



10-708 Probabilistic Graphical Models

Machine Learning Department
School of Computer Science
Carnegie Mellon University



Deep Generative Models

Matt Gormley
Lecture 21
Apr. 21, 2021

Reminders

- **Homework 5: Variational Inference**
 - Out: Thu, Apr. 8
 - Due: Wed, Apr. 21 at 11:59pm
- **Project Midway Milestones:**
 - **Midway Poster Session:**
Tue, Apr. 27 at 6:30pm – 8:30pm
 - **Midway Executive Summary**
Due: Tue, Apr. 27 at 11:59pm
 - **New requirement: must have baseline results**

VAE RESULTS

VAE Results

Kingma & Welling (2014)

- introduced VAEs
 - applied to image generation
- Model
- $p_\phi(z) \sim N(z; 0, I)$
 - $p_\phi(x | z)$ is a multivariate Gaussian with mean and variance computed by an MLP, fully connected neural network with a single hidden layer with parameters ϕ
 - $q_\theta(z | x)$ is a multivariate Gaussian with diagonal covariance structure and with mean and variance computed by an MLP with parameters θ

Auto-Encoding Variational Bayes

Diederik P. Kingma
Machine Learning Group
Universiteit van Amsterdam
dpk@csail.mit.edu

Max Welling
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

Abstract

How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variational inference and learning algorithm that scales to large datasets and, under some mild differentiability conditions, even works in the intractable case. Our contributions is two-fold. First, we show that a reparameterization of the variational lower bound yields a lower bound estimator that can be straightforwardly optimized using standard stochastic gradient methods. Second, we show that for i.i.d. datasets with continuous latent variables per datapoint, posterior inference can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed lower bound estimator. Theoretical advantages are reflected in experimental results.

VAE Results

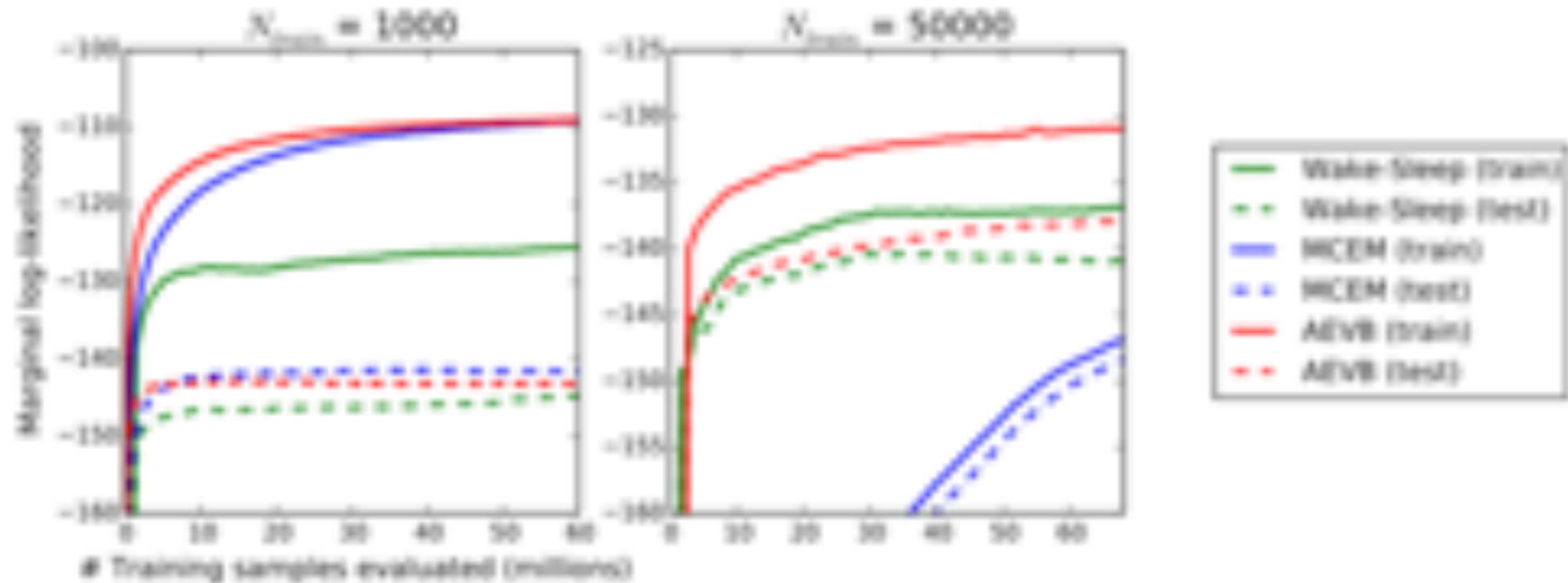


Figure 3: Comparison of AEVB to the wake-sleep algorithm and Monte Carlo EM, in terms of the estimated marginal likelihood, for a different number of training points. Monte Carlo EM is not an on-line algorithm, and (unlike AEVB and the wake-sleep method) can't be applied efficiently for the full MNIST dataset.

