Graphical Models

EE807: Recent Advances in Deep Learning

Lecture 5

Slide made by

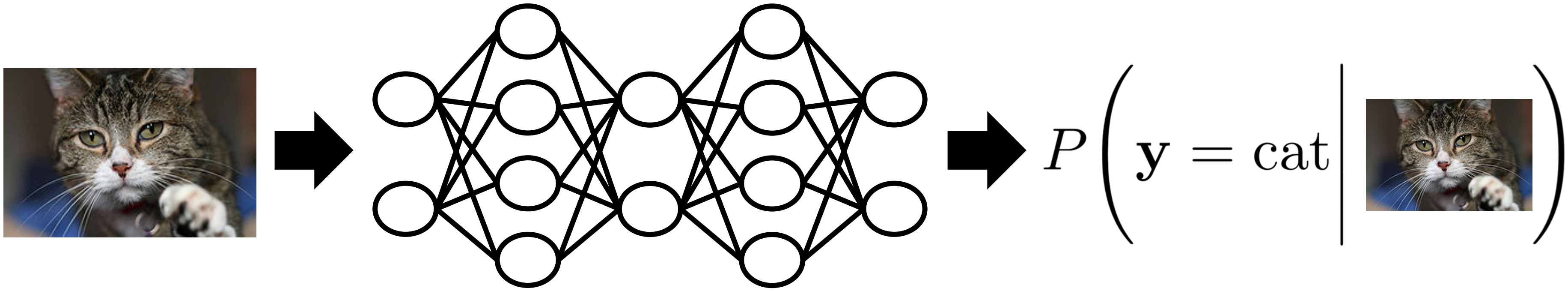
Sejun Park and Sungsoo Ahn

KAIST EE

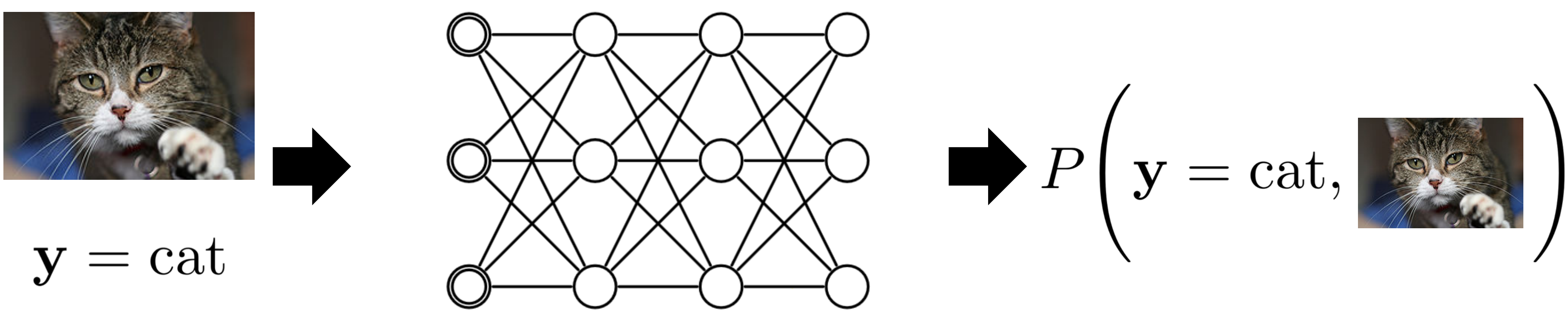
Table of Contents

Generative Model and Discriminative Model

- Given an observed variable $\mathbf{x}$ and a target variable $\mathbf{y}$
- **Discriminative model** is a model of a conditional distribution $P(\mathbf{y}|\mathbf{x})$
  - e.g., neural networks (supervised)



- **Generative model** is a model of a joint distribution $P(\mathbf{x}, \mathbf{y})$ (or $P(\mathbf{x})$)
  - e.g., Boltzmann machines, sum-product networks (unsupervised)

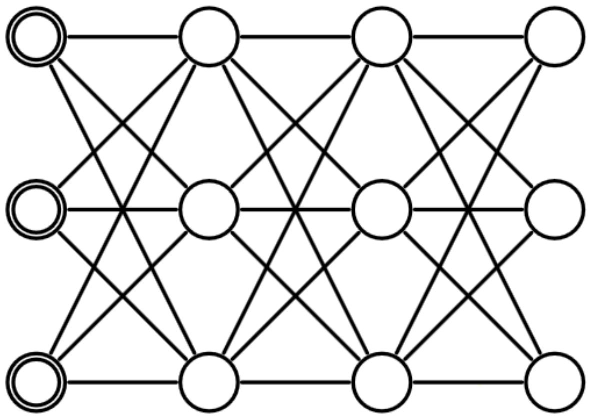


*reference : https://en.wikipedia.org/wiki/Cat#/media/File:Kittyply_edit1.jpg

Why Generative Model?

- Generative models model a full probability distribution given data
- $P(\mathbf{x}, \mathbf{y})$ enables us to generate new data similar to existing (training) data
  - This is impossible under discriminative models
- **Sampling methods** (e.g., Markov chain) are required for generation

$$P(\mathbf{x})$$



$\sim P(\mathbf{x})$

$\sim P(\mathbf{x})$

$\sim P(\mathbf{x})$

$\sim P(\mathbf{x})$

*reference : https://en.wikipedia.org/wiki/File:Cat_poster_1.jpg

Examples of Generative Models

- Modelling a joint distribution of $\mathbf{x}$
  - **Mean Fields**
    - $P(\mathbf{x}) = \prod_i P(x_i)$
    - Tractable inference, low expressive power
  - **Multivariate Gaussian distributions**
    - $P(\mathbf{x}) \propto \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)\Sigma^{-1}(\mathbf{x}-\mu)\right)$
    - Tractable inference, low expressive power
  - **Graphical models** (e.g., RBM, DBM, etc.)
    - $P(\mathbf{x}) \propto \exp\left(\sum_i b_i x_i + \sum_{i,j} w_{ij} x_i x_j\right)$
    - Intractable inference, high expressive power with compact representation
  - **Generative adversarial networks**
    - $P(\mathbf{x}) \propto |f^{-1}(\mathbf{x})|$ for some neural network
    - Intractable inference, high expressive power with complex expression

Table of Contents

Energy Based Model

- **Energy based model** (EBM) is a joint distribution on a vector $\mathbf{x}$ satisfying

$$P(\mathbf{x}) \propto \exp\left(-E(\mathbf{x})\right)$$

Energy

- Assignments with high energy appear less likely in EBM

- Examples of EBM
  - Gaussian distribution

$$P(\mathbf{x}) \propto \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

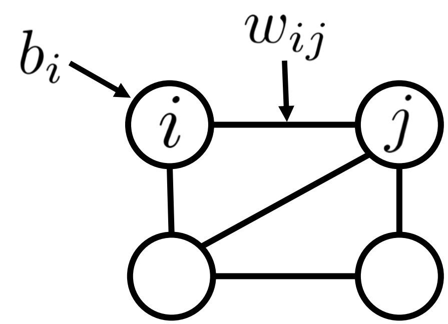
  - Bernoulli distribution

$$P(x) = \exp\left(\log(P(x=1))\mathbf{1}_{x=1} + \log(P(x=0))\mathbf{1}_{x=0}\right)$$

  - Poisson, binomial, ….

Boltzmann Machine

- Given a graph (V, E), **Boltzmann machine** (BM) is a joint distribution on a binary vector $\mathbf{x} = [x_i] \in \{0,1\}^{|V|}$ such that



$$P(\mathbf{x}) \propto \exp\left(\sum_{i \in V} b_i x_i + \sum_{(i,j) \in E} w_{ij} x_i x_j\right)$$

- Given a neighborhood of i, conditional distribution of x_i is
  - $\sigma(x) = 1/(1+\exp(-x))$: logistic sigmoid function

$$P(x_i = 1 | x_{\mathcal{N}(i)}) = \sigma\left(b_i + \sum_{j \in \mathcal{N}(i)} w_{ij} x_j\right)$$

where $\mathcal{N}(i) = \{j : (i,j) \in E\}$ is a set of neighbors of i

- To generate new data using BM, we need to learn parameters of BM

Learning BM

- Given training data $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$, learn the distribution $P(\mathbf{x})$
- **Goal (Maximum Likelihood Estimation)**:

Maximize $\ell(\mathbf{b}, \mathbf{w}) = \sum_{n=1}^{N} \log P_{\text{model}}(\mathbf{x}^{(n)})$

- $\ell(\mathbf{b}, \mathbf{w})$ is a convex function of parameters b_i, w_{ij}
- Gradient descent converges to the global optimum with gradients

$$\frac{\partial \ell}{\partial b_i} = E_{\text{data}}[x_i] - E_{\text{model}}[x_i] \qquad \frac{\partial \ell}{\partial w_{ij}} = E_{\text{data}}[x_i x_j] - E_{\text{model}}[x_i x_j]$$

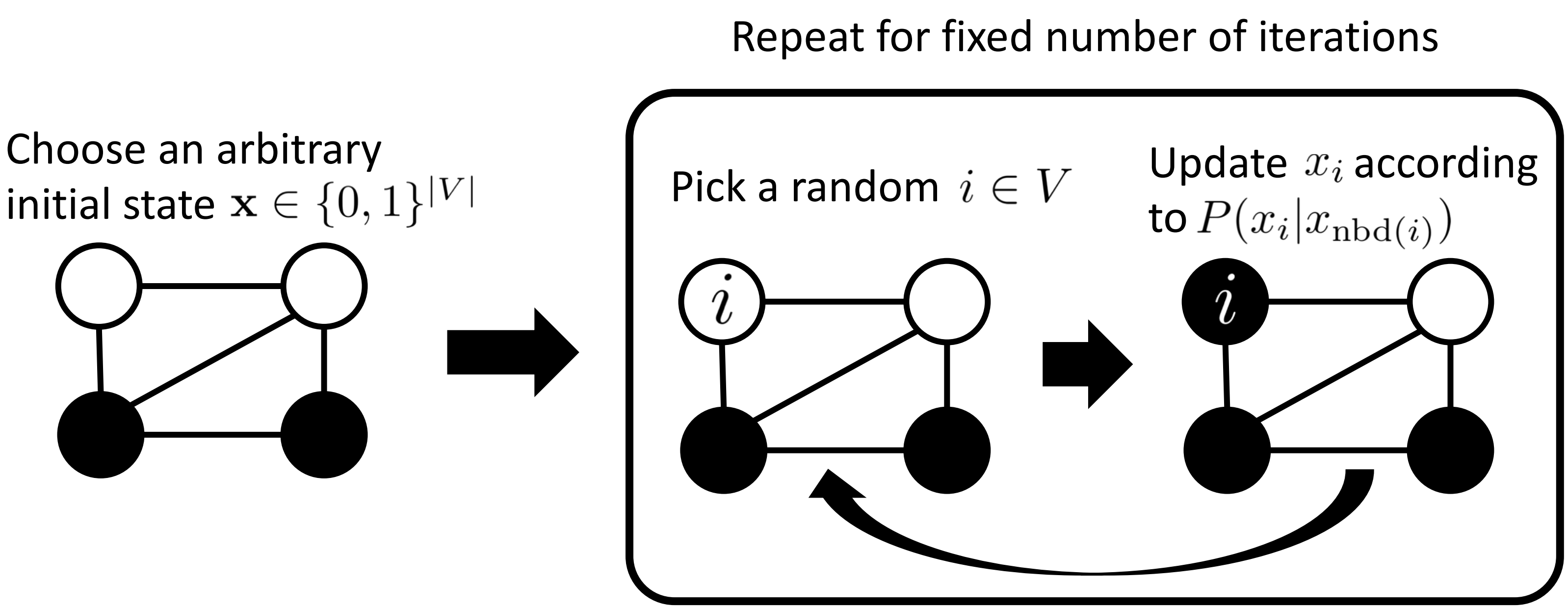
where $E_{\text{data}}[f(x)] = \frac{1}{N}\sum_{n=1}^{N} f(x^{(n)})$ is an empirical expectation

Learning BM

- **Problem**: Calculating $E_{\text{model}}[\cdot]$ is intractable in general, i.e., NP-Hard
  - Naïve approach requires $\Omega(2^{|V|})$ summations

- Instead of exact gradient, we approximate it using samples from $P_{\text{model}}(\mathbf{x})$

