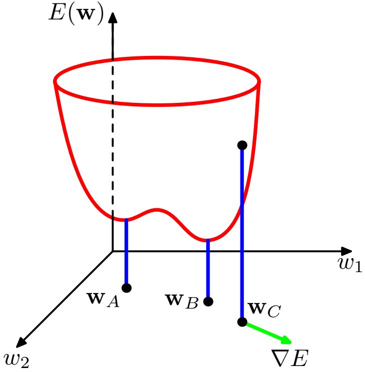


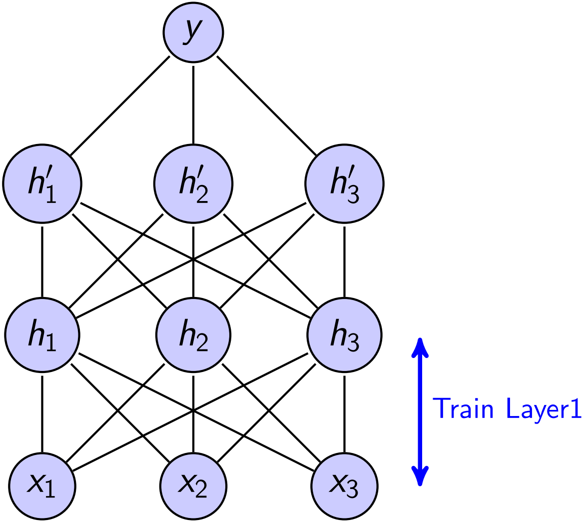
*Figure from Chapter 5, [Bishop, 2006]

Today's Topics

1. General Ideas in Deep Learning
   - Motivation for Deep Architectures and why is it hard?
   - Main Breakthrough in 2006: Layer-wise Pre-Training
2. Approach 1: Deep Belief Nets [Hinton et al., 2006]
   - Restricted Boltzmann Machines (RBM)
   - Training RBMs with Contrastive Divergence
   - Stacking RBMs to form Deep Belief Nets
3. Approach 2: Stacked Auto-Encoders [Bengio et al., 2006]
   - Auto-Encoders
   - Denoising Auto-Encoders
4. Discussions
   - Why it works, when it works, and the bigger picture

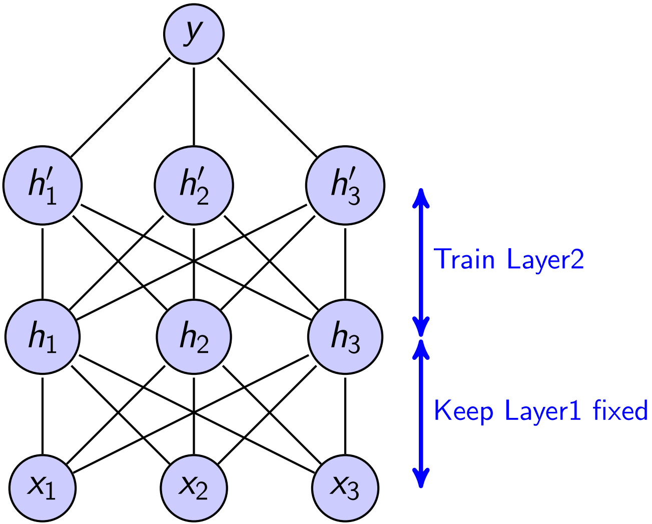
Layer-wise Pre-training [Hinton et al., 2006]

First, train one layer at a time, optimizing data-likelihood objective $P(x)$



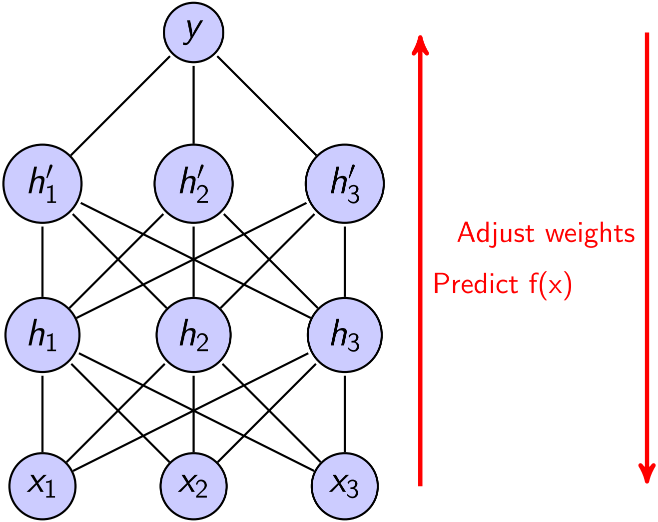
Layer-wise Pre-training [Hinton et al., 2006]

First, train one layer at a time, optimizing data-likelihood objective $P(x)$



Layer-wise Pre-training [Hinton et al., 2006]

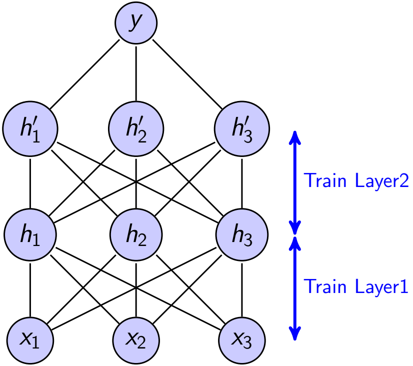
Finally, fine-tune labeled objective $P(y|x)$ by Backpropagation



Layer-wise Pre-training [Hinton et al., 2006]

Key Idea:
Focus on modeling the input $P(X)$ better with each successive layer.
Worry about optimizing the task $P(Y|X)$ later.

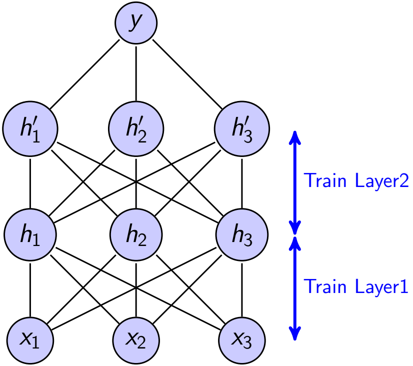
> *"If you want to do computer vision, first learn computer graphics." – Geoff Hinton*



Layer-wise Pre-training [Hinton et al., 2006]

Key Idea:
Focus on modeling the input $P(X)$ better with each successive layer. Worry about optimizing the task $P(Y|X)$ later.