VAE Results

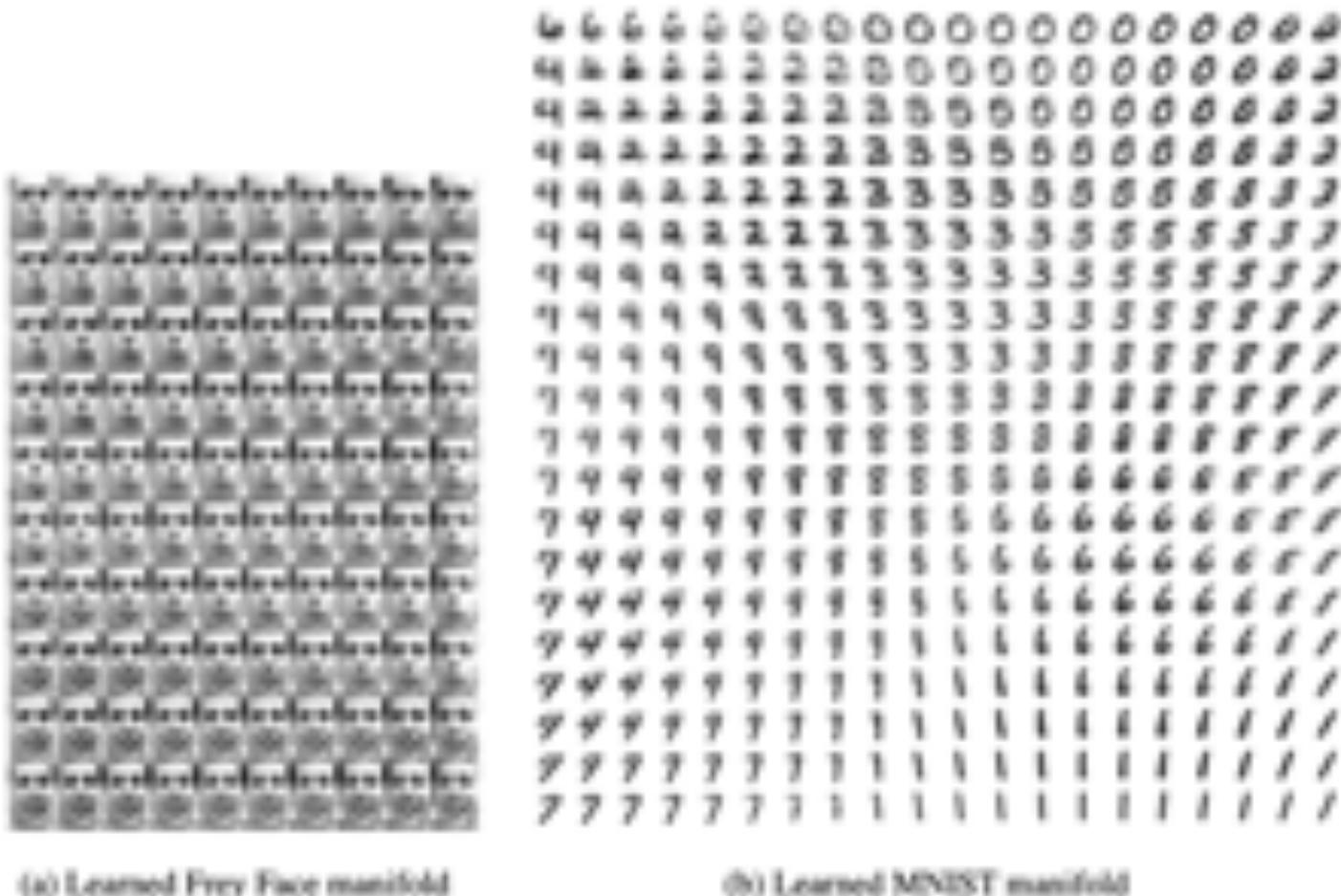


Figure 4: Visualisations of learned data manifold for generative models with two-dimensional latent space, learned with AEVB. Since the prior of the latent space is Gaussian, linearly spaced coordinates on the unit square were transformed through the inverse CDF of the Gaussian to produce values of the latent variables z . For each of these values z , we plotted the corresponding generative $p_\theta(x|z)$ with the learned parameters θ .

VAE Results

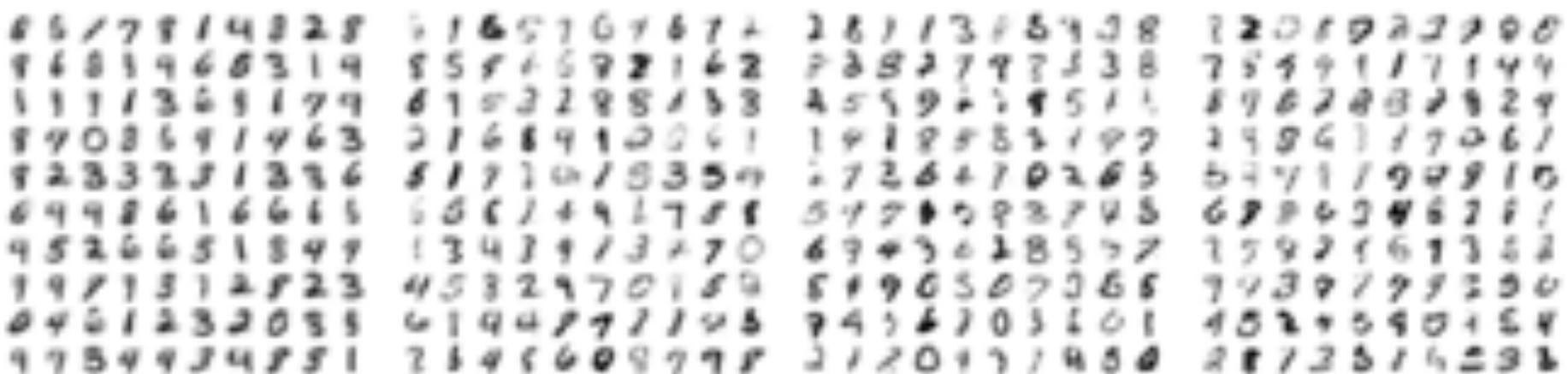


Figure 5: Random samples from learned generative models of MNIST for different dimensionalities of latent space.

