10-418 / 10-618 Machine Learning for Structured Data

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Variational Autoencoders
+
Deep Generative Models

Matt Gormley
Lecture 27
Dec. 4, 2019

Reminders

- **Final Exam**
  - Evening Exam
  - Thu, Dec. 5 at 6:30pm – 9:00pm

- **618 Final Poster:**
  - Submission: Tue, Dec. 10 at 11:59pm
  - Presentation: Wed, Dec. 11
    (time will be announced on Piazza)

FINAL EXAM LOGISTICS

Final Exam

- **Time / Location**
  - **Time:** Evening Exam
    Thu, Dec. 5 at 6:30pm – 9:00pm
  - **Room:** Doherty Hall A302
  - **Seats:** There will be **assigned seats**. Please arrive early to find yours.
  - Please watch Piazza carefully for announcements
- **Logistics**
  - Covered material: Lecture 1 – Lecture 26
    (not the new material in Lecture 27)
  - Format of questions:
    - Multiple choice
    - True / False (with justification)
    - Derivations
    - Short answers
    - Interpreting figures
    - Implementing algorithms on paper
  - No electronic devices
  - You are allowed to **bring** one 8½ x 11 sheet of notes (front and back)

Final Exam

- **Advice (for during the exam)**
  - Solve the easy problems first
    (e.g. multiple choice before derivations)
    - if a problem seems extremely complicated you're likely missing something
  - Don't leave any answer blank!
  - If you make an assumption, write it down
  - If you look at a question and don't know the answer:
    - we probably haven't told you the answer
    - but we've told you enough to work it out
    - imagine arguing for some answer and see if you like it

Final Exam

- **Exam Contents**
  - ~30% of material comes from topics covered **before** Midterm Exam
  - ~70% of material comes from topics covered **after** Midterm Exam

Topics from **before** Midterm Exam

- Search-Based Structured Prediction
  - Reductions to Binary Classification
  - Learning to Search
  - RNN-LMs
  - seq2seq models
- Graphical Model Representation
  - Directed GMs vs. Undirected GMs vs. Factor Graphs
  - Bayesian Networks vs. Markov Random Fields vs. Conditional Random Fields
- Graphical Model Learning
  - Fully observed Bayesian Network learning
  - Fully observed MRF learning
  - Fully observed CRF learning
  - Parameterization of a GM
  - Neural potential functions
- Exact Inference
  - Three inference problems:
(1) marginals
(2) partition function
(3) most probably assignment
  - Variable Elimination
  - Belief Propagation (sum-product and max-product)
  - MAP Inference via MILP

Topics from **after** Midterm Exam

- Learning for Structure Prediction
  - Structured Perceptron
  - Structured SVM
  - Neural network potentials
- Approximate MAP Inference
  - MAP Inference via MILP
  - MAP Inference via LP relaxation
- Approximate Inference by Sampling
  - Monte Carlo Methods
  - Gibbs Sampling
  - Metropolis-Hastings
  - Markov Chains and MCMC
- Approximate Inference by Optimization
  - Variational Inference
  - Mean Field Variational Inference
  - Coordinate Ascent V.I. (CAVI)
  - Variational EM
  - Variational Bayes
- Bayesian Nonparametrics
  - Dirichlet Process
  - DP Mixture Model
- ~~Deep Generative Models~~
  - ~~Variational Autoencoders~~

VARIATIONAL EM

Variational EM

Whiteboard

- Example: Unsupervised POS Tagging
- Variational Bayes
- Variational EM

Unsupervised POS Tagging

Bayesian Inference for HMMs

- **Task:** unsupervised POS tagging
- **Data:** 1 million words (i.e. unlabeled sentences) of WSJ text
- **Dictionary:** defines legal part-of-speech (POS) tags for each word type
- **Models:**
  - EM: standard HMM
  - VB: uncollapsed variational Bayesian HMM
  - Algo 1 (CVB): collapsed variational Bayesian HMM (strong indep. assumption)
  - Algo 2 (CVB): collapsed variational Bayesian HMM (weaker indep. assumption)
  - CGS: collapsed Gibbs Sampler for Bayesian HMM

Algo 1 mean field update:

$$q(z_t = k) \propto \frac{\mathbb{E}_{q(\mathbf{z}^{\neg t})}[C^{\neg t}_{k,w}] + \beta}{\mathbb{E}_{q(\mathbf{z}^{\neg t})}[C^{\neg t}_{k,\cdot}] + W\beta} \cdot \frac{\mathbb{E}_{q(\mathbf{z}^{\neg t})}[C^{\neg t}_{z_{t-1},k}] + \alpha}{\mathbb{E}_{q(\mathbf{z}^{\neg t})}[C^{\neg t}_{z_{t-1},\cdot}] + K\alpha} \cdot \frac{\mathbb{E}_{q(\mathbf{z}^{\neg t})}[C^{\neg t}_{k,z_{t+1}}] + \alpha + \mathbb{E}_{q(\mathbf{z}^{\neg t})}[\delta(z_{t-1} = k = z_{t+1})]}{\mathbb{E}_{q(\mathbf{z}^{\neg t})}[C^{\neg t}_{k,\cdot}] + K\alpha + \mathbb{E}_{q(\mathbf{z}^{\neg t})}[\delta(z_{t-1} = k)]}$$

CGS full conditional:

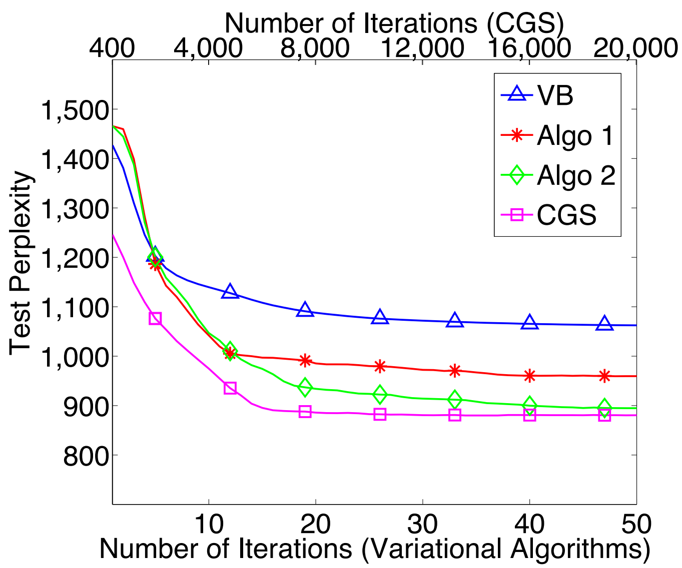
$$p(z_t = k|\mathbf{x}, \mathbf{z}^{\neg t}, \alpha, \beta) \propto \frac{C^{\neg t}_{k,w} + \beta}{C^{\neg t}_{k,\cdot} + W\beta} \cdot \frac{C^{\neg t}_{z_{t-1},k} + \alpha}{C^{\neg t}_{z_{t-1},\cdot} + K\alpha} \cdot \frac{C^{\neg t}_{k,z_{t+1}} + \alpha + \delta(z_{t-1} = k = z_{t+1})}{C^{\neg t}_{k,\cdot} + K\alpha + \delta(z_{t-1} = k)}$$

Figure from Wang & Blunsom (2013)

Unsupervised POS Tagging

Bayesian Inference for HMMs

- **Task:** unsupervised POS tagging
- **Data:** 1 million words (i.e. unlabeled sentences) of WSJ text
- **Dictionary:** defines legal part-of-speech (POS) tags for each word type
- **Models:**
  - EM: standard HMM
  - VB: uncollapsed variational Bayesian HMM
  - Algo 1 (CVB): collapsed variational Bayesian HMM (strong indep. assumption)
  - Algo 2 (CVB): collapsed variational Bayesian HMM (weaker indep. assumption)
  - CGS: collapsed Gibbs Sampler for Bayesian HMM



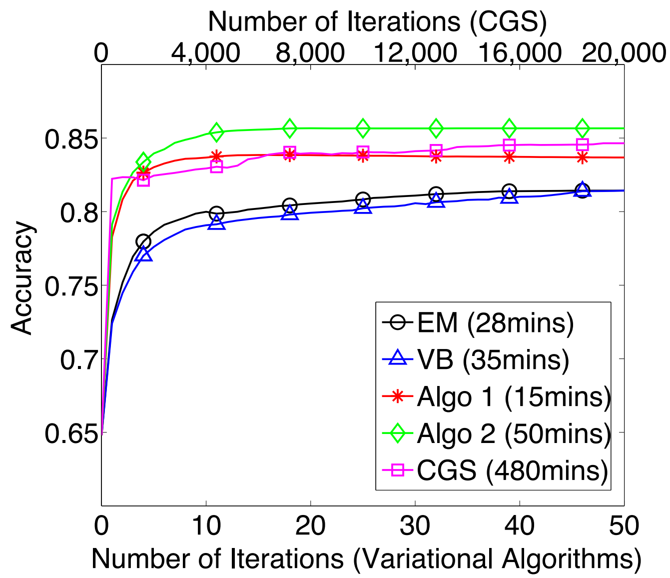


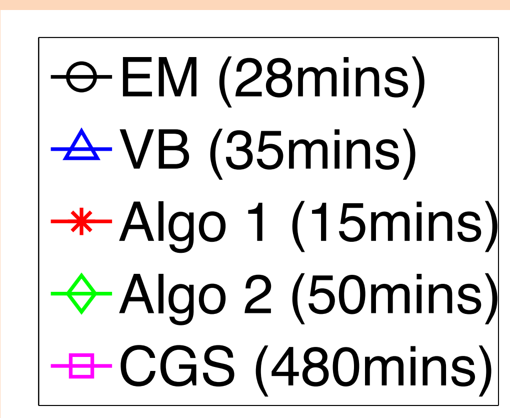
Figure from Wang & Blunsom (2013)

Unsupervised POS Tagging

Bayesian Inference for HMMs

- **Task:** unsupervised POS tagging
- **Data:** 1 million words (i.e. unlabeled sentences) of WSJ text
- **Dictionary:** defines legal part-of-speech (POS) tags for each word type
- **Models:**
  - EM: standard HMM
  - VB: uncollapsed variational Bayesian HMM
  - Algo 1 (CVB): collapsed variational Bayesian HMM (strong indep. assumption)
  - Algo 2 (CVB): collapsed variational Bayesian HMM (weaker indep. assumption)
  - CGS: collapsed Gibbs Sampler for Bayesian HMM

Speed:



- EM is slow b/c of log-space computations
- VB is slow b/c of digamma computations
- Algo 1 (CVB) is the fastest!
- Algo 2 (CVB) is slow b/c it computes dynamic parameters
- CGS: an order of magnitude slower than any deterministic algorithm

Figure from Wang & Blunsom (2013)

Stochastic Variational Bayesian HMM

- **Task:** Human Chromatin Segmentation
- **Goal:** unsupervised segmentation of the genome
- **Data:** from ENCODE, "250 million observations consisting of twelve assays carried out in the chronic myeloid leukemia cell line K562"
- **Metric:** "the false discovery rate (FDR) of predicting active promoter elements in the sequence"
- **Models:**
  - DBN HMM: dynamic Bayesian HMM trained with standard EM
  - SVIHMM: stochastic variational inference for a Bayesian HMM
- **Main Takeaway:**
  - the two models perform at similar levels of FDR
  - SVIHMM takes **one hour**
  - DBNHMM takes **days**

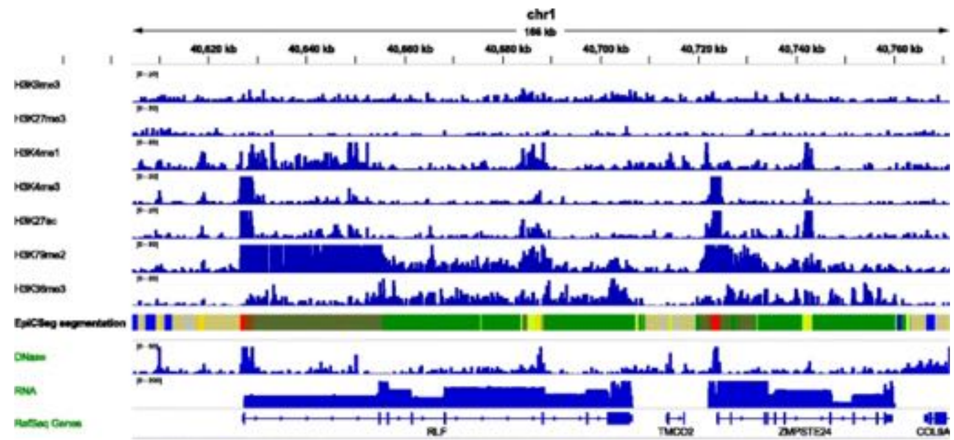


Figure from Foti et al. (2014)

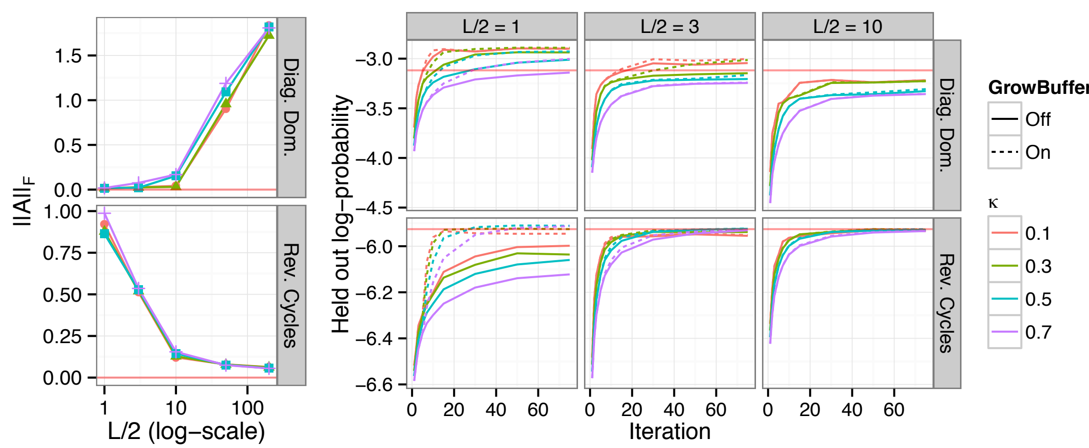


Figure from Mammana & Chung (2015)

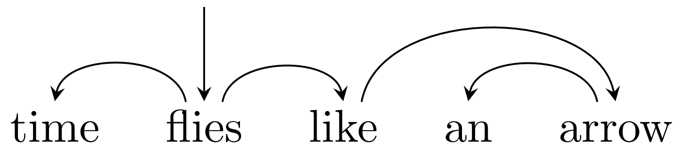
Grammar Induction

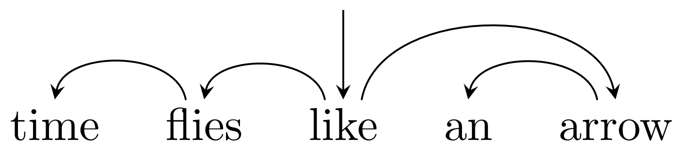
Question: Can maximizing (unsupervised) marginal likelihood produce useful results?

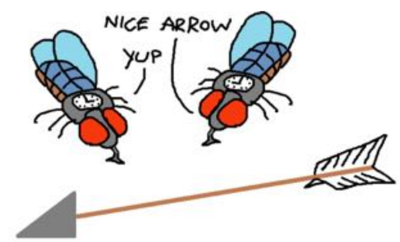
Answer: Let's look at an example…

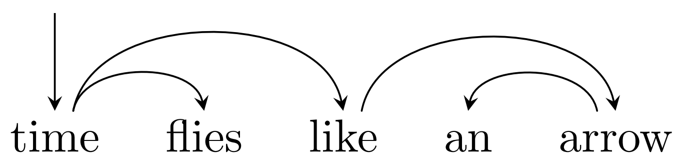
- **Babies** learn the syntax of their **native language** (e.g. English) just by **hearing** many sentences
- Can a **computer** similarly learn syntax of a **human language** just by looking at lots of example sentences?
  - This is the problem of Grammar Induction!
  - It's an unsupervised learning problem
  - We try to recover the **syntactic structure** for each sentence without any supervision

Grammar Induction

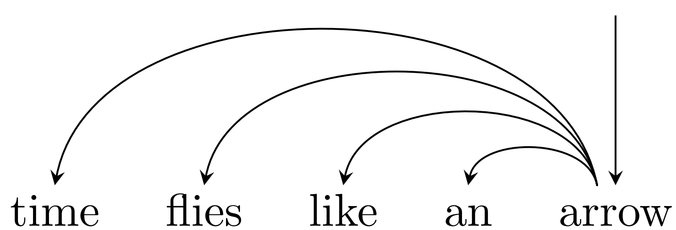








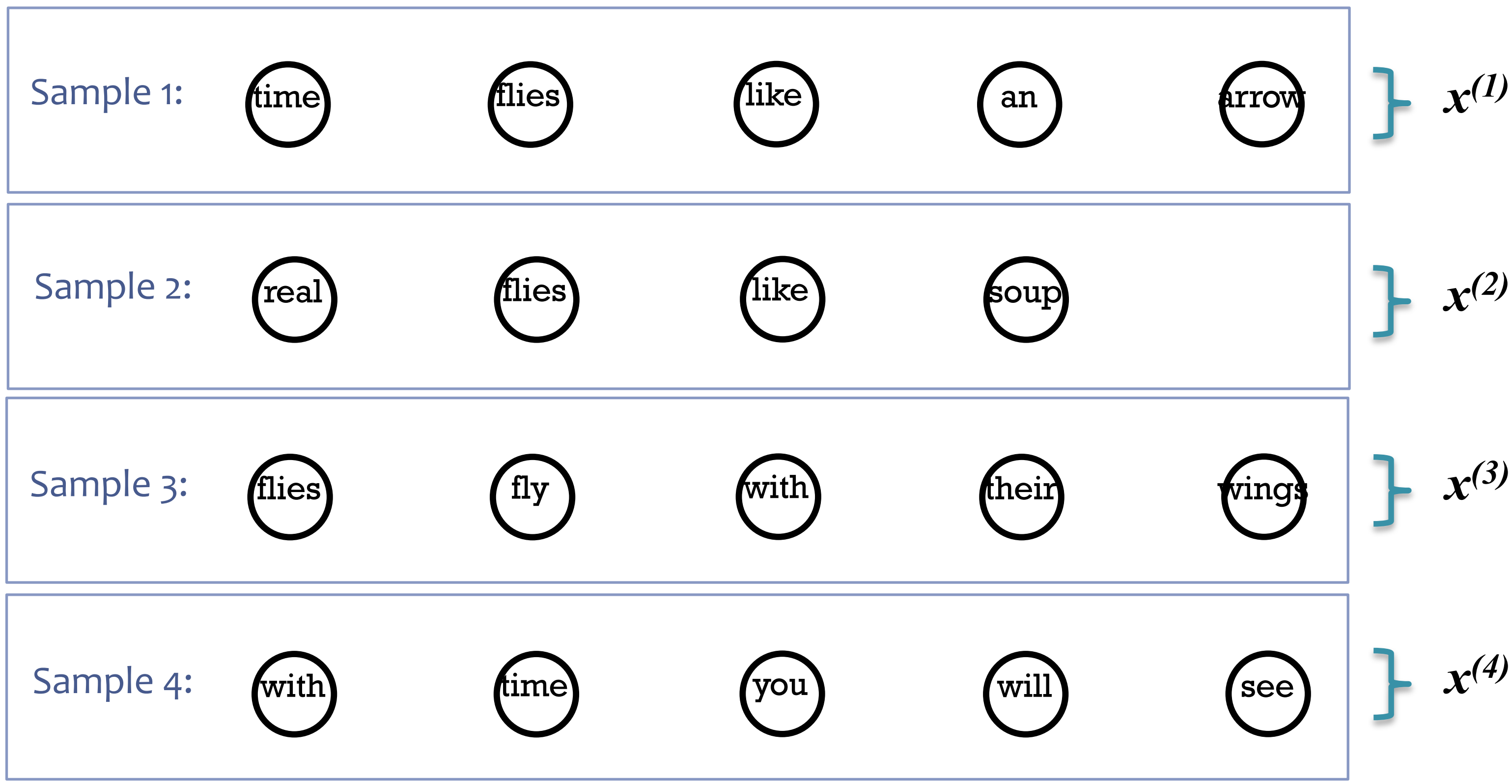
• • •



No semantic interpretation

Grammar Induction

Training Data: Sentences only, without parses



Test Data: Sentences **with** parses, so we can evaluate accuracy

Grammar Induction

Q: Does likelihood correlate with accuracy on a task we care about?