- **Gibbs sampler** is the most popular sampling algorithm in BM



Learning BM

- Learning BM
  1. Choose initial $\mathbf{b}, \mathbf{w}$
  2. Generate samples from $P_{\text{model}}(\mathbf{x})$ using Gibbs sampler
  3. Update parameters with approximated gradients from samples

$$b_i \leftarrow b_i + \gamma(E_{\text{data}}[x_i] - \widehat{E}_{\text{model}}[x_i])$$

$$w_{ij} \leftarrow w_{ij} + \gamma(E_{\text{data}}[x_i x_j] - \widehat{E}_{\text{model}}[x_i x_j])$$

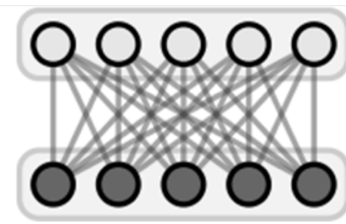
  4. Repeat 2-3 until convergence

Restricted Boltzmann Machine [Smolensky, 1986]

- **Restricted Boltzmann machine** (RBM) is a bipartite Boltzmann machine with visible nodes and hidden nodes

$$P(\mathbf{x}) \propto \sum_{\mathbf{h}} \exp\left(\sum_i b_i x_i + \sum_j c_j h_j + \sum_{i,j} w_{ij} x_i h_j\right)$$

Higher order potential



- Hidden nodes can be described by hidden features of visible nodes
- In RBM structure, all hidden nodes are conditionally independent given visible nodes and vise versa

$$P(x_i = 1|\mathbf{h}) = \sigma\left(\sum_j w_{ij} h_j + b_i\right) \quad P(h_j = 1|\mathbf{x}) = \sigma\left(\sum_i w_{ij} x_i + c_j\right)$$

*reference : http://arkitus.com/files/bms-eslami-slides.pptx

Learning RBM

- Given training data $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$, learn the distribution $P(\mathbf{x})$
- **Goal**: Maximize $\ell(\mathbf{b}, \mathbf{c}, \mathbf{w}) = \sum_{n=1}^{N} \log P_{\text{model}}(\mathbf{x}^{(n)})$

- $\ell(b, w)$ is a non-convex function of parameters b_i, w_{ij}
- But we still use gradient descent with gradients

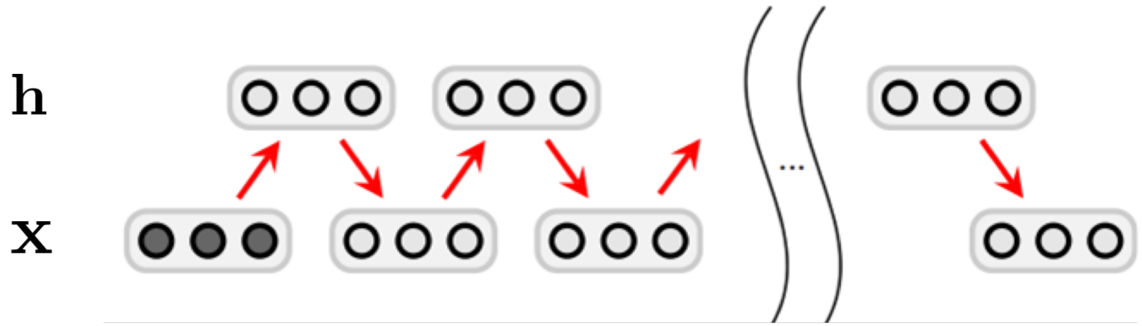
$$\frac{\partial \ell}{\partial b_i} = E_{\text{data}}[x_i] - E_{\text{model}}[x_i]$$

$$\frac{\partial \ell}{\partial c_j} = \frac{1}{N} \sum_{n=1}^{N} P_{\text{model}}(h_j = 1 | \mathbf{x}^{(n)}) - E_{\text{model}}[h_j]$$

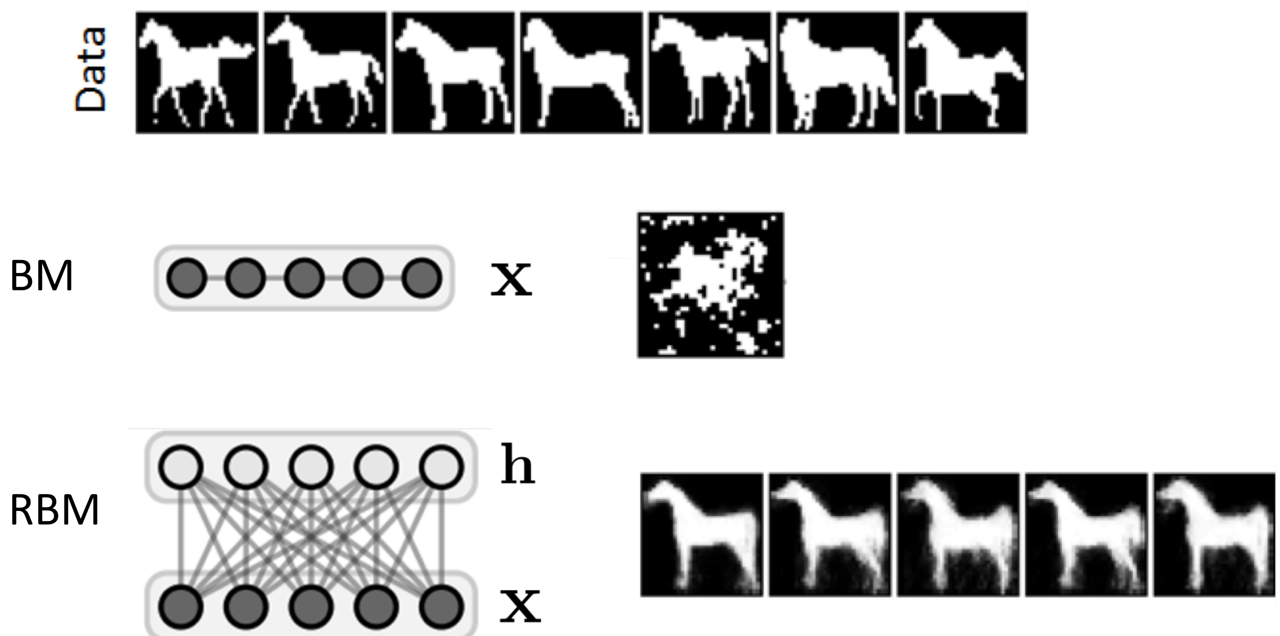
$$\frac{\partial \ell}{\partial w_{ij}} = \frac{1}{N} \sum_{n=1}^{N} x_i^{(n)} P_{\text{model}}(h_j = 1 | \mathbf{x}^{(n)}) - E_{\text{model}}[x_i h_j]$$

Learning RBM

- Due to conditional independence of RBM, block Gibbs sampling is possible



- Samples generated by BM and RBM



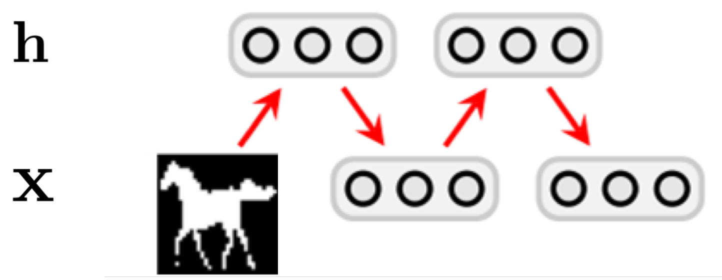
*reference : http://arkitus.com/files/bms-eslami-slides.pptx

Contrastive Divergence [Hinton, 2002]

- Convergence of a Gibbs sampler requires exponential number of iterations

- **Contrastive divergence** (CD-k) is a sampling method which runs only k iterations of a Markov chain without convergence guarantee

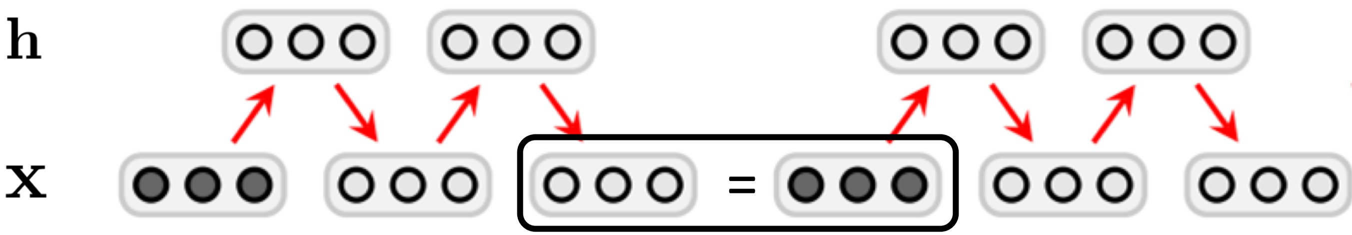
- For rapid mixing, CD chooses a initial state of the Markov chain from the training data



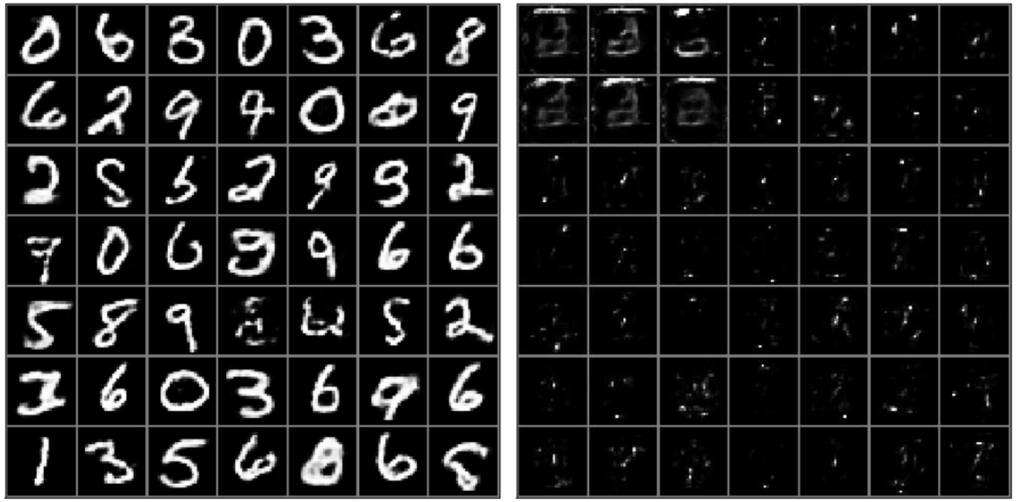
- CD-1 works surprisingly well in practice even though there is no guarantee

Persistent Contrastive Divergence [Tieleman, 2008]

- **Persistent Contrastive divergence** (PCD-k) runs k iterations of a Markov chain with initial state from the last Markov chain output
- We expect PCD chain approximates the full Markov chain of long iterations
- PCD requires more iterations than CD, but it shows better performance

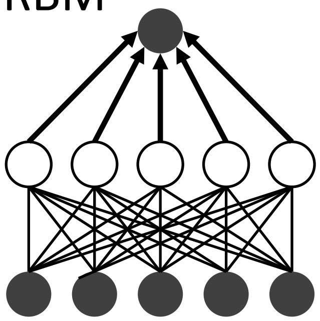


MNIST examples generated from RBM trained with PCD-1 and CD-1



Application: Classification using RBM [Nguyen et al., 2017]

- For a supervised learning using RBM, add a class variable to RBM
  - $\mathbf{x}$: binary input, y: multinomial label
  - Assume $P(y|\mathbf{x},\mathbf{h}) = P(y|\mathbf{h})$



$$P(\mathbf{x},\mathbf{h},y) = P(y|\mathbf{x},\mathbf{h})P(\mathbf{x},\mathbf{h}) = P(y|\mathbf{h})P(\mathbf{x},\mathbf{h})$$

$$P(y|\mathbf{h}) \propto \exp\left(d^y + \sum_j v_j^y h_j\right)$$

$$P(\mathbf{x},\mathbf{h}) \propto \exp\left(\sum_i b_i x_i + \sum_j c_j h_j + \sum_{i,j} w_{ij} x_i h_j\right)$$