VAE Results

Bowman et al. (2015)

- example of an application of VAEs to discrete data
- built on the sequence-to-sequence framework:
 - input is read in by an LSTM
 - output is generated by an LSTM-LM

Model

- $p_\phi(z) \sim N(z; 0, I)$
- $p_\phi(x | z)$ is an LSTM Language Model with parameters ϕ
- $q_\theta(z | x)$ is a multivariate Gaussian with mean and variance computed by an LSTM with parameters θ

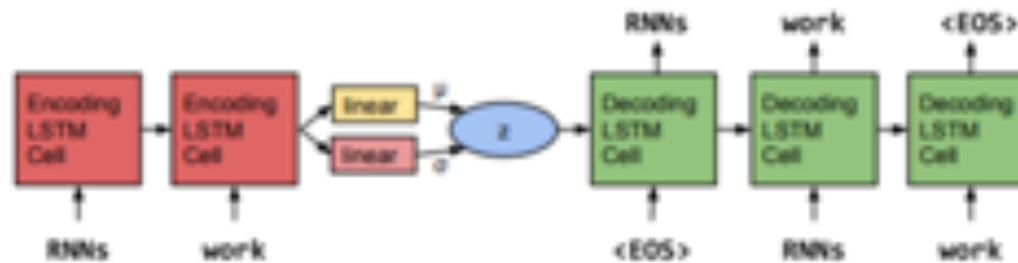


Figure 1: The core structure of our variational autoencoder language model. Words are represented using a learned dictionary of embedding vectors.

VAE Results

INPUT	we looked out at the setting sun .	i went to the kitchen .	how are you doing ?
MEAN	<i>they were laughing at the same time .</i>	<i>i went to the kitchen .</i>	<i>what are you doing ?</i>
SAMP. 1	<i>ill see you in the early morning .</i>	<i>i went to my apartment .</i>	<i>" are you sure ?</i>
SAMP. 2	<i>i looked up at the blue sky .</i>	<i>i looked around the room .</i>	<i>what are you doing ?</i>
SAMP. 3	<i>it was down on the dance floor .</i>	<i>i turned back to the table .</i>	<i>what are you doing ?</i>

Table 7: Three sentences which were used as inputs to the VAE, presented with greedy decodes from the mean of the posterior distribution, and from three samples from that distribution.

" i want to talk to you . "
<i>"i want to be with you . "</i>
<i>"i do n't want to be with you . "</i>
<i>i do n't want to be with you .</i>
she did n't want to be with him .
he was silent for a long moment .
<i>he was silent for a moment .</i>
<i>it was quiet for a moment .</i>
<i>it was dark and cold .</i>
<i>there was a pause .</i>
it was my turn .

Table 8: Paths between pairs of random points in VAE space: Note that intermediate sentences are grammatical, and that topic and syntactic structure are usually locally consistent.

GENERATIVE ADVERSARIAL NETWORK (GAN)



Generative Adversarial Nets (GANs)

- [Goodfellow et al., 2014]
- Generative model $\mathbf{x} = G_{\theta}(\mathbf{z})$, $\mathbf{z} \sim p(\mathbf{z})$
 - Map noise variable \mathbf{z} to data space \mathbf{x}
 - Define an **implicit distribution** over \mathbf{x} : $p_{g_{\theta}}(\mathbf{x})$
 - a stochastic process to simulate data \mathbf{x}
 - Intractable to evaluate likelihood
- Discriminator $D_{\phi}(\mathbf{x})$
 - Output the probability that \mathbf{x} came from the data rather than the generator
- No explicit inference model
- No obvious connection to previous models with inference networks like VAEs
 - We will build formal connections between GANs and VAEs later



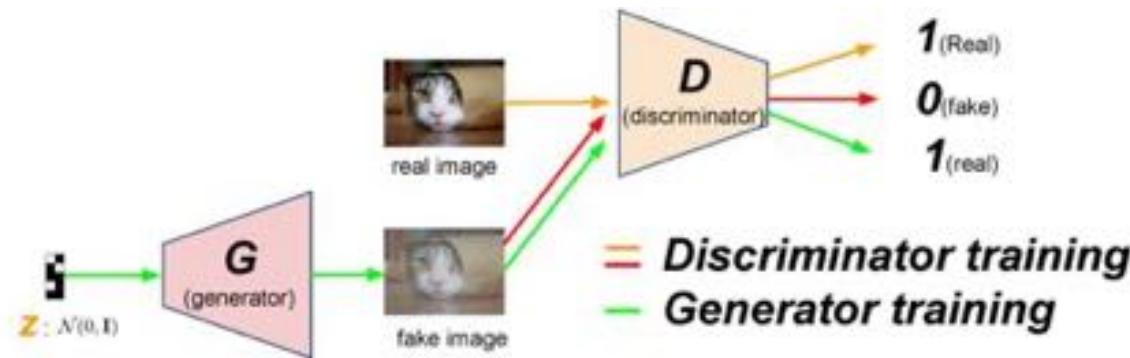


Generative Adversarial Nets (GANs)

- Learning
 - A minimax game between the generator and the discriminator
 - Train D to maximize the probability of assigning the correct label to both training examples and generated samples
 - Train G to fool the discriminator

$$\max_D \mathcal{L}_D = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim G(\mathbf{z}), \mathbf{z} \sim p(\mathbf{z})} [\log(1 - D(\mathbf{x}))]$$

$$\min_G \mathcal{L}_G = \mathbb{E}_{\mathbf{x} \sim G(\mathbf{z}), \mathbf{z} \sim p(\mathbf{z})} [\log(1 - D(\mathbf{x}))].$$



[Figure courtesy: Kim's slides]

© Eric Xing @ CMU, 2005-2020

31

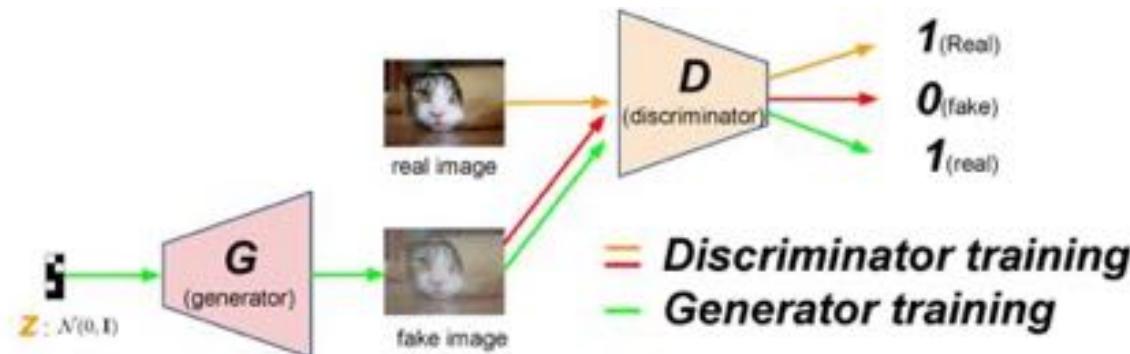




Generative Adversarial Nets (GANs)