A: Yes, but there is still a wide range of accuracies for a particular likelihood value

Dependency Model with Valence (Klein & Manning, 2004)

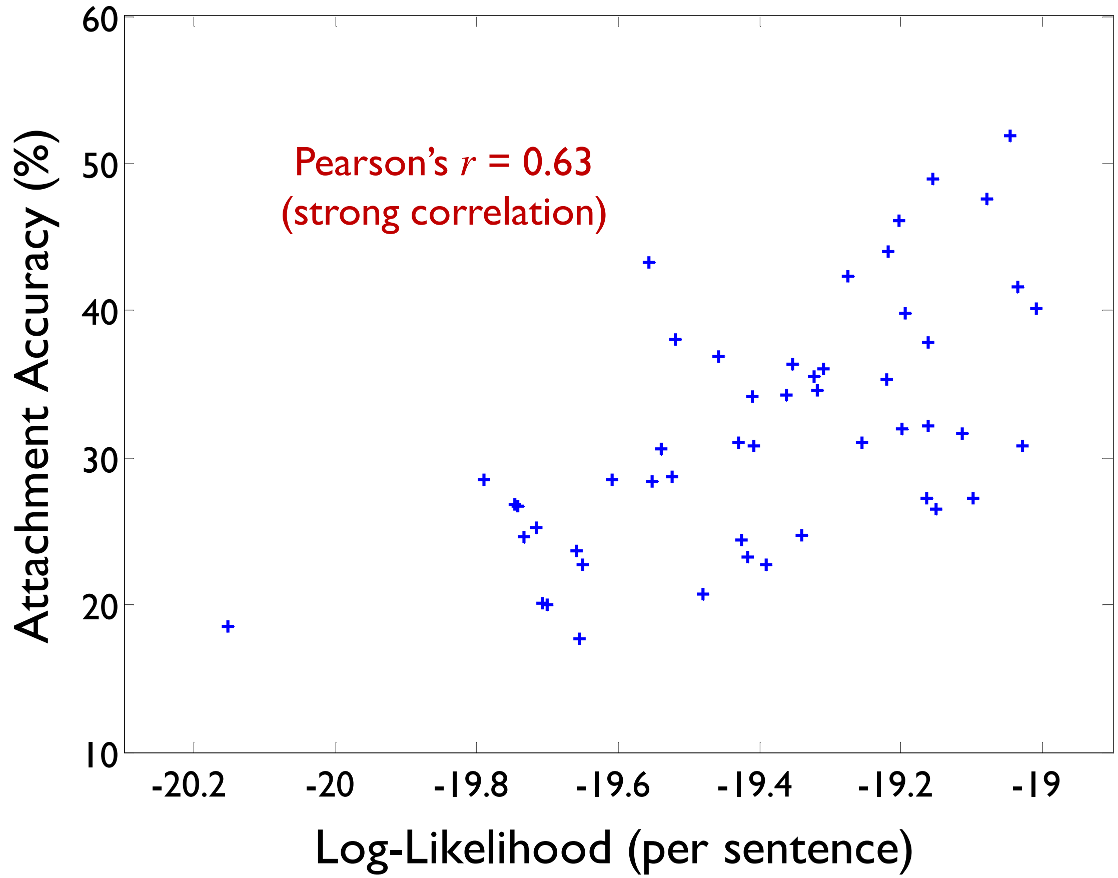
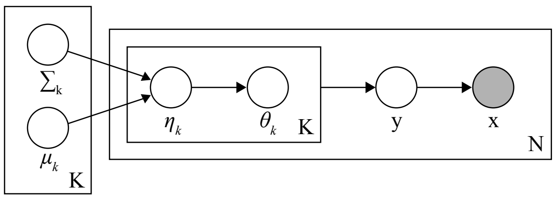


Figure from Gimpel & Smith (NAACL 2012) - slides

Grammar Induction

Graphical Model for Logistic Normal Probabilistic Grammar



y = syntactic parse

x = observed sentence

Settings:

EM Maximum likelihood estimate of $\boldsymbol{\theta}$ using the EM algorithm to optimize $p(\mathbf{x} \mid \boldsymbol{\theta})$ [14].

EM-MAP Maximum *a posteriori* estimate of $\boldsymbol{\theta}$ using the EM algorithm and a fixed symmetric Dirichlet prior with $\alpha > 1$ to optimize $p(\mathbf{x}, \boldsymbol{\theta} \mid \alpha)$. Tune α to maximize the likelihood of an unannotated development dataset, using grid search over $[1.1, 30]$.

VB-Dirichlet Use variational Bayes inference to estimate the posterior distribution $p(\boldsymbol{\theta} \mid \mathbf{x}, \alpha)$, which is a Dirichlet. Tune the symmetric Dirichlet prior's parameter α to maximize the likelihood of an unannotated development dataset, using grid search over $[0.0001, 30]$. Use the mean of the posterior Dirichlet as a point estimate for $\boldsymbol{\theta}$.

VB-EM-Dirichlet Use variational Bayes EM to optimize $p(\mathbf{x} \mid \boldsymbol{\alpha})$ with respect to $\boldsymbol{\alpha}$. Use the mean of the learned Dirichlet as a point estimate for $\boldsymbol{\theta}$ (similar to [5]).

VB-EM-Log-Normal Use variational Bayes EM to optimize $p(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Use the (exponentiated) mean of this Gaussian as a point estimate for $\boldsymbol{\theta}$.

Results:

	attachment accuracy (%)					
	Viterbi decoding			MBR decoding		
	$\lvert\mathbf{x}\rvert \le 10$	$\lvert\mathbf{x}\rvert \le 20$	all	$\lvert\mathbf{x}\rvert \le 10$	$\lvert\mathbf{x}\rvert \le 20$	all
Attach-Right	38.4	33.4	31.7	38.4	33.4	31.7
EM	45.8	39.1	34.2	46.1	39.9	35.9
EM-MAP, $\boldsymbol{\alpha} = 1.1$	45.9	39.5	34.9	46.2	40.6	36.7
VB-Dirichlet, $\boldsymbol{\alpha} = 0.25$	46.9	40.0	35.7	47.1	41.1	37.6
VB-EM-Dirichlet	45.9	39.4	34.9	46.1	40.6	36.9
VB-EM-Log-Normal, $\boldsymbol{\Sigma}_k^{(0)} = \mathbf{I}$	56.6	43.3	37.4	59.1	**45.9**	39.9
VB-EM-Log-Normal, families	**59.3**	**45.1**	**39.0**	**59.4**	**45.9**	**40.5**

Table 1: Attachment accuracy of different learning methods on unseen test data from the Penn Treebank of varying levels of difficulty imposed through a length filter. Attach-Right attaches each word to the word on its right and the last word to \$. EM and EM-MAP with a Dirichlet prior ($\alpha > 1$) are reproductions of earlier results [14, 18].

AUTOENCODERS

Idea #3: Unsupervised Pre-training

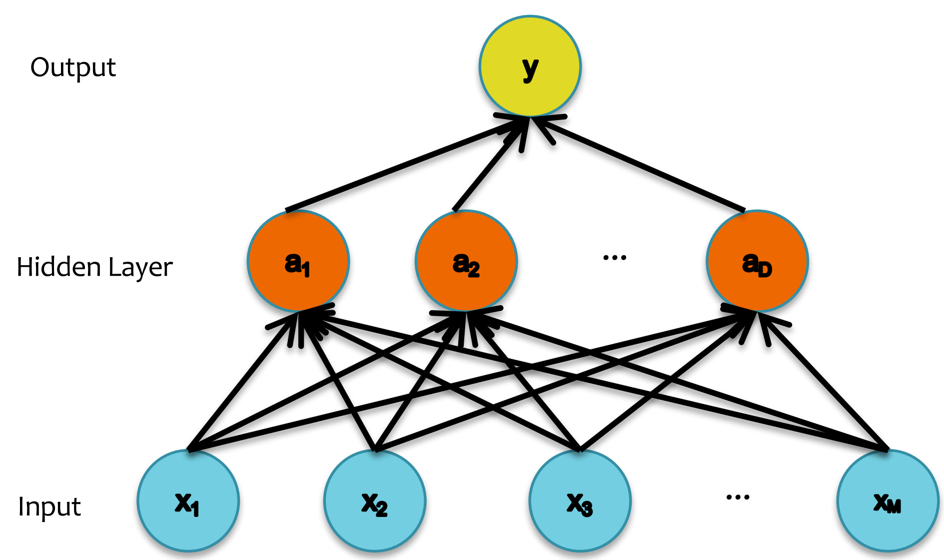
- **Idea: (Two Steps)**
  - Use supervised learning, but **pick a better starting point**
  - **Train each level** of the model in a **greedy** way

1. Unsupervised Pre-training
   - Use **unlabeled** data
   - Work bottom-up
     - Train hidden layer 1. Then fix its parameters.
     - Train hidden layer 2. Then fix its parameters.
     - …
     - Train hidden layer n. Then fix its parameters.
2. Supervised Fine-tuning
   - Use **labeled** data to train following “Idea #1”
   - Refine the features by backpropagation so that they become tuned to the end-task

The solution:
Unsupervised pre-training

Unsupervised pre-training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**

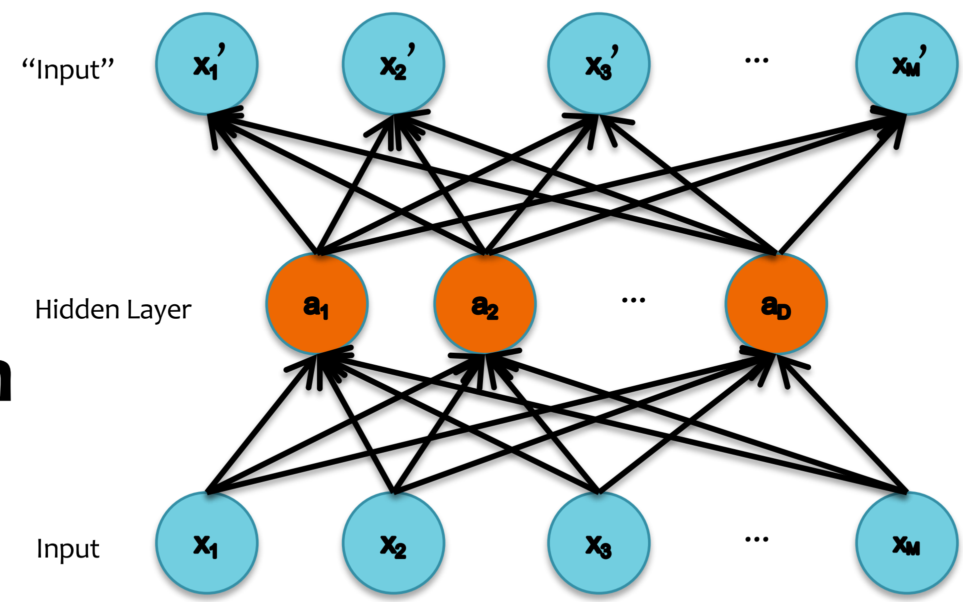


The solution:
Unsupervised pre-training

Unsupervised pre-training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**

This topology defines an Auto-encoder.



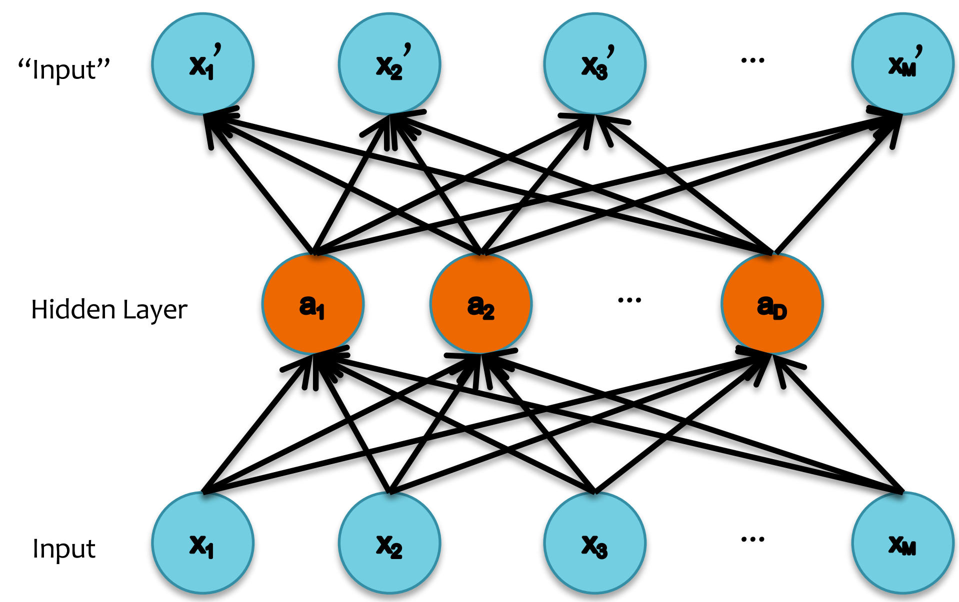
Auto-Encoders

Key idea: Encourage z to give small reconstruction error:

- x' is the *reconstruction* of x
- Loss = || x – DECODER(ENCODER(x)) ||^2
- Train with the same backpropagation algorithm for 2-layer Neural Networks with x_m as both input and output.

DECODER: x' = h(W'z)

ENCODER: z = h(Wx)

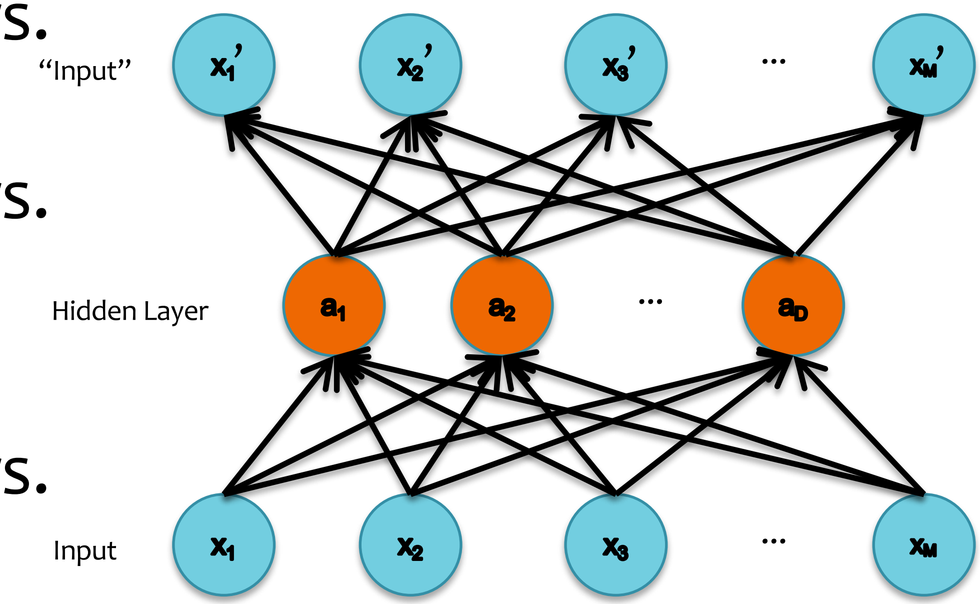


Slide adapted from Raman Arora

The solution:
Unsupervised pre-training

Unsupervised pre-training

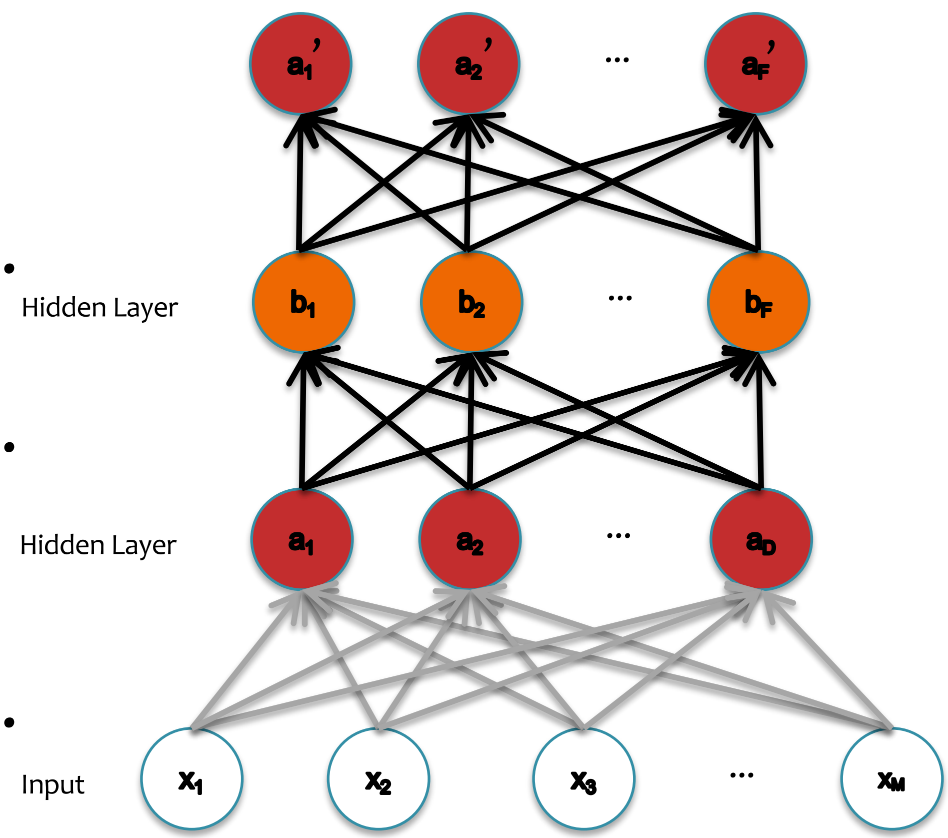
- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