"If you want to do computer vision, first learn computer graphics." – Geoff Hinton



Extra advantage: Can exploit large amounts of unlabeled data!

Today's Topics

1. General Ideas in Deep Learning
   - Motivation for Deep Architectures and why is it hard?
   - Main Breakthrough in 2006: Layer-wise Pre-Training
2. Approach 1: Deep Belief Nets [Hinton et al., 2006]
   - Restricted Boltzmann Machines (RBM)
   - Training RBMs with Contrastive Divergence
   - Stacking RBMs to form Deep Belief Nets
3. Approach 2: Stacked Auto-Encoders [Bengio et al., 2006]
   - Auto-Encoders
   - Denoising Auto-Encoders
4. Discussions
   - Why it works, when it works, and the bigger picture

General Approach for Deep Learning

- Recall the problem setup: Learn function $f : x \rightarrow y$

General Approach for Deep Learning

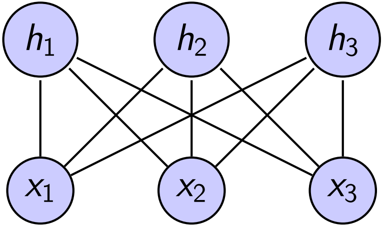
- Recall the problem setup: Learn function $f : x \rightarrow y$
- But rather doing this directly, we first learn hidden features h that model input x, i.e. $x \rightarrow h \rightarrow y$

General Approach for Deep Learning

- Recall the problem setup: Learn function $f : x \rightarrow y$
- But rather doing this directly, we first learn hidden features h that model input x, i.e. $x \rightarrow h \rightarrow y$
- How do we discover useful latent features h from data x?
  - Different Deep Learning methods differ by this basic component
  - e.g. Deep Belief Nets use Restricted Boltzmann Machines (RBMs)

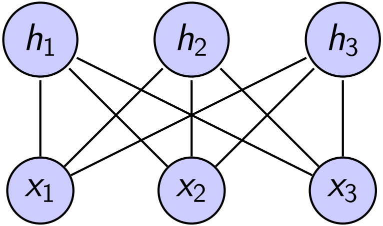
Restricted Boltzmann Machine (RBM)

- RBM is a simple energy-based model: $p(x, h) = \frac{1}{Z_\theta} \exp\left(-E_\theta(x, h)\right)$
  - with only h-x interactions: $E_\theta(x, h) = -x^T W h - b^T x - d^T h$
  - here, we assume h_j and x_i are binary variables
  - normalizer: $Z_\theta = \sum_{(x,h)} \exp(-E_\theta(x, h))$ is called partition function



Restricted Boltzmann Machine (RBM)

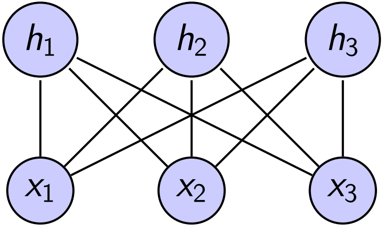
- RBM is a simple energy-based model: $p(x, h) = \frac{1}{Z_\theta} \exp\left(-E_\theta(x, h)\right)$
  - with only h-x interactions: $E_\theta(x, h) = -x^T W h - b^T x - d^T h$
  - here, we assume h_j and x_i are binary variables
  - normalizer: $Z_\theta = \sum_{(x,h)} \exp(-E_\theta(x, h))$ is called partition function



- Example:
  - Let weights (h_1, x_1), (h_1, x_3) be positive, others be zero, $b = d = 0$.

Restricted Boltzmann Machine (RBM)

- RBM is a simple energy-based model: $p(x,h) = \frac{1}{Z_\theta} \exp\left(-E_\theta(x,h)\right)$
  - with only h-x interactions: $E_\theta(x,h) = -x^T W h - b^T x - d^T h$
  - here, we assume h_j and x_i are binary variables
  - normalizer: $Z_\theta = \sum_{(x,h)} \exp(-E_\theta(x,h))$ is called partition function



- Example:
  - Let weights (h_1, x_1), (h_1, x_3) be positive, others be zero, $b = d = 0$.
  - Then this RBM defines a distribution over $[x_1, x_2, x_3, h_1, h_2, h_3]$ where $p(x_1 = 1, x_2 = 0, x_3 = 1, h_1 = 1, h_2 = 0, h_3 = 0)$ has high probability

Computing Posteriors in RBMs

- Computing $p(h|x)$ is easy due to factorization:

$$
\begin{aligned}
p(h|x) &= \frac{p(x,h)}{\sum_h p(x,h)} = \frac{1/Z_\theta \exp(-\mathrm{E}(\mathrm{x},\mathrm{h}))}{\sum_h 1/Z_\theta \exp(-\mathrm{E}(\mathrm{x},\mathrm{h}))} \\
&= \frac{\exp(x^T W h + b^T x + d^T h)}{\sum_h \exp(x^T W h + b^T x + d^T h)} \\
&= \frac{\prod_j \exp(x^T W_j h_j + d_j h_j) \cdot \exp(b^T x)}{\sum_{h_1 \in \{0,1\}} \sum_{h_2 \in \{0,1\}} \cdots \sum_{h_j} \prod_j \exp(x^T W_j h_j + d_j h_j) \cdot \exp(b^T x)} \\
&= \frac{\prod_j \exp(x^T W_j h_j + d_j h_j)}{\prod_j \sum_{h_j \in \{0,1\}} \exp(x^T W_j h_j + d_j h_j)} \\
&= \prod_j \frac{\exp(x^T W_j h_j + d_j h_j)}{\sum_{h_j \in \{0,1\}} \exp(x^T W_j h_j + d_j h_j)} = \prod_j p(h_j|x)
\end{aligned}
$$