- Learning
 - Train G to fool the discriminator
 - The original loss suffers from vanishing gradients when D is too strong
 - Instead use the following in practice

$$\max_G \mathcal{L}_G = \mathbb{E}_{\mathbf{x} \sim G(\mathbf{z}), \mathbf{z} \sim p(\mathbf{z})} [\log D(\mathbf{x})]$$



[Figure courtesy: Kim's slides]

© Eric Xing @ CMU, 2005-2020

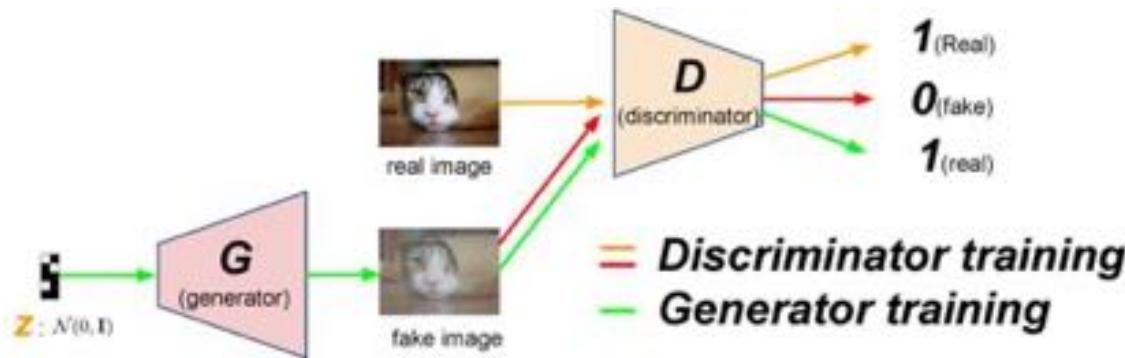
32





Generative Adversarial Nets (GANs)

- Learning
 - Aim to achieve equilibrium of the game
 - Optimal state:
 - $p_g(x) = p_{data}(x)$
 - $D(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)} = \frac{1}{2}$



[Figure courtesy: Kim's slides]

© Eric Xing @ CMU, 2005-2020

33





GANs: example results



Generated bedrooms [Radford et al., 2016]

© Eric Xing @ CMU, 2005-2020

34



Z Hu, Z YANG, R Salakhutdinov, E Xing,
“On Unifying Deep Generative Models”, arxiv 1706.00550

UNIFYING GANS AND VAEs



Generative Adversarial Nets (GANs):

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

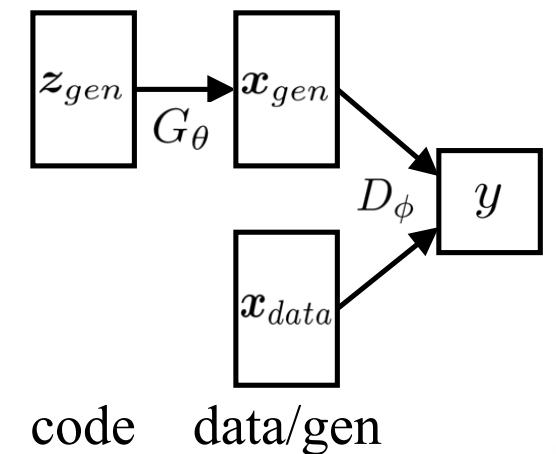
$$p_{\theta}(\mathbf{x}|y) = \begin{cases} p_{g_{\theta}}(\mathbf{x}) & y = 0 \\ p_{data}(\mathbf{x}) & y = 1. \end{cases}$$

(distribution of generated images)
(distribution of real images)

- $\mathbf{x} \sim p_{g_{\theta}}(\mathbf{x}) \Leftrightarrow \mathbf{x} = G_{\theta}(\mathbf{z}), \mathbf{z} \sim p(\mathbf{z}|y = 0)$

- $\mathbf{x} \sim p_{data}(\mathbf{x})$

- the code space of \mathbf{z} is degenerated
- sample directly from data



© Eric Xing @ CMU, 2005-2019

39





A new formulation

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

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

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

- ❑ Rewrite in the new form

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

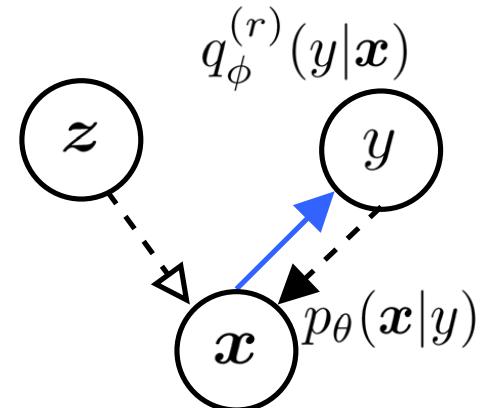
$$\mathbf{x} = G_{\theta}(\mathbf{z}), \mathbf{z} \sim p(\mathbf{z}|y)$$

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

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

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

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





GANs vs. Variational EM

Variational EM

- Objectives

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

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

- 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

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

In EVM, we minimize the following:

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

\rightarrow KL (inference model | posterior)

GAN

- Objectives

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

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

- 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

Eric Xing @ CMU 2005-2014 42





GANs vs VAEs side by side

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

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

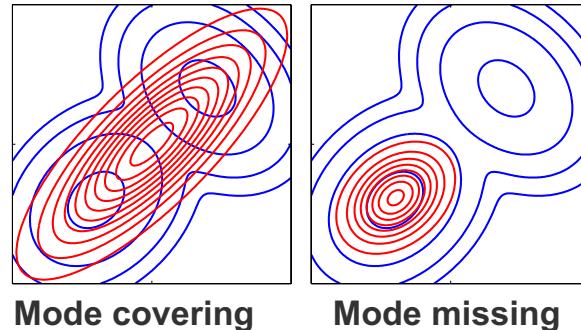




GANs vs VAEs side by side

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

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



[Figure courtesy: PRML]

