Lecture 17
Deep Neural Generative Models II

CMSC 35246: Deep Learning

Shubhendu Trivedi
&
Risi Kondor

University of Chicago

May 24, 2017

Recap: Linear Factor Models

- Sample latent factors $\mathbf{h} \sim P(\mathbf{h})$

Recap: Linear Factor Models

- Sample latent factors $\mathbf{h} \sim P(\mathbf{h})$
- Generate $\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$

Recap: Linear Factor Models

- Sample latent factors $\mathbf{h} \sim P(\mathbf{h})$
- Generate $\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$
- In Probabilistic PCA:
  - Latent Factors: $\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$

Recap: Linear Factor Models

- Sample latent factors $\mathbf{h} \sim P(\mathbf{h})$
- Generate $\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$
- In Probabilistic PCA:
  - Latent Factors: $\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$
  - Noise Model: $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$

Recap: Linear Factor Models

- Sample latent factors $\mathbf{h} \sim P(\mathbf{h})$
- Generate $\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$
- In Probabilistic PCA:
  - Latent Factors: $\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$
  - Noise Model: $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$
- Estimate $W, \mathbf{b}, \sigma^2$ by maximum likelihood estimation or Expectation Maximization (EM)

Recap: Linear Factor Models

- Sample latent factors $\mathbf{h} \sim P(\mathbf{h})$

Recap: Linear Factor Models

- Sample latent factors $\mathbf{h} \sim P(\mathbf{h})$
- Generate $\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$

Recap: Linear Factor Models

- Sample latent factors $\mathbf{h} \sim P(\mathbf{h})$
- Generate $\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$
- In Factor Analysis:
    - Latent Factors: $\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$

Recap: Linear Factor Models

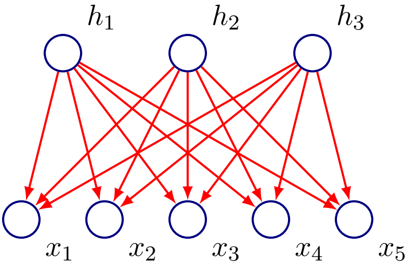
- Sample latent factors $\mathbf{h} \sim P(\mathbf{h})$
- Generate $\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$
- In Factor Analysis:
    - Latent Factors: $\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$
    - Noise Model: $\epsilon \sim \mathcal{N}(0, diag([\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2]))$

Recap: Linear Factor Models

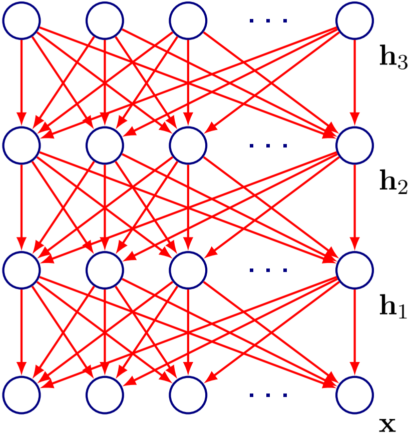
- Sample latent factors $\mathbf{h} \sim P(\mathbf{h})$
- Generate $\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$
- In Factor Analysis:
  - Latent Factors: $\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$
  - Noise Model: $\epsilon \sim \mathcal{N}(0, diag([\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2])$
- Estimate $W, \mathbf{b}, diag([\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2]$ by Expectation Maximization

Recap: Linear Factor Models

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

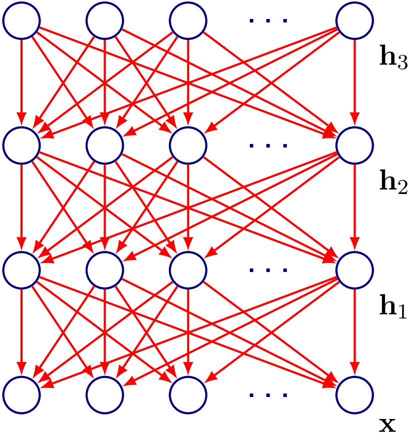


Recap: Sigmoid Belief Networks



- Just like a feedfoward network, but with arrows reversed

Recap: Sigmoid Belief Networks



- Just like a feedfoward network, but with arrows reversed
- What if we place a class as a latent variable at the top?

