# Lecture 16
# Deep Neural Generative Models

## CMSC 35246: Deep Learning

Shubhendu Trivedi
&
Risi Kondor

University of Chicago

May 22, 2017

# Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants

# Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants
- Example: PCA (and Linear Autoencoder) to Nonlinear-PCA (Non-linear (deep?) autoencoders)

# Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants
- Example: PCA (and Linear Autoencoder) to Nonlinear-PCA (Non-linear (deep?) autoencoders)
- Example: Sparse Coding (Sparse Autoencoder with linear decoding) to Deep Sparse Autoencoders

## Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants
- Example: PCA (and Linear Autoencoder) to Nonlinear-PCA (Non-linear (deep?) autoencoders)
- Example: Sparse Coding (Sparse Autoencoder with linear decoding) to Deep Sparse Autoencoders
- All the models we have considered so far are completely deterministic

# Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants
- Example: PCA (and Linear Autoencoder) to Nonlinear-PCA (Non-linear (deep?) autoencoders)
- Example: Sparse Coding (Sparse Autoencoder with linear decoding) to Deep Sparse Autoencoders
- All the models we have considered so far are completely deterministic
- The encoder and decoders have no stochasticity

# Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants
- Example: PCA (and Linear Autoencoder) to Nonlinear-PCA (Non-linear (deep?) autoencoders)
- Example: Sparse Coding (Sparse Autoencoder with linear decoding) to Deep Sparse Autoencoders
- All the models we have considered so far are completely deterministic
- The encoder and decoders have no stochasticity
- We don't construct a probabilistic model of the data

# Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants
- Example: PCA (and Linear Autoencoder) to Nonlinear-PCA (Non-linear (deep?) autoencoders)
- Example: Sparse Coding (Sparse Autoencoder with linear decoding) to Deep Sparse Autoencoders
- All the models we have considered so far are completely deterministic
- The encoder and decoders have no stochasticity
- We don't construct a probabilistic model of the data
- Can't sample from the model

# Representations

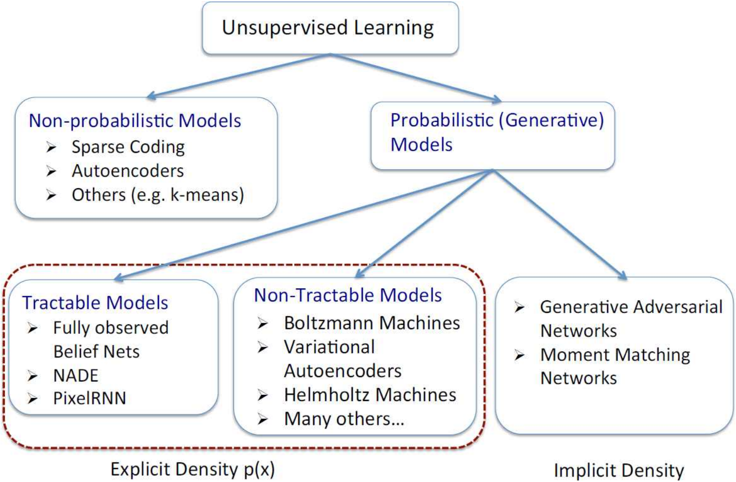



Figure: Ruslan Salakhutdinov

- To motivate Deep Neural Generative models, like before, let's seek inspiration from simple linear models first

# Linear Factor Model

- We want to build a probabilistic model of the input $\tilde{P}(\mathbf{x})$

# Linear Factor Model

- We want to build a probabilistic model of the input $\tilde{P}(\mathbf{x})$
- Like before, we are interested in latent factors $\mathbf{h}$ that *explain* $\mathbf{x}$

# Linear Factor Model

- We want to build a probabilistic model of the input $\tilde{P}(\mathbf{x})$
- Like before, we are interested in latent factors $\mathbf{h}$ that *explain* $\mathbf{x}$
- We then care about the marginal:

$$\tilde{P}(\mathbf{x}) = \mathbb{E}_{\mathbf{h}}\tilde{P}(\mathbf{x}|\mathbf{h})$$

# Linear Factor Model

- We want to build a probabilistic model of the input $\tilde{P}(\mathbf{x})$
- Like before, we are interested in latent factors $\mathbf{h}$ that *explain* $\mathbf{x}$
- We then care about the marginal:

$$\tilde{P}(\mathbf{x}) = \mathbb{E}_{\mathbf{h}} \tilde{P}(\mathbf{x}|\mathbf{h})$$

- $\mathbf{h}$ is a *representation* of the data

# Linear Factor Model

- The latent factors $\mathbf{h}$ are an *encoding* of the data

# Linear Factor Model

- The latent factors $\mathbf{h}$ are an *encoding* of the data
- Simplest decoding model: Get $\mathbf{x}$ after a linear transformation of $\mathbf{x}$ with some noise

# Linear Factor Model

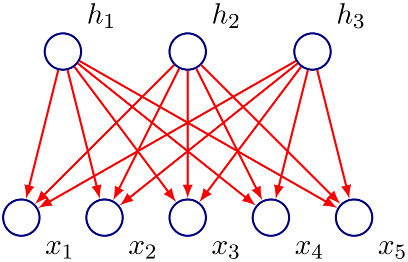
- The latent factors $\mathbf{h}$ are an *encoding* of the data
- Simplest decoding model: Get $\mathbf{x}$ after a linear transformation of $\mathbf{x}$ with some noise
- Formally: Suppose we sample the latent factors from a distribution $\mathbf{h} \sim P(\mathbf{h})$

# Linear Factor Model

- The latent factors $\mathbf{h}$ are an *encoding* of the data
- Simplest decoding model: Get $\mathbf{x}$ after a linear transformation of $\mathbf{x}$ with some noise
- Formally: Suppose we sample the latent factors from a distribution $\mathbf{h} \sim P(\mathbf{h})$
- Then: $\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$

# Linear Factor Model

- $P(\mathbf{h})$ is a factorial distribution




$$\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$$

- How do learn in such a model?
- Let's look at a simple example

# Probabilistic PCA

- Suppose underlying latent factor has a Gaussian distribution

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

# Probabilistic PCA

- Suppose underlying latent factor has a Gaussian distribution

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

- For the noise model: Assume $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$

# Probabilistic PCA

- Suppose underlying latent factor has a Gaussian distribution

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

- For the noise model: Assume $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$
- Then:

$$P(\mathbf{x}|\mathbf{h}) = \mathcal{N}(\mathbf{x}|W\mathbf{h} + \mathbf{b}, \sigma^2 I)$$

# Probabilistic PCA

- Suppose underlying latent factor has a Gaussian distribution

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

- For the noise model: Assume $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$
- Then:

$$P(\mathbf{x}|\mathbf{h}) = \mathcal{N}(\mathbf{x}|W\mathbf{h} + \mathbf{b}, \sigma^2 I)$$

- We care about the marginal $P(\mathbf{x})$ (predictive distribution):

$$P(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{b}, WW^T + \sigma^2 I)$$

## Probabilistic PCA

$$P(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{b}, WW^T + \sigma^2 I)$$

- How do we learn the parameters? (EM, ML Estimation)