© Eric Xing @ CMU, 2005-2019

50



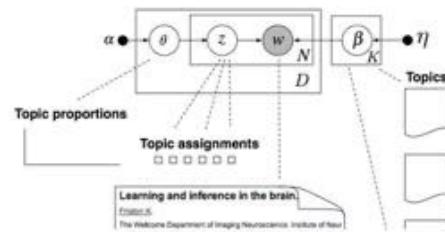
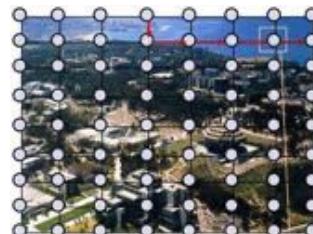
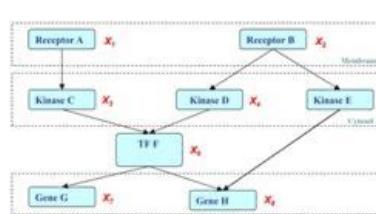
SIMILARITIES AND DIFFERENCES BETWEEN GMS AND NNS



Graphical models vs. Deep nets

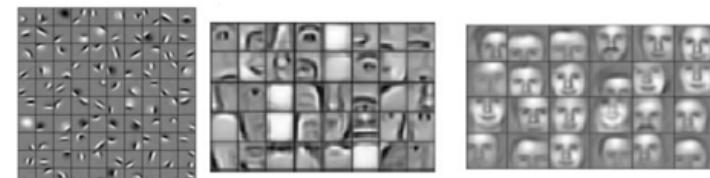
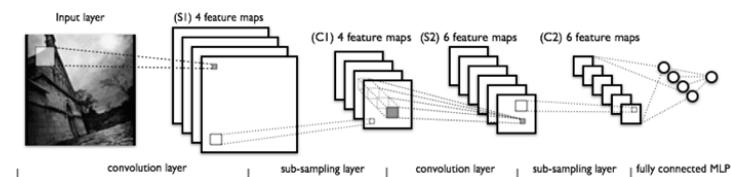
Graphical models

- Representation for encoding meaningful knowledge and the associated uncertainty in a graphical form



Deep neural networks

- Learn representations that facilitate computation and performance on the end-metric (intermediate representations are not guaranteed to be meaningful)



© Eric Xing @ CMU, 2005-2020

21





Graphical models vs. Deep nets

Graphical models

- Representation for encoding meaningful knowledge and the associated uncertainty in a graphical form
- Learning and inference are based on a rich toolbox of well-studied (structure-dependent) techniques (e.g., EM, message passing, VI, MCMC, etc.)
- Graphs represent models

Deep neural networks

- Learn representations that facilitate computation and performance on the end-metric (intermediate representations are not guaranteed to be meaningful)
- Learning is predominantly based on the gradient descent method (aka backpropagation); Inference is often trivial and done via a “forward pass”
- Graphs represent computation





Graphical models vs. Deep nets

Graphical models

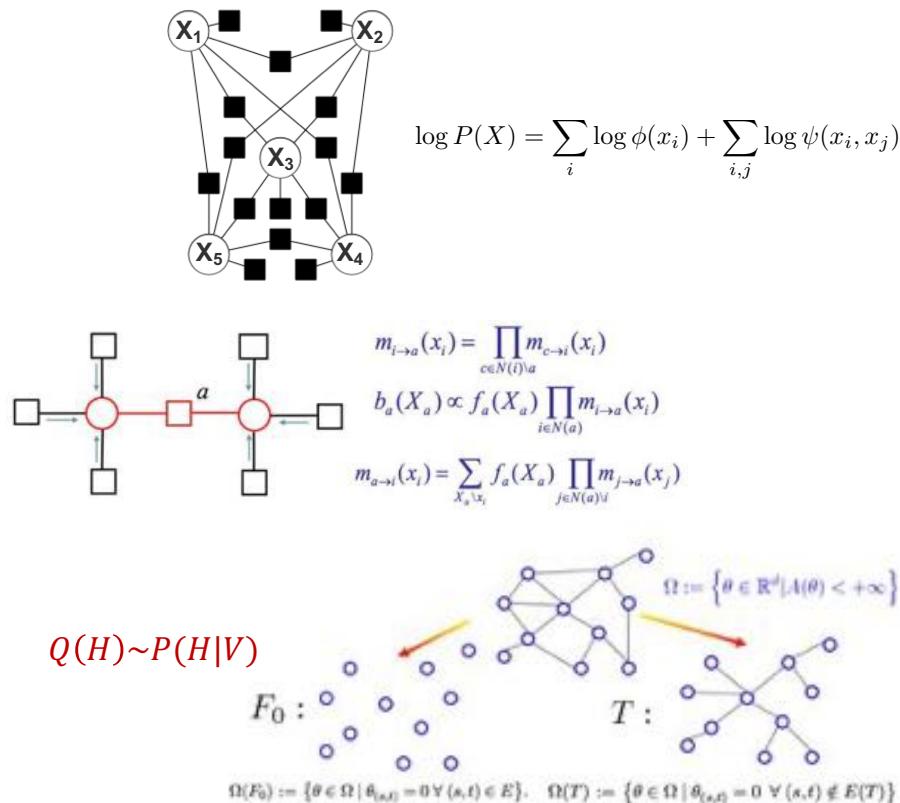
Utility of the graph

- ❑ A vehicle for synthesizing a global loss function from local structure
 - ❑ potential function, feature function, etc.
- ❑ A vehicle for designing sound and efficient inference algorithms
 - ❑ Sum-product, mean-field, etc.
- ❑ A vehicle to inspire approximation and penalization
 - ❑ Structured MF, Tree-approximation, etc.
- ❑ A vehicle for monitoring theoretical and empirical behavior and accuracy of inference