Recap: Sigmoid Belief Networks

- Joint probability factorizes as:

Recap: Sigmoid Belief Networks

- Joint probability factorizes as:

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

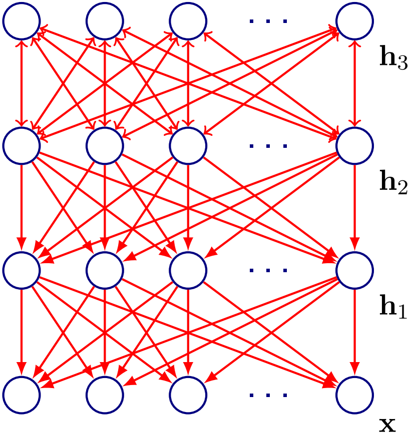
Recap: Sigmoid Belief Networks

- Joint probability factorizes as:

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

- Marginalization yields $P(\mathbf{x})$

Recap: Deep Belief Networks



- The top two layers are a Restricted Boltzmann Machine

Recap: Deep Belief Networks

- The joint probability factorizes as:

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

Recap: Energy Based Models

- We defined a probability distribution through an energy:

Recap: Energy Based Models

- We defined a probability distribution through an energy:

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

Recap: Energy Based Models

- We defined a probability distribution through an energy:

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

- Z is a normalizing factor called the Partition Function

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

Recap: Energy Based Models

- One formulation of the energy:

Recap: Energy Based Models

- One formulation of the energy:

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

Recap: Energy Based Models

- One formulation of the energy:

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

- This gave us the Product of Experts Model (PoE):

Recap: Energy Based Models

- One formulation of the energy:

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

- This gave us the Product of Experts Model (PoE):

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

Recap: Energy Based Models

- $\mathbf{x}$ is observed, $\mathbf{h}$ represents hidden factors

Recap: Energy Based Models

- $\mathbf{x}$ is observed, $\mathbf{h}$ represents hidden factors
- Joint probability:

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

Recap: Energy Based Models

- $\mathbf{x}$ is observed, $\mathbf{h}$ represents hidden factors
- Joint probability:

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

- We only care about the marginal:

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

Recap: Energy Based Models

- $\mathbf{x}$ is observed, $\mathbf{h}$ represents hidden factors
- Joint probability:

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

- We only care about the marginal:

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

- We can write the marginal in terms of free energy:

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

Recap: Energy Based Models

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

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

Recap: Energy Based Models

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

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

Recap: Energy Based Models

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

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

Recap: Energy Based Models

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

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

Recap: Energy Based Models

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

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

End of recap

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:

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

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

A Special Case

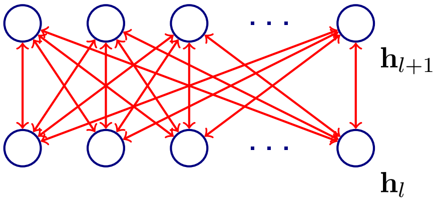
- Suppose the energy has the following form:

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

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

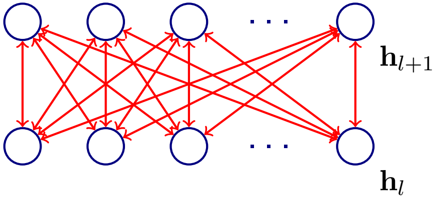
$$\begin{aligned} \text{FreeEnergy}(\mathbf{x}) &= -\log P(\mathbf{x}) - \log Z \\ &= -\beta - \sum_i \log \sum_{\mathbf{h}_i} \exp^{-\gamma_i(\mathbf{x}, \mathbf{h}_i)} \end{aligned}$$

Restricted Boltzmann Machines



- Form of energy:

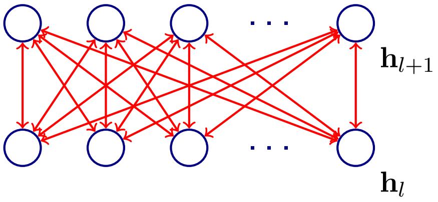
Restricted Boltzmann Machines



- 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

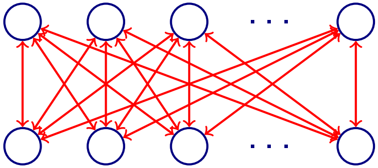


- Form of energy:

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

- 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



- 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

- With $\beta(\mathbf{x}) = \mathbf{b}^T\mathbf{x}$ and $\gamma_i(\mathbf{x}, \mathbf{h_i}) = \mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})$:

$$P(\mathbf{x}) = \frac{\exp^{\mathbf{b}^T\mathbf{x}}}{Z} \prod_i \sum_{\mathbf{h}_i} \exp^{\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})}$$

Restricted Boltzmann Machines

- With $\beta(\mathbf{x}) = \mathbf{b}^T\mathbf{x}$ and $\gamma_i(\mathbf{x}, \mathbf{h_i}) = \mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})$:

$$P(\mathbf{x}) = \frac{\exp^{\mathbf{b}^T\mathbf{x}}}{Z} \prod_i \sum_{\mathbf{h}_i} \exp^{\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})}$$

- Likewise, plugging in, we have:

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

Restricted Boltzmann Machines

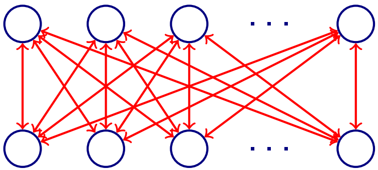
- With $\beta(\mathbf{x}) = \mathbf{b}^T\mathbf{x}$ and $\gamma_i(\mathbf{x}, \mathbf{h_i}) = \mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})$:

$$P(\mathbf{x}) = \frac{\exp^{\mathbf{b}^T\mathbf{x}}}{Z} \prod_i \sum_{\mathbf{h}_i} \exp^{\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})}$$

- Likewise, plugging in, we have:

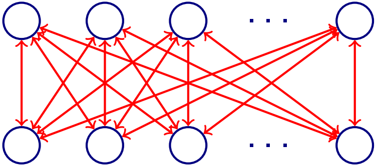
$$\text{FreeEnergy}(\mathbf{x}) = -\mathbf{b}^T\mathbf{x} - \sum_i \log \sum_{\mathbf{h}_i} \exp^{\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})}$$

Restricted Boltzmann Machines



- We have an expression for $P(\mathbf{x})$ and the Free Energy can be computed analytically

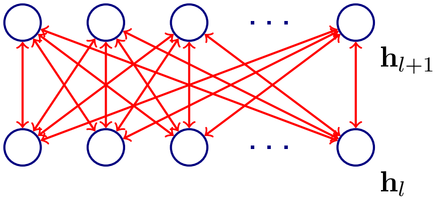
Restricted Boltzmann Machines



- We have an expression for $P(\mathbf{x})$ and the Free Energy can be computed analytically
- The conditional probability:

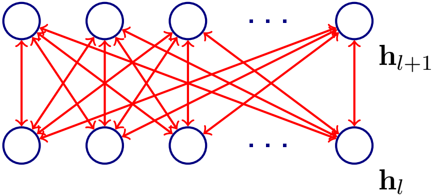
$$P(\mathbf{h}|\mathbf{x}) = \frac{\exp\left(\mathbf{b}^T\mathbf{x} + \mathbf{c}^T\mathbf{h} + \mathbf{h}^T W \mathbf{x}\right)}{\sum_{\tilde{\mathbf{h}}} \exp\left(\mathbf{b}^T\mathbf{x} + \mathbf{c}^T\tilde{\mathbf{h}} + \tilde{\mathbf{h}}^T W \mathbf{x}\right)} = \prod_i P(\mathbf{h}_i|\mathbf{x})$$

Restricted Boltzmann Machines



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

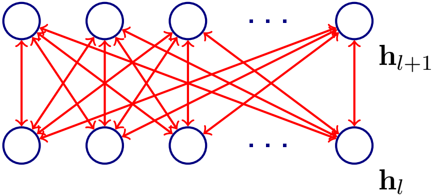
Restricted Boltzmann Machines



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

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

Restricted Boltzmann Machines

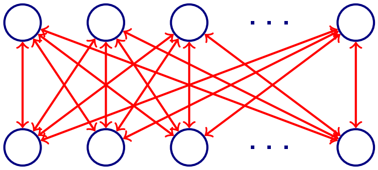


- $\mathbf{x}$ and $\mathbf{h}$ play symmetric roles:

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

- The common transfer (for the binary case):

Restricted Boltzmann Machines



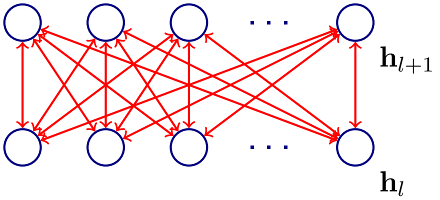
- $\mathbf{x}$ and $\mathbf{h}$ play symmetric roles:

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

- 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:

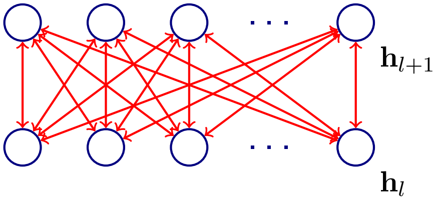
$$P(\mathbf{x}|\mathbf{h}) = \prod_i P(\mathbf{x}_i|\mathbf{h})$$

- 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

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

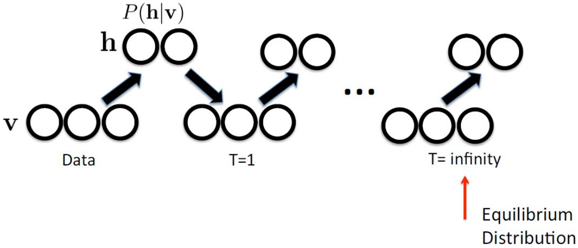
- We saw the expression for Free Energy for a RBM. But the second term was intractable. How do learn in this case?
- Replace the average over all possible input configurations by samples
- Run Markov Chain Monte Carlo (Gibbs Sampling):

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):
- We want $\tilde{P}(\mathbf{x}) \approx P(\mathbf{x})$
- 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 $\mathbf{x}_{k+1}$

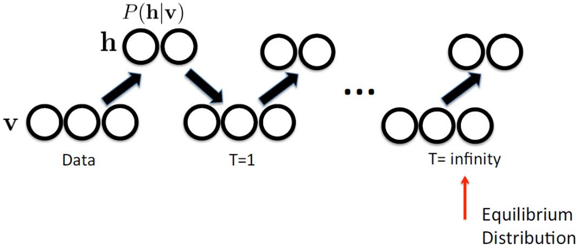
Approximate Learning, Alternating Gibbs Sampling



- 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?)

Gradient of Log-Likelihood Revisited

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = \frac{\partial \log \sum_{\mathbf{h}} \exp^{-\text{Energy}(\mathbf{x},\mathbf{h})}}{\partial \theta} - \frac{\partial \log \sum_{\tilde{\mathbf{x}},\mathbf{h}} \exp^{-\text{Energy}(\tilde{\mathbf{x}},\mathbf{h})}}{\partial \theta}$$

After basic manipulations and substitution:

Gradient of Log-Likelihood Revisited

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = \frac{\partial \log \sum_{\mathbf{h}} \exp^{-\text{Energy}(\mathbf{x},\mathbf{h})}}{\partial \theta} - \frac{\partial \log \sum_{\tilde{\mathbf{x}},\mathbf{h}} \exp^{-\text{Energy}(\tilde{\mathbf{x}},\mathbf{h})}}{\partial \theta}$$

After basic manipulations and substitution:

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{x}) \frac{\partial \text{Energy}(\mathbf{x},\mathbf{h})}{\partial \theta} + \sum_{\tilde{\mathbf{x}},\mathbf{h}} P(\tilde{\mathbf{x}},\mathbf{h}) \frac{\partial \text{Energy}(\tilde{\mathbf{x}},\mathbf{h})}{\partial \theta}$$