The solution:
Unsupervised pre-training

Unsupervised pre-training

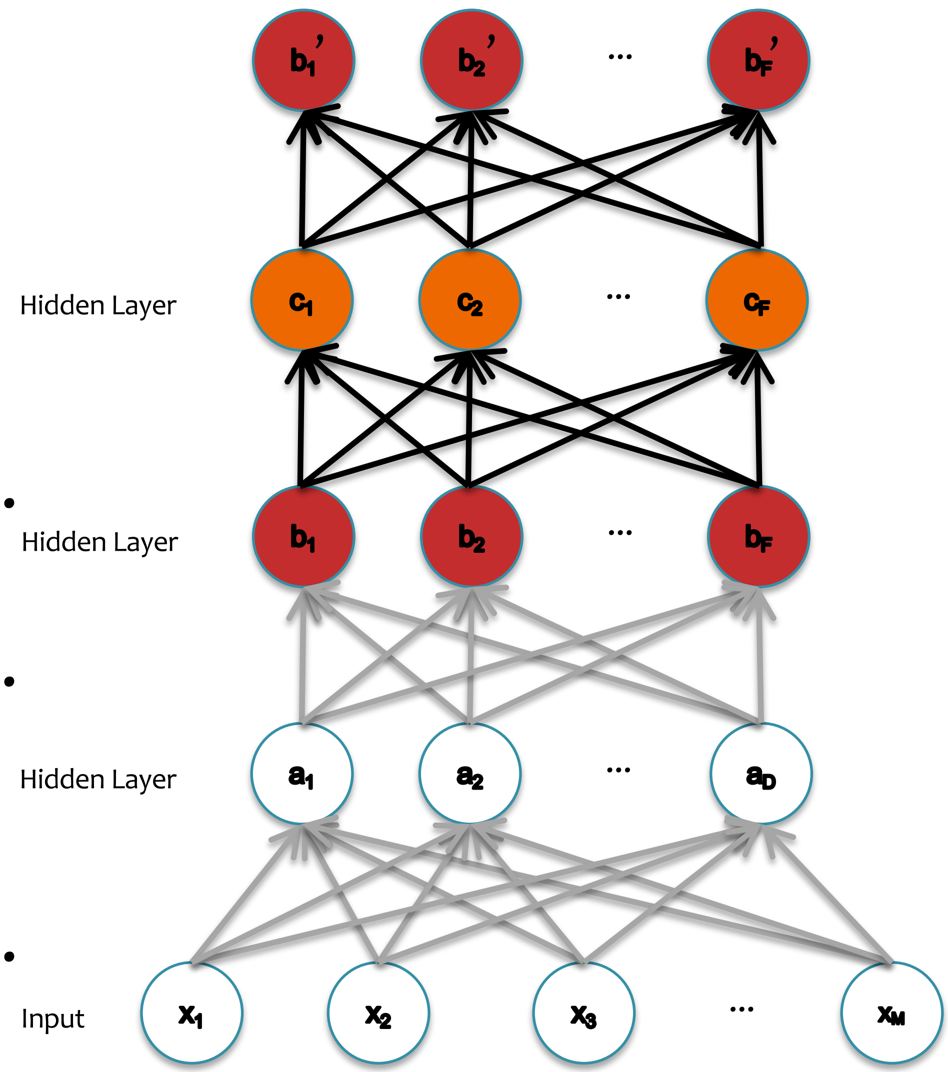
- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



The solution:
Unsupervised pre-training

Unsupervised pre-training

- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



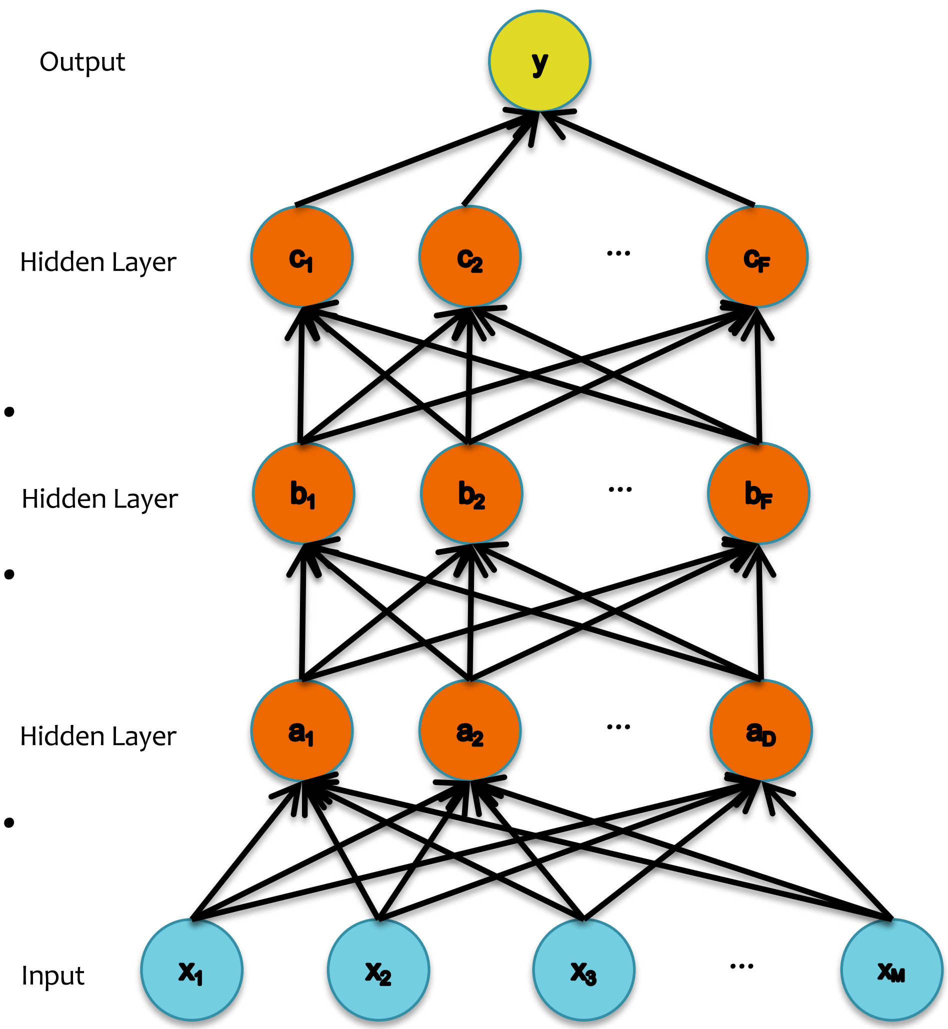
The solution:
Unsupervised pre-training

Unsupervised pre-training

- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.

Supervised fine-tuning

Backprop and update all parameters



Deep Network Training

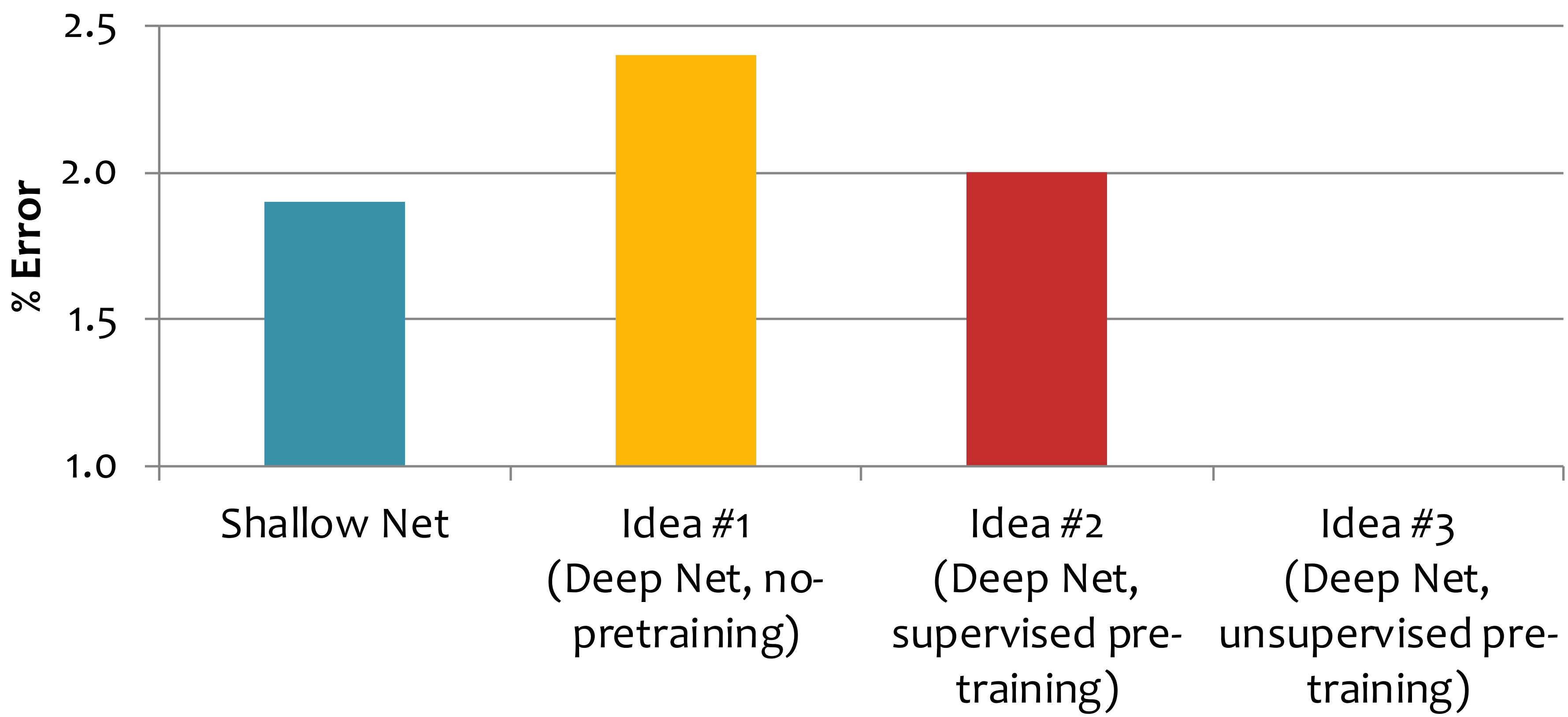
- **Idea #1:**
  1. Supervised fine-tuning only

- **Idea #2:**
  1. Supervised layer-wise pre-training
  2. Supervised fine-tuning

- **Idea #3:**
  1. Unsupervised layer-wise pre-training
  2. Supervised fine-tuning

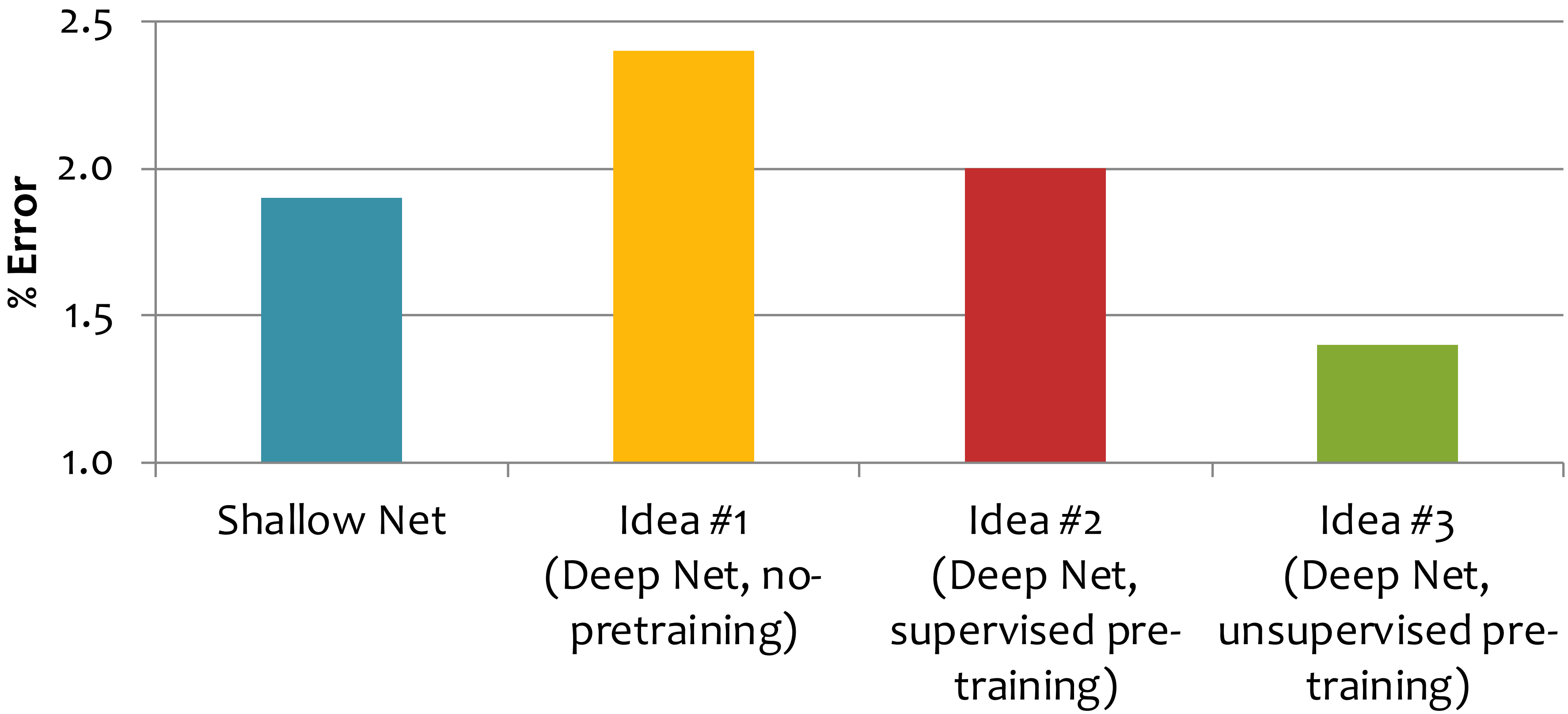
Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



VARIATIONAL AUTOENCODERS

Variational Autoencoders

Whiteboard

- Variational Autoencoder = VAE
- VAE as a Probability Model
- Parameterizing the VAE with Neural Nets
- Variational EM for VAEs

Reparameterization Trick

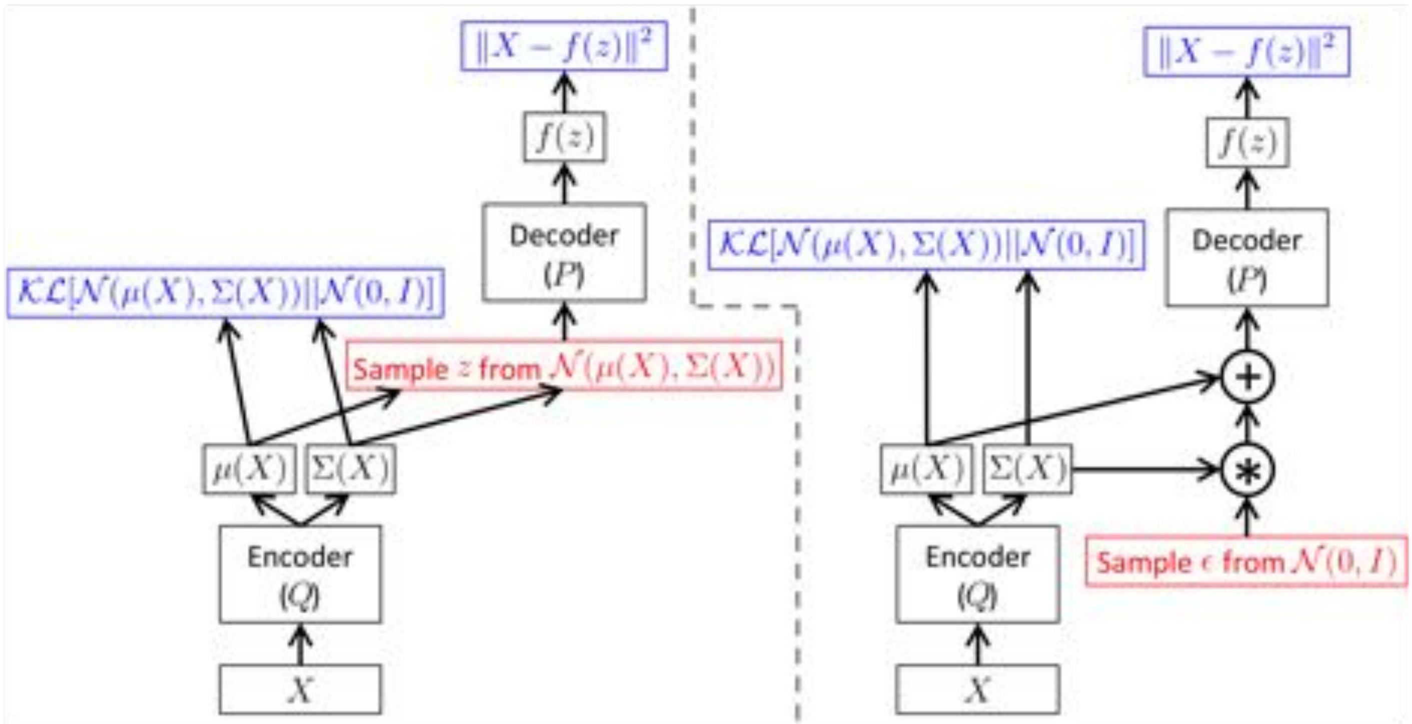


Figure 4: A training-time variational autoencoder implemented as a feed-forward neural network, where $P(X|z)$ is Gaussian. Left is without the "reparameterization trick", and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior of these networks is identical, but backpropagation can be applied only to the right network.