- Note $p(h_j = 1|x) = \exp(x^T W_j + d_j)/Z = \sigma(x^T W_j + d_j)$

Computing Posteriors in RBMs

- Computing $p(h|x)$ is easy due to factorization:

$$
\begin{aligned}
p(h|x) &= \frac{p(x,h)}{\sum_h p(x,h)} = \frac{1/Z_\theta \exp(-\mathrm{E(x,h)})}{\sum_h 1/Z_\theta \exp(-\mathrm{E(x,h)})} \\
&= \frac{\exp(x^T W h + b^T x + d^T h)}{\sum_h \exp(x^T W h + b^T x + d^T h)} \\
&= \frac{\prod_j \exp(x^T W_j h_j + d_j h_j) \cdot \exp(b^T x)}{\sum_{h_1 \in \{0,1\}} \sum_{h_2 \in \{0,1\}} \cdots \sum_{h_j} \prod_j \exp(x^T W_j h_j + d_j h_j) \cdot \exp(b^T x)} \\
&= \frac{\prod_j \exp(x^T W_j h_j + d_j h_j)}{\prod_j \sum_{h_j \in \{0,1\}} \exp(x^T W_j h_j + d_j h_j)} \\
&= \prod_j \frac{\exp(x^T W_j h_j + d_j h_j)}{\sum_{h_j \in \{0,1\}} \exp(x^T W_j h_j + d_j h_j)} = \prod_j p(h_j|x)
\end{aligned}
$$

- Note $p(h_j = 1|x) = \exp(x^T W_j + d_j)/Z = \sigma(x^T W_j + d_j)$
- Similarly, computing $p(x|h) = \prod_i p(x_i|h)$ is easy

Today's Topics

1. General Ideas in Deep Learning
   - Motivation for Deep Architectures and why is it hard?
   - Main Breakthrough in 2006: Layer-wise Pre-Training
2. Approach 1: Deep Belief Nets [Hinton et al., 2006]
   - Restricted Boltzmann Machines (RBM)
   - Training RBMs with Contrastive Divergence
   - Stacking RBMs to form Deep Belief Nets
3. Approach 2: Stacked Auto-Encoders [Bengio et al., 2006]
   - Auto-Encoders
   - Denoising Auto-Encoders
4. Discussions
   - Why it works, when it works, and the bigger picture

Training RBMs to optimize $P(X)$

Derivative of the Log-Likelihood: $\partial_{\mathrm{w_{ij}}} \log P_w(x = x^{(m)})$

$$= \partial_{\mathrm{w_{ij}}} \log \sum_h P_w(x = x^{(m)}, h) \quad (1)$$

$$= \partial_{\mathrm{w_{ij}}} \log \sum_h \frac{1}{Z_w} \exp\left(-\mathrm{E_w}(\mathrm{x}^{(\mathrm{m})}, \mathrm{h})\right) \quad (2)$$

$$= -\partial_{\mathrm{w_{ij}}} \log Z_w + \partial_{\mathrm{w_{ij}}} \log \sum_h \exp\left(-\mathrm{E_w}(\mathrm{x}^{(\mathrm{m})}, \mathrm{h})\right) \quad (3)$$

$$= \frac{1}{Z_w} \sum_{h,x} e^{(-\mathrm{E_w}(\mathrm{x},\mathrm{h}))} \partial_{\mathrm{w_{ij}}} \mathrm{E_w}(\mathrm{x}, \mathrm{h}) - \frac{1}{\sum_h e^{(-\mathrm{E_w}(\mathrm{x}^{(\mathrm{m})},\mathrm{h}))}} \sum_h e^{(-\mathrm{E_w}(\mathrm{x}^{(\mathrm{m})},\mathrm{h}))} \partial_{\mathrm{w_{ij}}} \mathrm{E_w}(\mathrm{x}^{(\mathrm{m})}, \mathrm{h})$$

$$= \sum_{h,x} P_w(x, h)[\partial_{\mathrm{w_{ij}}} \mathrm{E_w}(\mathrm{x}, \mathrm{h})] - \sum_h P_w(x^{(m)}, h)[\partial_{\mathrm{w_{ij}}} \mathrm{E_w}(\mathrm{x}^{(\mathrm{m})}, \mathrm{h})] \quad (4)$$

$$= -\mathbb{E}_{p(x,h)}[x_i \cdot h_j] + \mathbb{E}_{p(h|x=x^{(m)})}[x_i^{(m)} \cdot h_j] \quad (5)$$

Training RBMs to optimize $P(X)$

Derivative of the Log-Likelihood: $\partial_{\mathrm{w_{ij}}} \log P_w(x = x^{(m)})$

$$= \partial_{\mathrm{w_{ij}}} \log \sum_h P_w(x = x^{(m)}, h) \quad (1)$$

$$= \partial_{\mathrm{w_{ij}}} \log \sum_h \frac{1}{Z_w} \exp\left(-\mathrm{E_w}(\mathrm{x}^{(\mathrm{m})}, \mathrm{h})\right) \quad (2)$$

$$= -\partial_{\mathrm{w_{ij}}} \log Z_w + \partial_{\mathrm{w_{ij}}} \log \sum_h \exp\left(-\mathrm{E_w}(\mathrm{x}^{(\mathrm{m})}, \mathrm{h})\right) \quad (3)$$

$$= \frac{1}{Z_w} \sum_{h,x} e^{(-\mathrm{E_w}(\mathrm{x},\mathrm{h}))} \partial_{\mathrm{w_{ij}}} \mathrm{E_w}(\mathrm{x}, \mathrm{h}) - \frac{1}{\sum_h e^{(-\mathrm{E_w}(\mathrm{x}^{(\mathrm{m})},\mathrm{h}))}} \sum_h e^{(-\mathrm{E_w}(\mathrm{x}^{(\mathrm{m})},\mathrm{h}))} \partial_{\mathrm{w_{ij}}} \mathrm{E_w}(\mathrm{x}^{(\mathrm{m})}, \mathrm{h})$$

$$= \sum_{h,x} P_w(x, h)[\partial_{\mathrm{w_{ij}}} \mathrm{E_w}(\mathrm{x}, \mathrm{h})] - \sum_h P_w(x^{(m)}, h)[\partial_{\mathrm{w_{ij}}} \mathrm{E_w}(\mathrm{x}^{(\mathrm{m})}, \mathrm{h})] \quad (4)$$

$$= -\mathbb{E}_{p(x,h)}[x_i \cdot h_j] + \mathbb{E}_{p(h|x=x^{(m)})}[x_i^{(m)} \cdot h_j] \quad (5)$$

Second term (positive phase) increases probability of $x^{(m)}$; First term (negative phase) decreases probability of samples generated by the model

Contrastive Divergence Algorithm

- The negative phase term ($\mathbb{E}_{p(x,h)}[x_i \cdot h_j]$) is expensive because it requires sampling (x,h) from the model

Contrastive Divergence Algorithm

- The negative phase term ($\mathbb{E}_{p(x,h)}[x_i \cdot h_j]$) is expensive because it requires sampling (x,h) from the model
- Gibbs Sampling (sample x then h iteratively) works, but waiting for convergence at each gradient step is slow.