Gradient of Log-Likelihood Revisited

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{x}) \frac{\partial \text{Energy}(\mathbf{x}, \mathbf{h})}{\partial \theta} + \sum_{\tilde{\mathbf{x}}, \mathbf{h}} P(\tilde{\mathbf{x}}, \mathbf{h}) \frac{\partial \text{Energy}(\tilde{\mathbf{x}}, \mathbf{h})}{\partial \theta}$$

Gradient of Log-Likelihood Revisited

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{x}) \frac{\partial \text{Energy}(\mathbf{x}, \mathbf{h})}{\partial \theta} + \sum_{\tilde{\mathbf{x}}, \mathbf{h}} P(\tilde{\mathbf{x}}, \mathbf{h}) \frac{\partial \text{Energy}(\tilde{\mathbf{x}}, \mathbf{h})}{\partial \theta}$$

- Note that $\frac{\partial \text{Energy}(\mathbf{x}, \mathbf{h})}{\partial \theta}$ is easy to compute

Gradient of Log-Likelihood Revisited

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{x}) \frac{\partial \text{Energy}(\mathbf{x}, \mathbf{h})}{\partial \theta} + \sum_{\tilde{\mathbf{x}}, \mathbf{h}} P(\tilde{\mathbf{x}}, \mathbf{h}) \frac{\partial \text{Energy}(\tilde{\mathbf{x}}, \mathbf{h})}{\partial \theta}$$

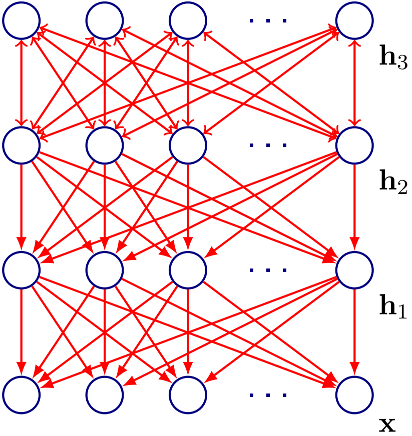
- Note that $\frac{\partial \text{Energy}(\mathbf{x}, \mathbf{h})}{\partial \theta}$ is easy to compute
- If we have a procedure to sample from $P(\mathbf{h}|\mathbf{x})$ and from $P(\tilde{\mathbf{x}}, \mathbf{h})$ we get an unbiased stochastic estimator of the gradient

Gradient of Log-Likelihood Revisited

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{x}) \frac{\partial \text{Energy}(\mathbf{x}, \mathbf{h})}{\partial \theta} + \sum_{\tilde{\mathbf{x}}, \mathbf{h}} P(\tilde{\mathbf{x}}, \mathbf{h}) \frac{\partial \text{Energy}(\tilde{\mathbf{x}}, \mathbf{h})}{\partial \theta}$$

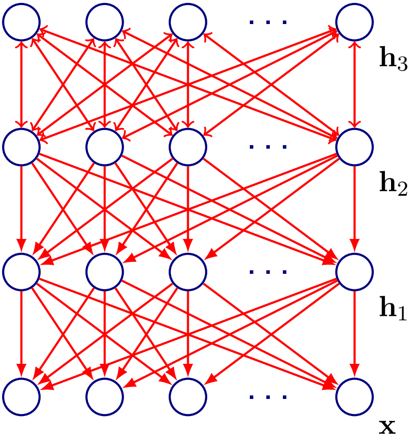
- Note that $\frac{\partial \text{Energy}(\mathbf{x}, \mathbf{h})}{\partial \theta}$ is easy to compute
- If we have a procedure to sample from $P(\mathbf{h}|\mathbf{x})$ and from $P(\tilde{\mathbf{x}}, \mathbf{h})$ we get an unbiased stochastic estimator of the gradient

Back to Deep Belief Networks



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

Back to Deep Belief Networks



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

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)