- Gradient descent for parameter learning
  - **Goal**: Maximize log likelihood $\ell(\mathbf{b},\mathbf{c},\mathbf{d},\mathbf{v},\mathbf{w}) = \sum_{n=1}^N \log P_{\text{model}}(\mathbf{x}^{(n)}, y^{(n)})$
  - Similar to learning RBM, computing gradient requires $P_{\text{model}}(h_j = 1|\mathbf{x}^{(n)}, y^{(n)})$ but block Gibbs sampler is not available (inefficient sampling)
    - $P(\mathbf{h}|\mathbf{x},y) \neq \prod_j P(h_j|\mathbf{x},y)$

Application: Classification using RBM [Nguyen et al., 2017]

- Gradient descent for parameter learning
  - **Solution**: Use $Q(\mathbf{h}) = \prod_j Q(h_j)$ for approximating $P_{\text{model}}(\mathbf{h}|\mathbf{x}^{(n)}, y^{(n)})$
    - Such an approximation is called the **mean field approximation**
    - Solve the optimization minimizing KL divergence

$$\min_Q KL(Q(\mathbf{h})||P(\mathbf{h}|\mathbf{x}, y))$$

$$= \min_Q \left(E_{Q(\mathbf{h})}[\log Q(\mathbf{h})] - E_{Q(\mathbf{h})}[\log P(\mathbf{x}, \mathbf{h}, y)] \right) - \log P(\mathbf{x}, y)$$

  - Formulation $E_{Q(\mathbf{h})}[\log Q(\mathbf{h})]$ as a function of $\mu_j = Q(h_j = 1)$

$$\begin{aligned}
E_{Q(\mathbf{h})}[\log Q(\mathbf{h})] &= \sum_{\mathbf{h}} \left(\prod_j Q(h_j) \right) \log \left(\prod_j Q(h_j) \right) = \sum_{\mathbf{h}} \left(\prod_j Q(h_j) \right) \left(\sum_j \log Q(h_j) \right) \\
&= \sum_{\mathbf{h}} \left(\prod_j Q(h_j) \right) \left(\sum_j \log Q(h_j = 1) \mathbf{1}_{h_j=1} + \log Q(h_j = 0) \mathbf{1}_{h_j=0} \right) \\
&= \sum_{\mathbf{h}} \left(\prod_j Q(h_j) \right) \left(\sum_j \log Q(h_j = 1) \mathbf{1}_{h_j=1} \right) + \sum_{\mathbf{h}} \left(\prod_j Q(h_j) \right) \left(\sum_j \log Q(h_j = 0) \mathbf{1}_{h_j=0} \right) \\
&= \sum_j \log Q(h_j = 1) \left(\sum_{\mathbf{h}:h_j=1} \prod_{j'} Q(h_{j'}) \right) + \sum_j \log Q(h_j = 0) \left(\sum_{\mathbf{h}:h_j=0} \prod_{j'} Q(h_{j'}) \right) \\
&= \sum_j Q(h_j = 1) \log Q(h_j = 1) + \sum_j Q(h_j = 0) \log Q(h_j = 0) \\
&= \boxed{\sum_j \mu_j \log \mu_j + (1 - \mu_j) \log(1 - \mu_j)}
\end{aligned}$$

1 : Indicator function

Gradient w.r.t. μ_j is tractable

Application: Classification using RBM [Nguyen et al., 2017]

- Gradient descent for parameter learning
  - Formulation of $E_{Q(\mathbf{h})}[\log P(\mathbf{x},\mathbf{h},y)]$ as a function of $\mu_j = Q(h_j = 1)$

$$E_{Q(\mathbf{h})}[\log P(\mathbf{x},\mathbf{h},y)] = E_{Q(\mathbf{h})}[\log(P(y|\mathbf{h})P(\mathbf{x},\mathbf{h})]$$

$$= E_{Q(\mathbf{h})}\left[\log\left(\frac{\exp\left(d^y + \sum_j v_j^y h_j\right)}{\sum_y \exp\left(d^y + \sum_j v_j^y h_j\right)} \frac{\exp\left(\sum_i b_i x_i + \sum_j c_j h_j + \sum_{i,j} w_{ij} x_i h_j\right)}{\sum_{\mathbf{x},\mathbf{h}} \exp\left(\sum_i b_i x_i + \sum_j c_j h_j + \sum_{i,j} w_{ij} x_i h_j\right)}\right)\right]$$

$$= E_{Q(\mathbf{h})}\left[d^y + \sum_j v_j^y h_j + \sum_i b_i x_i + \sum_j c_j h_j + \sum_{i,j} w_{ij} x_i h_j - A(\mathbf{d},\mathbf{v},\mathbf{h}) - B(\mathbf{b},\mathbf{c},\mathbf{w})\right]$$

$$= E_{Q(\mathbf{h})}\left[\sum_j v_j^y h_j + \sum_j c_j h_j + \sum_{i,j} w_{ij} x_i h_j - A(\mathbf{d},\mathbf{v},\mathbf{h})\right] + E_{Q(\mathbf{h})}\left[d^y + \sum_i b_i x_i - B(\mathbf{b},\mathbf{c},\mathbf{w})\right]$$

$$= \boxed{\sum_j \mu_j \left(v_j^y + c_j + \sum_i w_{ij} x_i\right)} - \boxed{E_{Q(\mathbf{h})}[A(\mathbf{d},\mathbf{v},\mathbf{h})]} + \boxed{d^y + \sum_i b_i x_i - B(\mathbf{b},\mathbf{c},\mathbf{w})}$$

Gradient w.r.t. μ_j is tractable

where $A(\mathbf{d},\mathbf{v},\mathbf{h}) = \log\left(\sum_y \exp\left(d^y + \sum_j v_j^y h_j\right)\right)$

$B(\mathbf{b},\mathbf{c},\mathbf{w}) = \log\left(\sum_{\mathbf{x},\mathbf{h}} \exp\left(\sum_i b_i x_i + \sum_j c_j h_j + \sum_{i,j} w_{ij} x_i h_j\right)\right)$

- Gradient descent for parameter learning
  - **Problem**: Computing below is hard

$$\frac{\partial}{\partial \mu_j} E_{Q(\mathbf{h})}[A(\mathbf{d},\mathbf{v},\mathbf{h})] = \frac{\partial}{\partial \mu_j} \sum_{\mathbf{h}} \left(\prod_j \mu_j^{h_j} (1-\mu_j)^{1-h_j} \right) \log \left(\sum_y \exp \left(d^y + \sum_j v_j^y h_j \right) \right)$$

$$\boxed{\begin{aligned} &= \sum_{\mathbf{h}:h_j=1} \left(\prod_{j' \neq j} \mu_{j'}^{h_{j'}} (1-\mu_{j'})^{1-h_{j'}} \right) \log \left(\sum_y \exp \left(d^y + \sum_j v_j^y h_j \right) \right) \\ &- \sum_{\mathbf{h}:h_j=0} \left(\prod_{j' \neq j} \mu_{j'}^{h_{j'}} (1-\mu_{j'})^{1-h_{j'}} \right) \log \left(\sum_y \exp \left(d^y + \sum_j v_j^y h_j \right) \right) \end{aligned}}$$

  - **Solution**: Approximate $E_{Q(\mathbf{h})}[A(\mathbf{d},\mathbf{v},\mathbf{h})]$ into tractable expression
    - 1st order Taylor series approximation at $\mathbf{h} = \boldsymbol{\mu}$

$$\begin{aligned} E_{Q(\mathbf{h})}[A(\mathbf{d},\mathbf{v},\mathbf{h})] &\approx E_{Q(\mathbf{h})}[A(\mathbf{d},\mathbf{v},\boldsymbol{\mu}) + (\mathbf{h}-\boldsymbol{\mu})^T \nabla_{\mathbf{h}} A(\mathbf{d},\mathbf{v},\mathbf{h})|_{\mathbf{h}=\boldsymbol{\mu}}] \\ &= E_{Q(\mathbf{h})}[A(\mathbf{d},\mathbf{v},\boldsymbol{\mu})] + E_{Q(\mathbf{h})}[(\mathbf{h}-\boldsymbol{\mu})^T \nabla_{\mathbf{h}} A(\mathbf{d},\mathbf{v},\mathbf{h})|_{\mathbf{h}=\boldsymbol{\mu}}] \\ &= A(\mathbf{d},\mathbf{v},\boldsymbol{\mu}) + E_{Q(\mathbf{h})}[(\mathbf{h}-\boldsymbol{\mu})^T] \nabla_{\mathbf{h}} A(\mathbf{d},\mathbf{v},\mathbf{h})|_{\mathbf{h}=\boldsymbol{\mu}} \\ &= \boxed{A(\mathbf{d},\mathbf{v},\boldsymbol{\mu})} \end{aligned}$$

$$\frac{\partial}{\partial \mu_j} A(\mathbf{d},\mathbf{v},\boldsymbol{\mu}) = \frac{\sum_y v_j^y \exp\left(d^y + \sum_j v_j^y \mu_j\right)}{\sum_y \exp\left(d^y + \sum_j v_j^y \mu_j\right)}$$

Application: Classification using RBM [Nguyen et al., 2017]

- Gradient descent for parameter learning
  - Approximate $E_{Q(\mathbf{h})}[A(\mathbf{d},\mathbf{v},\mathbf{h})]$ into polynomial
    - 2nd order Taylor series approximation at $\mathbf{h}=\boldsymbol{\mu}$

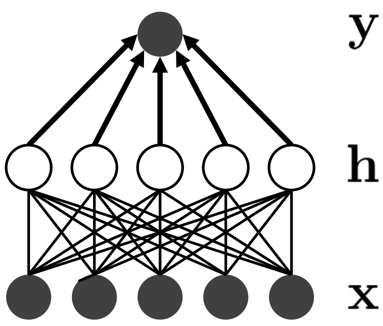
$$
\begin{aligned}
E_{Q(\mathbf{h})}[A(\mathbf{d},\mathbf{v},\mathbf{h})] &\approx E_{Q(\mathbf{h})}[A(\mathbf{d},\mathbf{v},\boldsymbol{\mu}) + (\mathbf{h}-\boldsymbol{\mu})^T \nabla_{\mathbf{h}} A(\mathbf{d},\mathbf{v},\mathbf{h})|_{\mathbf{h}=\boldsymbol{\mu}}] \\
&\quad + E_{Q(\mathbf{h})}\left[\frac{1}{2}(\mathbf{h}-\boldsymbol{\mu})^T \nabla^2_{\mathbf{h}} A(\mathbf{d},\mathbf{v},\mathbf{h})|_{\mathbf{h}=\boldsymbol{\mu}}(\mathbf{h}-\boldsymbol{\mu})\right] \\
&= A(\mathbf{d},\mathbf{v},\boldsymbol{\mu}) + E_{Q(\mathbf{h})}\left[\frac{1}{2}\sum_{j,j'}(h_j-\mu_j)(h_{j'}-\mu_{j'})\frac{\partial}{\partial h_j}\frac{\partial}{\partial h_{j'}}A(\mathbf{d},\mathbf{v},\mathbf{h})|_{\mathbf{h}=\boldsymbol{\mu}}\right] \\
&= A(\mathbf{d},\mathbf{v},\boldsymbol{\mu}) + \frac{1}{2}\sum_{j,j'} E_{Q(\mathbf{h})}\left[(h_j-\mu_j)(h_{j'}-\mu_{j'})\right]\frac{\partial}{\partial h_j}\frac{\partial}{\partial h_{j'}}A(\mathbf{d},\mathbf{v},\mathbf{h})|_{\mathbf{h}=\boldsymbol{\mu}} \\
&= A(\mathbf{d},\mathbf{v},\boldsymbol{\mu}) + \frac{1}{2}\sum_{j} E_{Q(\mathbf{h})}\left[(h_j-\mu_j)^2\right]\frac{\partial}{\partial h_j}\frac{\partial}{\partial h_{j'}}A(\mathbf{d},\mathbf{v},\mathbf{h})|_{\mathbf{h}=\boldsymbol{\mu}} \\
&= \boxed{A(\mathbf{d},\mathbf{v},\boldsymbol{\mu}) + \frac{1}{2}\sum_{j} \mu_j(1-\mu_j)\frac{\partial^2}{\partial h_j^2}A(\mathbf{d},\mathbf{v},\mathbf{h})|_{\mathbf{h}=\boldsymbol{\mu}}} \longrightarrow \text{Closed form gradient exists}
\end{aligned}
$$

  - After Taylor series approximation, $\boldsymbol{\mu}$ can be optimized using coordinate descent
  - Also, gradient can be approximated by approximating $Q(\mathbf{h}) \approx P(\mathbf{h}|\mathbf{x},y)$