Contrastive Divergence Algorithm

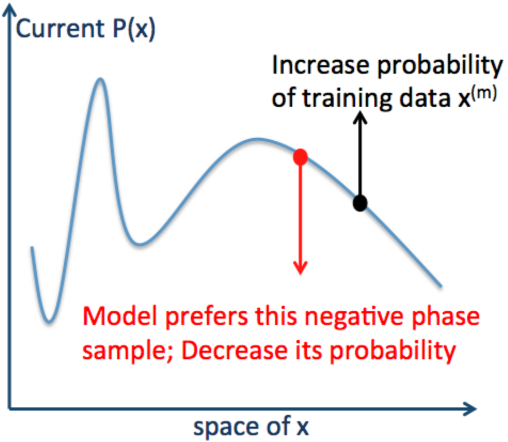
- The negative phase term ($\mathbb{E}_{p(x,h)}[x_i \cdot h_j]$) is expensive because it requires sampling (x,h) from the model
- Gibbs Sampling (sample x then h iteratively) works, but waiting for convergence at each gradient step is slow.
- Contrastive Divergence is a faster but biased method: initialize with training point and wait only a few (usu. 1) sampling steps

Contrastive Divergence Algorithm

- The negative phase term ($\mathbb{E}_{p(x,h)}[x_i \cdot h_j]$) is expensive because it requires sampling (x,h) from the model
- Gibbs Sampling (sample x then h iteratively) works, but waiting for convergence at each gradient step is slow.
- Contrastive Divergence is a faster but biased method: initialize with training point and wait only a few (usu. 1) sampling steps
  1. Let $x^{(m)}$ be training point, $W = [w_{ij}]$ be current model weights
  2. Sample $\hat{h}_j \in \{0,1\}$ from $p(h_j|x = x^{(m)}) = \sigma(\sum_i w_{ij} x_i^{(m)} + d_j)\ \forall j$.
  3. Sample $\tilde{x}_i \in \{0,1\}$ from $p(x_i|h = \hat{h}) = \sigma(\sum_j w_{ij} \hat{h}_j + b_i)\ \forall i$.
  4. Sample $\tilde{h}_j \in \{0,1\}$ from $p(h_j|x = \tilde{x}) = \sigma(\sum_i w_{ij} \tilde{x}_i + d_j)\ \forall j$.
  5. $w_{ij} \leftarrow w_{ij} + \gamma(x_i^{(m)} \cdot \hat{h}_j - \tilde{x}_i \cdot \tilde{h}_j)$

Pictorial View of Contrastive Divergence

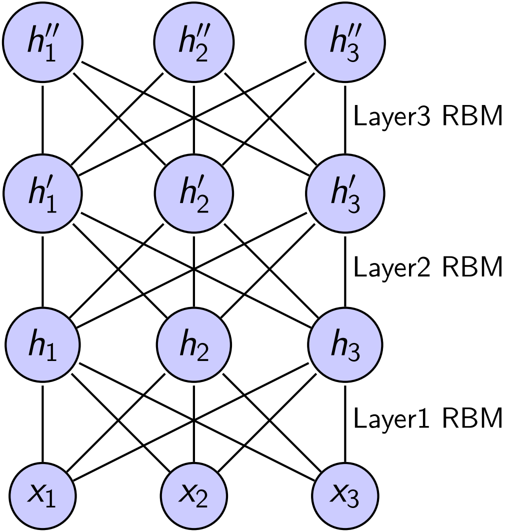
- Goal: Make RBM $p(x, h)$ have high probability on training samples
- To do so, we'll "steal" probability mass from nearby samples that incorrectly preferred by the model
- For detailed analysis, see [Carreira-Perpinan and Hinton, 2005]



Today's Topics

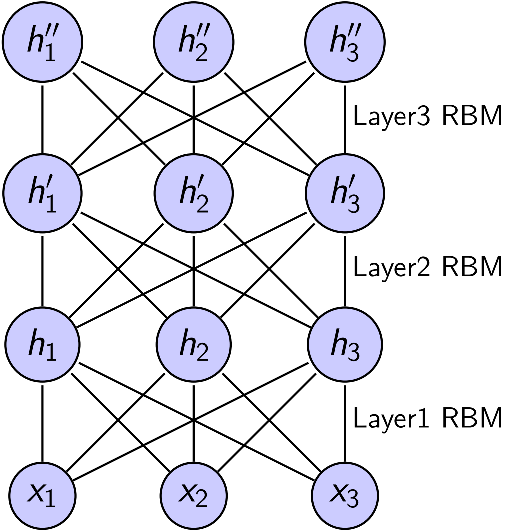
1. General Ideas in Deep Learning
   - Motivation for Deep Architectures and why is it hard?
   - Main Breakthrough in 2006: Layer-wise Pre-Training
2. Approach 1: Deep Belief Nets [Hinton et al., 2006]
   - Restricted Boltzmann Machines (RBM)
   - Training RBMs with Contrastive Divergence
   - Stacking RBMs to form Deep Belief Nets
3. Approach 2: Stacked Auto-Encoders [Bengio et al., 2006]
   - Auto-Encoders
   - Denoising Auto-Encoders
4. Discussions
   - Why it works, when it works, and the bigger picture

Deep Belief Nets (DBN) = Stacked RBM



- DBN defines a probabilistic generative model $p(x) = \sum_{h,h',h''} p(x|h)p(h|h')p(h',h'')$ (top 2 layers is interpreted as a RBM; lower layers are directed sigmoids)