Figure from Doersch (2016)

Z Hu, Z YANG, R Salakhutdinov, E Xing,

"On Unifying Deep Generative Models", arxiv 1706.00550

(Slides in this section from Eric Xing)

UNIFYING GANS AND VAES

Generative Adversarial Nets (GANs):

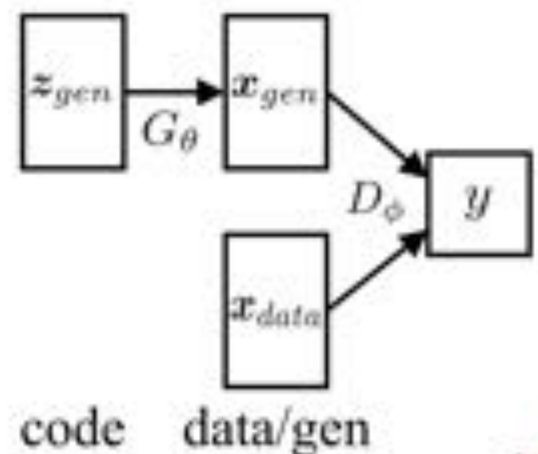
- Implicit distribution over $\boldsymbol{x} \sim p_\theta(\boldsymbol{x}|y)$

$$p_\theta(\boldsymbol{x}|y) = \begin{cases} p_{g_\theta}(\boldsymbol{x}) & y = 0 \\ p_{data}(\boldsymbol{x}) & y = 1. \end{cases}$$

(distribution of generated images)

(distribution of real images)

- $\boldsymbol{x} \sim p_{g_\theta}(\boldsymbol{x}) \Leftrightarrow \boldsymbol{x} = G_\theta(\boldsymbol{z}),\ \boldsymbol{z} \sim p(\boldsymbol{z}|y = 0)$
- $\boldsymbol{x} \sim p_{data}(\boldsymbol{x})$
  - the code space of $\boldsymbol{z}$ is degenerated
  - sample directly from data



A new formulation

- Rewrite GAN objectives in the "variational-EM" format
- Recap: conventional formulation:

$$\max_{\phi} \mathcal{L}_{\phi} = \mathbb{E}_{\boldsymbol{x}=G_{\theta}(\boldsymbol{z}), \boldsymbol{z}\sim p(\boldsymbol{z}|y=0)} \left[\log(1 - D_{\phi}(\boldsymbol{x}))\right] + \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})} \left[\log D_{\phi}(\boldsymbol{x})\right]$$

$$\begin{aligned}\max_{\boldsymbol{\theta}} \mathcal{L}_{\theta} &= \mathbb{E}_{\boldsymbol{x}=G_{\theta}(\boldsymbol{z}), \boldsymbol{z}\sim p(\boldsymbol{z}|y=0)} \left[\log D_{\phi}(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})} \left[\log(1 - D_{\phi}(\boldsymbol{x}))\right] \\ &= \mathbb{E}_{\boldsymbol{x}=G_{\theta}(\boldsymbol{z}), \boldsymbol{z}\sim p(\boldsymbol{z}|y=0)} \left[\log D_{\phi}(\boldsymbol{x})\right]\end{aligned}$$

- Rewrite in the new form
  - Implicit distribution over $\boldsymbol{x} \sim p_{\theta}(\boldsymbol{x}|y)$

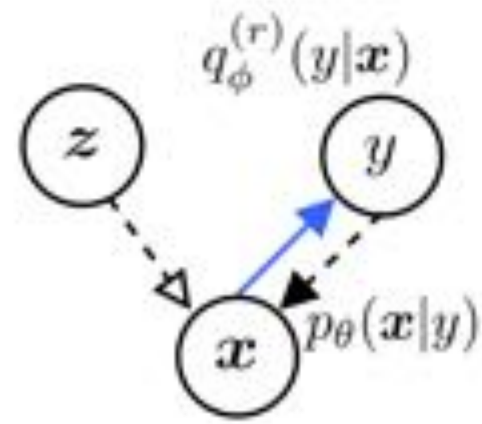
    $$\boldsymbol{x} = G_{\theta}(\boldsymbol{z}), \;\; \boldsymbol{z} \sim p(\boldsymbol{z}|y)$$

  - Discriminator distribution $q_{\phi}(y|\boldsymbol{x})$

    $$q^{r}_{\phi}(y|\boldsymbol{x}) = q_{\phi}(1 - y|\boldsymbol{x}) \quad \text{(reverse)}$$

$$\max_{\boldsymbol{\phi}} \mathcal{L}_{\phi} = \mathbb{E}_{p_{\theta}(\boldsymbol{x}|y)p(y)} \left[\log q_{\phi}(y|\boldsymbol{x})\right]$$

$$\max_{\boldsymbol{\theta}} \mathcal{L}_{\theta} = \mathbb{E}_{p_{\theta}(\boldsymbol{x}|y)p(y)} \left[\log q^{r}_{\phi}(y|\boldsymbol{x})\right]$$



GANs vs. Variational EM

- Interpret x as latent variables
- Interpret generation of x as performing inference over latent

In EVM, we minimize the following:

$$F(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = -\log p(\boldsymbol{x}) + KL\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \,||\, p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right)$$

→ KL (inference model | posterior)

Variational EM

- Objectives

$$\max_{\phi} \mathcal{L}_{\phi,\theta} = \mathbb{E}_{q_{\phi}(z|x)}[\log p_{\theta}(x|z)] + KL\left(q_{\phi}(z|x)||p(z)\right)$$
$$\max_{\theta} \mathcal{L}_{\phi,\theta} = \mathbb{E}_{q_{\phi}(z|x)}[\log p_{\theta}(x|z)] + KL\left(q_{\phi}(z|x)||p(z)\right)$$

  - Single objective for both θ and ϕ
  - Extra prior regularization by $p(z)$
- The reconstruction term: maximize the conditional log-likelihood of x with the generative distribution $p_{\theta}(x|z)$ conditioning on the latent code z inferred by $q_{\phi}(z|x)$

- $p_{\theta}(x|z)$ is the generative model
- $q_{\phi}(z|x)$ is the inference model

GAN

- Objectives

$$\max_{\phi} \mathcal{L}_{\phi} = \mathbb{E}_{p_{\theta}(\boldsymbol{x}|y)p(y)}\left[\log q_{\phi}(y|\boldsymbol{x})\right]$$
$$\max_{\theta} \mathcal{L}_{\theta} = \mathbb{E}_{p_{\theta}(\boldsymbol{x}|y)p(y)}\left[\log q_{\phi}^{r}(y|\boldsymbol{x})\right]$$

  - Two objectives
  - Have global optimal state in the game theoretic view
- The objectives: maximize the conditional log-likelihood of y (or $1-y$) with the distribution $q_{\phi}(y|x)$ conditioning on data/generation x inferred by $p_{\theta}(x|y)$

- Interpret $q_{\phi}(y|x)$ as the generative model
- Interpret $p_{\theta}(x|y)$ as the inference model

GANs vs VAEs side by side

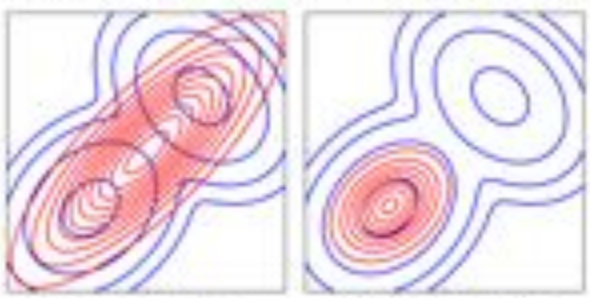
$p_\theta(\boldsymbol{z}, y|\boldsymbol{x}) \propto p_\theta(\boldsymbol{x}|\boldsymbol{z}, y)p(\boldsymbol{z}|y)p(y)$

	GANs (InfoGAN)	VAEs
Generative distribution	$p_\theta(\boldsymbol{x}\|y) = \begin{cases} p_{g_\theta}(\boldsymbol{x}) & y = 0 \\ p_{data}(\boldsymbol{x}) & y = 1. \end{cases}$	$p_\theta(\boldsymbol{x}\|\boldsymbol{z}, y) = \begin{cases} p_\theta(\boldsymbol{x}\|\boldsymbol{z}) & y = 0 \\ p_{data}(\boldsymbol{x}) & y = 1. \end{cases}$
Discriminator distribution	$q_\phi(y\|\boldsymbol{x})$	$q_*(y\|\boldsymbol{x})$, perfect, degenerated
$\boldsymbol{z}$-inference model	$q_\eta(\boldsymbol{z}\|\boldsymbol{x}, y)$ of InfoGAN	$q_\eta(\boldsymbol{z}\|\boldsymbol{x}, y)$
KLD to minimize	$\min_\theta \mathrm{KL}\left(p_\theta(\boldsymbol{x}\|y) \;\|\|\; q^r(\boldsymbol{x}\|\boldsymbol{z}, y)\right)$ $\sim \min_\theta \mathrm{KL}(P_\theta \;\|\|\; Q)$	$\min_\theta \mathrm{KL}\left(q_\eta(\boldsymbol{z}\|\boldsymbol{x}, y)q_*^r(y\|\boldsymbol{x}) \;\|\|\; p_\theta(\boldsymbol{z}, y\|\boldsymbol{x})\right)$ $\sim \min_\theta \mathrm{KL}(Q \;\|\|\; P_\theta)$

GANs vs VAEs side by side

	GANs (InfoGAN)	VAEs
KLD to minimize	$\min_\theta \mathrm{KL}\left(p_\theta(\boldsymbol{x}\mid y) \mid\mid q^r(\boldsymbol{x}\mid \boldsymbol{z}, y)\right)$ $\sim \min_\theta \mathrm{KL}(P_\theta \mid\mid Q)$	$\min_\theta \mathrm{KL}(q_\eta(\boldsymbol{z}\mid \boldsymbol{x}, y) q_*^r(y\mid \boldsymbol{x}) \mid\mid p_\theta(\boldsymbol{z}, y\mid \boldsymbol{x}))$ $\sim \min_\theta \mathrm{KL}(Q \mid\mid P_\theta)$

- Asymmetry of KLDs inspires combination of GANs and VAEs
  - GANs: $\min_\theta \mathrm{KL}(P_\theta \mid\mid Q)$ tends to missing mode
  - VAEs: $\min_\theta \mathrm{KL}(Q \mid\mid P_\theta)$ tends to cover regions with small values of p_{data}



Mode covering Mode missing

[Figure courtesy: PRML]

DEEP GENERATIVE MODELS

Question:

How does this relate to Graphical Models?

The first "Deep Learning" papers in 2006 were innovations in training a particular flavor of Belief Network.

Those models happen to also be neural nets.

DBNs

MNIST Digit **Generation**

- This section: Suppose you want to build a **generative** model capable of explaining handwritten digits
- Goal:
  - To have a model p(x) **from which we can sample digits** that look realistic
  - Learn **unsupervised** hidden representation of an image

Figure from (Hinton et al., 2006)

Sigmoid Belief Networks

- Directed graphical model of binary variables in fully connected layers
- Only bottom layer is observed
- Specific parameterization of the conditional probabilities:

$$p(x_i|\text{parents}(x_i)) = \frac{1}{1 + \exp(-\sum_j w_{ij} x_j)}$$

Note: this is a GM diagram not a NN!

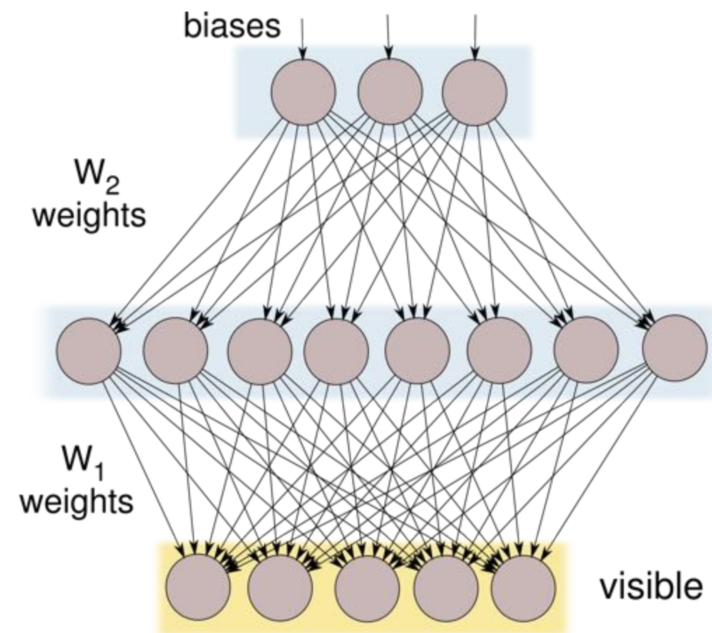


Figure from Marcus Frean, MLSS Tutorial 2010

DBNs

Contrastive Divergence Training

Contrastive Divergence is a general tool for learning a generative distribution, where the derivative of the log partition function is intractable to compute.

$$\begin{aligned}
\log L &= \log P(\mathcal{D}) \\
&= \sum_{\mathbf{v}\in\mathcal{D}} \log P(\mathbf{v}) \\
&= \sum_{\mathbf{v}\in\mathcal{D}} \log \left(P^\star(\mathbf{v})/Z\right) \\
&= \sum_{\mathbf{v}\in\mathcal{D}} \left(\log P^\star(\mathbf{v}) \quad - \quad \log Z\right) \\
&\propto \quad \frac{1}{N}\sum_{\mathbf{v}\in\mathcal{D}} \log P^\star(\mathbf{v}) \qquad - \quad \log Z
\end{aligned}$$

Slide from Marcus Frean, MLSS Tutorial 2010

Contrastive Divergence Training

$$\frac{\partial}{\partial w} \log L \quad \propto$$

$$\underbrace{\frac{1}{N}\sum_{\mathbf{v}\in\mathcal{D}}}_{\text{data}} \underbrace{\sum_{\mathbf{h}} P(\mathbf{h} \mid \mathbf{v})}_{\text{av. over posterior}} \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x}) \;-\; \underbrace{\sum_{\mathbf{v},\mathbf{h}} P(\mathbf{v},\mathbf{h})}_{\text{av. over joint}} \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x})$$

Contrastive Divergence estimates the second term with a Monte Carlo estimate from 1-step of a Gibbs sampler!

Both terms involve averaging over $\frac{\partial}{\partial w} \log P^{\star}(\mathbf{x})$.

Another way to write it:

$$\left\langle \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x}) \right\rangle_{\mathbf{v}\in\mathcal{D},\;\; \mathbf{h}\sim P(\mathbf{h}|\mathbf{v})} \quad - \quad \left\langle \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x}) \right\rangle_{\mathbf{x}\sim P(\mathbf{x})}$$

clamped / wake phase

$\uparrow\uparrow\uparrow$ conditioned hypotheses

unclamped / sleep / free phase

$\downarrow\downarrow\downarrow$ random fantasies

Slide from Marcus Frean, MLSS Tutorial 2010

Contrastive Divergence Training

For a belief net the joint is automatically normalised: Z is a constant 1

- 2nd term is zero!
- for the weight w_{ij} from j into i, the gradient $\frac{\partial \log L}{\partial w_{ij}} = (x_i - p_i)x_j$
- stochastic gradient ascent:

$$\Delta w_{ij} \quad \propto \quad \underbrace{(x_i - p_i)x_j}_{\text{the "delta rule"}}$$

So this is a stochastic version of the EM algorithm, that you may have heard of. We iterate the following two steps:

E step: get samples from the posterior

M step: apply the learning rule that makes them more likely

Sigmoid Belief Networks

- In practice, applying CD to a Deep Sigmoid Belief Nets fails
- Sampling from the posterior of many (deep) hidden layers doesn't approach the equilibrium distribution quickly enough

Note: this is a GM diagram not a NN!

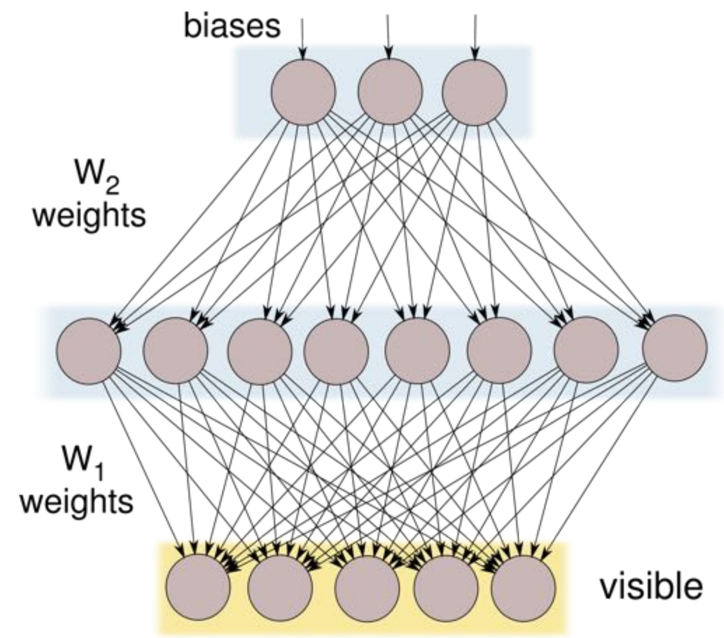


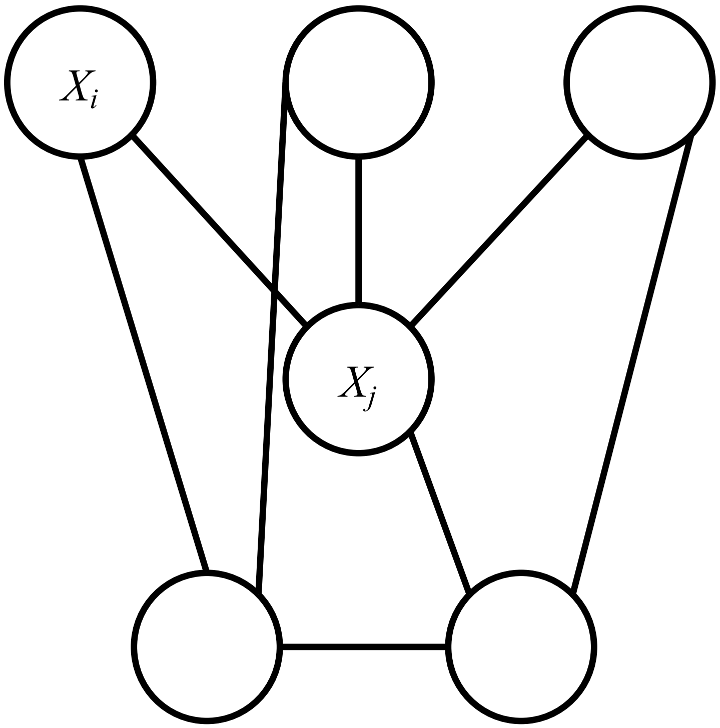
Figure from Marcus Frean, MLSS Tutorial 2010

Boltzman Machines

- Undirected graphical model of binary variables with pairwise potentials
- Parameterization of the potentials:

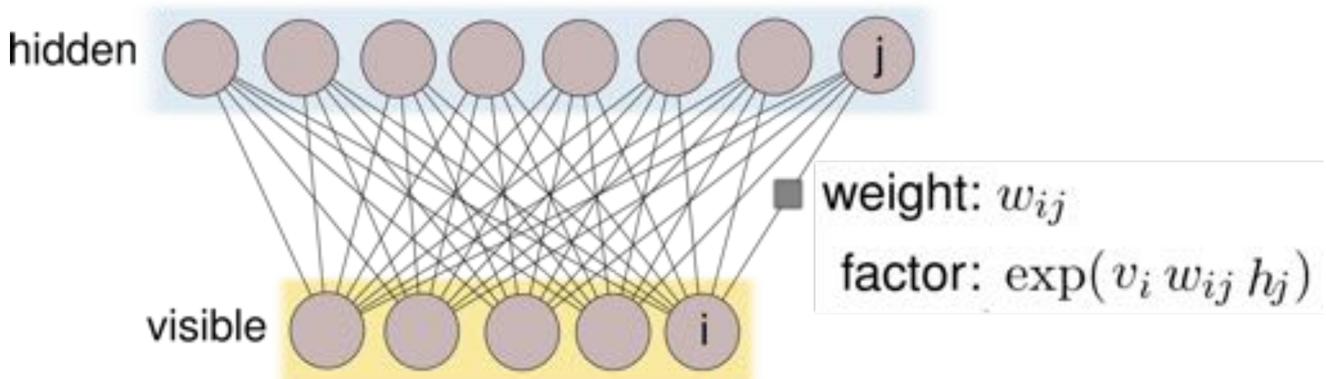
$$\psi_{ij}(x_i, x_j) = \exp(x_i W_{ij} x_j)$$

(In English: higher value of parameter W_{ij} leads to higher correlation between X_i and X_j on value 1)



Restricted Boltzman Machines

- Assume visible units are one layer, and hidden units are another.
- Throw out all the connections within each layer.