# Probabilistic PCA

$$P(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{b}, WW^T + \sigma^2 I)$$

- How do we learn the parameters? (EM, ML Estimation)
- Let's look at the ML Estimation:

# Probabilistic PCA

$$P(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{b}, WW^T + \sigma^2 I)$$

- How do we learn the parameters? (EM, ML Estimation)
- Let's look at the ML Estimation:
- Let $C = WW^T + \sigma^2 I$

# Probabilistic PCA

$$P(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{b}, WW^T + \sigma^2 I)$$

- How do we learn the parameters? (EM, ML Estimation)
- Let's look at the ML Estimation:
- Let $C = WW^T + \sigma^2 I$
- We want to maximize $\ell(\theta; X) = \sum_i \log P(\mathbf{x}_i|\theta)$

# Probabilistic PCA: ML Estimation

$$\ell(\theta; X) = \sum_i \log P(\mathbf{x}_i|\theta)$$

$$= -\frac{N}{2}\log|C| - \frac{1}{2}\sum_i (\mathbf{x}_i - \mathbf{b})C^{-1}(\mathbf{x}_i - \mathbf{b})^T$$

$$= -\frac{N}{2}\log|C| - \frac{1}{2}Tr[(C^{-1}\sum_i \mathbf{x}_i - \mathbf{b})(\mathbf{x}_i - \mathbf{b})^T]$$

$$= \frac{N}{2}\log|C| - \frac{1}{2}Tr[(C^{-1}S]$$

# Probabilistic PCA: ML Estimation

$$\ell(\theta; X) = \sum_i \log P(\mathbf{x}_i|\theta)$$

$$= -\frac{N}{2}\log|C| - \frac{1}{2}\sum_i (\mathbf{x}_i - \mathbf{b})C^{-1}(\mathbf{x}_i - \mathbf{b})^T$$

$$= -\frac{N}{2}\log|C| - \frac{1}{2}Tr[(C^{-1}\sum_i \mathbf{x}_i - \mathbf{b})(\mathbf{x}_i - \mathbf{b})^T]$$

$$= \frac{N}{2}\log|C| - \frac{1}{2}Tr[(C^{-1}S]$$

- Now fit the parameters $\theta = W, \mathbf{b}, \sigma$ to maximize log-likelihood

# Probabilistic PCA: ML Estimation

$$
\begin{aligned}
\ell(\theta; X) &= \sum_i \log P(\mathbf{x}_i|\theta) \\
&= -\frac{N}{2}\log|C| - \frac{1}{2}\sum_i (\mathbf{x}_i - \mathbf{b})C^{-1}(\mathbf{x}_i - \mathbf{b})^T \\
&= -\frac{N}{2}\log|C| - \frac{1}{2}Tr[(C^{-1}\sum_i \mathbf{x}_i - \mathbf{b})(\mathbf{x}_i - \mathbf{b})^T] \\
&= \frac{N}{2}\log|C| - \frac{1}{2}Tr[(C^{-1}S]
\end{aligned}
$$

- Now fit the parameters $\theta = W, \mathbf{b}, \sigma$ to maximize log-likelihood
- Can also use EM

# Factor Analysis

- Fix the latent factor prior to be the unit Gaussian as before:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

# Factor Analysis

- Fix the latent factor prior to be the unit Gaussian as before:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

- Noise is sampled from a Gaussian with a diagonal covariance:

# Factor Analysis

- Fix the latent factor prior to be the unit Gaussian as before:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

- Noise is sampled from a Gaussian with a diagonal covariance:

$$\Psi = diag([\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2])$$

# Factor Analysis

- Fix the latent factor prior to be the unit Gaussian as before:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

- Noise is sampled from a Gaussian with a diagonal covariance:

$$\Psi = diag([\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2])$$

- Still consider linear relationship between inputs and observed variables: Marginal $P(\mathbf{x}) \sim \mathcal{N}(\mathbf{x}; b, WW^T + \Psi)$

# Factor Analysis

- On deriving the posterior $P(\mathbf{h}|\mathbf{x}) = \mathcal{N}(\mathbf{h}|\mu, \Lambda)$, we get:

# Factor Analysis

- On deriving the posterior $P(\mathbf{h}|\mathbf{x}) = \mathcal{N}(\mathbf{h}|\mu, \Lambda)$, we get:
- $\mu = W^T(WW^T + \Psi)^{-1}(\mathbf{x} - \mathbf{b})$

# Factor Analysis

- On deriving the posterior $P(\mathbf{h}|\mathbf{x}) = \mathcal{N}(\mathbf{h}|\mu, \Lambda)$, we get:
- $\mu = W^T(WW^T + \Psi)^{-1}(\mathbf{x} - \mathbf{b})$
- $\Lambda = I - W^T(WW^T + \Psi)^{-1}W$

# Factor Analysis

- On deriving the posterior $P(\mathbf{h}|\mathbf{x}) = \mathcal{N}(\mathbf{h}|\mu, \Lambda)$, we get:
- $\mu = W^T(WW^T + \Psi)^{-1}(\mathbf{x} - \mathbf{b})$
- $\Lambda = I - W^T(WW^T + \Psi)^{-1}W$
- Parameters are coupled, makes ML estimation difficult

# Factor Analysis

- On deriving the posterior $P(\mathbf{h}|\mathbf{x}) = \mathcal{N}(\mathbf{h}|\mu, \Lambda)$, we get:
- $\mu = W^T(WW^T + \Psi)^{-1}(\mathbf{x} - \mathbf{b})$
- $\Lambda = I - W^T(WW^T + \Psi)^{-1}W$
- Parameters are coupled, makes ML estimation difficult
- Need to employ EM (or non-linear optimization)

# More General Models

- Suppose $P(\mathbf{h})$ can not be assumed to have a nice Gaussian form

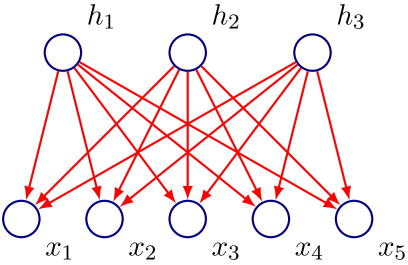
# More General Models

- Suppose $P(\mathbf{h})$ can not be assumed to have a nice Gaussian form
- The decoding of the input from the latent states can be a complicated non-linear function

# More General Models

- Suppose $P(\mathbf{h})$ can not be assumed to have a nice Gaussian form
- The decoding of the input from the latent states can be a complicated non-linear function
- Estimation and inference can get complicated!

# Earlier we had:



# Quick Review

- Generative models can be modeled as directed graphical models

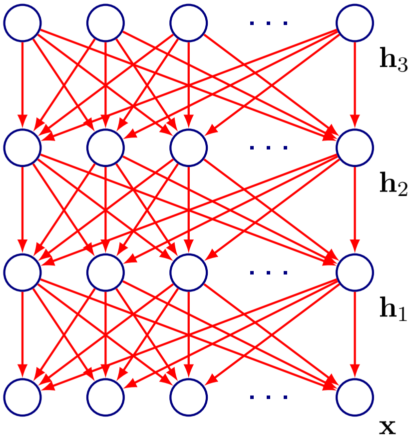
# Quick Review

- Generative models can be modeled as directed graphical models
- The nodes represent random variables and arcs indicate dependency

# Quick Review

- Generative models can be modeled as directed graphical models
- The nodes represent random variables and arcs indicate dependency
- Some of the random variables are observed, others are hidden

# Sigmoid Belief Networks




- Just like a feedfoward network, but with arrows reversed.