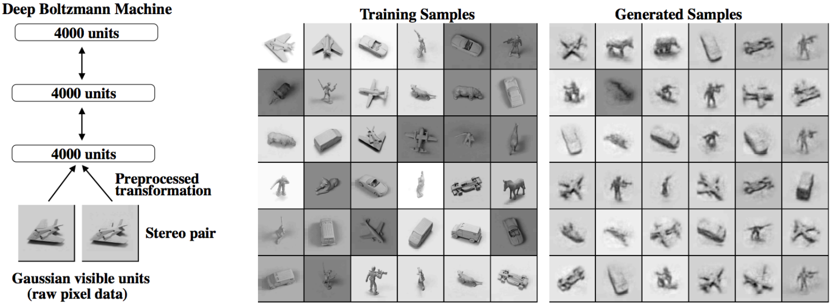
Deep Belief Nets (DBN) = Stacked RBM



- DBN defines a probabilistic generative model $p(x) = \sum_{h,h',h''} p(x|h)p(h|h')p(h',h'')$ (top 2 layers is interpreted as a RBM; lower layers are directed sigmoids)
- Stacked RBMs can also be used to initialize a Deep Neural Network (DNN)

Generating Data from a Deep Generative Model

After training on 20k images, the generative model of [Salakhutdinov and Hinton, 2009]* can generate random images (dimension=8976) that are amazingly realistic!



This model is a Deep Boltzmann Machine (DBM), different from Deep Belief Nets (DBN) but also built by stacking RBMs.

Summary: Things to remember about DBNs

1. Layer-wise pre-training is the innovation that rekindled interest in deep architectures.

Summary: Things to remember about DBNs

1. Layer-wise pre-training is the innovation that rekindled interest in deep architectures.
2. Pre-training focuses on optimizing likelihood on the data, not the target label. First model $p(x)$ to do better $p(y|x)$.

Summary: Things to remember about DBNs

1. Layer-wise pre-training is the innovation that rekindled interest in deep architectures.
2. Pre-training focuses on optimizing likelihood on the data, not the target label. First model $p(x)$ to do better $p(y|x)$.
3. Why RBM? $p(h|x)$ is tractable, so it's easy to stack.

Summary: Things to remember about DBNs

1. Layer-wise pre-training is the innovation that rekindled interest in deep architectures.
2. Pre-training focuses on optimizing likelihood on the data, not the target label. First model $p(x)$ to do better $p(y|x)$.
3. Why RBM? $p(h|x)$ is tractable, so it's easy to stack.
4. RBM training can be expensive. Solution: contrastive divergence

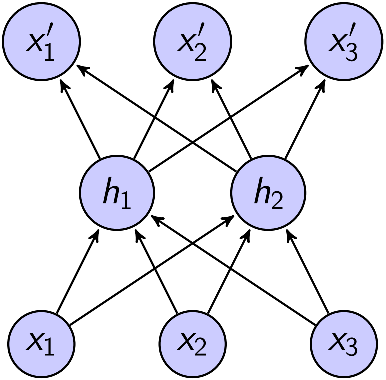
Summary: Things to remember about DBNs

1. Layer-wise pre-training is the innovation that rekindled interest in deep architectures.
2. Pre-training focuses on optimizing likelihood on the data, not the target label. First model $p(x)$ to do better $p(y|x)$.
3. Why RBM? $p(h|x)$ is tractable, so it's easy to stack.
4. RBM training can be expensive. Solution: contrastive divergence
5. DBN formed by stacking RBMs is a probabilistic generative model

Today's Topics

1. General Ideas in Deep Learning
   - Motivation for Deep Architectures and why is it hard?
   - Main Breakthrough in 2006: Layer-wise Pre-Training
2. Approach 1: Deep Belief Nets [Hinton et al., 2006]
   - Restricted Boltzmann Machines (RBM)
   - Training RBMs with Contrastive Divergence
   - Stacking RBMs to form Deep Belief Nets
3. Approach 2: Stacked Auto-Encoders [Bengio et al., 2006]
   - Auto-Encoders
   - Denoising Auto-Encoders
4. Discussions
   - Why it works, when it works, and the bigger picture

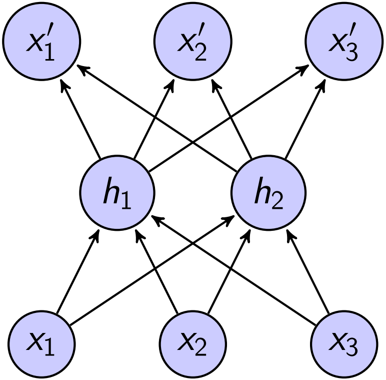
Auto-Encoders: simpler alternatives to RBMs



Decoder: $x' = \sigma(W'h + d)$

Encoder: $h = \sigma(Wx + b)$

Auto-Encoders: simpler alternatives to RBMs



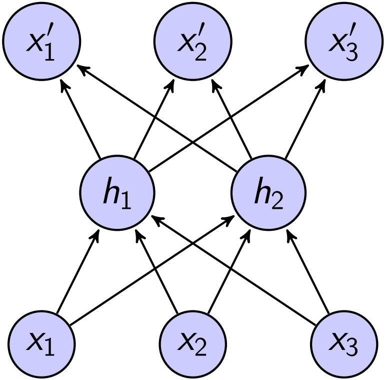
Decoder: $x' = \sigma(W'h + d)$

Encoder: $h = \sigma(Wx + b)$

Encourage h to give small reconstruction error:

- e.g. $Loss = \sum_m ||x^{(m)} - DECODER(ENCODER(x^{(m)}))||^2$

Auto-Encoders: simpler alternatives to RBMs



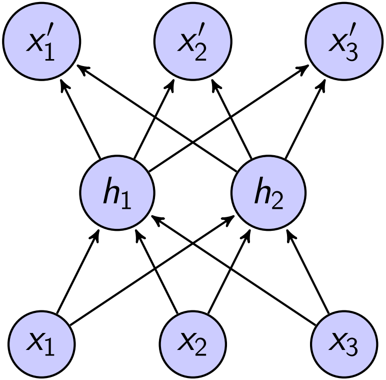
Decoder: $x' = \sigma(W'h + d)$

Encoder: $h = \sigma(Wx + b)$

Encourage h to give small reconstruction error:

- e.g. $Loss = \sum_m ||x^{(m)} - DECODER(ENCODER(x^{(m)}))||^2$
- Reconstruction: $x' = \sigma(W'\sigma(Wx + b) + d)$