Utility of the loss function

- ❑ A major measure of quality of the learning algorithm and the model

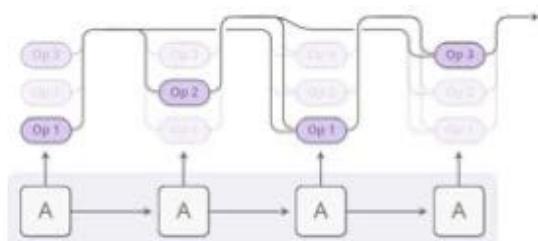
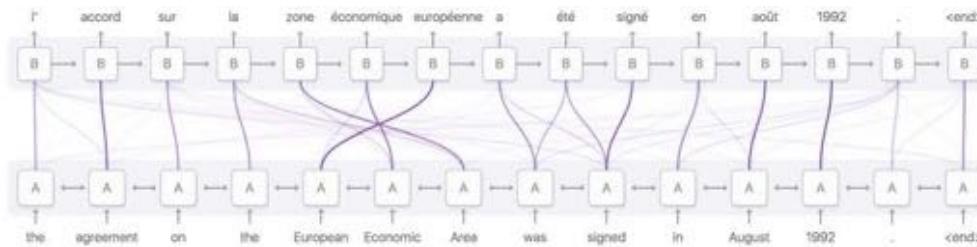
$$\theta = \operatorname{argmax}_{\theta} P_{\theta}(V)$$





Graphical models vs. Deep nets

Deep neural networks



Images from Distill.pub

Utility of the network

- A vehicle to conceptually synthesize complex decision hypothesis
 - stage-wise projection and aggregation
- A vehicle for organizing computational operations
 - stage-wise update of latent states
- A vehicle for designing processing steps and computing modules
 - Layer-wise parallelization
- No obvious utility in evaluating DL inference algorithms

Utility of the Loss Function

- Global loss? Well it is complex and non-convex...

© Eric Xing @ CMU, 2005-2020

24





Graphical models vs. Deep nets

Graphical models

Utility of the graph

- ❑ A vehicle for synthesizing a global loss function from local structure
 - potential function, feature function, etc.
- ❑ A vehicle for designing sound and efficient inference algorithms
 - Sum-product, mean-field, etc.
- ❑ A vehicle to inspire approximation and penalization
 - Structured MF, Tree-approximation, etc.
- ❑ A vehicle for monitoring theoretical and empirical behavior and accuracy of inference

Utility of the loss function

- ❑ A major measure of quality of the learning algorithm and the model

Deep neural networks

Utility of the network

- A vehicle to conceptually synthesize complex decision hypothesis
 - stage-wise projection and aggregation
 - A vehicle for organizing computational operations
 - stage-wise update of latent states
 - A vehicle for designing processing steps and computing modules
 - Layer-wise parallelization
 - No obvious utility in evaluating DL inference algorithms
- Utility of the Loss Function
- Global loss? Well it is complex and non-convex...

© Eric Xing @ CMU, 2005-2020

25



	DL	ML (e.g., GM)
Empirical goal:	e.g., classification, feature learning	e.g., latent variable inference, transfer learning
Structure:	Graphical	Graphical
Objective:	Something aggregated from local functions	Something aggregated from local functions
Vocabulary:	Neuron, activation function, ...	Variable, potential function, ...
Algorithm:	A single, unchallenged, inference algorithm – Backpropagation (BP)	A major focus of open research, many algorithms, and more to come
Evaluation:	On a black-box score – end performance	On almost every intermediate quantity
Implementation:	Many tricks	More or less standardized
Experiments:	Massive, real data (GT unknown)	Modest, often simulated data (GT known)



Graphical Models vs. Deep Nets

- So far:
 - Graphical models are representations of probability distributions
 - Neural networks are function approximators (with no probabilistic meaning)
- Some of the neural nets are in fact proper graphical models (i.e., units/neurons represent random variables):
 - Boltzmann machines (Hinton & Sejnowsky, 1983)
 - Restricted Boltzmann machines (Smolensky, 1986)
 - Learning and Inference in sigmoid belief networks (Neal, 1992)
 - Fast learning in deep belief networks (Hinton, Osindero, Teh, 2006)
 - Deep Boltzmann machines (Salakhutdinov and Hinton, 2009)



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.

- **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

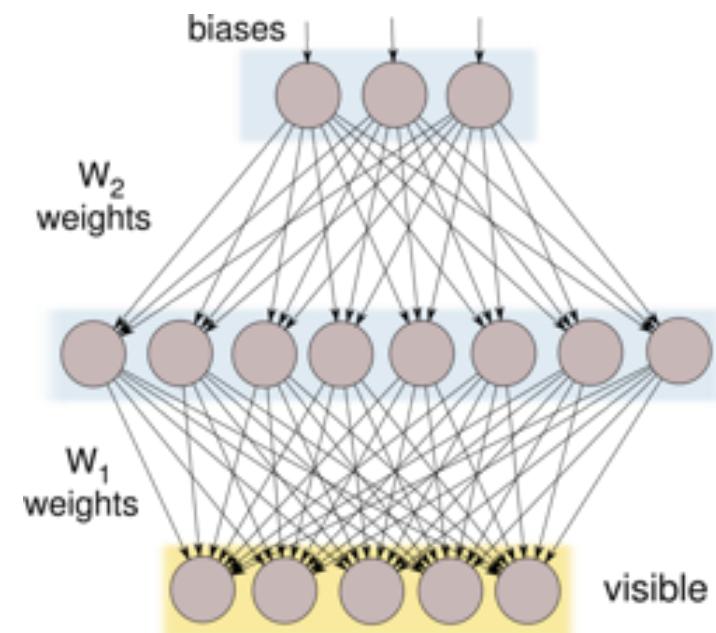


SIGMOID BELIEF NETWORK (SBN)

- 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!



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.