# Sigmoid Belief Networks

- Let $\mathbf{x} = \mathbf{h}^0$. Consider binary activations, then:

# Sigmoid Belief Networks

- Let $\mathbf{x} = \mathbf{h}^0$. Consider binary activations, then:

$$P(\mathbf{h}_i^k = 1|\mathbf{h}^{k+1}) = sigm(b_i^k + \sum_j W_{i,j}^{k+1}\mathbf{h}_j^{k+1})$$

# Sigmoid Belief Networks

- Let $\mathbf{x} = \mathbf{h}^0$. Consider binary activations, then:

$$P(\mathbf{h}_i^k = 1|\mathbf{h}^{k+1}) = sigm(b_i^k + \sum_j W_{i,j}^{k+1}\mathbf{h}_j^{k+1})$$

- The joint probability factorizes as:

# Sigmoid Belief Networks

- Let $\mathbf{x} = \mathbf{h}^0$. Consider binary activations, then:

$$P(\mathbf{h}_i^k = 1 | \mathbf{h}^{k+1}) = sigm(b_i^k + \sum_j W_{i,j}^{k+1} \mathbf{h}_j^{k+1})$$

- The joint probability factorizes as:

$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = P(\mathbf{h}^l) \Big( \prod_{k=1}^{l-1} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \Big) P(\mathbf{x} | \mathbf{h}^1)$$

# Sigmoid Belief Networks

- Let $\mathbf{x} = \mathbf{h}^0$. Consider binary activations, then:

$$P(\mathbf{h}_i^k = 1|\mathbf{h}^{k+1}) = sigm(b_i^k + \sum_j W_{i,j}^{k+1}\mathbf{h}_j^{k+1})$$

- The joint probability factorizes as:

$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = P(\mathbf{h}^l)\Bigg(\prod_{k=1}^{l-1} P(\mathbf{h}^k|\mathbf{h}^{k+1})\Bigg)P(\mathbf{x}|\mathbf{h}^1)$$

- Marginalization yields $P(\mathbf{x})$, intractable in practice except for very small models

# Sigmoid Belief Networks

$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = P(\mathbf{h}^l)\Big(\prod_{k=1}^{l-1} P(\mathbf{h}^k|\mathbf{h}^{k+1})\Big)P(\mathbf{x}|\mathbf{h}^1)$$

# Sigmoid Belief Networks

$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = P(\mathbf{h}^l)\Big(\prod_{k=1}^{l-1} P(\mathbf{h}^k|\mathbf{h}^{k+1})\Big)P(\mathbf{x}|\mathbf{h}^1)$$

- The top level prior is chosen as factorizable: $P(\mathbf{h}^l) = \prod_i P(\mathbf{h}_i^l)$

# Sigmoid Belief Networks

$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = P(\mathbf{h}^l)\Big(\prod_{k=1}^{l-1} P(\mathbf{h}^k|\mathbf{h}^{k+1})\Big)P(\mathbf{x}|\mathbf{h}^1)$$

- The top level prior is chosen as factorizable: $P(\mathbf{h}^l) = \prod_i P(\mathbf{h}_i^l)$
- A single (Bernoulli) parameter is needed for each $\mathbf{h}_i$ in case of binary units

# Sigmoid Belief Networks

$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = P(\mathbf{h}^l)\Big(\prod_{k=1}^{l-1} P(\mathbf{h}^k|\mathbf{h}^{k+1})\Big)P(\mathbf{x}|\mathbf{h}^1)$$

- The top level prior is chosen as factorizable: $P(\mathbf{h}^l) = \prod_i P(\mathbf{h}_i^l)$
- A single (Bernoulli) parameter is needed for each $\mathbf{h}_i$ in case of binary units
- Deep Belief Networks are like Sigmoid Belief Networks except for the top two layers

# Sigmoid Belief Networks

- General case models are called Helmholtz Machines

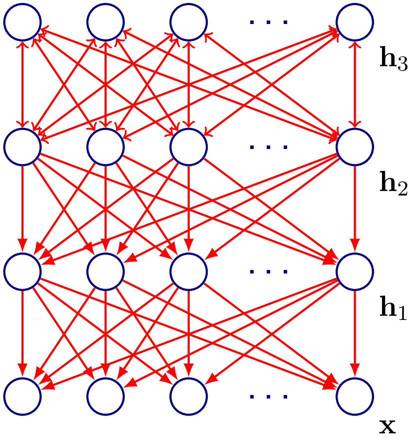
# Sigmoid Belief Networks

- General case models are called Helmholtz Machines
- Two key references:
  - G. E. Hinton, P. Dayan, B. J. Frey, R. M. Neal: The Wake-Sleep Algorithm for Unsupervised Neural Networks, In **Science**, 1995

# Sigmoid Belief Networks

- General case models are called Helmholtz Machines
- Two key references:
  - G. E. Hinton, P. Dayan, B. J. Frey, R. M. Neal: The Wake-Sleep Algorithm for Unsupervised Neural Networks, In **Science**, 1995
  - R. M. Neal: Connectionist Learning of Belief Networks, In **Artificial Intelligence**, 1992

# Deep Belief Networks




- The top two layers now have undirected edges

# Deep Belief Networks

- The joint probability changes as:

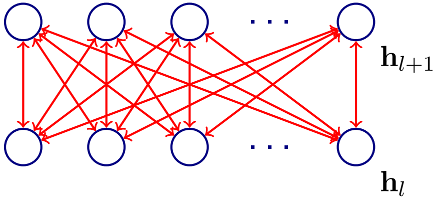
$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = P(\mathbf{h}^l, \mathbf{h}^{l-1}) \Big( \prod_{k=1}^{l-2} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \Big) P(\mathbf{x} | \mathbf{h}^1)$$

# Deep Belief Networks

- The joint probability changes as:

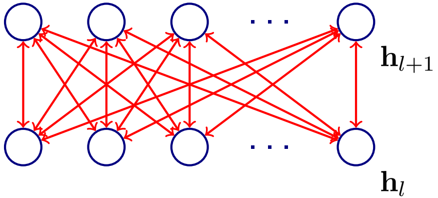
$$P(\mathbf{x}, \mathbf{h}^1, \dots, \mathbf{h}^l) = P(\mathbf{h}^l, \mathbf{h}^{l-1}) \Big( \prod_{k=1}^{l-2} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \Big) P(\mathbf{x} | \mathbf{h}^1)$$

# Deep Belief Networks



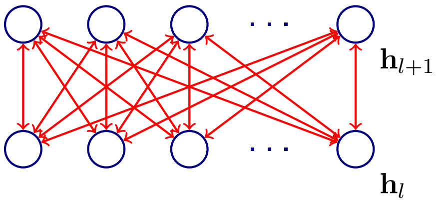

- The top two layers are a Restricted Boltzmann Machine

# Deep Belief Networks




- The top two layers are a Restricted Boltzmann Machine
- A RBM has the joint distribution:

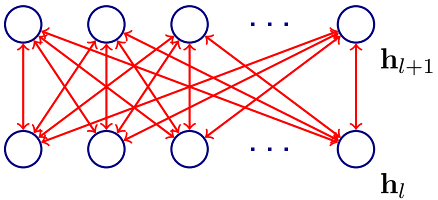
# Deep Belief Networks




- The top two layers are a Restricted Boltzmann Machine
- A RBM has the joint distribution:

$$P(\mathbf{h}^{l+1}, \mathbf{h}^{l}) \propto \exp(\mathbf{b}'\mathbf{h}^{l-1} + \mathbf{c}'\mathbf{h}^{l} + \mathbf{h}^{l} W \mathbf{h}^{l-1})$$

# Deep Belief Networks




- The top two layers are a Restricted Boltzmann Machine
- A RBM has the joint distribution:

$$P(\mathbf{h}^{l+1}, \mathbf{h}^l) \propto \exp(\mathbf{b}'\mathbf{h}^{l-1} + \mathbf{c}'\mathbf{h}^l + \mathbf{h}^l W \mathbf{h}^{l-1})$$

- We will return to RBMs and training procedures in a while, but first we look at the mathematical machinery that will make our task easier

# Energy Based Models

- Energy-Based Models assign a scalar energy with *every configuration* of variables under consideration

# Energy Based Models

- Energy-Based Models assign a scalar energy with *every configuration* of variables under consideration
- Learning: Change the energy function so that its final shape has some desirable properties

# Energy Based Models

- Energy-Based Models assign a scalar energy with *every configuration* of variables under consideration
- Learning: Change the energy function so that its final shape has some desirable properties
- We can define a probability distribution through an energy:

$$P(\mathbf{x}) = \frac{\exp^{-(\text{Energy}(\mathbf{x}))}}{Z}$$

# Energy Based Models

- Energy-Based Models assign a scalar energy with *every configuration* of variables under consideration
- Learning: Change the energy function so that its final shape has some desirable properties
- We can define a probability distribution through an energy:

$$P(\mathbf{x}) = \frac{\exp^{-(\text{Energy}(\mathbf{x}))}}{Z}$$

- Energies are in the log-probability domain:

$$\text{Energy}(\mathbf{x}) = \log \frac{1}{(ZP(\mathbf{x}))}$$

# Energy Based Models

$$P(\mathbf{x}) = \frac{\exp^{-(\textsf{Energy}(\mathbf{x}))}}{Z}$$

# Energy Based Models

$$P(\mathbf{x}) = \frac{\exp^{-(\text{Energy}(\mathbf{x}))}}{Z}$$

- Z is a normalizing factor called the Partition Function

$$Z = \sum_{\mathbf{x}} \exp(-\text{Energy}(\mathbf{x}))$$

# Energy Based Models

$$P(\mathbf{x}) = \frac{\exp^{-(\text{Energy}(\mathbf{x}))}}{Z}$$

- Z is a normalizing factor called the Partition Function

$$Z = \sum_{\mathbf{x}} \exp(-\text{Energy}(\mathbf{x}))$$

- How do we specify the energy function?

# Product of Experts Formulation

- In this formulation, the energy function is:

$$\text{Energy}(\mathbf{x}) = \sum_i f_i(\mathbf{x})$$

# Product of Experts Formulation

- In this formulation, the energy function is:

$$\text{Energy}(\mathbf{x}) = \sum_i f_i(\mathbf{x})$$

- Therefore:

$$P(\mathbf{x}) = \frac{\exp^{-(\sum_i f_i(\mathbf{x}))}}{Z}$$

# Product of Experts Formulation

- In this formulation, the energy function is:

$$\text{Energy}(\mathbf{x}) = \sum_i f_i(\mathbf{x})$$

- Therefore:

$$P(\mathbf{x}) = \frac{\exp^{-(\sum_i f_i(\mathbf{x}))}}{Z}$$

- We have the product of experts:

$$P(\mathbf{x}) \propto \prod_i P_i(\mathbf{x}) \propto \prod_i \exp^{(-f_i(\mathbf{x}))}$$

# Product of Experts Formulation

$$P(\mathbf{x}) \propto \prod_i P_i(\mathbf{x}) \propto \prod_i \exp^{(-f_i(\mathbf{x}))}$$

- Every expert $f_i$ can be seen as enforcing a constraint on $\mathbf{x}$

# Product of Experts Formulation

$$P(\mathbf{x}) \propto \prod_i P_i(\mathbf{x}) \propto \prod_i \exp^{(-f_i(\mathbf{x}))}$$

- Every expert $f_i$ can be seen as enforcing a constraint on $\mathbf{x}$
- If $f_i$ is large $\implies$ $P_i(\mathbf{x})$ is small i.e. the expert thinks $\mathbf{x}$ is implausible (constraint violated)

# Product of Experts Formulation

$$P(\mathbf{x}) \propto \prod_i P_i(\mathbf{x}) \propto \prod_i \exp^{(-f_i(\mathbf{x}))}$$

- Every expert $f_i$ can be seen as enforcing a constraint on $\mathbf{x}$
- If $f_i$ is large $\implies P_i(\mathbf{x})$ is small i.e. the expert thinks $\mathbf{x}$ is implausible (constraint violated)
- If $f_i$ is small $\implies P_i(\mathbf{x})$ is large i.e. the expert thinks $\mathbf{x}$ is plausible (constraint satisfied)

# Product of Experts Formulation

$$P(\mathbf{x}) \propto \prod_i P_i(\mathbf{x}) \propto \prod_i \exp^{(-f_i(\mathbf{x}))}$$

- Every expert $f_i$ can be seen as enforcing a constraint on $\mathbf{x}$
- If $f_i$ is large $\implies$ $P_i(\mathbf{x})$ is small i.e. the expert thinks $\mathbf{x}$ is implausible (constraint violated)
- If $f_i$ is small $\implies$ $P_i(\mathbf{x})$ is large i.e. the expert thinks $\mathbf{x}$ is plausible (constraint satisfied)
- Contrast this with mixture models

# Latent Variables

- $\mathbf{x}$ is observed, let's say $\mathbf{h}$ are hidden factors that *explain* $\mathbf{x}$

# Latent Variables

- $\mathbf{x}$ is observed, let's say $\mathbf{h}$ are hidden factors that *explain* $\mathbf{x}$
- The probability then becomes:

$$P(\mathbf{x}, \mathbf{h}) = \frac{\exp^{-(\text{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

# Latent Variables

- $\mathbf{x}$ is observed, let's say $\mathbf{h}$ are hidden factors that *explain* $\mathbf{x}$
- The probability then becomes:

$$P(\mathbf{x}, \mathbf{h}) = \frac{\exp^{-(\text{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

- We only care about the marginal:

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{\exp^{-(\text{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