Auto-Encoders: simpler alternatives to RBMs



Decoder: $x' = \sigma(W'h + d)$

Encoder: $h = \sigma(Wx + b)$

Encourage h to give small reconstruction error:

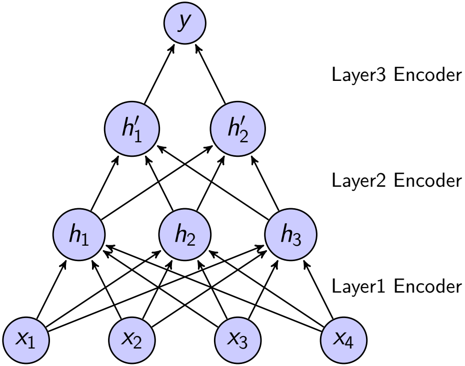
- e.g. $Loss = \sum_m ||x^{(m)} - DECODER(ENCODER(x^{(m)}))||^2$
- Reconstruction: $x' = \sigma(W'\sigma(Wx + b) + d)$
- This can be trained with the same Backpropagation algorithm for 2-layer nets, with $x^{(m)}$ as both input and output

Stacked Auto-Encoders (SAE)

- The encoder/decoder gives same form $p(h|x)$, $p(x|h)$ as RBMs, so can be stacked in the same way to form Deep Architectures

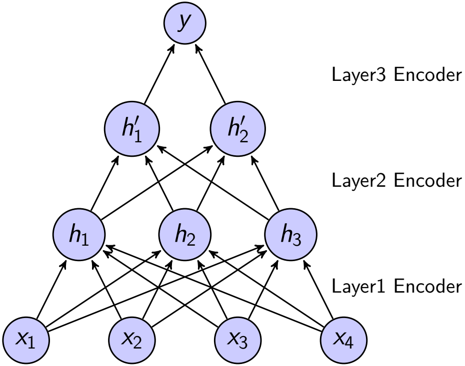
Stacked Auto-Encoders (SAE)

- The encoder/decoder gives same form $p(h|x)$, $p(x|h)$ as RBMs, so can be stacked in the same way to form Deep Architectures



Stacked Auto-Encoders (SAE)

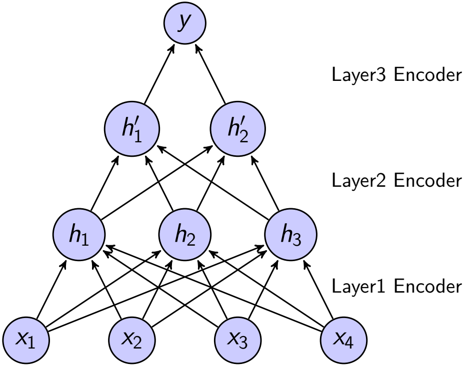
- The encoder/decoder gives same form $p(h|x)$, $p(x|h)$ as RBMs, so can be stacked in the same way to form Deep Architectures



- Unlike RBMs, Auto-encoders are deterministic.
  - $h = \sigma(Wx + b)$, not $p(h = \{0, 1\}) = \sigma(Wx + b)$

Stacked Auto-Encoders (SAE)

- The encoder/decoder gives same form $p(h|x)$, $p(x|h)$ as RBMs, so can be stacked in the same way to form Deep Architectures



- Unlike RBMs, Auto-encoders are deterministic.
  - $h = \sigma(Wx + b)$, not $p(h = \{0, 1\}) = \sigma(Wx + b)$
  - Disadvantage: Can't form deep generative model
  - Advantage: Fast to train, and useful still for Deep Neural Nets

Many Variants of Auto-Encoders

- Enforce compression to get latent factors (lower dimensional h)

Many Variants of Auto-Encoders

- Enforce compression to get latent factors (lower dimensional h)
- Linear encoder/decoder with squared reconstruction error learns same subspace of PCA [Bourlard and Kamp, 1988]

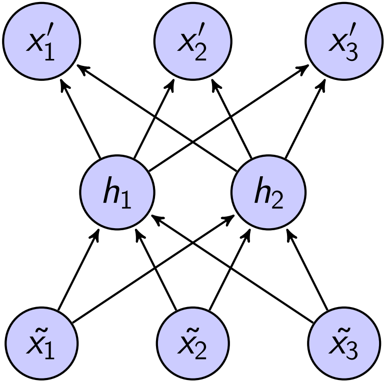
Many Variants of Auto-Encoders

- Enforce compression to get latent factors (lower dimensional h)
- Linear encoder/decoder with squared reconstruction error learns same subspace of PCA [Bourlard and Kamp, 1988]
- Enforce sparsity and over-complete representations (high dimensional h) [Ranzato et al., 2006]
- Enforce binary hidden layers to build hash codes [Salakhutdinov and Hinton, 2007]
- Incorporate domain knowledge, e.g. denoising auto-encoders [Vincent et al., 2010]

Today's Topics

1. General Ideas in Deep Learning
   - Motivation for Deep Architectures and why is it hard?
   - Main Breakthrough in 2006: Layer-wise Pre-Training
2. Approach 1: Deep Belief Nets [Hinton et al., 2006]
   - Restricted Boltzmann Machines (RBM)
   - Training RBMs with Contrastive Divergence
   - Stacking RBMs to form Deep Belief Nets
3. Approach 2: Stacked Auto-Encoders [Bengio et al., 2006]
   - Auto-Encoders
   - Denoising Auto-Encoders
4. Discussions
   - Why it works, when it works, and the bigger picture

Denoising Auto-Encoders



Decoder: $x' = \sigma(W'h + d)$

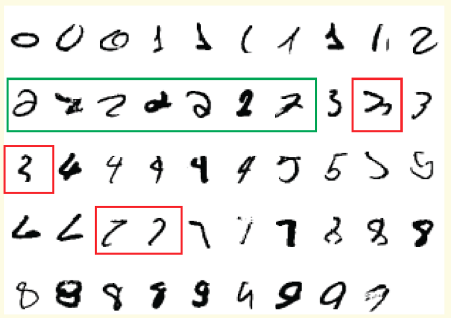
Encoder: $h = \sigma(W\tilde{x} + b)$

$\tilde{x} = x +$ noise

1. Perturb input data x to $\tilde{x}$ using invariance from domain knowledge.
2. Train weights to reduce reconstruction error with respect to original input: $||x - x'||$

Denoising Auto-Encoders

- Example: Randomly shift, rotate, and scale input image; add Gaussian or salt-and-pepper noise.
- A "2" is a "2" no matter how you add noise, so the auto-encoder will be forced to cancel the variations that are not important.