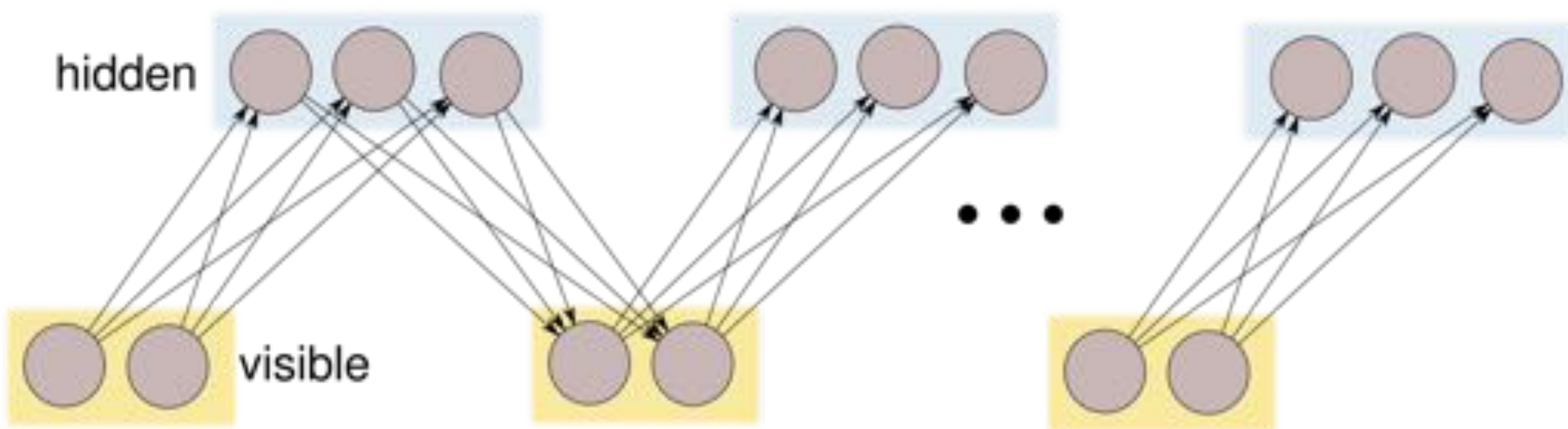
- $h_j \perp\!\!\!\perp h_k \mid \mathbf{v}$
- the posterior $P(\mathbf{h} \mid \mathbf{v})$ factors
  c.f. in a belief net, the *prior* $P(\mathbf{h})$ factors
- no explaining away

Slide from Marcus Frean, MLSS Tutorial 2010

Restricted Boltzman Machines

Alternating Gibbs sampling

Since none of the units within a layer are interconnected, we can do Gibbs sampling by updating the whole layer at a time.

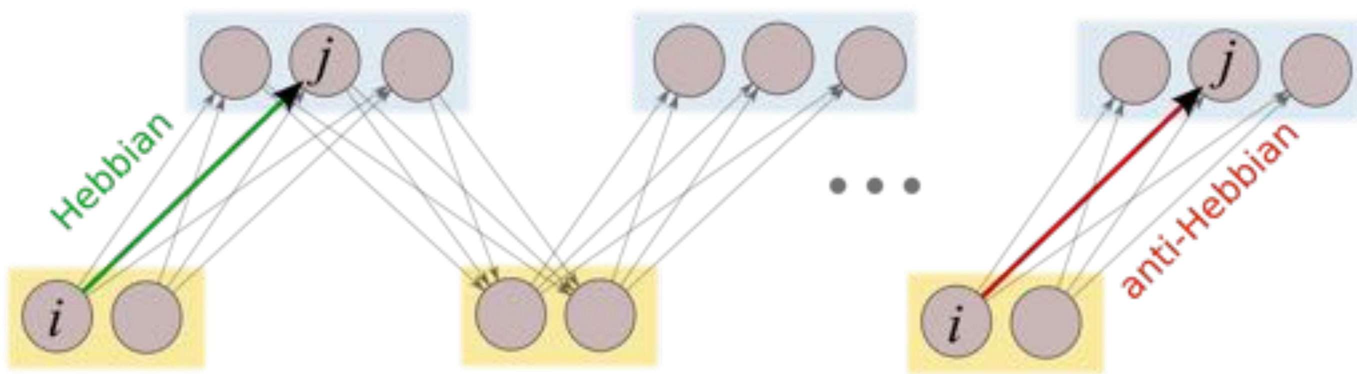


(with time running from left $\longrightarrow$ right)

Slide from Marcus Frean, MLSS Tutorial 2010

Restricted Boltzman Machines

learning in an RBM



Repeat for all data:

1. start with a training vector on the visible units
2. then alternate between updating all the hidden units in parallel and updating all the visible units in parallel

$$\Delta w_{ij} = \eta \left[\langle v_i \, h_j \rangle^0 - \langle v_i \, h_j \rangle^\infty \right]$$

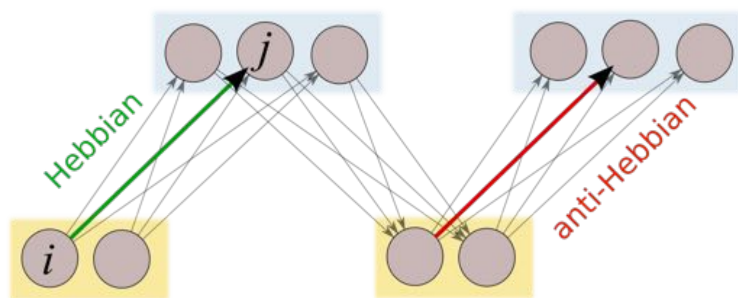
restricted connectivity is trick #1:

it saves waiting for equilibrium in the clamped phase.

Slide from Marcus Frean, MLSS Tutorial 2010

Restricted Boltzman Machines

trick # 2: curtail the Markov chain during learning



Repeat for all data:

1. start with a training vector on the visible units
2. update all the hidden units in parallel
3. update all the visible units in parallel to get a "reconstruction"
4. update the hidden units again

$$\Delta w_{ij} = \eta \left[\langle v_i \, h_j \rangle^0 - \langle v_i \, h_j \rangle^1 \right]$$

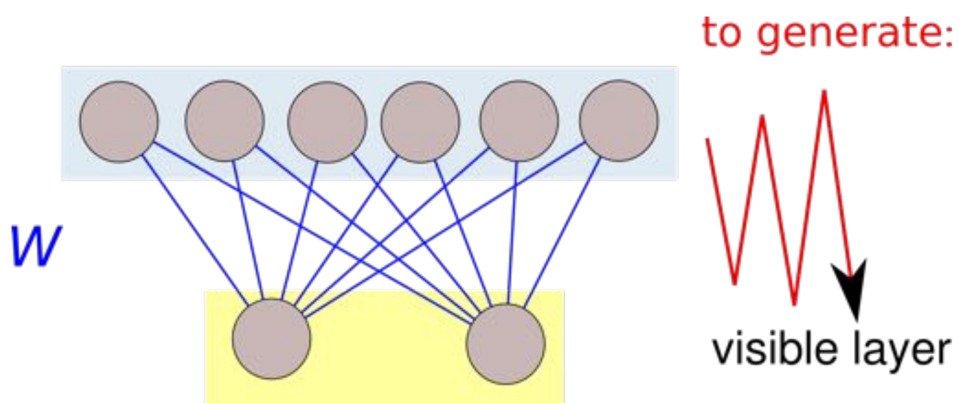
This is not following the correct gradient, but works well in practice. Geoff Hinton calls it learning by "contrastive divergence".

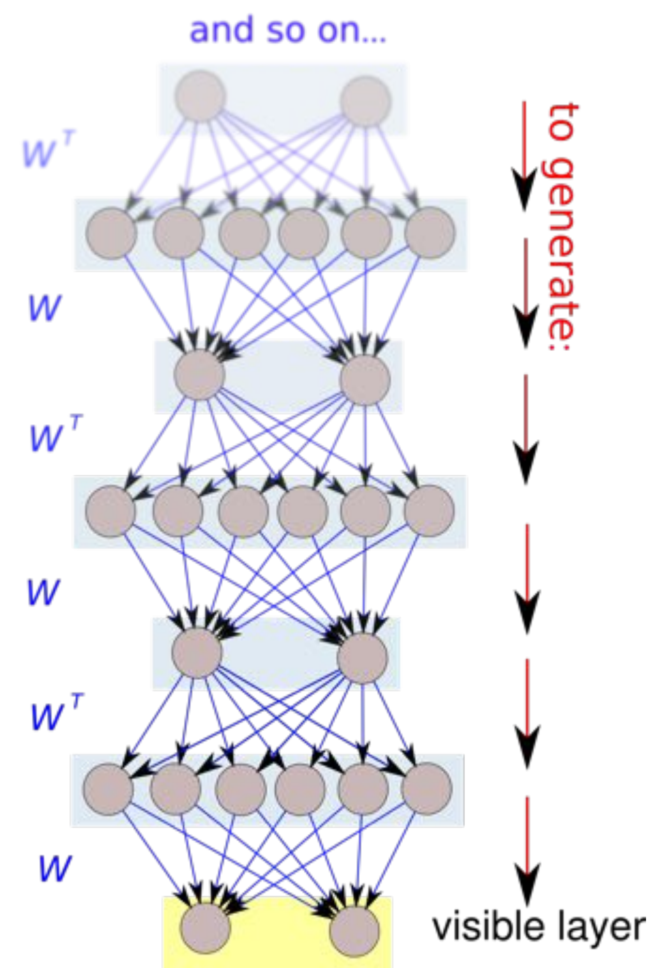
Slide from Marcus Frean, MLSS Tutorial 2010

DBNs

Deep Belief Networks (DBNs)

RBMs are equivalent to infinitely deep belief networks



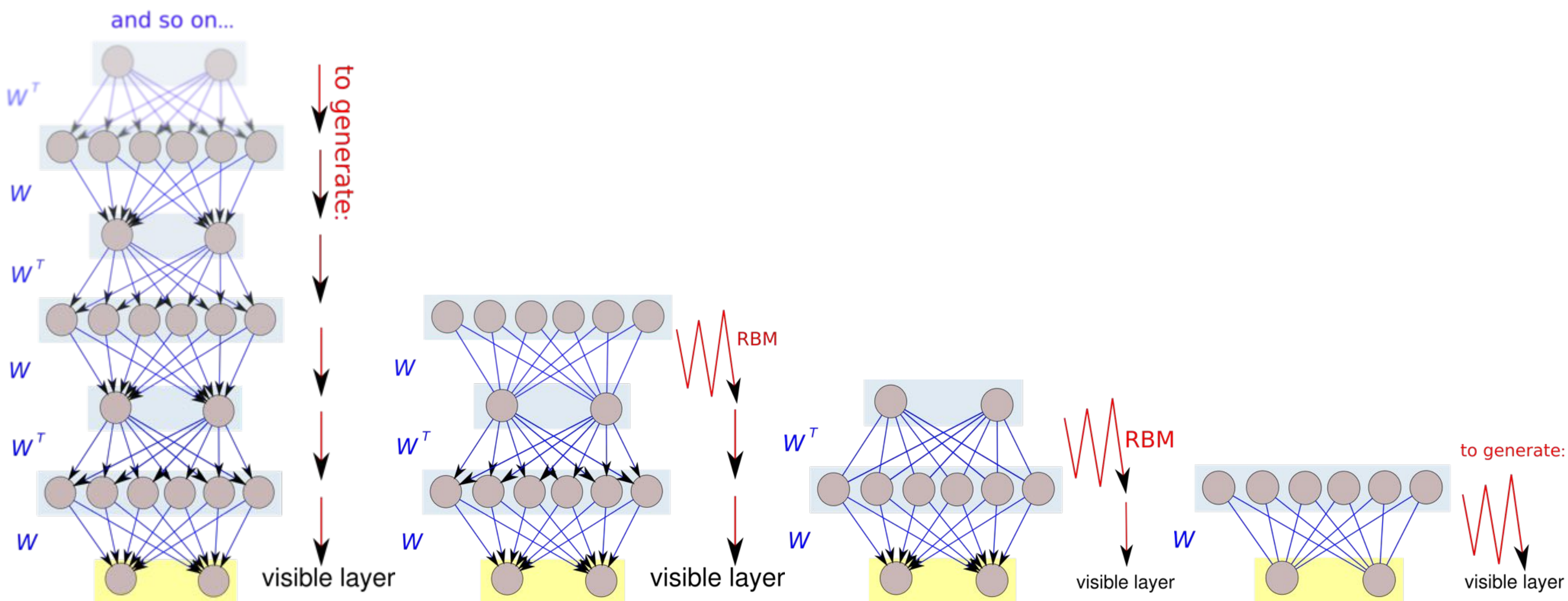


sampling from this is the same as sampling from the network on the right.

Slide from Marcus Frean, MLSS Tutorial 2010

Deep Belief Networks (DBNs)

RBMs are equivalent to infinitely deep belief networks



- So when we train an RBM, we're really training an ∞^{ly} deep sigmoid belief net!
- It's just that the weights of all layers are tied.

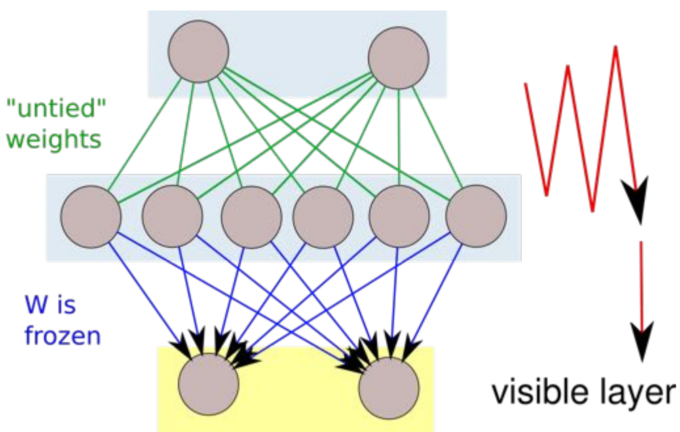
Slide from Marcus Frean, MLSS Tutorial 2010

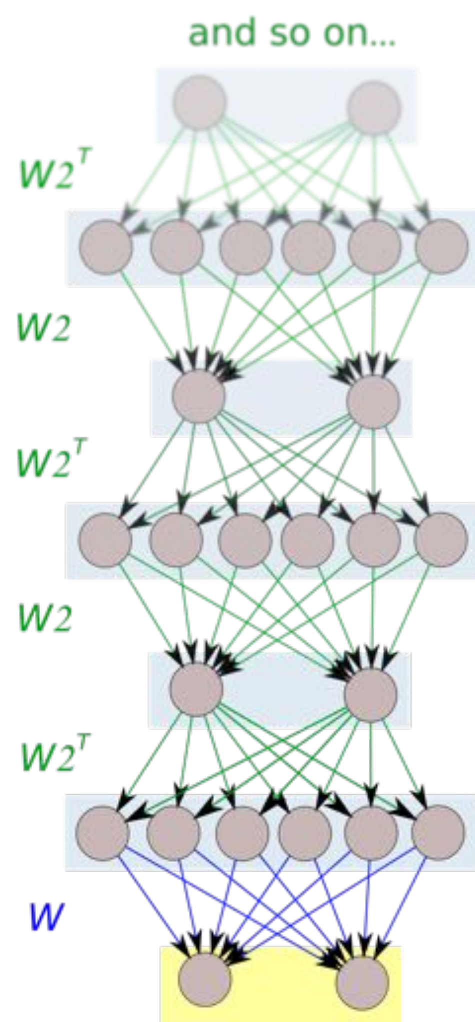
DBNs

Deep Belief Networks (DBNs)

Un-tie the weights from layers 2 to infinity

If we freeze the first RBM, and then train another RBM atop it, we are untying the weights of layers 2+ in the ∞ net (which remain tied together).