# Latent Variables

- $\mathbf{x}$ is observed, let's say $\mathbf{h}$ are hidden factors that *explain* $\mathbf{x}$
- The probability then becomes:

$$P(\mathbf{x}, \mathbf{h}) = \frac{\exp^{-(\text{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

- We only care about the marginal:

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{\exp^{-(\text{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

# Latent Variables

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{\exp^{-(\text{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

# Latent Variables

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{\exp^{-(\text{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

- We introduce another term in analogy from statistical physics: free energy:

$$P(\mathbf{x}) = \frac{\exp^{-(\text{FreeEnergy}(\mathbf{x}))}}{Z}$$

# Latent Variables

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{\exp^{-(\text{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

- We introduce another term in analogy from statistical physics: free energy:

$$P(\mathbf{x}) = \frac{\exp^{-(\text{FreeEnergy}(\mathbf{x}))}}{Z}$$

- Free Energy is just a marginalization of energies in the log-domain:

$$\text{FreeEnergy}(\mathbf{x}) = -\log \sum_{\mathbf{h}} \exp^{-(\text{Energy}(\mathbf{x},\mathbf{h}))}$$

# Latent Variables

$$P(\mathbf{x}) = \frac{\exp^{-(\text{FreeEnergy}(\mathbf{x}))}}{Z}$$

# Latent Variables

$$P(\mathbf{x}) = \frac{\exp^{-(\text{FreeEnergy}(\mathbf{x}))}}{Z}$$

- Likewise, the partition function:

$$Z = \sum_{\mathbf{x}} \exp^{-\text{FreeEnergy}(\mathbf{x})}$$

# Latent Variables

$$P(\mathbf{x}) = \frac{\exp^{-(\text{FreeEnergy}(\mathbf{x}))}}{Z}$$

- Likewise, the partition function:

$$Z = \sum_{\mathbf{x}} \exp^{-\text{FreeEnergy}(\mathbf{x})}$$

- We have an expression for $P(\mathbf{x})$ (and hence for the data log-likelihood). Let us see how the gradient looks like

# Data Log-Likelihood Gradient

$$P(\mathbf{x}) = \frac{\exp^{-(\text{FreeEnergy}(\mathbf{x}))}}{Z}$$

# Data Log-Likelihood Gradient

$$P(\mathbf{x}) = \frac{\exp^{-(\text{FreeEnergy}(\mathbf{x}))}}{Z}$$

- The gradient is simply working from the above:

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = -\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta} + \frac{1}{Z} \sum_{\tilde{\mathbf{x}}} \exp^{-(\text{FreeEnergy}(\tilde{\mathbf{x}}))} \frac{\partial \text{FreeEnergy}(\tilde{\mathbf{x}})}{\partial \theta}$$

# Data Log-Likelihood Gradient

$$P(\mathbf{x}) = \frac{\exp^{-(\textsf{FreeEnergy}(\mathbf{x}))}}{Z}$$

- The gradient is simply working from the above:

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = -\frac{\partial \textsf{FreeEnergy}(\mathbf{x})}{\partial \theta} + \frac{1}{Z} \sum_{\tilde{\mathbf{x}}} \exp^{-(\textsf{FreeEnergy}(\tilde{\mathbf{x}}))} \frac{\partial \textsf{FreeEnergy}(\tilde{\mathbf{x}})}{\partial \theta}$$

- Note that $P(\tilde{\mathbf{x}}) = \exp^{-(\textsf{FreeEnergy}(\tilde{\mathbf{x}}))}$

# Data Log-Likelihood Gradient

- The expected log-likelihood gradient over the training set has the following form:

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

# Data Log-Likelihood Gradient

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

- $\tilde{P}$ is the empirical training distribution

# Data Log-Likelihood Gradient

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

- $\tilde{P}$ is the empirical training distribution
- Easy to compute!

# Data Log-Likelihood Gradient

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

- $P$ is the model distribution (exponentially many configurations!)

# Data Log-Likelihood Gradient

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

- $P$ is the model distribution (exponentially many configurations!)
- Usually very hard to compute!

# Data Log-Likelihood Gradient

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \mathsf{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \mathsf{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

- $P$ is the model distribution (exponentially many configurations!)
- Usually very hard to compute!
- Resort to Markov Chain Monte Carlo to get a stochastic estimator of the gradient

# A Special Case

- Suppose the energy has the following form:

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\beta(\mathbf{x}) + \sum_i \gamma_i(\mathbf{x}, \mathbf{h}_i)$$

# A Special Case

- Suppose the energy has the following form:

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\beta(\mathbf{x}) + \sum_i \gamma_i(\mathbf{x}, \mathbf{h}_i)$$

- The free energy, and numerator of log likelihood can be computed tractably!

# A Special Case

- Suppose the energy has the following form:

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\beta(\mathbf{x}) + \sum_i \gamma_i(\mathbf{x}, \mathbf{h}_i)$$

- The free energy, and numerator of log likelihood can be computed tractably!
- What is $P(\mathbf{x})$?

# A Special Case

- Suppose the energy has the following form:

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\beta(\mathbf{x}) + \sum_i \gamma_i(\mathbf{x}, \mathbf{h}_i)$$

- The free energy, and numerator of log likelihood can be computed tractably!
- What is $P(\mathbf{x})$?
- What is the $\text{FreeEnergy}(\mathbf{x})$?

# A Special Case

- The likelihood term:

$$P(\mathbf{x}) = \frac{\exp^{\beta(\mathbf{x})}}{Z} \prod_i \sum_{\mathbf{h}_i} \exp^{-\gamma_i(\mathbf{x},\mathbf{h}_i)}$$

# A Special Case

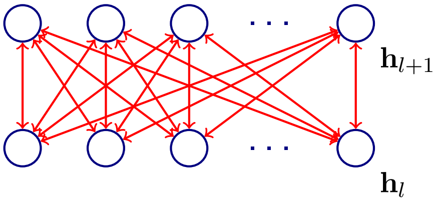
- The likelihood term:

$$P(\mathbf{x}) = \frac{\exp^{\beta(\mathbf{x})}}{Z} \prod_i \sum_{\mathbf{h}_i} \exp^{-\gamma_i(\mathbf{x},\mathbf{h}_i)}$$

- The Free Energy term:

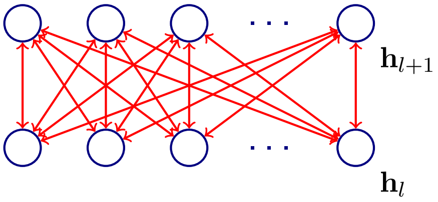
$$\begin{aligned} \text{FreeEnergy}(\mathbf{x}) &= -\log P(\mathbf{x}) - \log Z \\ &= -\beta - \sum_i \log \sum_{\mathbf{h}_i} \exp^{-\gamma_i(\mathbf{x},\mathbf{h}_i)} \end{aligned}$$

# Restricted Boltzmann Machines




- Recall the form of energy:

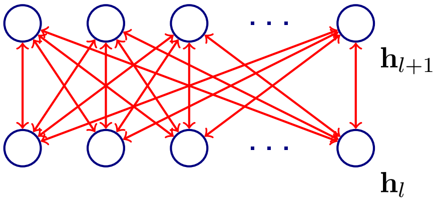
# Restricted Boltzmann Machines




- Recall the form of energy:

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\mathbf{b}^T\mathbf{x} - \mathbf{c}^T\mathbf{h} - \mathbf{h}^T W\mathbf{x}$$

# Restricted Boltzmann Machines

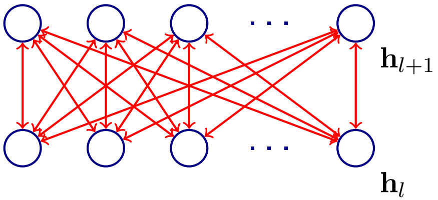



- Recall the form of energy:

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\mathbf{b}^T\mathbf{x} - \mathbf{c}^T\mathbf{h} - \mathbf{h}^T W \mathbf{x}$$

- Takes the earlier nice form with $\beta(\mathbf{x}) = \mathbf{b}^T\mathbf{x}$ and $\gamma_i(\mathbf{x}, \mathbf{h_i}) = \mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})$
- Originally proposed by Smolensky (1987) who called them *Harmoniums* as a special case of Boltzmann Machines

# Restricted Boltzmann Machines

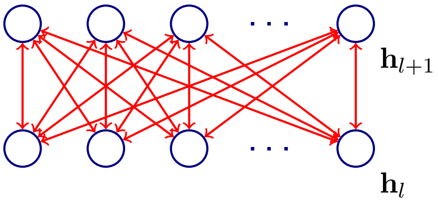



- Recall the form of energy:

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\mathbf{b}^T\mathbf{x} - \mathbf{c}^T\mathbf{h} - \mathbf{h}^T W\mathbf{x}$$

- Takes the earlier nice form with $\beta(\mathbf{x}) = \mathbf{b}^T\mathbf{x}$ and $\gamma_i(\mathbf{x}, \mathbf{h_i}) = \mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})$
- Originally proposed by Smolensky (1987) who called them *Harmoniums* as a special case of Boltzmann Machines

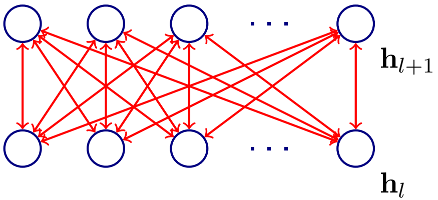
# Restricted Boltzmann Machines




- As seen before, the Free Energy can be computed efficiently:

$$\text{FreeEnergy}(\mathbf{x}) = -\mathbf{b}^T\mathbf{x} - \sum_i \log \sum_{\mathbf{h}_i} \exp^{\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})}$$

# Restricted Boltzmann Machines



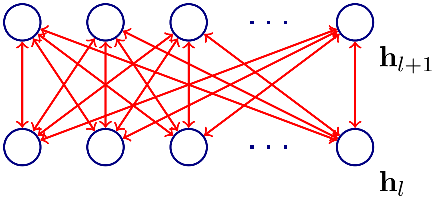

- As seen before, the Free Energy can be computed efficiently:

$$\text{FreeEnergy}(\mathbf{x}) = -\mathbf{b}^T\mathbf{x} - \sum_i \log \sum_{\mathbf{h}_i} \exp^{\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})}$$

- The conditional probability:

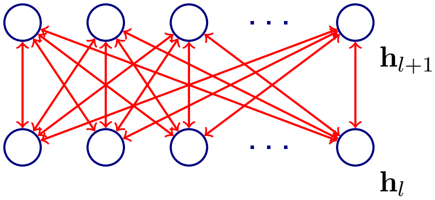
$$P(\mathbf{h}|\mathbf{x}) = \frac{\exp\left(\mathbf{b}^T\mathbf{x} + \mathbf{c}^T\mathbf{h} + \mathbf{h}^T W\mathbf{x}\right)}{\sum_{\tilde{\mathbf{h}}} \exp\left(\mathbf{b}^T\mathbf{x} + \mathbf{c}^T\tilde{\mathbf{h}} + \tilde{\mathbf{h}}^T W\mathbf{x}\right)} = \prod_i P(\mathbf{h}_i|\mathbf{x})$$

# Restricted Boltzmann Machines




- $\mathbf{x}$ and $\mathbf{h}$ play symmetric roles:

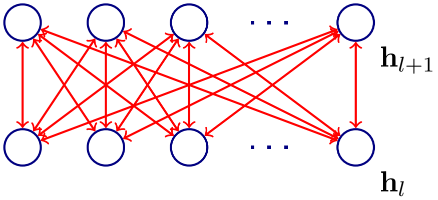
# Restricted Boltzmann Machines




- $\mathbf{x}$ and $\mathbf{h}$ play symmetric roles:

$$P(\mathbf{x}|\mathbf{h}) = \prod_i P(\mathbf{x}_i|\mathbf{h})$$

# Restricted Boltzmann Machines

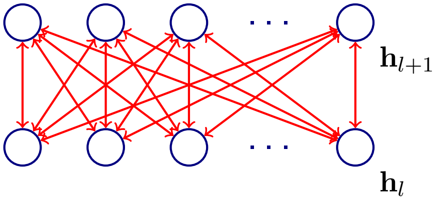



- $\mathbf{x}$ and $\mathbf{h}$ play symmetric roles:

$$P(\mathbf{x}|\mathbf{h}) = \prod_i P(\mathbf{x}_i|\mathbf{h})$$

- The common transfer (for the binary case):

# Restricted Boltzmann Machines



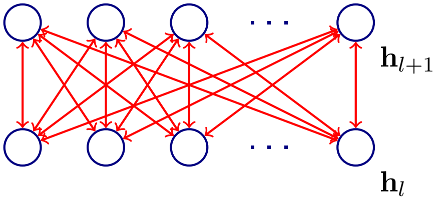

- $\mathbf{x}$ and $\mathbf{h}$ play symmetric roles:

$$P(\mathbf{x}|\mathbf{h}) = \prod_i P(\mathbf{x}_i|\mathbf{h})$$

- The common transfer (for the binary case):

$$P(\mathbf{h}_i = 1|\mathbf{x}) = \sigma(\mathbf{c_i} + W_i\mathbf{x})$$

# Restricted Boltzmann Machines




- $\mathbf{x}$ and $\mathbf{h}$ play symmetric roles:

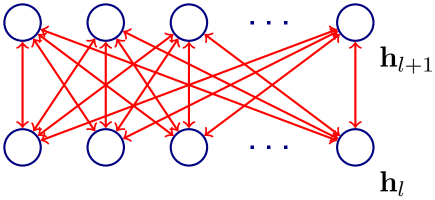
$$P(\mathbf{x}|\mathbf{h}) = \prod_i P(\mathbf{x}_i|\mathbf{h})$$

- The common transfer (for the binary case):

$$P(\mathbf{h}_i = 1|\mathbf{x}) = \sigma(\mathbf{c_i} + W_i\mathbf{x})$$

$$P(\mathbf{x}_j = 1|\mathbf{h}) = \sigma(\mathbf{b_j} + W_{:,j}^T\mathbf{h})$$

# Restricted Boltzmann Machines




- $\mathbf{x}$ and $\mathbf{h}$ play symmetric roles:

$$P(\mathbf{x}|\mathbf{h}) = \prod_i P(\mathbf{x}_i|\mathbf{h})$$

- The common transfer (for the binary case):

$$P(\mathbf{h}_i = 1|\mathbf{x}) = \sigma(\mathbf{c_i} + W_i\mathbf{x})$$

$$P(\mathbf{x}_j = 1|\mathbf{h}) = \sigma(\mathbf{b_j} + W_{:,j}^T\mathbf{h})$$

# Approximate Learning and Gibbs Sampling

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

- We saw the expression for Free Energy for a RBM. But the second term was intractable. How do learn in this case?