- **Evaluation**: Exact inference is intractable, use approximated inference

$$P(y|\mathbf{x}) = \sum_{\mathbf{h}} P(y|\mathbf{h})P(\mathbf{h}|\mathbf{x}) \approx \frac{\exp\left(\sum_j w_j^y \nu_j + b^y\right)}{\sum_y \exp\left(\sum_j w_j^y \nu_j + b^y\right)}$$

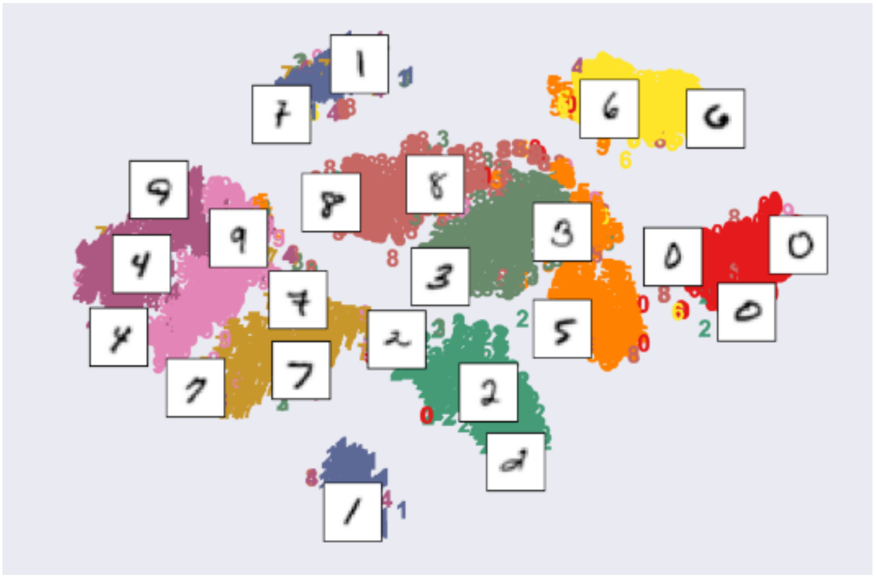
where $\nu_j = P(h_j = 1|\mathbf{x})$



- Classification using
  1. Above approximated output
  2. kNN, SVN on the hidden feature space of RBM

Table 1: Classification errors (%) on testing sets.

	MNIST	20 Newsgroups
RBM+kNN	3.03	56.15
RBM+SVM	1.76	41.79
ClassRBM+kNN	2.98	57.80
ClassRBM+SVM	1.68	40.88
ClassRBM	3.39	24.9
sRBM+kNN	2.94	55.89
sRBM+SVM	1.42	38.43
sRBM-1st	2.27	24.1
sRBM-2nd	**2.21**	**23.2**



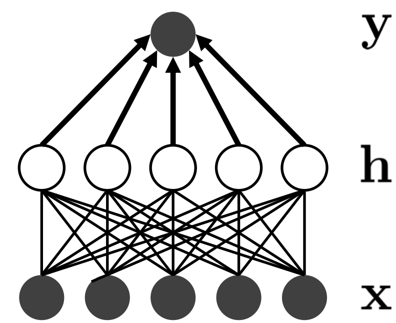
Hidden feature projection of MNIST examples

Application: Classification using RBM [Nguyen et al., 2017]

- **Evaluation**: Exact inference is intractable, use approximated inference

$$P(y|\mathbf{x}) = \sum_{\mathbf{h}} P(y|\mathbf{h})P(\mathbf{h}|\mathbf{x}) \approx \frac{\exp\left(\sum_j w_j^y \nu_j + b^y\right)}{\sum_y \exp\left(\sum_j w_j^y \nu_j + b^y\right)}$$

where $\nu_j = P(h_j = 1|\mathbf{x})$



- Classification using
  1. Above approximated output
  2. kNN, SVN on the hidden feature space of RBM
- Generation of RBM using Gibbs sampler

RBM VAE GAN

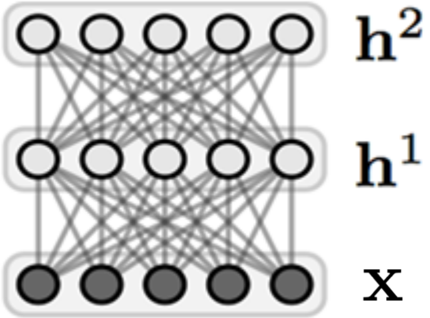
Deep Boltzmann Machine [Salakhutdinov et al., 2009]

- **Deep Boltzmann machine** (DBM) has a deeper structure than RBM
- Higher expressive power as DBM becomes deeper

$$P(\mathbf{x}) \propto \sum_{\mathbf{h}^1,\mathbf{h}^2} \exp\Big(\sum_i b_i x_i + \sum_j c_j^1 h_j^1 + \sum_k c_k^2 h_k^2$$

Higher order & complex potential

$$+ \sum_{i,j} w_{ij}^1 x_i h_j^1 + \sum_{j,k} w_{jk}^1 h_j^1 h_k^2\Big)$$



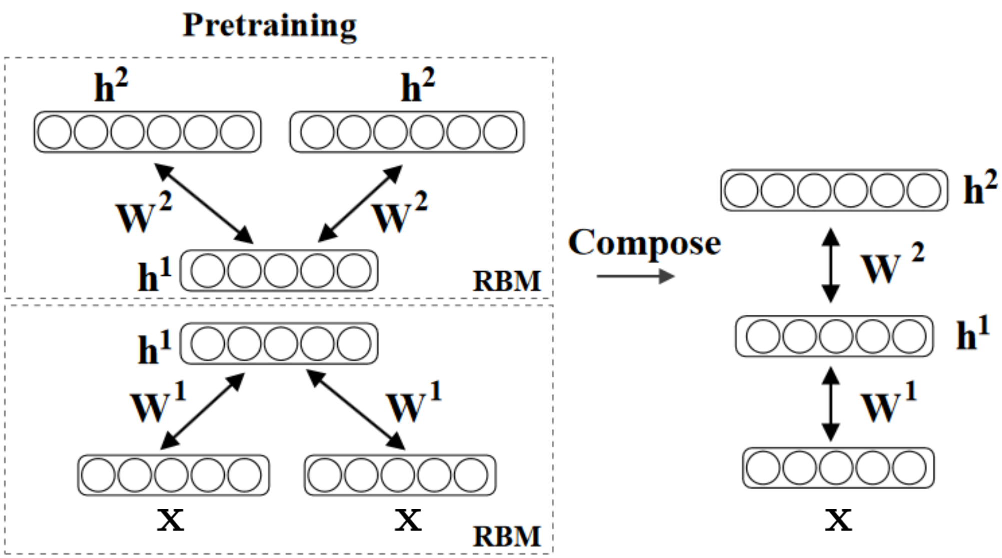
- DBM also contains conditionally independence

$$P(x_i = 1|\mathbf{h}^1) = \sigma\left(\sum_j w_{ij}^1 h_j^1 + b_i\right)$$
$$P(h_j^1 = 1|\mathbf{x}, \mathbf{h}^2) = \sigma\left(\sum_i w_{ij}^1 x_i + \sum_k w_{jk}^2 h_k^2 + c_j^1\right)$$
$$P(h_k^2 = 1|\mathbf{h}^1) = \sigma\left(\sum_j w_{jk}^2 h_j^1 + c_k^2\right)$$

*reference : http://arkitus.com/files/bms-eslami-slides.pptx

Learning DBM

- Learning DBM is similar to learning RBM
- **Pretraining** each layer with RBM empirically achieves better performance

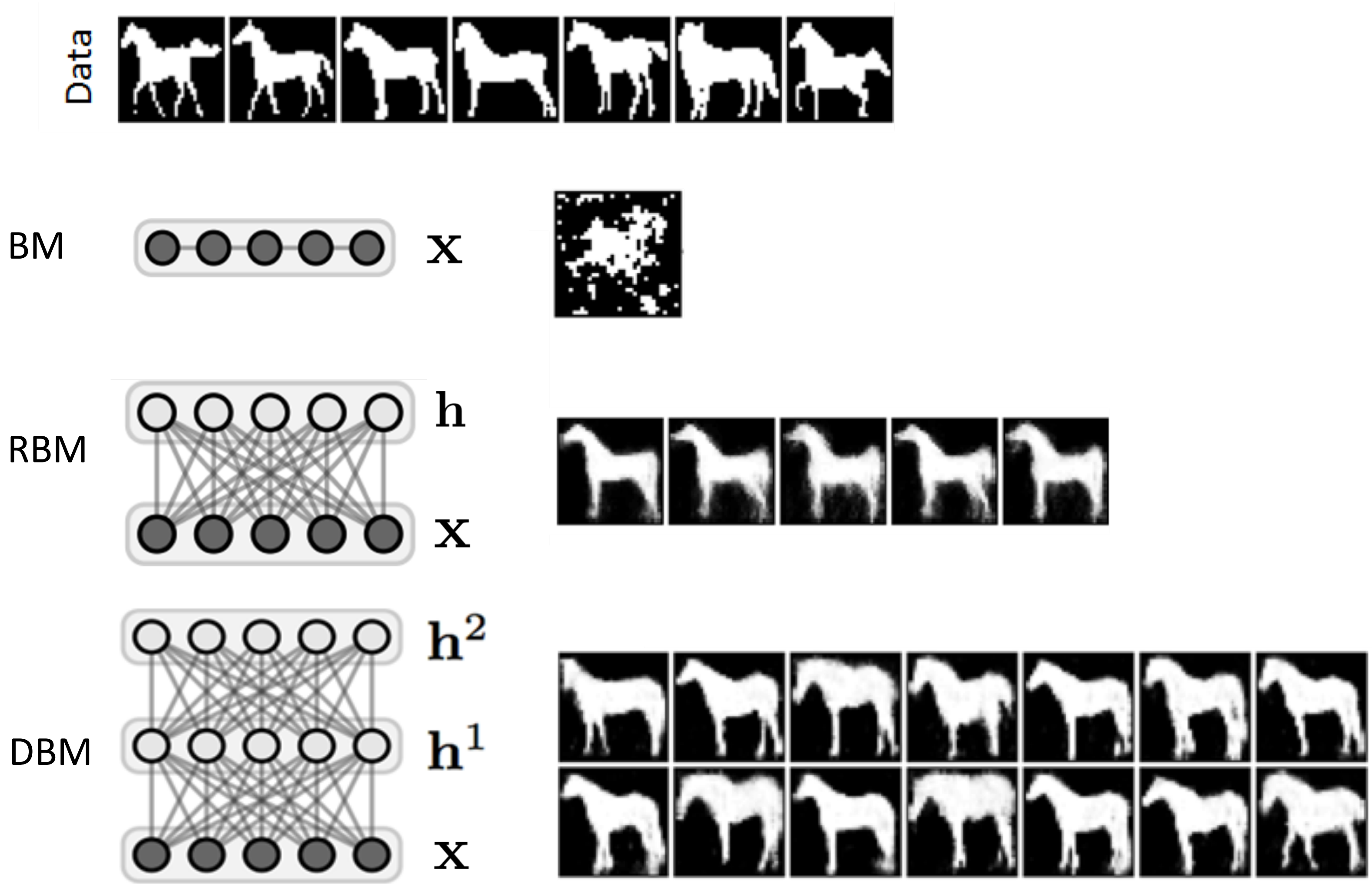


- As used in [Nguyen et al., 2017], the mean field approximation is a good alternative for slow Gibbs sampler in DBM [Salakhutdinov, 2010]