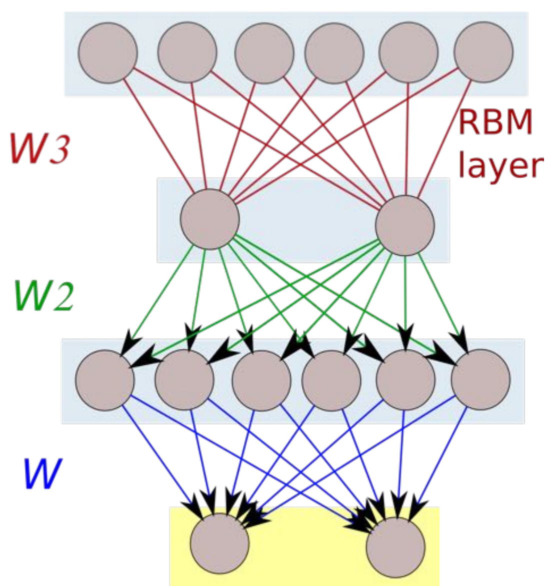
Slide from Marcus Frean, MLSS Tutorial 2010

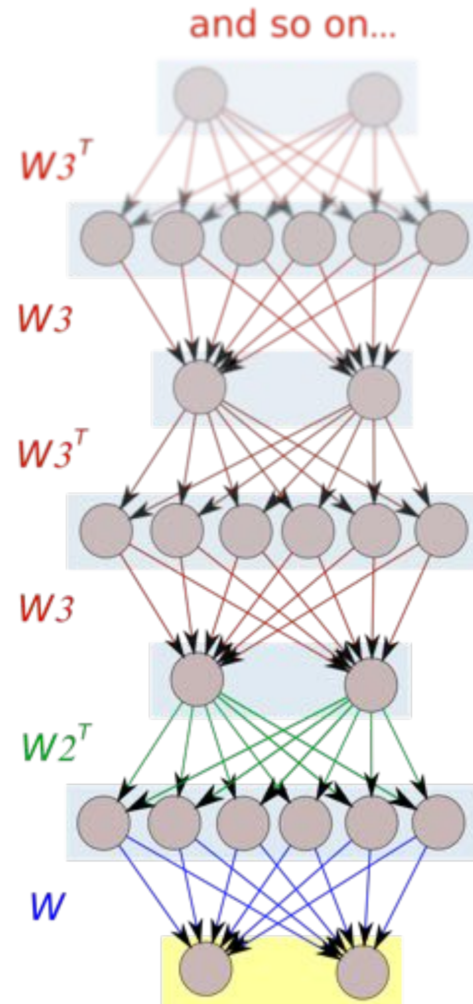
DBNs

Deep Belief Networks (DBNs)

Un-tie the weights from layers 3 to infinity

and ditto for the 3rd layer...





Slide from Marcus Frean, MLSS Tutorial 2010

Deep Belief Networks (DBNs)

fine-tuning with the wake-sleep algorithm

So far, the up and down weights have been symmetric, as required by the Boltzmann machine learning algorithm. And we didn't change the lower levels after "freezing" them.

- wake: do a bottom-up pass, starting with a pattern from the training set. Use the delta rule to make this more likely *under the generative model*.
- sleep: do a top-down pass, starting from an equilibrium sample from the top RBM. Use the delta rule to make this more likely *under the recognition model*.

[CD version: start top RBM at the sample from the wake phase, and don't wait for equilibrium before doing the top-down pass].

wake-sleep learning algorithm

unties the recognition weights from the generative ones

Slide from Marcus Frean, MLSS Tutorial 2010

Unsupervised Learning of DBNs

Setting A: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion

II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)

III. Fine-tune the parameters using backpropagation

Figure from (Hinton & Salakhutinov, 2006)

DBNs

Unsupervised Learning of DBNs

Setting A: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion
II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)
III. Fine-tune the parameters using backpropagation

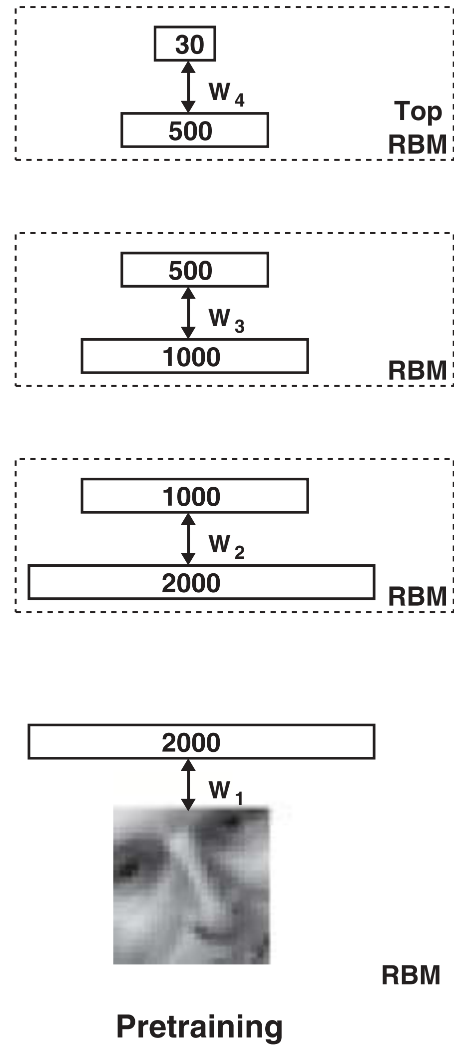


Figure from (Hinton & Salakhutinov, 2006)

Unsupervised Learning of DBNs

Setting A: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion

II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)

III. Fine-tune the parameters using backpropagation

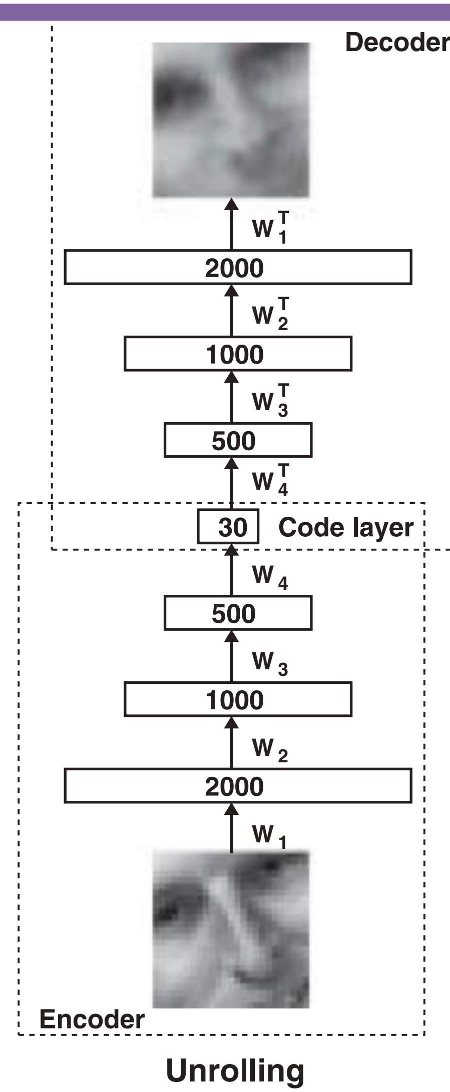


Figure from (Hinton & Salakhutinov, 2006)

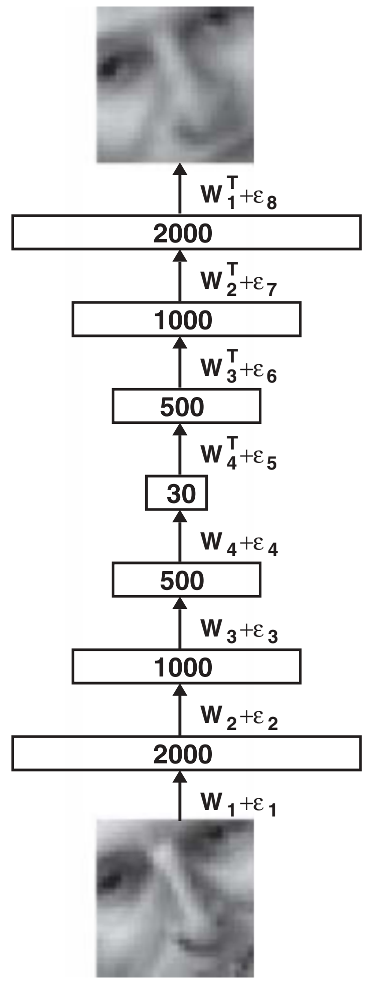
Unsupervised Learning of DBNs

Setting A: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion

II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)

III. Fine-tune the parameters using backpropagation



Fine-tuning

Figure from (Hinton & Salakhutinov, 2006)

Supervised Learning of DBNs

Setting B: DBN classifier

I. Pre-train a stack of RBMs in greedy layerwise fashion (unsupervised)

II. Fine-tune the parameters using backpropagation by minimizing classification error on the training data

Figure from (Hinton & Salakhutinov, 2006)

MNIST Digit **Generation**

- Comparison of deep autoencoder, logistic PCA, and PCA
- Each method projects the *real data* down to a vector of 30 real numbers
- Then reconstructs the data from the low-dimensional projection

Figure from Hinton, NIPS Tutorial 2007

Learning Deep Belief Networks (DBNs)

Setting B: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion

II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)

III. Fine-tune the parameters using backpropagation

Figure from (Hinton & Salakhutinov, 2006)

DBNs

MNIST Digit **Generation**

- This section: Suppose you want to build a **generative** model capable of explaining handwritten digits
- Goal:
  - To have a model p(x) **from which we can sample digits** that look realistic
  - Learn **unsupervised** hidden representation of an image

Figure 8: Each row shows 10 samples from the generative model with a particular label clamped on. The top-level associative memory is run for 1000 iterations of alternating Gibbs sampling between samples.

Samples from a DBN trained on MNIST

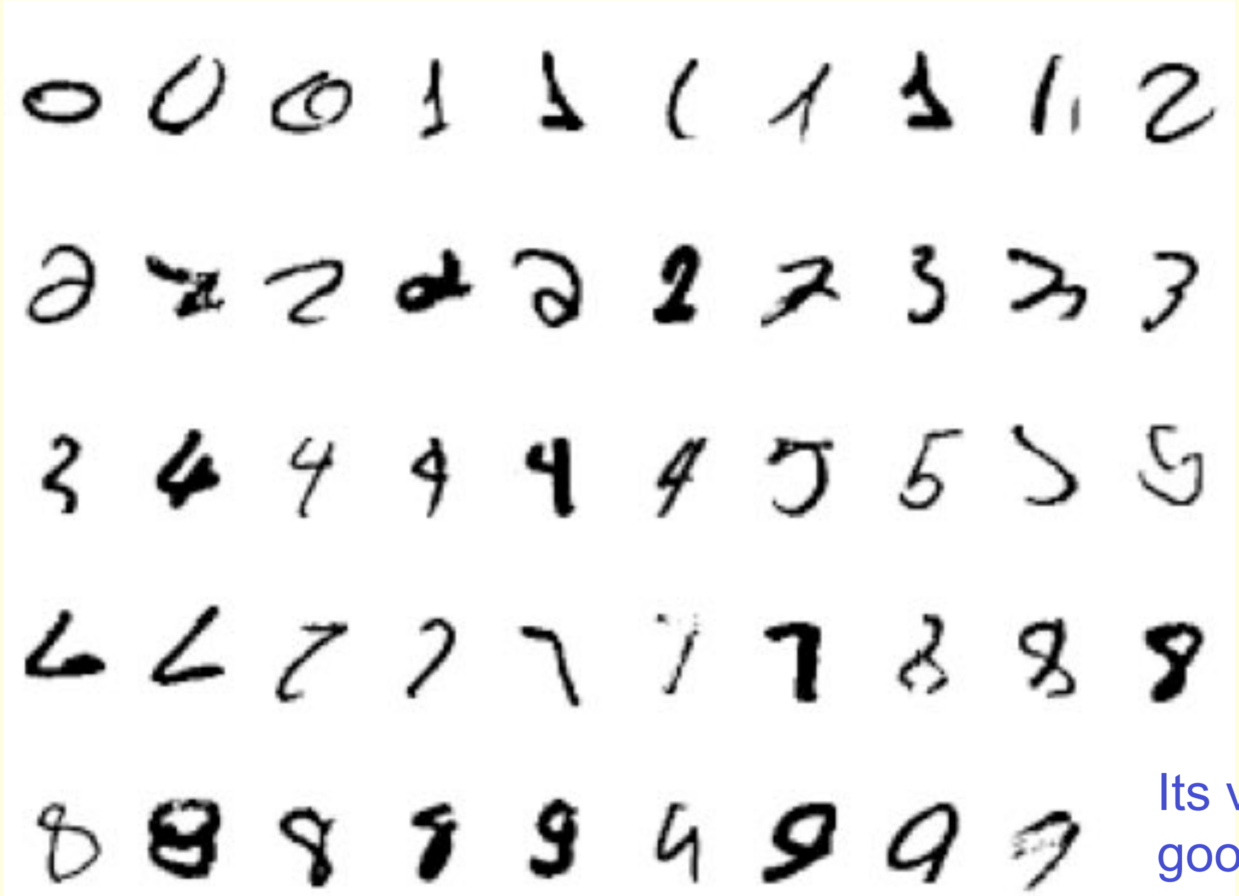
Figure from (Hinton et al., 2006)

DBNs

MNIST Digit **Recognition**

Experimental evaluation of DBN with greedy layer-wise pre-training and fine-tuning via the wake-sleep algorithm

Examples of correctly recognized handwritten digits that the neural network had never seen before



Its very good

Slide from Hinton, NIPS Tutorial 2007

MNIST Digit **Recognition**

Experimental evaluation of DBN with greedy layer-wise pre-training and fine-tuning via the wake-sleep algorithm

How well does it discriminate on MNIST test set with no extra information about geometric distortions?

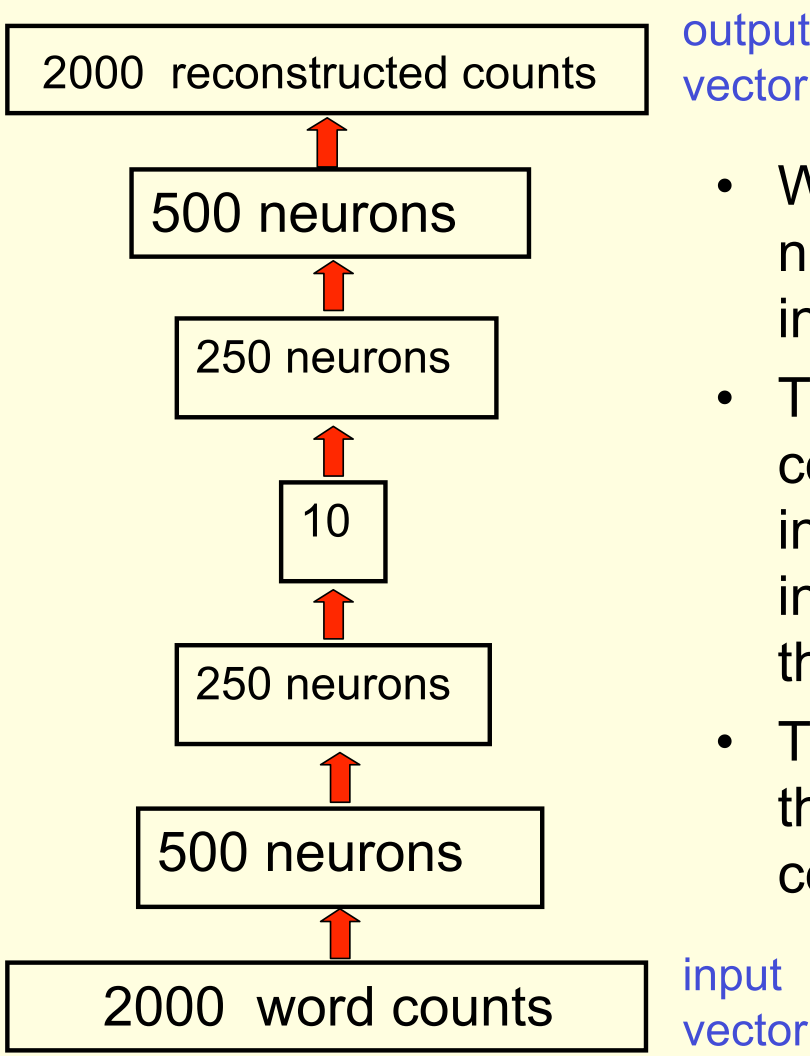
Generative model based on RBM's	1.25%
Support Vector Machine (Decoste et. al.)	1.4%
Backprop with 1000 hiddens (Platt)	~1.6%
Backprop with 500 -->300 hiddens	~1.6%
K-Nearest Neighbor	~ 3.3%
See Le Cun et. al. 1998 for more results	

- Its better than backprop and much more neurally plausible because the neurons only need to send one kind of signal, and the teacher can be another sensory input.

Slide from Hinton, NIPS Tutorial 2007

DBNs

Document Clustering and Retrieval



- We train the neural network to reproduce its input vector as its output
- This forces it to compress as much information as possible into the 10 numbers in the central bottleneck.
- These 10 numbers are then a good way to compare documents.

Slide from Hinton, NIPS Tutorial 2007

Document Clustering and Retrieval

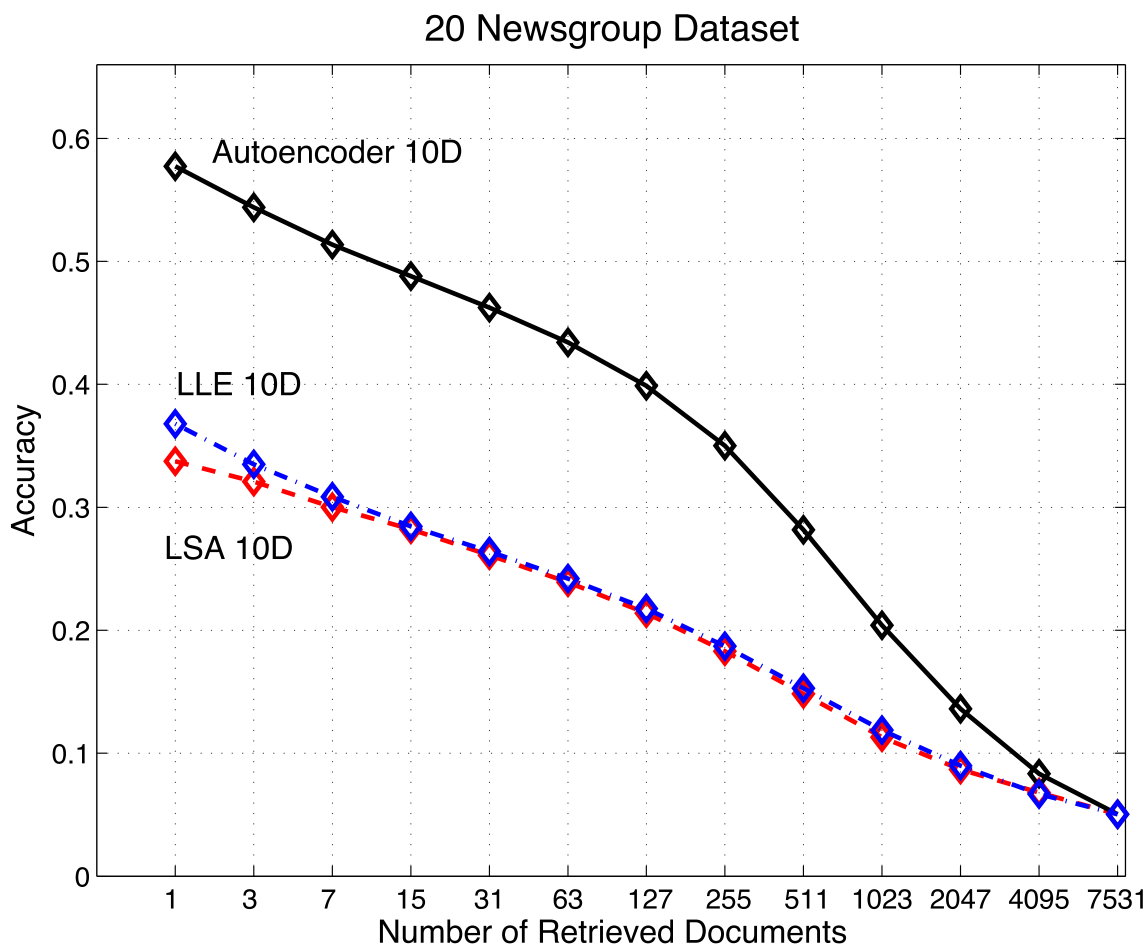
Performance of the autoencoder at document retrieval

- Train on bags of 2000 words for 400,000 training cases of business documents.
  - First train a stack of RBM's. Then fine-tune with backprop.
- Test on a separate 400,000 documents.
  - Pick one test document as a query. Rank order all the other test documents by using the cosine of the angle between codes.
  - Repeat this using each of the 400,000 test documents as the query (requires 0.16 trillion comparisons).
- Plot the number of retrieved documents against the proportion that are in the same hand-labeled class as the query document.