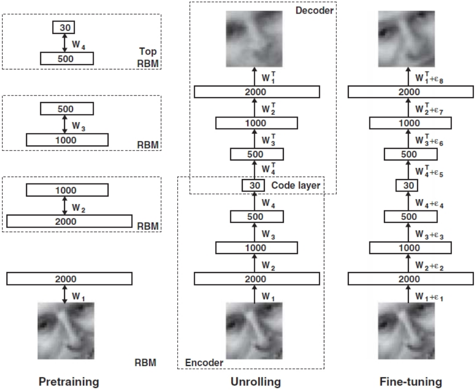
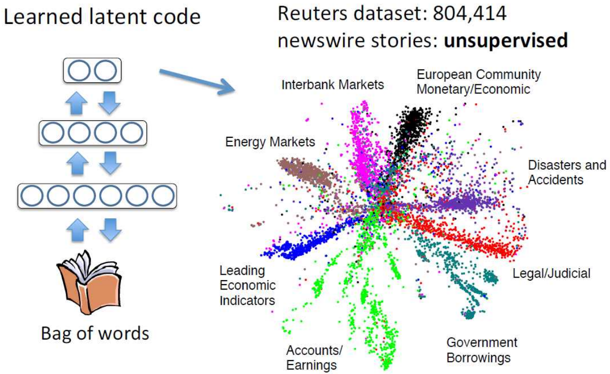


Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create 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

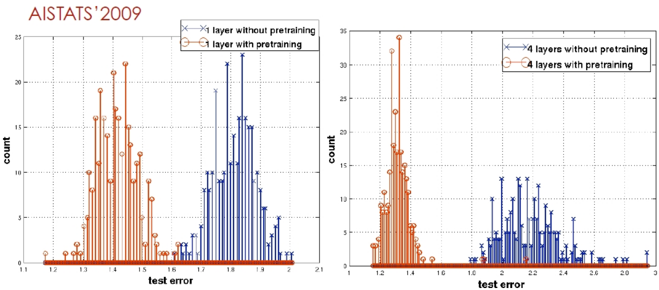
Why does Unsupervised Pre-training work?

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

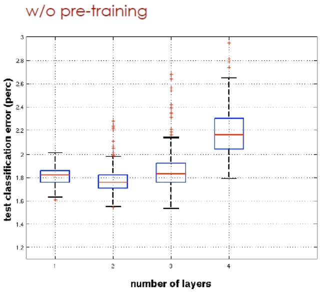
Why does Unsupervised Pre-training work?

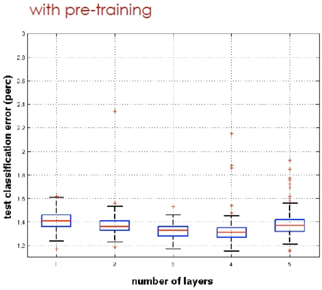
- Regularization. Feature representations that are good for $P(x)$ are good for $P(y|x)$
- Optimization: Unsupervised pre-training leads to better regions of the space i.e. better than random initialization

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

Generative Adversarial Networks

Representations

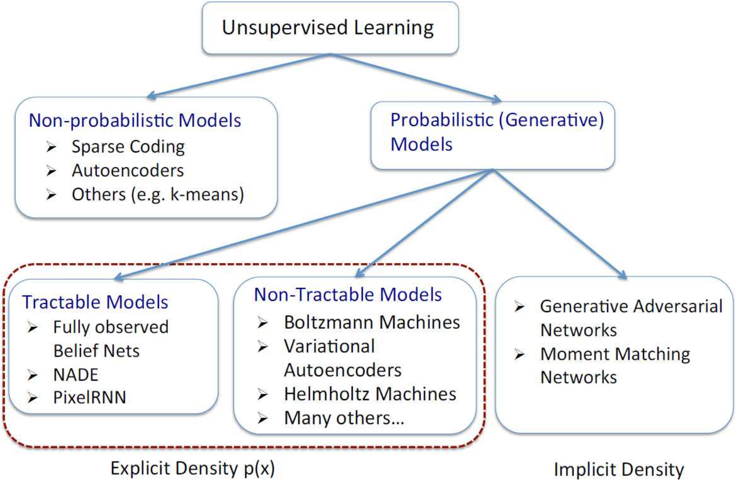


Figure: Ruslan Salakhutdinov

Motivation

- We don't want to write down the formula for $P(X)$

Motivation

- We don't want to write down the formula for $P(X)$
- Thus want to avoid variational learning, ML estimation, MCMC etc

Motivation

- We don't want to write down the formula for $P(X)$
- Thus want to avoid variational learning, ML estimation, MCMC etc
- By playing an adversarial game!