Summary: things to remember about SAE

1. Auto-Encoders are cheaper alternatives to RBMs.
   - Not probabilistic, but fast to train using Backpropagation or SGD

Summary: things to remember about SAE

1. Auto-Encoders are cheaper alternatives to RBMs.
   - Not probabilistic, but fast to train using Backpropagation or SGD
2. Auto-Encoders learn to "compress" and "re-construct" input data. Again, the focus is on modeling $p(x)$ first.

Summary: things to remember about SAE

1. Auto-Encoders are cheaper alternatives to RBMs.
    - Not probabilistic, but fast to train using Backpropagation or SGD
2. Auto-Encoders learn to "compress" and "re-construct" input data. Again, the focus is on modeling $p(x)$ first.
3. Many variants, some provide ways to incorporate domain knowledge.

Today's Topics

1. General Ideas in Deep Learning
   - Motivation for Deep Architectures and why is it hard?
   - Main Breakthrough in 2006: Layer-wise Pre-Training
2. Approach 1: Deep Belief Nets [Hinton et al., 2006]
   - Restricted Boltzmann Machines (RBM)
   - Training RBMs with Contrastive Divergence
   - Stacking RBMs to form Deep Belief Nets
3. Approach 2: Stacked Auto-Encoders [Bengio et al., 2006]
   - Auto-Encoders
   - Denoising Auto-Encoders
4. Discussions
   - Why it works, when it works, and the bigger picture

Why does Layer-wise Pre-Training work?

One Hypothesis [Bengio, 2009, Erhan et al., 2010]:

- A deep net can fit the training data in many ways (non-convex):
  1. By optimizing upper-layers really hard
  2. By optimizing lower-layers really hard

Why does Layer-wise Pre-Training work?

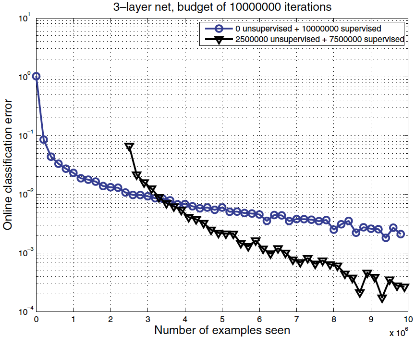
One Hypothesis [Bengio, 2009, Erhan et al., 2010]:

- A deep net can fit the training data in many ways (non-convex):
  1. By optimizing upper-layers really hard
  2. By optimizing lower-layers really hard
- Top-down vs. Bottom-up information
  1. Even if lower-layers are random weights, upper-layer may still fit well. But this might not generalize to new data
  2. Pre-training with objective on $P(x)$ learns more generalizable features

Why does Layer-wise Pre-Training work?

One Hypothesis [Bengio, 2009, Erhan et al., 2010]:

- A deep net can fit the training data in many ways (non-convex):
  1. By optimizing upper-layers really hard
  2. By optimizing lower-layers really hard
- Top-down vs. Bottom-up information
  1. Even if lower-layers are random weights, upper-layer may still fit well. But this might not generalize to new data
  2. Pre-training with objective on $P(x)$ learns more generalizable features
- Pre-training seems to help put weights at a better local optimum



Is Layer-wise Pre-Training always necessary?

Is Layer-wise Pre-Training always necessary?

Answer in 2006: Yes!

Is Layer-wise Pre-Training always necessary?

Answer in 2006: Yes!
Answer in 2014: No!

1. If initialization is done well by design (e.g. sparse connections and convolutional nets), maybe won't have vanishing gradient problem

Is Layer-wise Pre-Training always necessary?

Answer in 2006: Yes!
Answer in 2014: No!

1. If initialization is done well by design (e.g. sparse connections and convolutional nets), maybe won't have vanishing gradient problem
2. If you have an extremely large datasets, maybe won't overfit. (But maybe that also means you want an ever deeper net)

Is Layer-wise Pre-Training always necessary?

Answer in 2006: Yes!

Answer in 2014: No!

1. If initialization is done well by design (e.g. sparse connections and convolutional nets), maybe won't have vanishing gradient problem
2. If you have an extremely large datasets, maybe won't overfit. (But maybe that also means you want an ever deeper net)
3. New architectures are emerging:
   - Stacked SVM's with random projections [Vinyals et al., 2012]
   - Sum-Product Networks [Poon and Domingos, 2011]

Connections with other Machine Learning concepts

- A RBM is like a product-of-expert model and forms a distributed representation of the data
  - Compared with clustering (which compresses data but loses information), distributed representations (multi-clustering) are richer representations
  - Like a mixture model with 2^n hidden components $p(x) = \sum_h p(h)p(x|h)$, but much more compact

Connections with other Machine Learning concepts

- A RBM is like a product-of-expert model and forms a distributed representation of the data
  - Compared with clustering (which compresses data but loses information), distributed representations (multi-clustering) are richer representations
  - Like a mixture model with 2^n hidden components $p(x) = \sum_h p(h)p(x|h)$, but much more compact
- Neural Net as kernel for SVM [Li et al., 2005] and SVM training for Neural Nets [Collobert and Bengio, 2004]

Connections with other Machine Learning concepts

- A RBM is like a product-of-expert model and forms a distributed representation of the data
  - Compared with clustering (which compresses data but loses information), distributed representations (multi-clustering) are richer representations
  - Like a mixture model with 2^n hidden components $p(x) = \sum_h p(h)p(x|h)$, but much more compact
- Neural Net as kernel for SVM [Li et al., 2005] and SVM training for Neural Nets [Collobert and Bengio, 2004]
- Decision trees are deep (but no distributed representation). Random forests are both deep and distributed. They do well in practice too!