Slide from Hinton, NIPS Tutorial 2007

DBNs

Document Clustering and Retrieval



Retrieval Results

- Goal: given a query document, retrieve the relevant test documents
- Figure shows accuracy for varying numbers of retrieved test docs

Figure from (Hinton and Salakhutdinov, 2006)

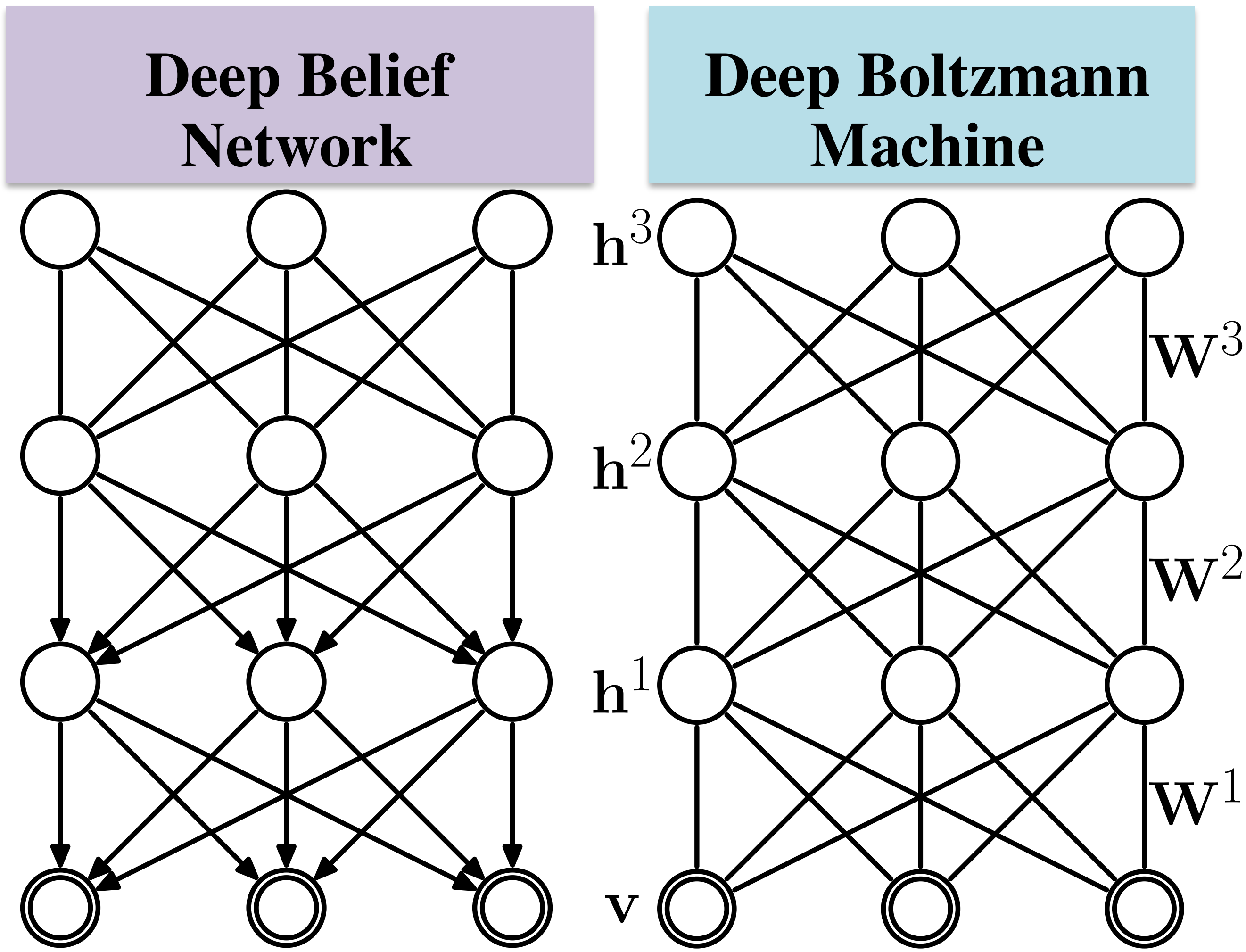
Outline

- **Motivation**
- **Deep Neural Networks (DNNs)**
  - Background: Decision functions
  - Background: Neural Networks
  - Three ideas for training a DNN
  - Experiments: MNIST digit classification
- **Deep Belief Networks (DBNs)**
  - Sigmoid Belief Network
  - Contrastive Divergence learning
  - Restricted Boltzman Machines (RBMs)
  - RBMs as infinitely deep Sigmoid Belief Nets
  - Learning DBNs
- **Deep Boltzman Machines (DBMs)**
  - Boltzman Machines
  - Learning Boltzman Machines
  - Learning DBMs

DBMs

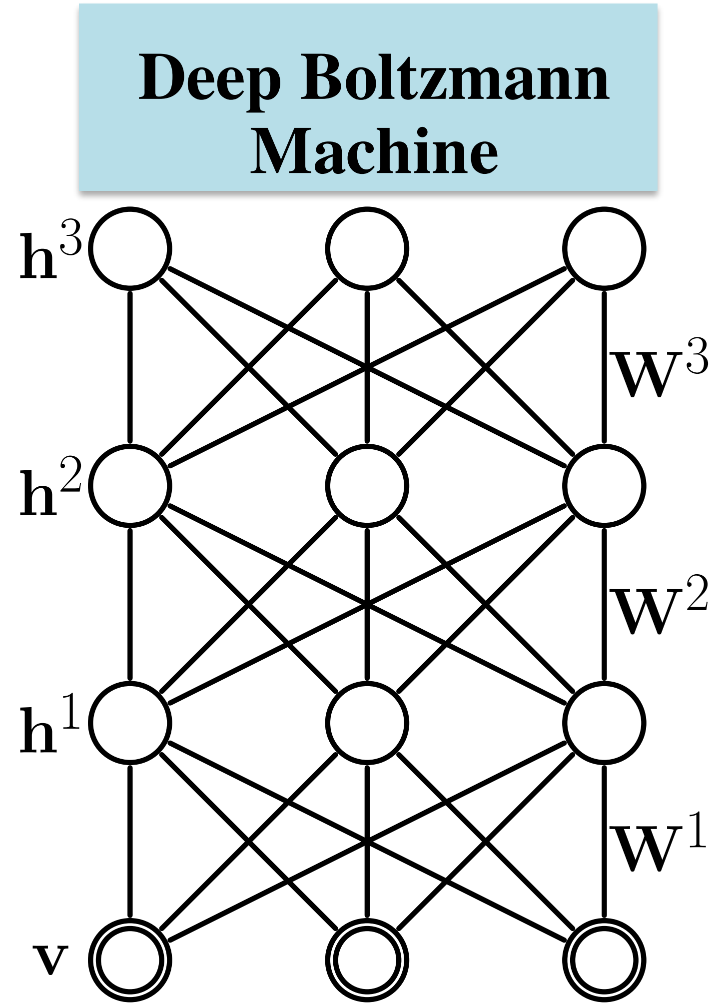
Deep Boltzman Machines

- DBNs are a hybrid directed/undirected graphical model
- DBMs are a purely undirected graphical model



Deep Boltzman Machines

Can we use the same techniques to train a DBM?



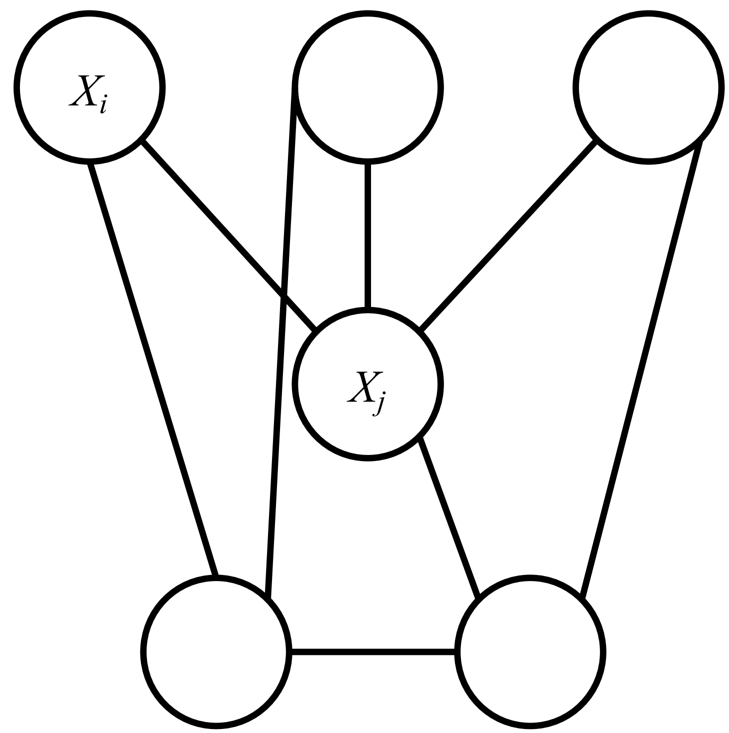
DBMs

Learning Standard Boltzman Machines

- Undirected graphical model of binary variables with pairwise potentials
- Parameterization of the potentials:

$$\psi_{ij}(x_i, x_j) = \exp(x_i W_{ij} x_j)$$

(In English: higher value of parameter W_{ij} leads to higher correlation between X_i and X_j on value 1)



DBMs

Learning Standard Boltzman Machines

Visible units: $\mathbf{v} \in \{0,1\}^D$

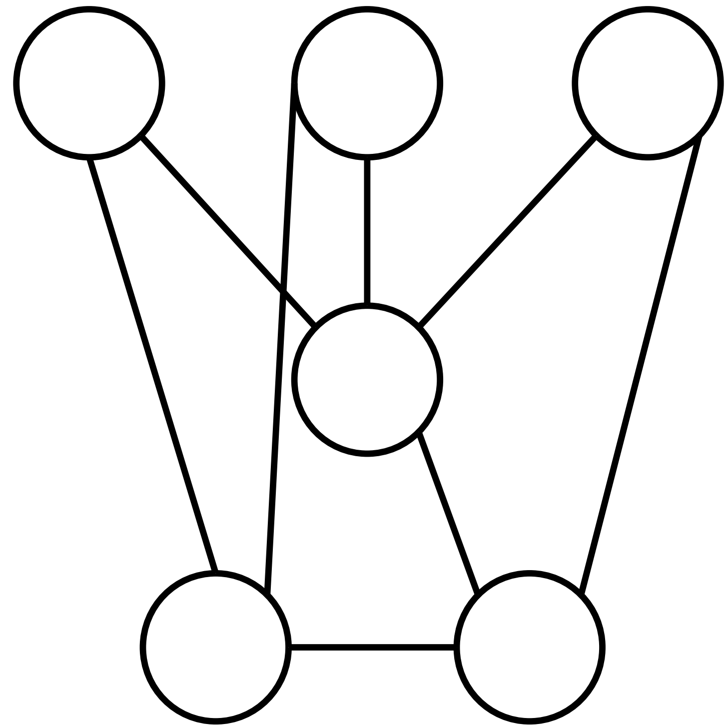
Hidden units: $\mathbf{h} \in \{0,1\}^P$

Likelihood:

$$E(\mathbf{v},\mathbf{h};\theta) = -\frac{1}{2}\mathbf{v}^\top \mathbf{L}\mathbf{v} - \frac{1}{2}\mathbf{h}^\top \mathbf{J}\mathbf{h} - \mathbf{v}^\top \mathbf{W}\mathbf{h},$$

$$p(\mathbf{v};\theta) = \frac{p^*(\mathbf{v};\theta)}{Z(\theta)} = \frac{1}{Z(\theta)}\sum_h \exp\left(-E(\mathbf{v},\mathbf{h};\theta)\right),$$

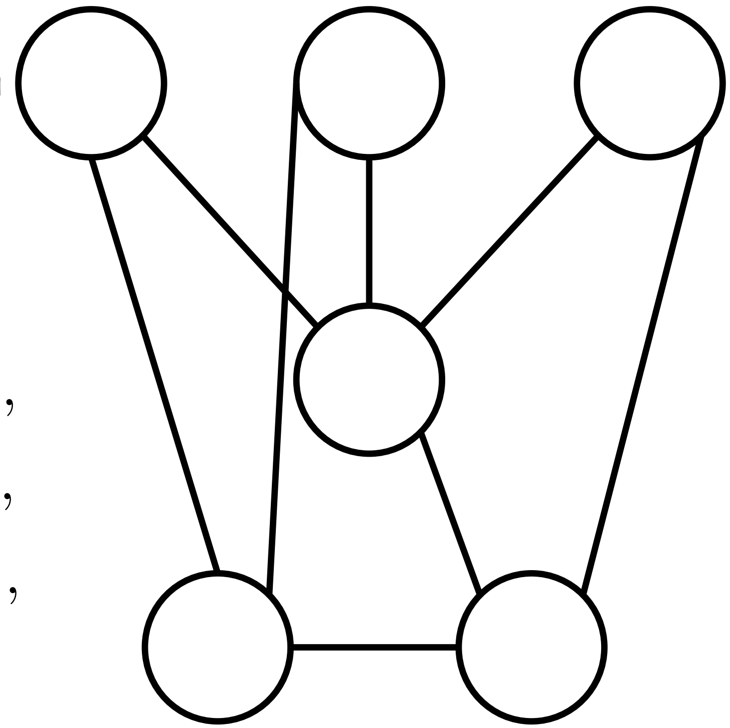
$$Z(\theta) = \sum_{\mathbf{v}}\sum_{\mathbf{h}} \exp\left(-E(\mathbf{v},\mathbf{h};\theta)\right),$$



DBMs

Learning Standard Boltzman Machines

(Old) idea from Hinton & Sejnowski (1983): For each iteration of optimization, run a separate MCMC chain for each of the data and model expectations to approximate the parameter updates.



Delta updates to each of model parameters:

$$\begin{aligned}
\Delta \mathbf{W} &= \alpha\left(\mathrm{E}_{P_{\text{data}}}[\mathbf{v}\mathbf{h}^\top] - \mathrm{E}_{P_{\text{model}}}[\mathbf{v}\mathbf{h}^\top]\right), \\
\Delta \mathbf{L} &= \alpha\left(\mathrm{E}_{P_{\text{data}}}[\mathbf{v}\mathbf{v}^\top] - \mathrm{E}_{P_{\text{model}}}[\mathbf{v}\mathbf{v}^\top]\right), \\
\Delta \mathbf{J} &= \alpha\left(\mathrm{E}_{P_{\text{data}}}[\mathbf{h}\mathbf{h}^\top] - \mathrm{E}_{P_{\text{model}}}[\mathbf{h}\mathbf{h}^\top]\right),
\end{aligned}$$

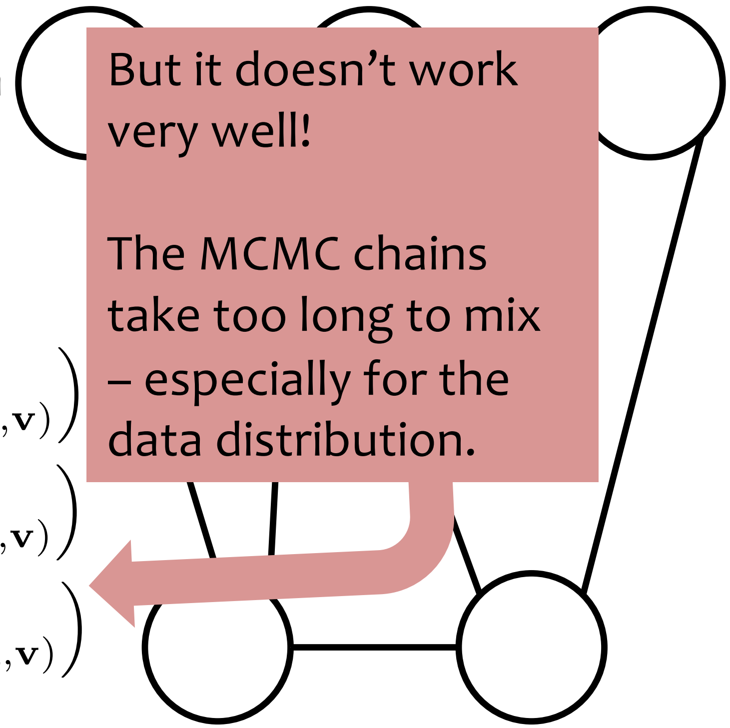
Full conditionals for Gibbs sampler:

$$p(h_j = 1|\mathbf{v}, \mathbf{h}_{-j}) = \sigma\Big(\sum_{i=1}^{D} W_{ij} v_i + \sum_{m=1\backslash j}^{P} J_{jm} h_j\Big),$$

$$p(v_i = 1|\mathbf{h}, \mathbf{v}_{-i}) = \sigma\Big(\sum_{j=1}^{P} W_{ij} h_j + \sum_{k=1\backslash i}^{D} L_{ik} v_j\Big),$$

Learning Standard Boltzman Machines

(Old) idea from Hinton & Sejnowski (1983): For each iteration of optimization, run a separate MCMC chain for each of the data and model expectations to approximate the parameter updates.