$$Q^{MF}(\mathbf{h}^1, \mathbf{h}^2 | x) = \prod_j \prod_k q(h_j^1) q(h_k^2)$$

Learning DBM

- Samples generated by BM, RBM and DBM

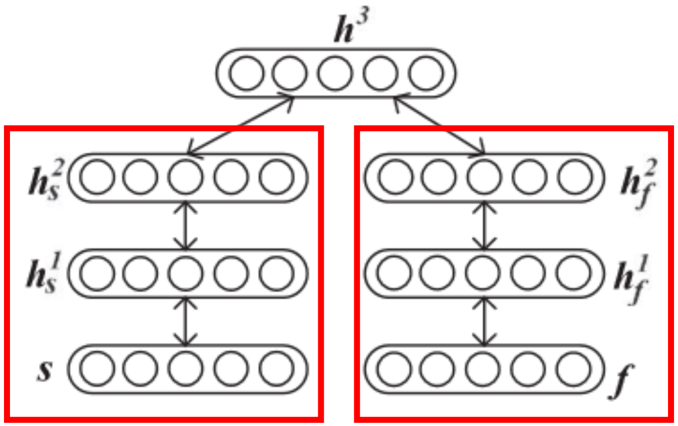


*reference : http://arkitus.com/files/bms-eslami-slides.pptx

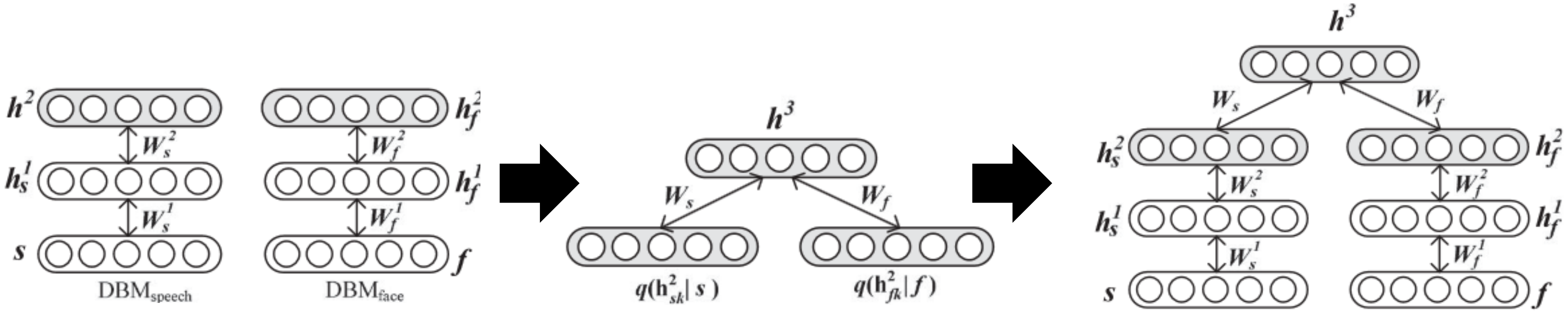
Application of DBM: Multimodal Learning [Pang et al., 2017]

- **Goal**: Extract a joint feature from image and audio data

- Joint DBM consisting of sub-DBMs corresponding to modalities



- Learning joint DBM
    1. Extract features from each modality
    2. Pretrain sub-DBM for each modality using extracted features
    3. Pretrain the shared parameters
    4. Fine tune the joint DBM with pretrained parameter

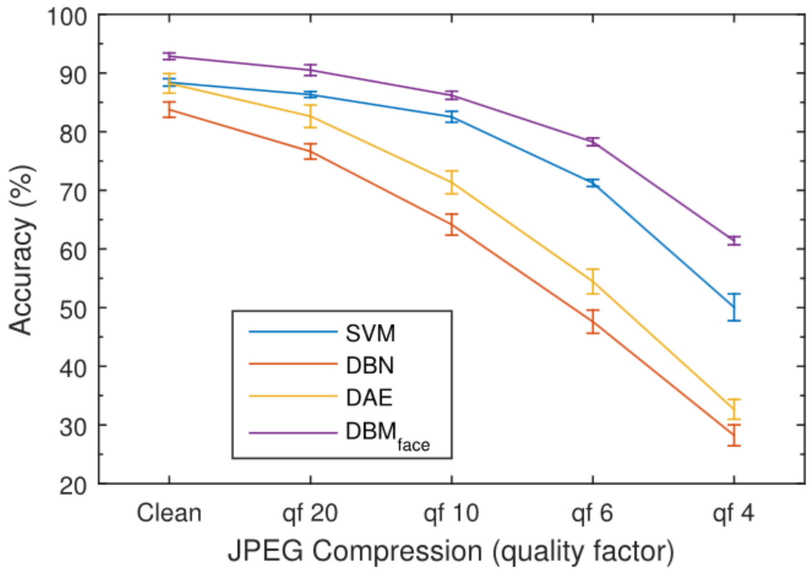


Application of DBM: Multimodal Learning [Pang et al., 2017]

- **Evaluation**: Person identification using face image and audio data
  - Logistic regression classifier is used for extracted DBM features

Method	IR (%)
SVM (concatenated features)	94.33 ± 0.087
Bimodal DAE [9]	88.70 ± 0.256
Bimodal DBN (Fig. 1c)	88.33 ± 0.465
Bimodal DBM (conventional training) [11]	93.90 ± 0.114
score fusion [14], [15]	93.70 ± 0.247
jDBM (proposed three-step training)	$\mathbf{96.33} \pm 0.074$

- DBM also shows robustness on noisy unimodal data



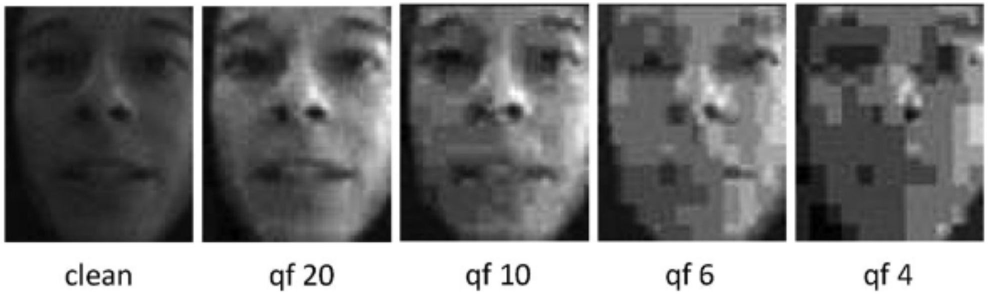


Table of Contents

Complex and tractable model?

- Modelling a joint distribution of $\mathbf{x}$
  - **Mean Fields**
    - $P(\mathbf{x}) = \prod_i P(x_i)$
    - Tractable inference, low expressive power
  - **Multivariate Gaussian distributions**
    - $P(\mathbf{x}) \propto \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)\Sigma^{-1}(\mathbf{x}-\mu)\right)$
    - Tractable inference, low expressive power

  Simple and tractable models

  - **Graphical models** (e.g., RBM, DBM, etc.)
    - $P(\mathbf{x}) \propto \exp\left(\sum_i b_i x_i + \sum_{i,j} w_{ij} x_i x_j\right)$
    - Intractable inference, complex but simple expression
  - **Generative adversarial networks**
    - $P(\mathbf{x}) \propto |f^{-1}(\mathbf{x})|$ for some neural network
    - Intractable inference, complex expression

  Complex and intractable models

Network Polynomial

- **Goal**: Model tractable distributions over $\mathbf{x} \in \{0,1\}^n$

- Any distribution can be represented by **network polynomial**

$$P(x_1, x_2) \propto F(x_1 = 0, x_2 = 0)\bar{x}_1\bar{x}_2 + F(x_1 = 0, x_2 = 1)x_1\bar{x}_2 \\ + F(x_1 = 1, x_2 = 0)\bar{x}_1 x_2 + F(x_1 = 1, x_2 = 1)x_1 x_2$$

- **Idea**: Learn coefficients of network polynomial which enables tractable inference
  - Example: mean field

Iterative sum and product

$$P(\mathbf{x}) \propto F(\mathbf{x}) = \prod_i \left(f_i(x_i = 1)x_i + f_i(x_i = 0)\bar{x}_i\right)$$

$$\sum_{\mathbf{x}} F(\mathbf{x}) = \sum_{\mathbf{x}} \prod_i \left(f_i(x_i = 1)x_i + f_i(x_i = 0)\bar{x}_i\right)$$

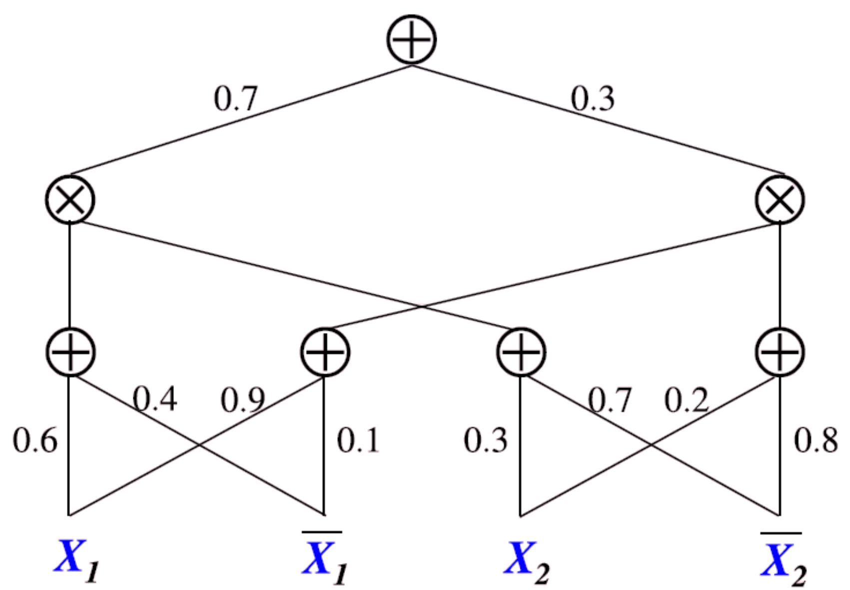
$$= \prod_i \left(\sum_{x_i} \left(f_i(x_i = 1)x_i + f_i(x_i = 0)\bar{x}_i\right) \right)$$

$$= \prod_i \left(f_i(x_i = 1) + f_i(x_i = 0)\right)$$

Sum-product Network [Poon et al., 2012]

- **Sum-product network** (SPN) provides tractable distribution over $\mathbf{x} \in \{0,1\}^n$
  - $P(\mathbf{x}) = F(x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n)/Z, \; F(\cdot)$ is SPN
  - $Z = \sum_{\mathbf{x} \in \{0,1\}^n} F(x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n)$

- SPN consists of iterative sum and product nodes (operations)
  - e.g. $$F(x_1, \bar{x}_1, x_2, \bar{x}_2) = 0.7(0.6x_1 + 0.4\bar{x}_1)(0.3x_2 + 0.7\bar{x}_2) + 0.3(0.9x_1 + 0.1\bar{x}_1)(0.2x_2 + 0.8\bar{x}_2)$$



*reference : https://jmhldotorg.files.wordpress.com/2013/11/slidescambridgesumproductnetworks2013.pdf

Sum-product Network [Poon et al., 2012]

- **Sum-product network** (SPN) provides tractable distribution over $\mathbf{x} \in \{0,1\}^n$
  - $P(\mathbf{x}) = F(x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n)/Z,\ F(\cdot)$ is SPN
  - $Z = \sum_{\mathbf{x}\in\{0,1\}^n} F(x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n)$
  - Direct calculation of Z requires 2^n number of summations but…

- **Observation**: If F is a polynomial with a form

$$F(x_1, \bar{x}_1, x_2, \bar{x}_2) = F(0,1,0,1)\bar{x}_1\bar{x}_2 + F(1,0,0,1)x_1\bar{x}_2 + F(0,1,1,0)\bar{x}_1 x_2 + F(1,0,1,0)x_1 x_2$$

then $F(1,1,1,1) = \sum_{\mathbf{x}\in\{0,1\}^2} F(\mathbf{x}) = Z$

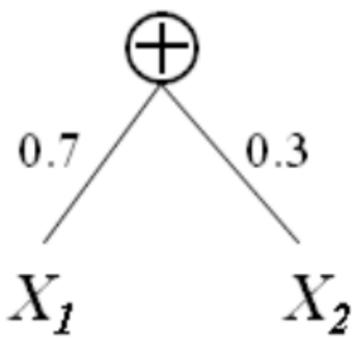
- What if there exist monomials such as x_i or $x_i\bar{x}_i$?

*reference : https://jmhldotorg.files.wordpress.com/2013/11/slidescambridgesumproductnetworks2013.pdf

Any SPN Produces Tractable and Valid Distribution?