Connections with other Machine Learning concepts

- A RBM is like a product-of-expert model and forms a distributed representation of the data
  - Compared with clustering (which compresses data but loses information), distributed representations (multi-clustering) are richer representations
  - Like a mixture model with 2^n hidden components $p(x) = \sum_h p(h)p(x|h)$, but much more compact
- Neural Net as kernel for SVM [Li et al., 2005] and SVM training for Neural Nets [Collobert and Bengio, 2004]
- Decision trees are deep (but no distributed representation). Random forests are both deep and distributed. They do well in practice too!
- Philosophical connections to:
  - Semi-supervised Learning: exploit both labeled and unlabeled data
  - Curriculum Learning: start on easy task, gradually level-up
  - Multi-task Learning: learn and share sub-tasks

History

- Early days of AI. Invention of artificial neuron [McCulloch and Pitts, 1943] & perceptron [Rosenblatt, 1958]

History

- Early days of AI. Invention of artificial neuron [McCulloch and Pitts, 1943] & perceptron [Rosenblatt, 1958]
- AI Winter. [Minsky and Papert, 1969] showed perceptron only learns linearly separable concepts

History

- Early days of AI. Invention of artificial neuron [McCulloch and Pitts, 1943] & perceptron [Rosenblatt, 1958]
- AI Winter. [Minsky and Papert, 1969] showed perceptron only learns linearly separable concepts
- Revival in 1980s: Multi-layer Perceptrons (MLP) and Back-propagation [Rumelhart et al., 1986]

History

- Early days of AI. Invention of artificial neuron [McCulloch and Pitts, 1943] & perceptron [Rosenblatt, 1958]
- AI Winter. [Minsky and Papert, 1969] showed perceptron only learns linearly separable concepts
- Revival in 1980s: Multi-layer Perceptrons (MLP) and Back-propagation [Rumelhart et al., 1986]
- Other directions (1990s - present): SVMs, Bayesian Networks

History

- Early days of AI. Invention of artificial neuron [McCulloch and Pitts, 1943] & perceptron [Rosenblatt, 1958]
- AI Winter. [Minsky and Papert, 1969] showed perceptron only learns linearly separable concepts
- Revival in 1980s: Multi-layer Perceptrons (MLP) and Back-propagation [Rumelhart et al., 1986]
- Other directions (1990s - present): SVMs, Bayesian Networks
- Revival in 2006: Deep learning [Hinton et al., 2006]

History

- Early days of AI. Invention of artificial neuron [McCulloch and Pitts, 1943] & perceptron [Rosenblatt, 1958]
- AI Winter. [Minsky and Papert, 1969] showed perceptron only learns linearly separable concepts
- Revival in 1980s: Multi-layer Perceptrons (MLP) and Back-propagation [Rumelhart et al., 1986]
- Other directions (1990s - present): SVMs, Bayesian Networks
- Revival in 2006: Deep learning [Hinton et al., 2006]
- Successes in applications: Speech at IBM/Toronto [Sainath et al., 2011], Microsoft [Dahl et al., 2012]. Vision at Google/Stanford [Le et al., 2012]

References I

Bengio, Y. (2009).
Learning Deep Architectures for AI, volume Foundations and Trends in Machine Learning.
NOW Publishers.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2006).
Greedy layer-wise training of deep networks.
In *NIPS'06*, pages 153–160.

Bishop, C. (2006).
Pattern Recognition and Machine Learning.
Springer.

Bourlard, H. and Kamp, Y. (1988).
Auto-association by multilayer perceptrons and singular value decomposition.
Biological Cybernetics, 59:291–294.

References II

Carreira-Perpinan, M. A. and Hinton, G. E. (2005).
On contrastive divergence learning.
In *AISTATS*.

Collobert, R. and Bengio, S. (2004).
Links between perceptrons, MLPs and SVMs.
In *ICML*.

Dahl, G., Yu, D., Deng, L., and Acero, A. (2012).
Context-dependent pre-trained deep neural networks for large vocabulary speech recognition.
IEEE Transactions on Audio, Speech, and Language Processing, Special Issue on Deep Learning for Speech and Langauge Processing.

References III

Erhan, D., Bengio, Y., Courville, A., Manzagol, P., Vincent, P., and Bengio, S. (2010).
Why does unsupervised pre-training help deep learning?
Journal of Machine Learning Research, 11:625–660.

Erhan, D., Manzagol, P., Bengio, Y., Bengio, S., and Vincent, P. (2009).
The difficulty of training deep architectures and the effect of unsupervised pre-training.
In *AISTATS*.

Hinton, G., Osindero, S., and Teh, Y.-W. (2006).
A fast learning algorithm for deep belief nets.
Neural Computation, 18:1527–1554.

References IV

Le, Q. V., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G. S., Dean, J., and Ng, A. Y. (2012).
Building high-level features using large scale unsupervised learning.
In *ICML*.

Li, X., Bilmes, J., and Malkin, J. (2005).
Maximum margin learning and adaptation of MLP classifiers.
In *Interspeech*.

McCulloch, W. S. and Pitts, W. H. (1943).
A logical calculus of the ideas immanent in nervous activity.
In *Bulletin of Mathematical Biophysics*, volume 5, pages 115–137.

Minsky, M. and Papert, S. (1969).
Perceptrons: an introduction to computational geometry.
MIT Press.

References V

Poon, H. and Domingos, P. (2011).
Sum-product networks.
In *UAI*.

Ranzato, M., Boureau, Y.-L., and LeCun, Y. (2006).
Sparse feature learning for deep belief networks.
In *NIPS*.

Rosenblatt, F. (1958).
The perceptron: A probabilistic model for information storage and organization in the brain.
Psychological Review, 65:386–408.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986).
Learning representations by back-propagating errors.
Nature, 323:533–536.

References VI

Sainath, T. N., Kingsbury, B., Ramabhadran, B., Fousek, P., Novak, P., and Mohamed, A. (2011).
Making deep belief networks effective for large vocabulary continuous speech recognition.
In *ASRU*.

Salakhutdinov, R. and Hinton, G. (2007).
Semantic hashing.
In *SIGIR*.

Salakhutdinov, R. and Hinton, G. (2009).
Deep Boltzmann machines.
In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pages 448–455.

References VII

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010).
Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion.
Journal of Machine Learning Research, 11:3371–3408.

Vinyals, O., Jia, Y., Deng, L., and Darrell, T. (2012).
Learning with recursive perceptual representations.
In *NIPS*.