But it doesn't work very well!

The MCMC chains take too long to mix – especially for the data distribution.

Delta updates to each of model parameters:

$$\Delta \mathbf{W} = \alpha \left(\langle \mathbf{v}\mathbf{h}^T \rangle_{\mathbf{v} \in \mathcal{D}, \mathbf{h} \sim p(\mathbf{h}|\mathbf{v})} - \langle \mathbf{v}\mathbf{h}^T \rangle_{\mathbf{v}, \mathbf{h} \sim p(\mathbf{h}, \mathbf{v})} \right)$$

$$\Delta \mathbf{L} = \alpha \left(\langle \mathbf{v}\mathbf{v}^T \rangle_{\mathbf{v} \in \mathcal{D}, \mathbf{h} \sim p(\mathbf{h}|\mathbf{v})} - \langle \mathbf{v}\mathbf{v}^T \rangle_{\mathbf{v}, \mathbf{h} \sim p(\mathbf{h}, \mathbf{v})} \right)$$

$$\Delta \mathbf{J} = \alpha \left(\langle \mathbf{h}\mathbf{h}^T \rangle_{\mathbf{v} \in \mathcal{D}, \mathbf{h} \sim p(\mathbf{h}|\mathbf{v})} - \langle \mathbf{h}\mathbf{h}^T \rangle_{\mathbf{v}, \mathbf{h} \sim p(\mathbf{h}, \mathbf{v})} \right)$$

Full conditionals for Gibbs sampler:

$$p(h_j = 1 | \mathbf{v}, \mathbf{h}_{-j}) = \sigma\Big(\sum_{i=1}^{D} W_{ij} v_i + \sum_{m=1 \setminus j}^{P} J_{jm} h_j \Big),$$

$$p(v_i = 1 | \mathbf{h}, \mathbf{v}_{-i}) = \sigma\Big(\sum_{j=1}^{P} W_{ij} h_j + \sum_{k=1 \setminus i}^{D} L_{ik} v_j \Big),$$

Learning Standard Boltzman Machines

(New) idea from Salakhutinov & Hinton (2009):

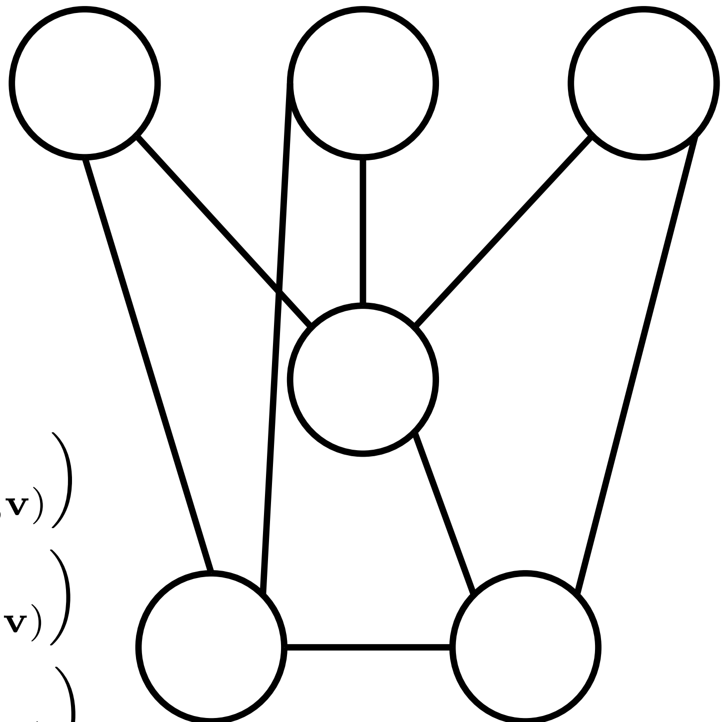
- Step 1) Approximate the data distribution by variational inference.
- Step 2) Approximate the model distribution with a "persistent" Markov chain (from iteration to iteration)

Delta updates to each of model parameters:

$$\Delta \mathbf{W} = \alpha \left(\left\langle \mathbf{v}\mathbf{h}^T \right\rangle_{\mathbf{v}\in\mathcal{D},\mathbf{h}\sim p(\mathbf{h}|\mathbf{v})} - \left\langle \mathbf{v}\mathbf{h}^T \right\rangle_{\mathbf{v},\mathbf{h}\sim p(\mathbf{h},\mathbf{v})} \right)$$

$$\Delta \mathbf{L} = \alpha \left(\left\langle \mathbf{v}\mathbf{v}^T \right\rangle_{\mathbf{v}\in\mathcal{D},\mathbf{h}\sim p(\mathbf{h}|\mathbf{v})} - \left\langle \mathbf{v}\mathbf{v}^T \right\rangle_{\mathbf{v},\mathbf{h}\sim p(\mathbf{h},\mathbf{v})} \right)$$

$$\Delta \mathbf{J} = \alpha \left(\left\langle \mathbf{h}\mathbf{h}^T \right\rangle_{\mathbf{v}\in\mathcal{D},\mathbf{h}\sim p(\mathbf{h}|\mathbf{v})} - \left\langle \mathbf{h}\mathbf{h}^T \right\rangle_{\mathbf{v},\mathbf{h}\sim p(\mathbf{h},\mathbf{v})} \right)$$

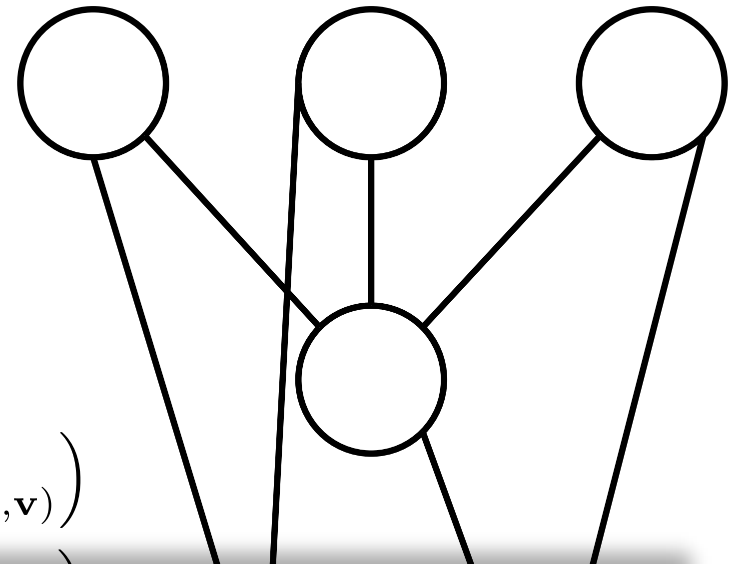


DBMs

Learning Standard Boltzman Machines

(New) idea from Salakhutinov & Hinton (2009):

- Step 1) Approximate the data distribution by variational inference.
- Step 2) Approximate the model distribution with a "persistent" Markov chain (from iteration to iteration)



Delta updates to each of model parameters:

$$\Delta \mathbf{W} = \alpha \left(\langle \mathbf{v}\mathbf{h}^T \rangle_{\mathbf{v}\in\mathcal{D},\mathbf{h}\sim p(\mathbf{h}|\mathbf{v})} - \langle \mathbf{v}\mathbf{h}^T \rangle_{\mathbf{v},\mathbf{h}\sim p(\mathbf{h},\mathbf{v})} \right)$$

Step 1) Approximate the data distribution…

Mean-field approximation:

$$q(\mathbf{h};\mu) = \prod_{j=1}^{P} q(h_i)$$

$$q(h_i = 1) = \mu_i$$

Variational lower-bound of log-likelihood:

$$\ln p(\mathbf{v};\theta) \geq \sum_{\mathbf{h}} q(\mathbf{h}|\mathbf{v};\mu) \ln p(\mathbf{v},\mathbf{h};\theta) + \mathcal{H}(q)$$

Fixed-point equations for variational params:

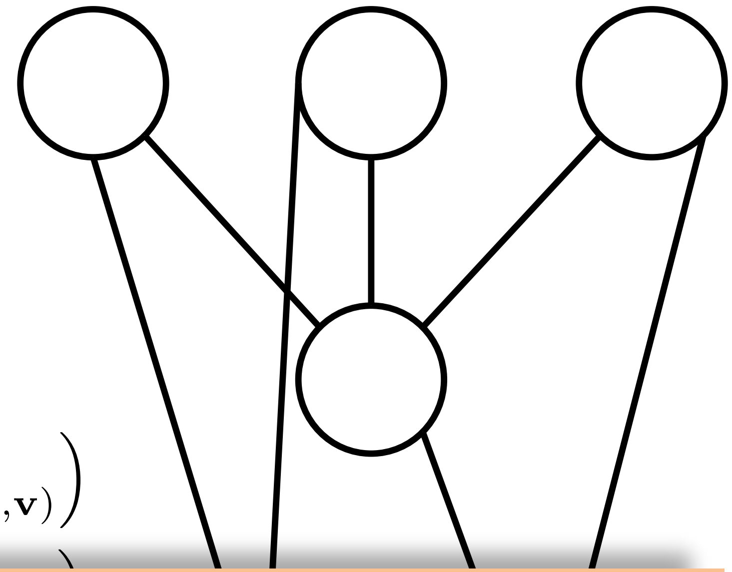
$$\mu_j \leftarrow \sigma\Big(\sum_i W_{ij} v_i + \sum_{m\backslash j} J_{mj}\mu_m\Big)$$

DBMs

Learning Standard Boltzman Machines

(New) idea from Salakhutinov & Hinton (2009):

- Step 1) Approximate the data distribution by variational inference.
- Step 2) Approximate the model distribution with a "persistent" Markov chain (from iteration to iteration)



Delta updates to each of model parameters:

$$\Delta \mathbf{W} = \alpha \left(\langle \mathbf{v}\mathbf{h}^T \rangle_{\mathbf{v}\in\mathcal{D},\mathbf{h}\sim p(\mathbf{h}|\mathbf{v})} - \langle \mathbf{v}\mathbf{h}^T \rangle_{\mathbf{v},\mathbf{h}\sim p(\mathbf{h},\mathbf{v})} \right)$$

Step 2) Approximate the model distribution…

Why not use variational inference for the model expectation as well?

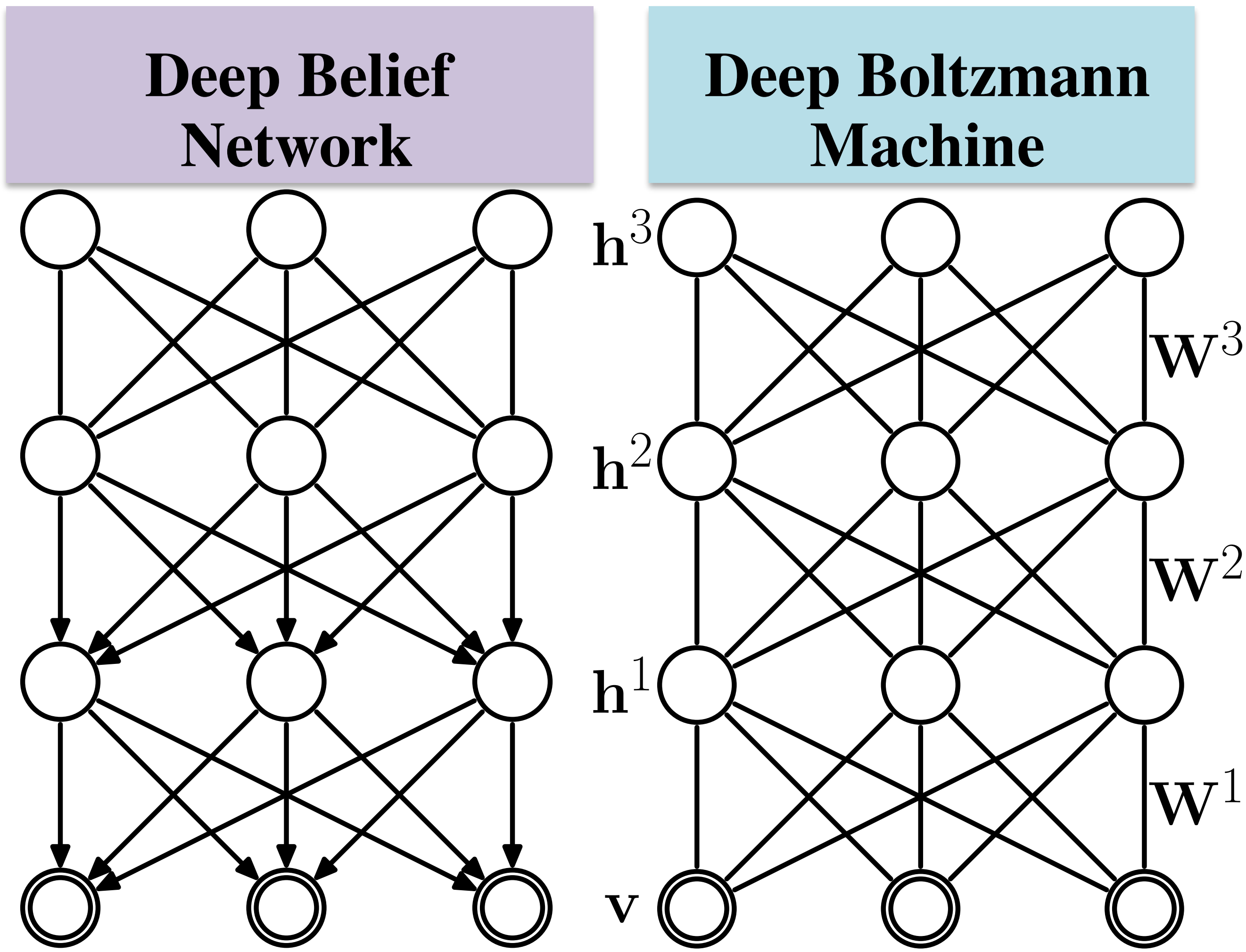
Difference of the two mean-field approximated expectations above would cause learning algorithm to **maximize** divergence between true and mean-field distributions.

Persistent CD adds correlations between successive iterations, but not an issue.

DBMs

Deep Boltzman Machines

- DBNs are a hybrid directed/undirected graphical model
- DBMs are a purely undirected graphical model



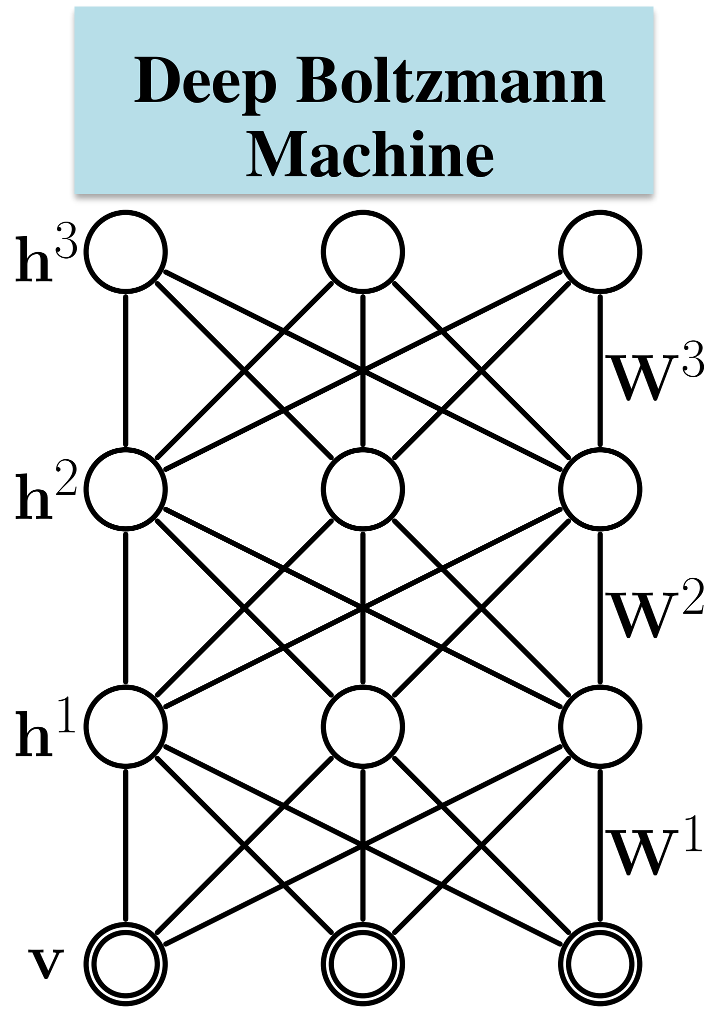
DBMs

Learning Deep Boltzman Machines

Can we use the same techniques to train a DBM?

I. Pre-train a stack of RBMs in greedy layerwise fashion (requires some caution to avoid double counting)

II. Use those parameters to initialize two step mean-field approach to learning full Boltzman machine (i.e. the full DBM)



Document Clustering and Retrieval

Clustering Results

- Goal: cluster related documents
- Figures show projection to 2 dimensions
- Color shows true categories

PCA

DBN

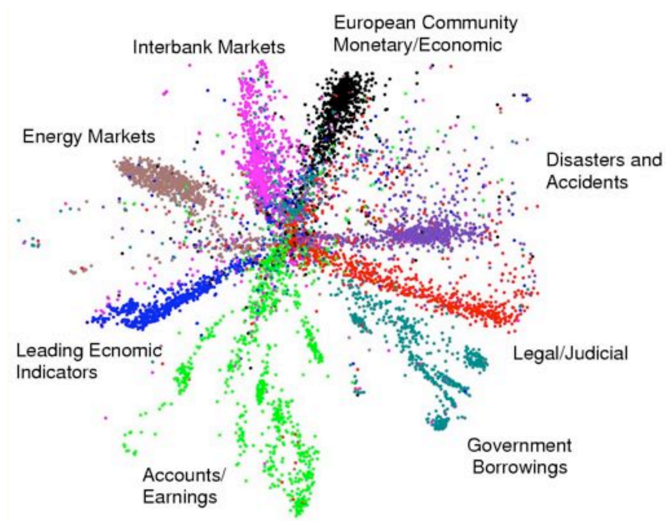


Figure from (Salakhutdinov and Hinton, 2009)

Course Level Objectives

You should be able to…

1. Formalize new tasks as structured prediction problems.
2. Develop new models by incorporating domain knowledge about constraints on or interactions between the outputs
3. Combine deep neural networks and graphical models
4. Identify appropriate inference methods, either exact or approximate, for a probabilistic graphical model
5. Employ learning algorithms that make the best use of available data
6. Implement from scratch state-of-the-art approaches to learning and inference for structured prediction models

Q&A