# Approximate Learning and Gibbs Sampling

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

- We saw the expression for Free Energy for a RBM. But the second term was intractable. How do learn in this case?
- Replace the average over all possible input configurations by samples

# Approximate Learning and Gibbs Sampling

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

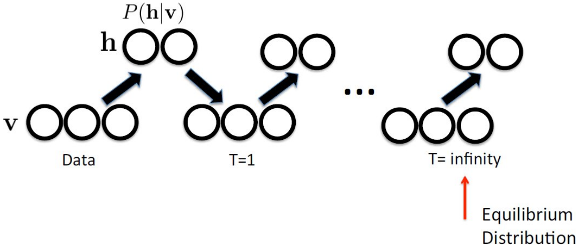
- We saw the expression for Free Energy for a RBM. But the second term was intractable. How do learn in this case?
- Replace the average over all possible input configurations by samples
- Run Markov Chain Monte Carlo (Gibbs Sampling):

# Approximate Learning and Gibbs Sampling

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

- We saw the expression for Free Energy for a RBM. But the second term was intractable. How do learn in this case?
- Replace the average over all possible input configurations by samples
- Run Markov Chain Monte Carlo (Gibbs Sampling):
- First sample $\mathbf{x}_1 \sim \tilde{P}(\mathbf{x})$, then $\mathbf{h}_1 \sim P(\mathbf{h}|\mathbf{x}_1)$, then $\mathbf{x}_2 \sim P(\mathbf{x}|\mathbf{h}_1)$, then $\mathbf{h}_2 \sim P(\mathbf{h}|\mathbf{x}_2)$ till $\mathbf{x}_{k+1}$

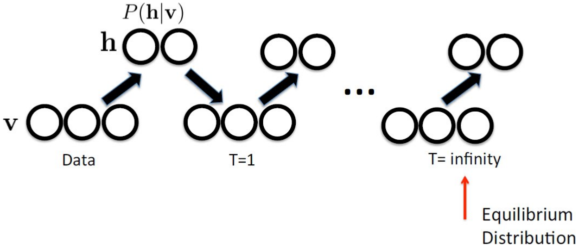
# Approximate Learning, Alternating Gibbs Sampling




- We have already seen: $P(\mathbf{x}|\mathbf{h}) = \prod_i P(\mathbf{x}_i|\mathbf{h})$ and

$$P(\mathbf{h}|\mathbf{x}) = \prod_i P(\mathbf{h}_i|\mathbf{x})$$

# Approximate Learning, Alternating Gibbs Sampling




- We have already seen: $P(\mathbf{x}|\mathbf{h}) = \prod_i P(\mathbf{x}_i|\mathbf{h})$ and

  $$P(\mathbf{h}|\mathbf{x}) = \prod_i P(\mathbf{h}_i|\mathbf{x})$$

- With: $P(\mathbf{h}_i = 1|\mathbf{x}) = \sigma(\mathbf{c_i} + W_i\mathbf{x})$ and
  $P(\mathbf{x}_j = 1|\mathbf{h}) = \sigma(\mathbf{b_j} + W_{:,j}^T\mathbf{h})$

# Training a RBM: The Contrastive Divergence Algorithm

- Start with a training example on the visible units

# Training a RBM: The Contrastive Divergence Algorithm

- Start with a training example on the visible units
- Update all the hidden units in parallel

# Training a RBM: The Contrastive Divergence Algorithm

- Start with a training example on the visible units
- Update all the hidden units in parallel
- Update all the visible units in parallel to obtain a reconstruction

# Training a RBM: The Contrastive Divergence Algorithm

- Start with a training example on the visible units
- Update all the hidden units in parallel
- Update all the visible units in parallel to obtain a reconstruction
- Update all the hidden units again

# Training a RBM: The Contrastive Divergence Algorithm

- Start with a training example on the visible units
- Update all the hidden units in parallel
- Update all the visible units in parallel to obtain a reconstruction
- Update all the hidden units again
- Update model parameters

# Training a RBM: The Contrastive Divergence Algorithm

- Start with a training example on the visible units
- Update all the hidden units in parallel
- Update all the visible units in parallel to obtain a reconstruction
- Update all the hidden units again
- Update model parameters
- Aside: Easy to extend RBM (and contrastive divergence) to the continuous case

# Boltzmann Machines

- A model in which the energy has the form:

# Boltzmann Machines

- A model in which the energy has the form:

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\mathbf{b}^T\mathbf{x} - \mathbf{c}^T\mathbf{h} - \mathbf{h}^T W\mathbf{x} - \mathbf{x}^T U\mathbf{x} - \mathbf{h}^T V\mathbf{h}$$

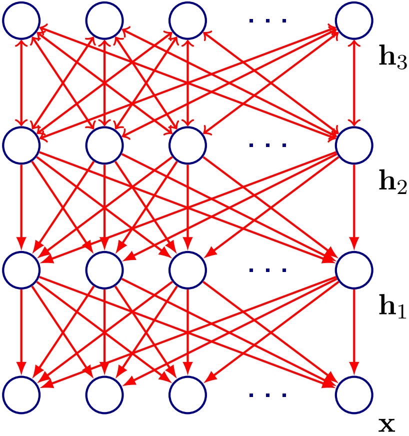
# Boltzmann Machines

- A model in which the energy has the form:

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\mathbf{b}^T\mathbf{x} - \mathbf{c}^T\mathbf{h} - \mathbf{h}^T W\mathbf{x} - \mathbf{x}^T U\mathbf{x} - \mathbf{h}^T V\mathbf{h}$$

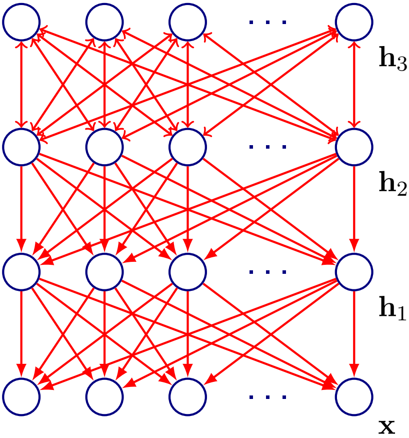
- Originally proposed by Hinton and Sejnowski (1983)
- Important historically. But very difficult to train (why?)