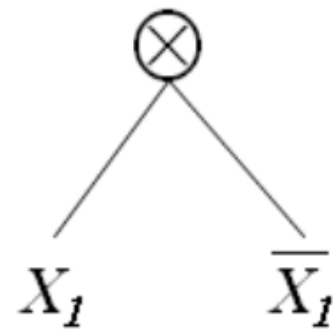
- **Theorem**: SPN is valid ($Z = F(1,\ldots,1)$) if
  - (consistent) There is no monomial containing both $x_i, \bar{x}_i$
  - (complete) Children of sum node have same set of descendant leaf node

Incomplete



$$F(x_1, \bar{x}_1, x_2, \bar{x}_2) = 0.7x_1 + 0.3x_2$$
$$Z = 2 > F(1,1,1,1) = 1$$

Inconsistent



$$F(x_1, \bar{x}_1) = x_1\bar{x}_1$$
$$Z = 0 < F(1,1) = 1$$

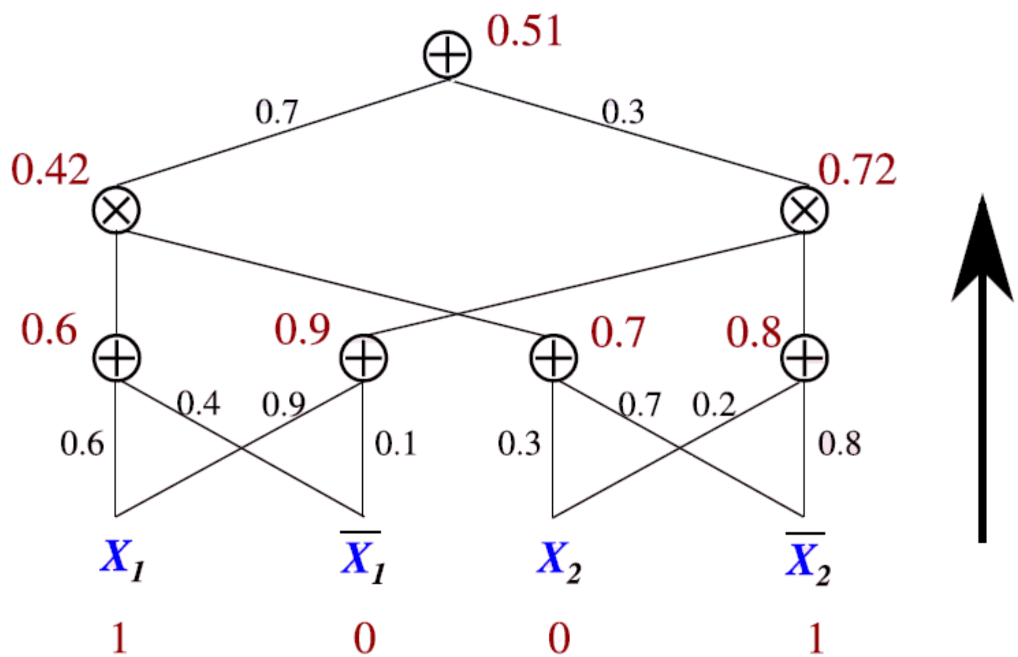
- Theorem implies that SPN is valid if it only contains n-th order monomials which consist of all x_i or $\bar{x}_i$ for all i

*reference : https://jmhldotorg.files.wordpress.com/2013/11/slidescambridgesumproductnetworks2013.pdf

Joint Probability in Valid SPN

- **Goal**: calculate joint probability in valid SPN
  - $P(\mathbf{x}) = F(x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n)/Z$
  - $Z = \sum_{\mathbf{x}} F(x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n)$

$$= F(x_1 = 1, \bar{x}_1 = 1, x_2 = 1, \bar{x}_2 = 1, \ldots, x_n = 1, \bar{x}_n = 1)$$

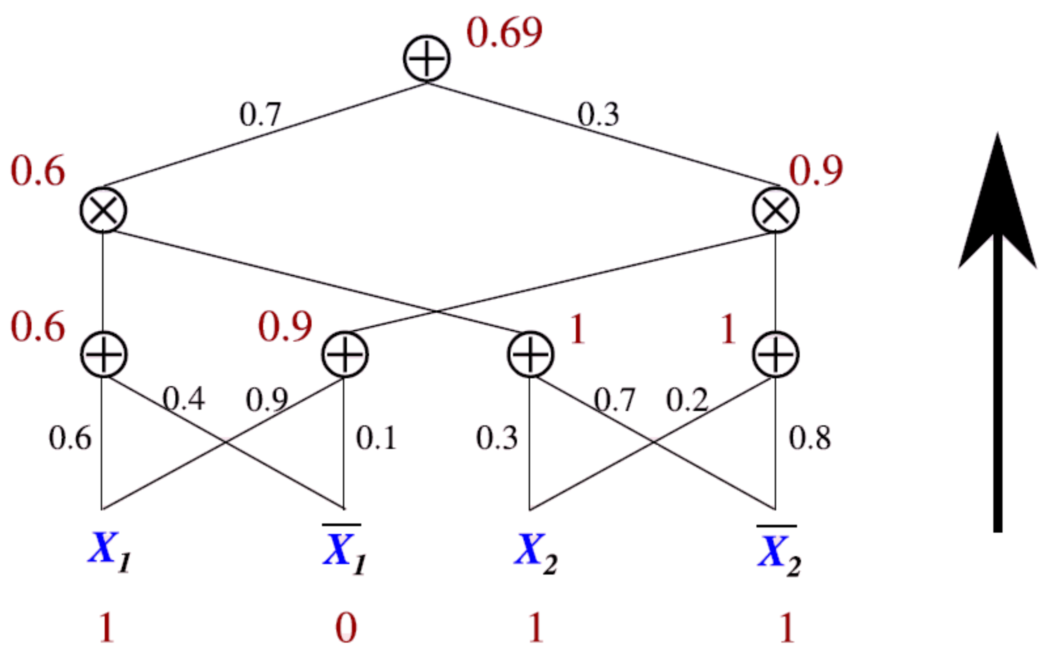


$$P(x_1 = 1, x_2 = 0) = \frac{F(x_1 = 1, \bar{x}_1 = 0, x_2 = 0, \bar{x}_2 = 1)}{Z}$$

*reference : https://jmhldotorg.files.wordpress.com/2013/11/slidescambridgesumproductnetworks2013.pdf

Marginal Probability in Valid SPN

- **Goal**: calculate marginal probability in valid SPN
  - $P(x_1 = 1) = F(x_1 = 1, \bar{x}_1 = 0, x_2 = 1, \bar{x}_2 = 1, \ldots, x_n = 1, \bar{x}_n = 1)/Z$



$$P(x_1 = 1) = \frac{F(x_1 = 1, \bar{x}_1 = 0, x_2 = 1, \bar{x}_2 = 1)}{Z}$$

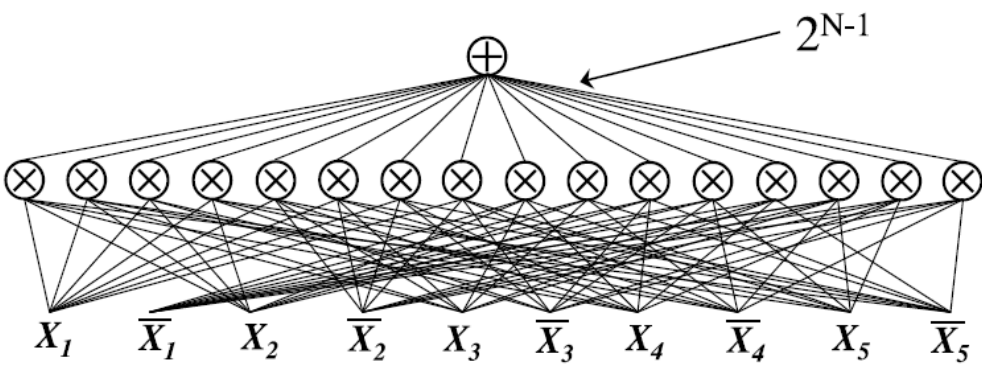
*reference : https://jmhldotorg.files.wordpress.com/2013/11/slidescambridgesumproductnetworks2013.pdf

Deep vs Shallow SPNs

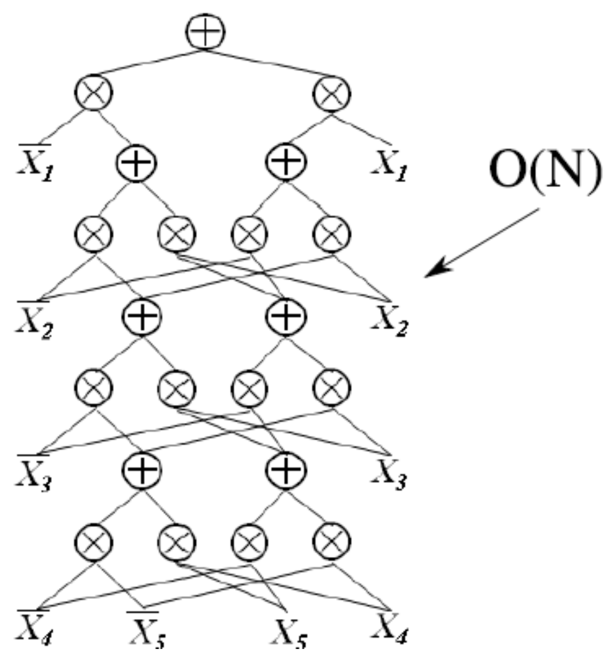
- **Any distribution** can be encoded using shallow large SPN [Poon et al., 2012]

- **Some distribution** can be encoded using compact deep SPN
  - e.g. uniform distribution over states with even number of 1's

Shallow large SPN



Deep compact SPN

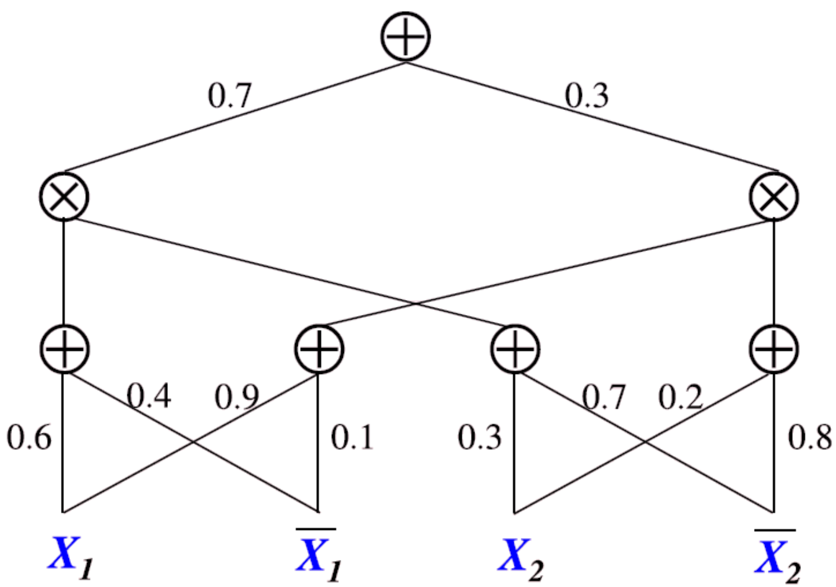


*reference : https://jmhldotorg.files.wordpress.com/2013/11/slidescambridgesumproductnetworks2013.pdf

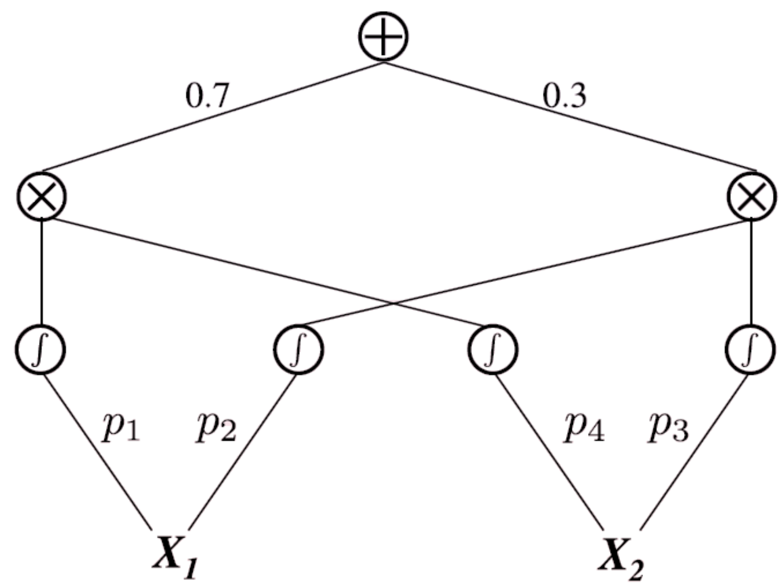
Extension to Multivariate/Continuous Models