Goal

- Assume we have training samples $\mathcal{D} = \{X | X \sim P_{\text{data}}, X \in \mathcal{X}\}$

Goal

- Assume we have training samples $\mathcal{D} = \{X | X \sim P_{\text{data}}, X \in \mathcal{X}\}$
- We want a generative model P_{model} from which we can draw new samples $X \sim P_{\text{model}}$

Goal

- Assume we have training samples $\mathcal{D} = \{X | X \sim P_{\text{data}}, X \in \mathcal{X}\}$
- We want a generative model P_{model} from which we can draw new samples $X \sim P_{\text{model}}$
- Such that $P_{\text{model}} \approx P_{\text{data}}$

$\mathbf{x} \sim p_{\text{data}}$

$\longrightarrow$

$\mathbf{x} \sim p_{\text{model}}$

Figure by Gilles Louppe

Generative Adversarial Networks (Goodfellow et al. 2014)

- Don't assume any form, instead use a neural network to produce similar samples

Generative Adversarial Networks (Goodfellow et al. 2014)

- Don't assume any form, instead use a neural network to produce similar samples
- Setup a two-player game between:

Generative Adversarial Networks (Goodfellow et al. 2014)

- Don't assume any form, instead use a neural network to produce similar samples
- Setup a two-player game between:
  - A Generator G

Generative Adversarial Networks (Goodfellow et al. 2014)

- Don't assume any form, instead use a neural network to produce similar samples
- Setup a two-player game between:
  - A Generator G
  - A Discriminator D

Generative Adversarial Networks (Goodfellow et al. 2014)

- Don't assume any form, instead use a neural network to produce similar samples
- Setup a two-player game between:
  - A Generator G
  - A Discriminator D
- The discriminator D tries to distinguish between a sample from P_{model} and a sample from G

Generative Adversarial Networks (Goodfellow et al. 2014)

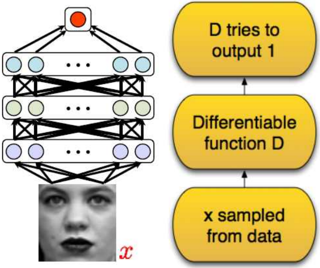
- Don't assume any form, instead use a neural network to produce similar samples
- Setup a two-player game between:
  - A Generator G
  - A Discriminator D
- The discriminator D tries to distinguish between a sample from P_{model} and a sample from G
- The generator G tries to *fool* D by producing samples that are hard to discriminate from the real data