# Back to Deep Belief Networks




$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = P(\mathbf{h}^l, \mathbf{h}^{l-1}) \left( \prod_{k=1}^{l-2} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \right) P(\mathbf{x} | \mathbf{h}^1)$$

# Back to Deep Belief Networks




$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = P(\mathbf{h}^l, \mathbf{h}^{l-1}) \Big( \prod_{k=1}^{l-2} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \Big) P(\mathbf{x} | \mathbf{h}^1)$$

# Greedy Layer-wise Training of DBNs

- Reference: G. E. Hinton, S. Osindero and Y-W Teh: A Fast Learning Algorithm for Deep Belief Networks, In **Neural Computation**, 2006.

# Greedy Layer-wise Training of DBNs

- Reference: G. E. Hinton, S. Osindero and Y-W Teh: A Fast Learning Algorithm for Deep Belief Networks, In **Neural Computation**, 2006.
- First Step: Construct a RBM with input $\mathbf{x}$ and a hidden layer $\mathbf{h}$, train the RBM

# Greedy Layer-wise Training of DBNs

- Reference: G. E. Hinton, S. Osindero and Y-W Teh: A Fast Learning Algorithm for Deep Belief Networks, In **Neural Computation**, 2006.
- First Step: Construct a RBM with input $\mathbf{x}$ and a hidden layer $\mathbf{h}$, train the RBM
- Stack another layer on top of the RBM to form a new RBM. Fix $W^1$, sample from $P(\mathbf{h}^1|\mathbf{x})$, train $W^2$ as RBM

# Greedy Layer-wise Training of DBNs

- Reference: G. E. Hinton, S. Osindero and Y-W Teh: A Fast Learning Algorithm for Deep Belief Networks, In **Neural Computation**, 2006.
- First Step: Construct a RBM with input $\mathbf{x}$ and a hidden layer $\mathbf{h}$, train the RBM
- Stack another layer on top of the RBM to form a new RBM. Fix $W^1$, sample from $P(\mathbf{h}^1|\mathbf{x})$, train $W^2$ as RBM
- Continue till $k$ layers

# Greedy Layer-wise Training of DBNs

- Reference: G. E. Hinton, S. Osindero and Y-W Teh: A Fast Learning Algorithm for Deep Belief Networks, In **Neural Computation**, 2006.
- First Step: Construct a RBM with input $\mathbf{x}$ and a hidden layer $\mathbf{h}$, train the RBM
- Stack another layer on top of the RBM to form a new RBM. Fix $W^1$, sample from $P(\mathbf{h}^1|\mathbf{x})$, train $W^2$ as RBM
- Continue till $k$ layers
- Implicitly defines $P(\mathbf{x})$ and $P(\mathbf{h})$ (variational bound justifies layerwise training)

# Greedy Layer-wise Training of DBNs

- Reference: G. E. Hinton, S. Osindero and Y-W Teh: A Fast Learning Algorithm for Deep Belief Networks, In **Neural Computation**, 2006.
- First Step: Construct a RBM with input $\mathbf{x}$ and a hidden layer $\mathbf{h}$, train the RBM
- Stack another layer on top of the RBM to form a new RBM. Fix $W^1$, sample from $P(\mathbf{h}^1|\mathbf{x})$, train $W^2$ as RBM
- Continue till $k$ layers
- Implicitly defines $P(\mathbf{x})$ and $P(\mathbf{h})$ (variational bound justifies layerwise training)
- Can then be discriminatively fine-tuned using backpropagation

# Deep Autoencoders (2006)

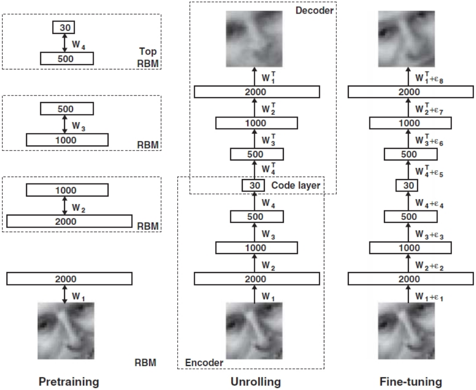
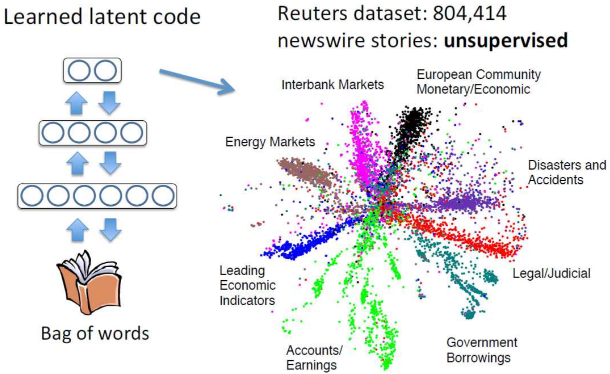



**Fig. 1.** Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

G. E. Hinton, R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science, 2006

From last time: Was hard to train deep networks from scratch in 2006!

# Semantic Hashing




G. Hinton and R. Salakhutdinov, "Semantic Hashing", 2006

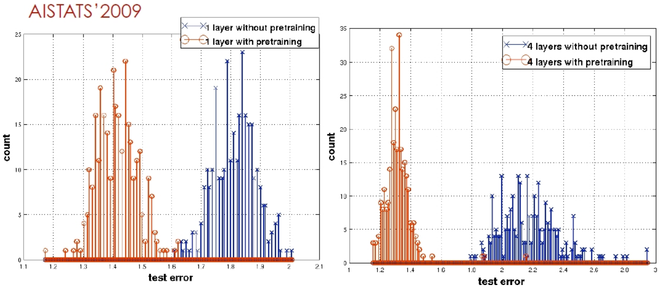
# Why does Unsupervised Pre-training work?

- Regularization. Feature representations that are good for $P(x)$ are good for $P(y|x)$

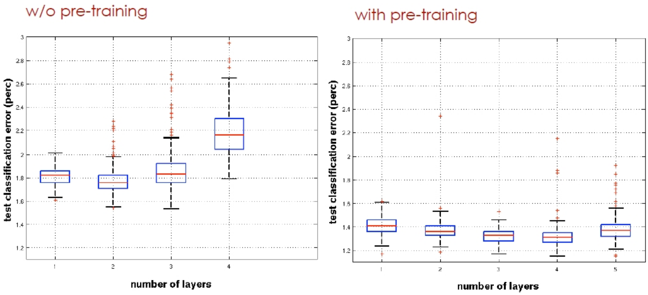
# Why does Unsupervised Pre-training work?

- Regularization. Feature representations that are good for $P(x)$ are good for $P(y|x)$
- Optimization: Unsupervised pre-training leads to better regions of the space i.e. better than random initialization

# Effect of Unsupervised Pre-training



# Effect of Unsupervised Pre-training



- Important topics we didn't talk about in detail/at all:
  - Joint unsupervised training of all layers (Wake-Sleep algorithm)
  - Deep Boltzmann Machines
  - Variational bounds justifying greedy layerwise training
  - Conditional RBMs, Multimodal RBMs, Temporal RBMs etc

# Next Time

- Some Applications of methods we just considered

# Next Time

- Some Applications of methods we just considered
- Generative Adversarial Networks