- Replace sum nodes by integral (or weighted sum) nodes
  - Integral for continuous models and weighted sum for multivariate models

- When all p are Gaussian, SPN defines very large mixture of Gaussian



Binary SPN



Continuous/multivariate SPN

*reference : https://jmhldotorg.files.wordpress.com/2013/11/slidescambridgesumproductnetworks2013.pdf

Learning SPN

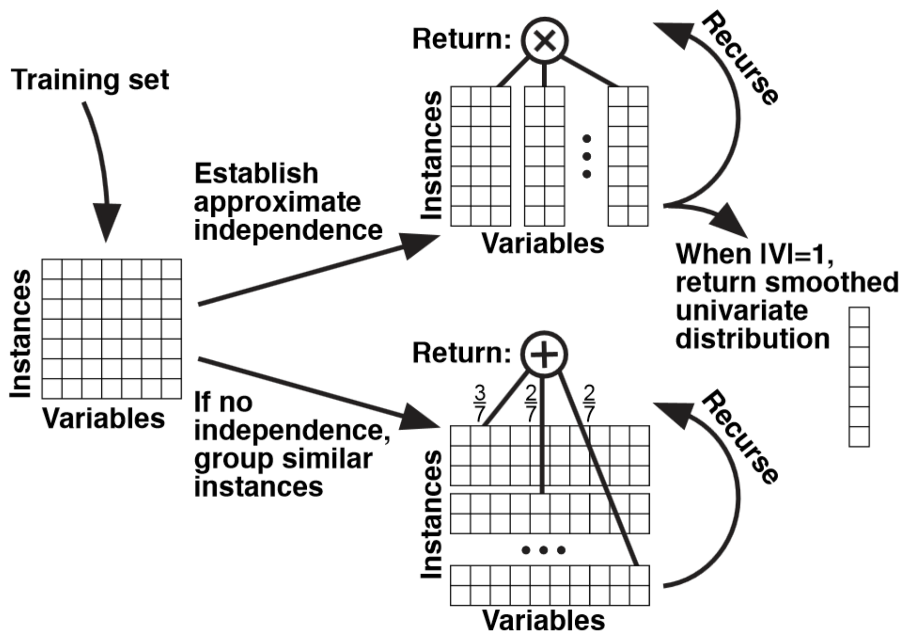
- Learning SPN is similar to learning neural networks
    - Select an appropriate structure for target dataset
    - Define target loss function from target dataset/distribution
    - Iteratively update weights using back-propagation

- Learning SPN
    1. **Initialize** SPN with some ‘valid’ structure and random parameter
    2. **Choose** an appropriate loss function (e.g., maximum likelihood: $\max \sum_n P(\mathbf{x}^{(n)})$)
    3. **Update** weight until convergence
        - Weight update algorithm can be generally chosen (e.g., gradient descent)
        - Gradient descent can be done using back-propagation
    4. **Prune** zero weighted edges

Structure Learning of SPN [Gens et al., 2013]

- **Recall**: Intuition behind SPN
  - Each sum node represents the mixture of distributions
  - Each product node represents the independence of variables

- Given samples, structure learning of SPN is an iterative procedure of
  - Finding independence (independence test, e.g., G-test of pairwise independence)
  - Finding similar instances (clustering methods, e.g., k-means clustering)



Application of SPN: Image Completion [Poon et al., 2012]

- Completing the missing left half of images
  - SPN is trained using Caltech-101, Olivetti datasets
  - Consider each pixel as a mixture of Gaussian with unit variance
- Comparison with several algorithms

Top to bottom: original, SPN, DBM, DBN, PCA, nearest neighbor

Application of SPN: Discriminative Learning [Gens et al., 2012]

- **Goal**: Use SPN as an image classifier

- **Problem**
  - Finding deep valid structure for high dimensional variable may decrease the classification accuracy as it restrict the number of edges
  - Modelling continuous image pixels into Gaussian distribution may not be realistic

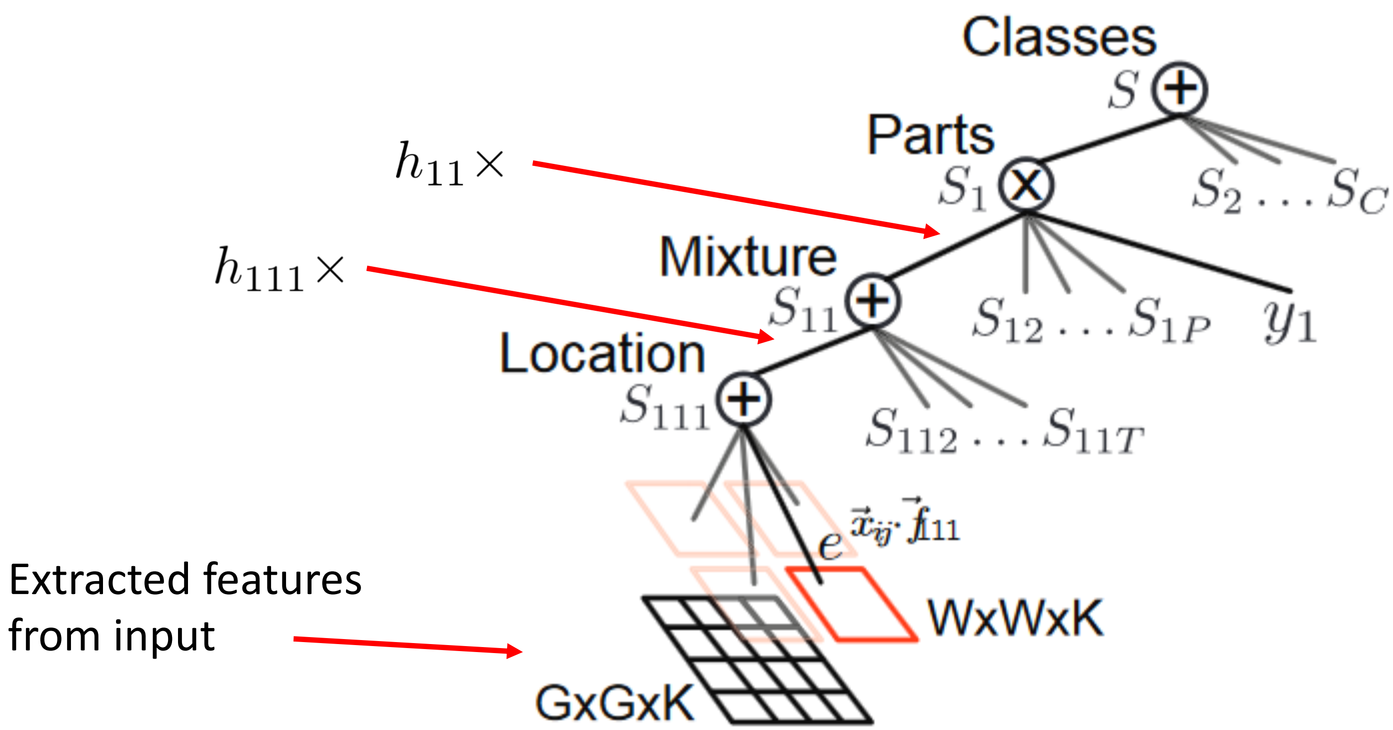
- **Idea**: Ignore the distribution of input $\mathbf{x}$. Only model the distribution of target variable y given $\mathbf{x}$

- Learn SPN which maximizes $\log \sum_{\mathbf{h}} p(\mathbf{y}, \mathbf{h}|\mathbf{x})$
  - Hidden variables are introduced to enhance the expressive power

Application of SPN: Discriminative Learning [Gens et al., 2012]

- Structure of SPN follows some convolutional structure
  - Each parts affects the weight of a target variable y_c
  - Each mixture extracts image features using convolutional filters

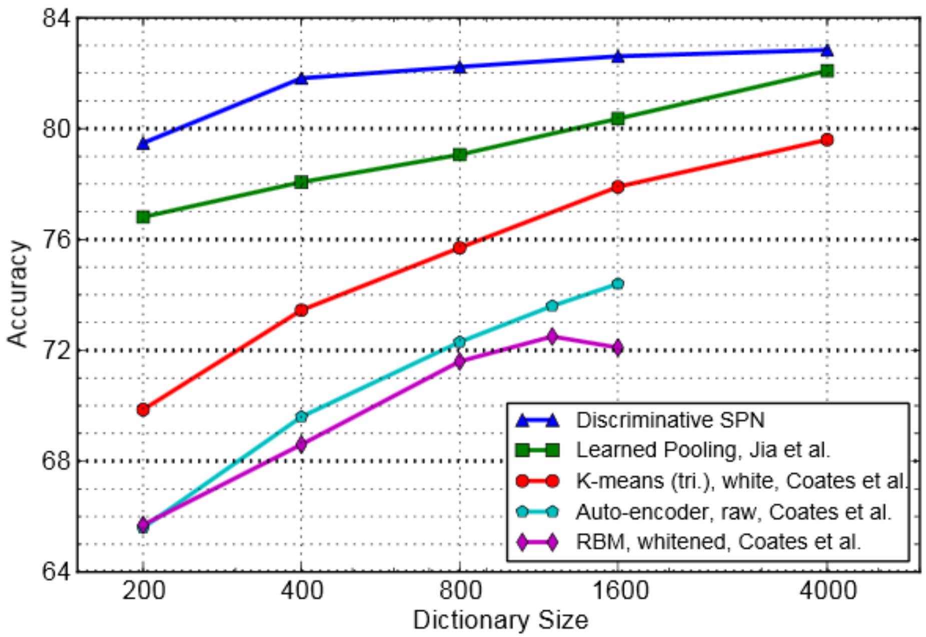
- Hidden variable determine whether discard the part of filter/location of images



Application of SPN: Discriminative Learning [Gens et al., 2012]

- Test accuracy on CIFAR-10 dataset
  - SPN estimation $\max_c P(y_c|\mathbf{x}, \mathbf{h})$
  - Input images are preprocessed by feature extraction algorithm [Coates et al., 2011]
  - Vary number of extracted features for quality measure

- SPN works well even with a small number of feature size

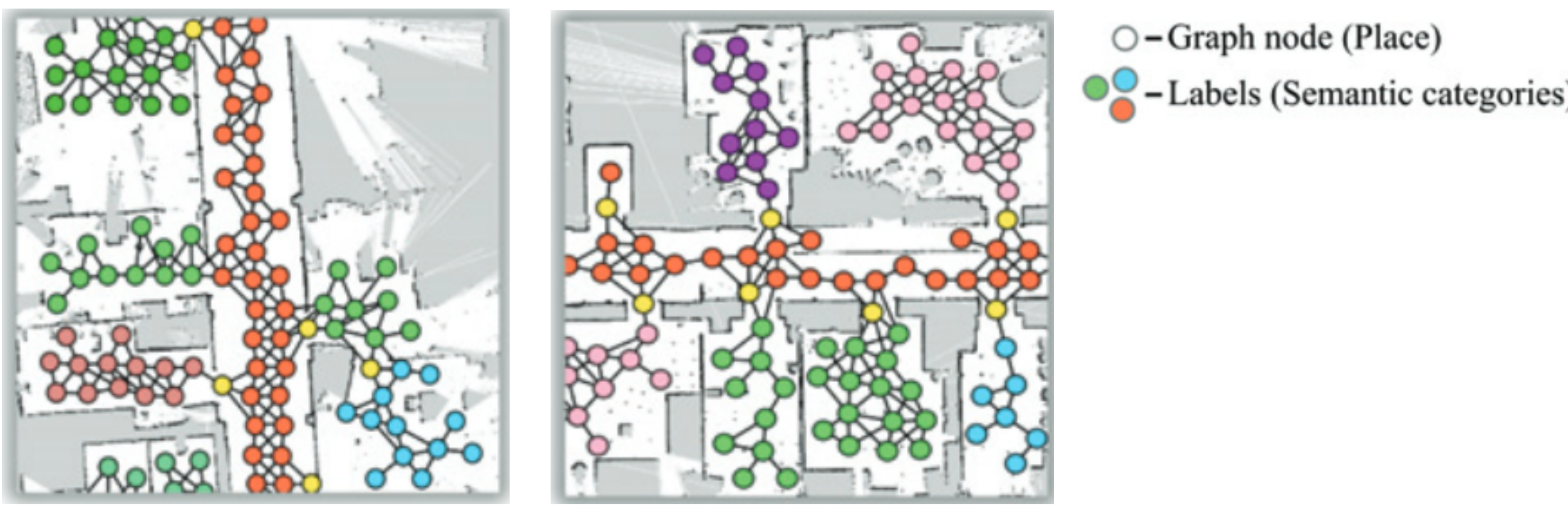


Method	Dictionary	Accuracy
Logistic Regression [24]		36.0%
SVM [5]		39.5%
SIFT [5]		65.6%
mcRBM [24]		68.3%
mcRBM-DBN [24]		71.0%
Convolutional RBM [10]		78.9%
K-means (Triangle) [10]	4000, 4x4 grid	79.6 %
HKDES [4]		80.0%
3-Layer Learned RF [12]	1600, 9x9 grid	82.0%
Learned Pooling [20]	6000, 4x4 grid	83.11%
Discriminative SPN	**400, 7x7 grid**	**83.96%**