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

Contrastive Divergence Training

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

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

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

Another way to write it:

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

clamped / wake phase

$\uparrow\uparrow\uparrow$ conditioned hypotheses

unclamped / sleep / free phase

$\downarrow\downarrow\downarrow$ random fantasies

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



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:

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

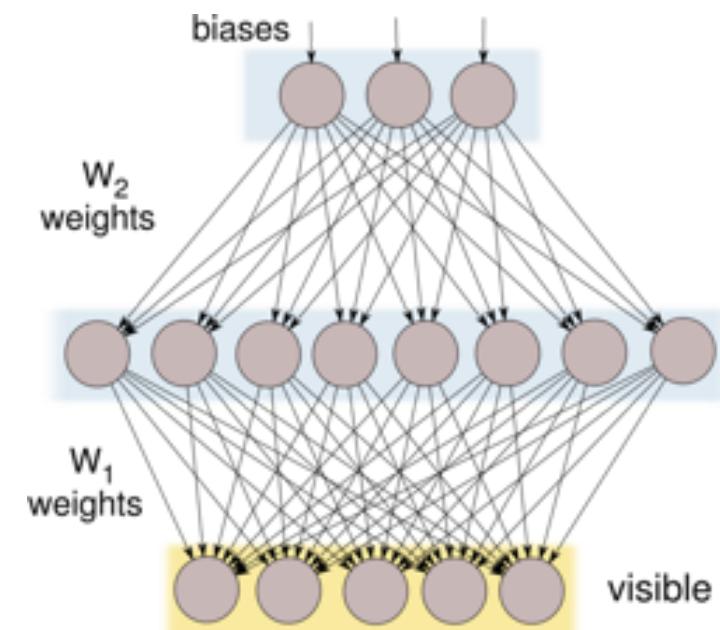
So this is a stochastic version of the EM algorithm, that you may have heard of. We iterate the following two steps:

E step: get samples from the posterior

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

- 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!

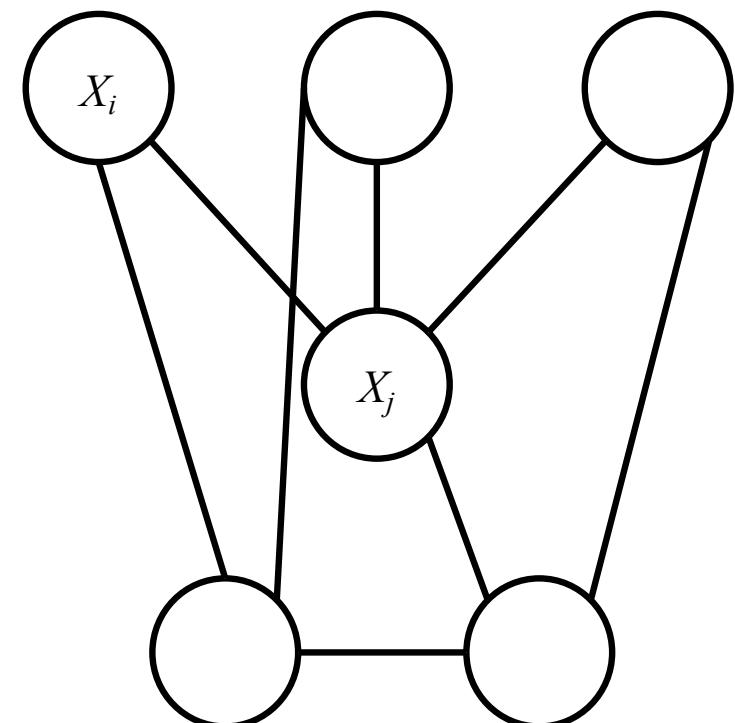


RESTRICTED BOLTZMAN MACHINE (RBM)

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

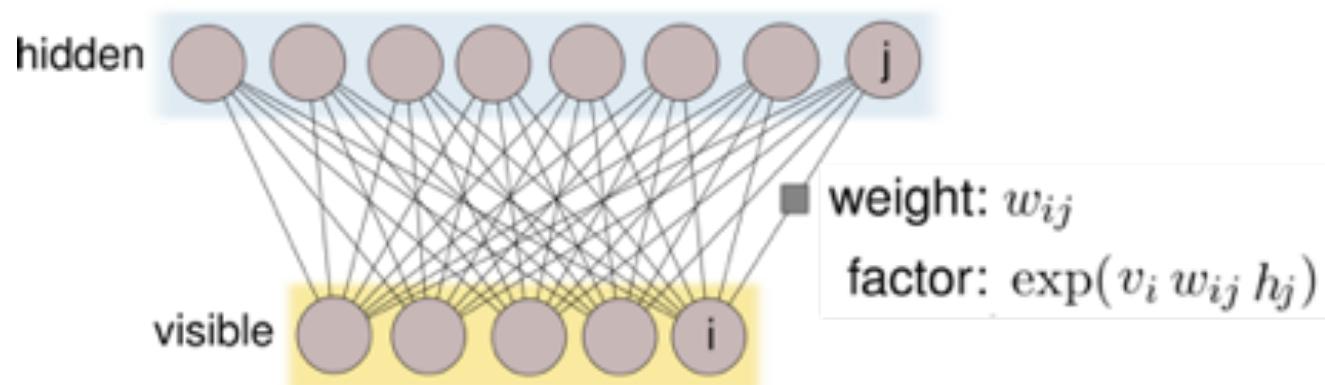
$$\psi_{ij}(x_i, x_j) = \exp(x_i W_{ij} x_j)$$

(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.