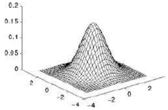
Catch me if you can

Leo is G Tom is D

Slide adapted from Gilles Louppe

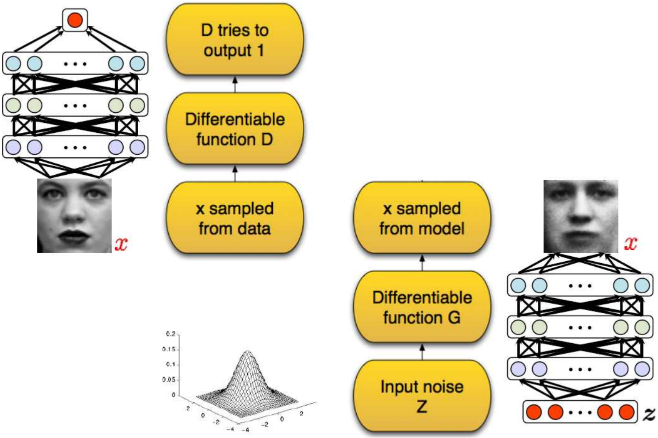
Generative Adversarial Networks





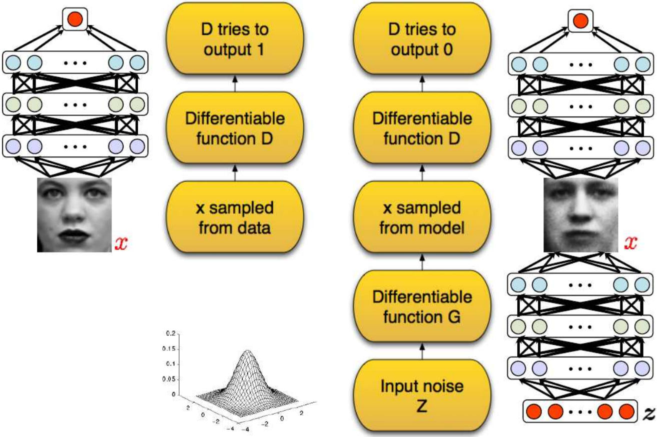
Slide adapted from Ian Goodfellow

Generative Adversarial Networks



Slide adapted from Ian Goodfellow

Generative Adversarial Networks



Slide adapted from Ian Goodfellow

Generative Adversarial Networks

- Minimax value function

Generator: generate samples that D would classify as real

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Discriminator: Pushes up

Discriminator: Classify data as being real

Discriminator: Classify generator samples as being fake

Generator: Pushes down

- Optimal strategy for Discriminator is:

$$D(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}$$

Slide adapted from Ian Goodfellow

Generative Adversarial Networks

- The value function:

$$V(D,G) = \mathbb{E}_{X\sim P_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim P_{\text{noise}}}[\log(1 - D(G(X)))]$$

Generative Adversarial Networks

- The value function:

$$V(D,G) = \mathbb{E}_{X\sim P_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim P_{\text{noise}}}[\log(1 - D(G(X)))]$$

- For training we want to:

Generative Adversarial Networks

- The value function:

$$V(D,G) = \mathbb{E}_{X\sim P_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim P_{\text{noise}}}[\log(1 - D(G(X)))]$$

- For training we want to:
  - Fix G, find D which **maximizes** $V(D,G)$

Generative Adversarial Networks

- The value function:

$$V(D,G) = \mathbb{E}_{X\sim P_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim P_{\text{noise}}}[\log(1 - D(G(X)))]$$

- For training we want to:
  - Fix G, find D which **maximizes** $V(D,G)$
  - Fix D, find G which **minimizes** $V(D,G)$

Generative Adversarial Networks

- The value function:

$$V(D,G) = \mathbb{E}_{X\sim P_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim P_{\text{noise}}}[\log(1 - D(G(X)))]$$

- For training we want to:
  - Fix G, find D which **maximizes** $V(D,G)$
  - Fix D, find G which **minimizes** $V(D,G)$
- Alternate till convergence

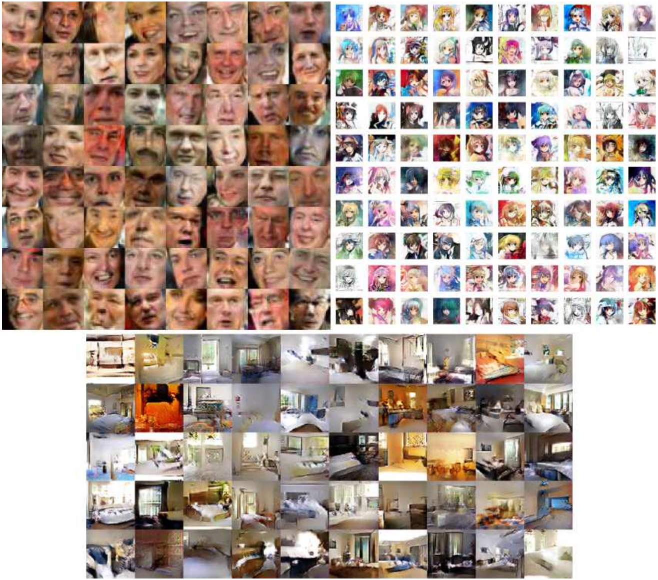
Generative Adversarial Networks

- The value function:

$$V(D,G) = \mathbb{E}_{X\sim P_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim P_{\text{noise}}}[\log(1-D(G(X)))]$$

- For training we want to:
  - Fix G, find D which **maximizes** $V(D,G)$
  - Fix D, find G which **minimizes** $V(D,G)$
- Alternate till convergence
- This is good since we can use the machinery for neural networks

Samples



Samples

- Open Question: How do you evaluate goodness of generated samples?

Next Time

- GANs wrap-up
- Quiz