Size of features

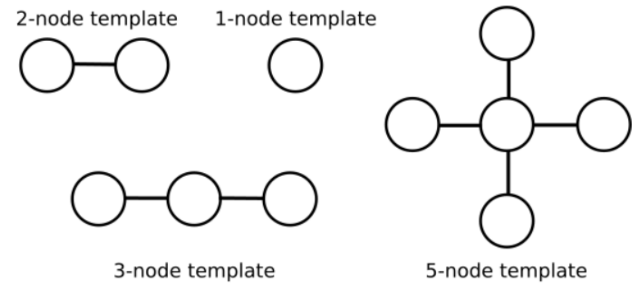
Application of SPN: Learning Graph Structured Data [Zheng et al, 2018]

- **Goal**: Learn the graph structured semantic map for predicting the labels of unvisited location
  - Each node represents a location of semantic place that a robot can visit
  - Each node associate with some local observation X_i (vision, sound, …) and its hidden label Y_i (office, corridor, …)
  - Each edge represents a spatial relation between nodes representing navigability

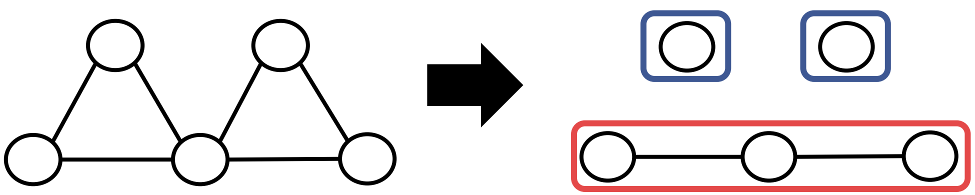


Application of SPN: Learning Graph Structured Data [Zheng et al, 2018]

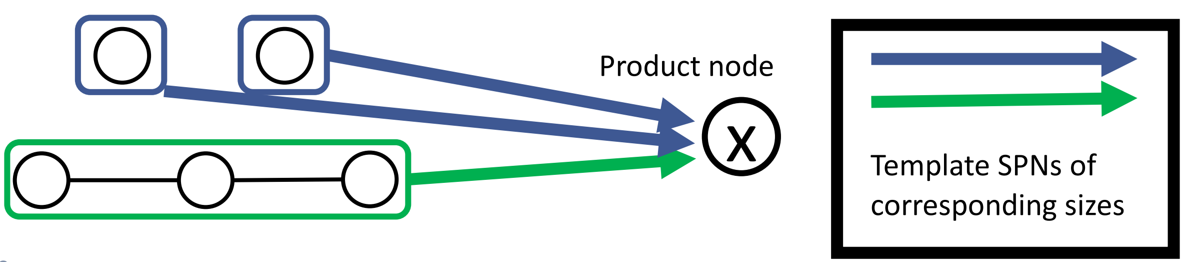
- Encoding samples of arbitrary size of graph into SPN is hard
  - Instead, learn SPN for small templates
  - Given training data, learn template SPNs for modelling distribution of X_i, Y_i from subgraphs



- Modelling distribution of test data
  1. Given a graph, decompose a graph into random disjoint templates



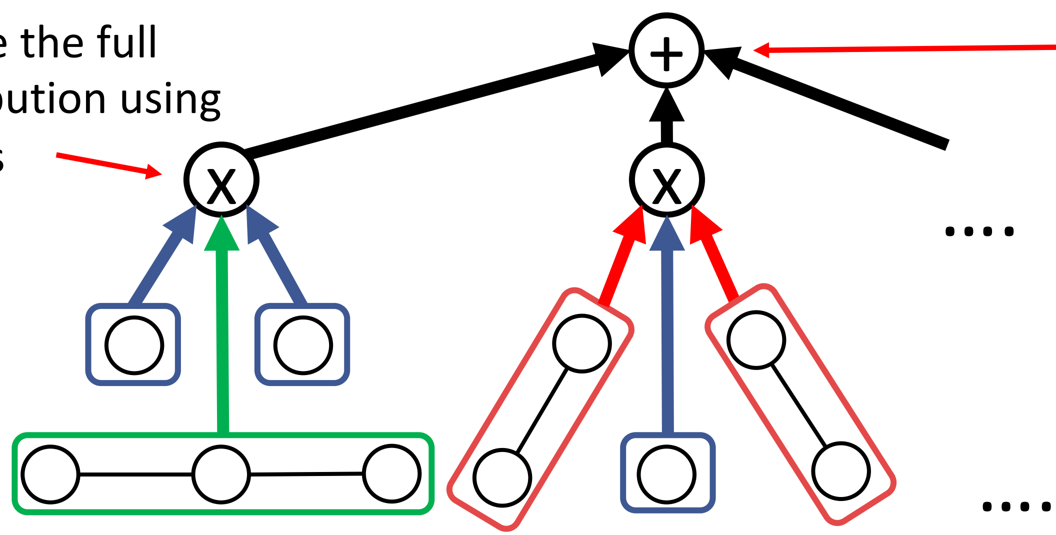
  2. Add a product node and trained template SPNs on the graph as its children



Application of SPN: Learning Graph Structured Data [Zheng et al, 2018]

- Modelling distribution of data
  1. Given a graph, decompose a graph into random disjoint templates
  2. Add a product node and trained template SPNs on the graph as its children
  3. Repeat 1-2 for fixed number
  4. For added product nodes, add a sum node as their parent. Use resulting SPN as a modelled distribution

Approximate the full graph distribution using smaller ones



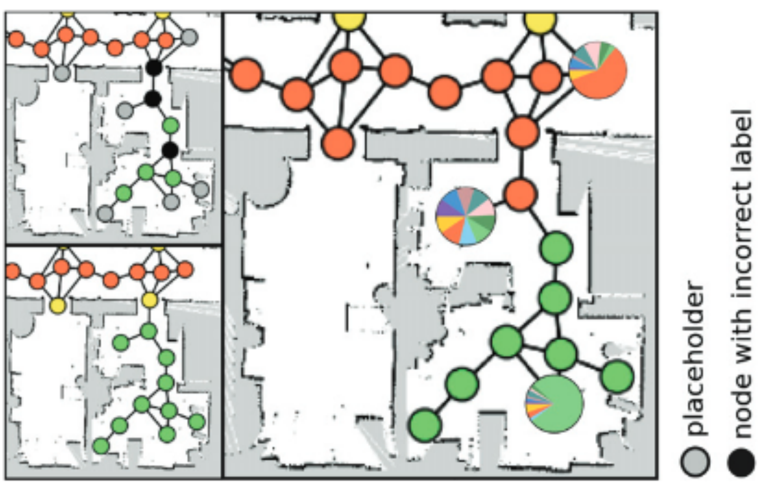
The full distribution is modelled as a mixture of several approximations

Application of SPN: Learning Graph Structured Data [Zheng et al, 2018]

- Experiment setup
  - Generate labels and observations under some ground truth distribution of X_i, Y_i
  - Add some noisy labels which are most likely to be correct except for the true label
  - Add placeholders which have no label and observation
  - Train SPN with above corrupted/missing data

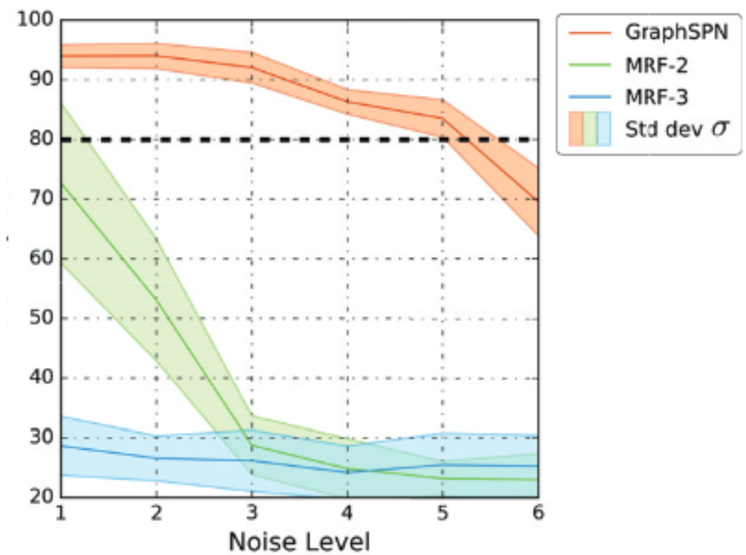
- Comparison with usual graphical models (Markov random field) of pairwise potentials and three nodes potentials

Corrupted training graph



Ground truth

Estimated result of SPN



Summary

- Generative models enables us to model the probability distribution of training samples

- BM, RBM, DBM have high expressive power but both inference and learning are intractable
    - They lead major breakthrough of deep learning before 2012 (AlexNet arrives)

- Sum-product network exhibits complex structure, tractable inference and efficient learning using back-propagation

References

- [Smolensky, 1986] “Chapter 6: Information Processing in Dynamical Systems: Foundations of Harmony Theory”, Parallel Distributed Processing 1986
  link : http://stanford.edu/~jlmcc/papers/PDP/Volume%201/Chap6_PDP86.pdf
- [Hinton, 2002] “Training Products of Experts by Minimizing Contrastive Divergence”, Neural Computation 2002
  link : http://www.cs.toronto.edu/~fritz/absps/tr00-004.pdf
- [Hinton et al., 2006] “A fast learning algorithm for deep belief nets”, Neural Computation 2006
  link : https://www.cs.toronto.edu/~hinton/absps/fastnc.pdf
- [Wainwright et al., 2008] “Graphical Models, Exponential Families, and Variational Inference“, FTML 2008
  link : https://people.eecs.berkeley.edu/~wainwrig/Papers/WaiJor08_FTML.pdf
- [Tieleman, 2008] “Training Restricted Boltzmann Machines using Approximations to the Likelihood Gradient”, ICML 2008
  link : https://arxiv.org/pdf/1705.11140.pdf
- [Salakhutdinov et al., 2009] “Deep Boltzmann Machines”, AISTATS 2009
  link : http://www.cs.toronto.edu/~fritz/absps/dbm.pdf
- [Salakhutdinov et al., 2010] “Efficient Learning of Deep Boltzmann Machines”, AISTATS 2010
  link : http://proceedings.mlr.press/v9/salakhutdinov10a/salakhutdinov10a.pdf
- [Poon et al., 2012], “Sum-Product Networks: A New Deep Architecture”, UAI 2012
  link : https://arxiv.org/ftp/arxiv/papers/1202/1202.3732.pdf
- [Gens et al., 2012] “Discriminative Learning of Sum-Product Networks”, NIPS 2012
  link : https://papers.nips.cc/paper/4516-discriminative-learning-of-sum-product-networks.pdf
- [Gens et al., 2013] “Learning the Structure of Sum-Product Networks”, ICML 2013
  link : http://proceedings.mlr.press/v28/gens13.pdf

References

- [Nguyen et al., 2017], "Supervised Restricted Boltzmann Machines", UAI 2017
  link : http://auai.org/uai2017/proceedings/papers/106.pdf
- [Pang et al., 2017], "A Joint Deep Boltzmann Machine (jDBM) Model for Person Identification Using Mobile Phone Data", IEEE Transactions on Multimedia
  link : https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7583729
- [Zheng et al., 2018], "Learning Graph-Structured Sum-Product Networks for Probabilistic Semantic Maps", AAAI 2018
  link : http://kaiyuzheng.me/documents/papers/zheng2018aaai.pdf