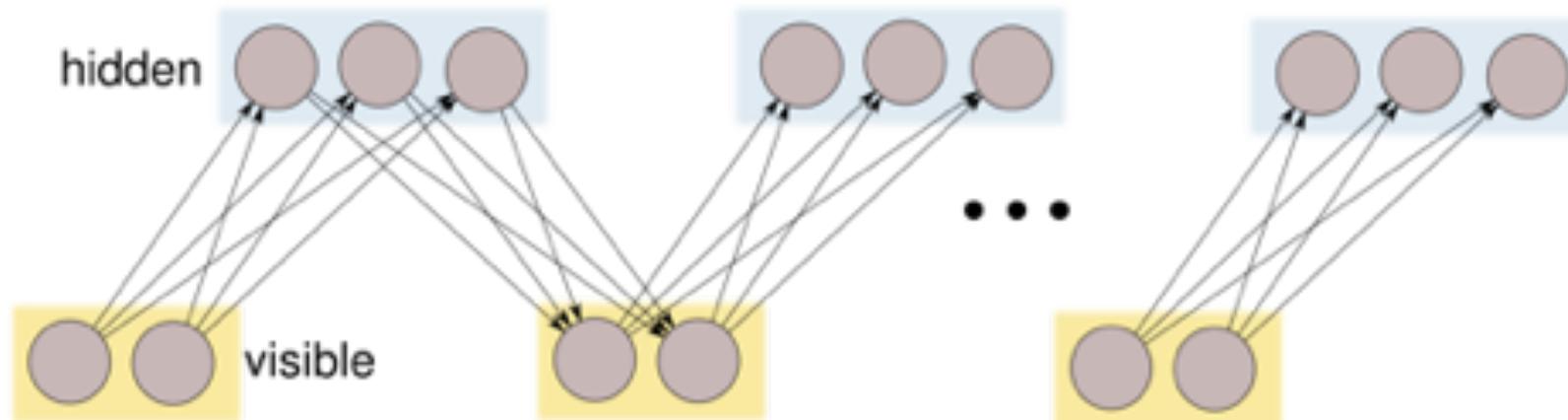


- $h_j \perp\!\!\!\perp h_k \mid \mathbf{v}$
- the posterior $P(\mathbf{h} \mid \mathbf{v})$ factors
c.f. in a belief net, the *prior* $P(\mathbf{h})$ factors
- no explaining away

Restricted Boltzmann Machines

Alternating Gibbs sampling

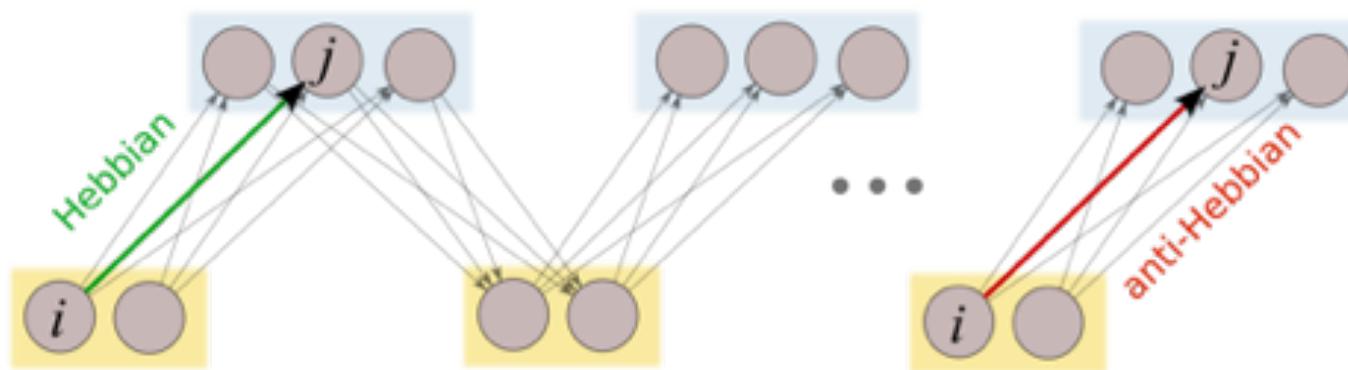
Since none of the units within a layer are interconnected, we can do Gibbs sampling by updating the whole layer at a time.



(with time running from left → right)

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

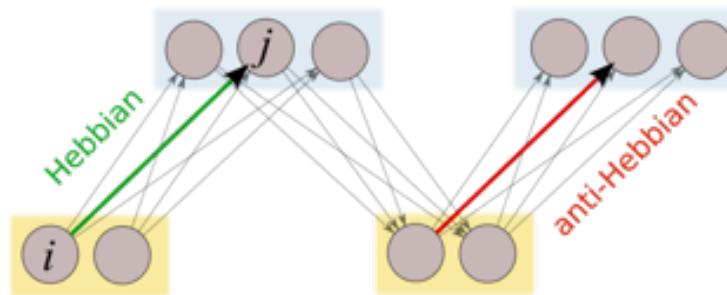
$$\Delta w_{ij} = \eta [\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^\infty]$$

restricted connectivity is trick #1:

it saves waiting for equilibrium in the clamped phase.

Restricted Boltzmann 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

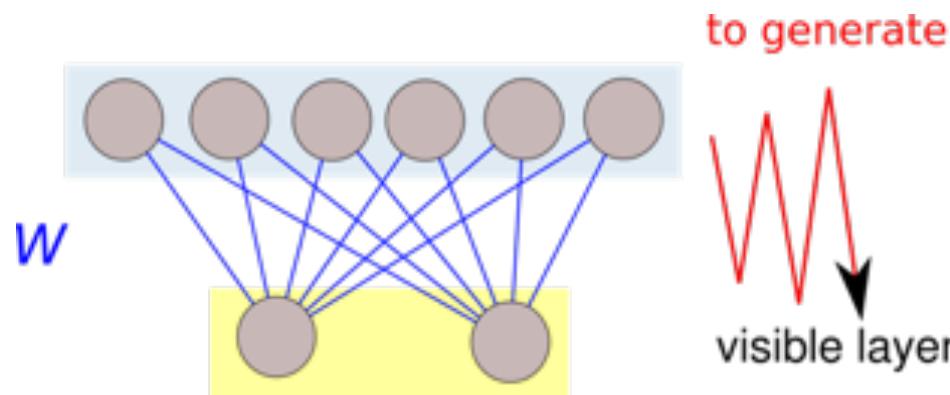
$$\Delta w_{ij} = \eta [\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1]$$

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

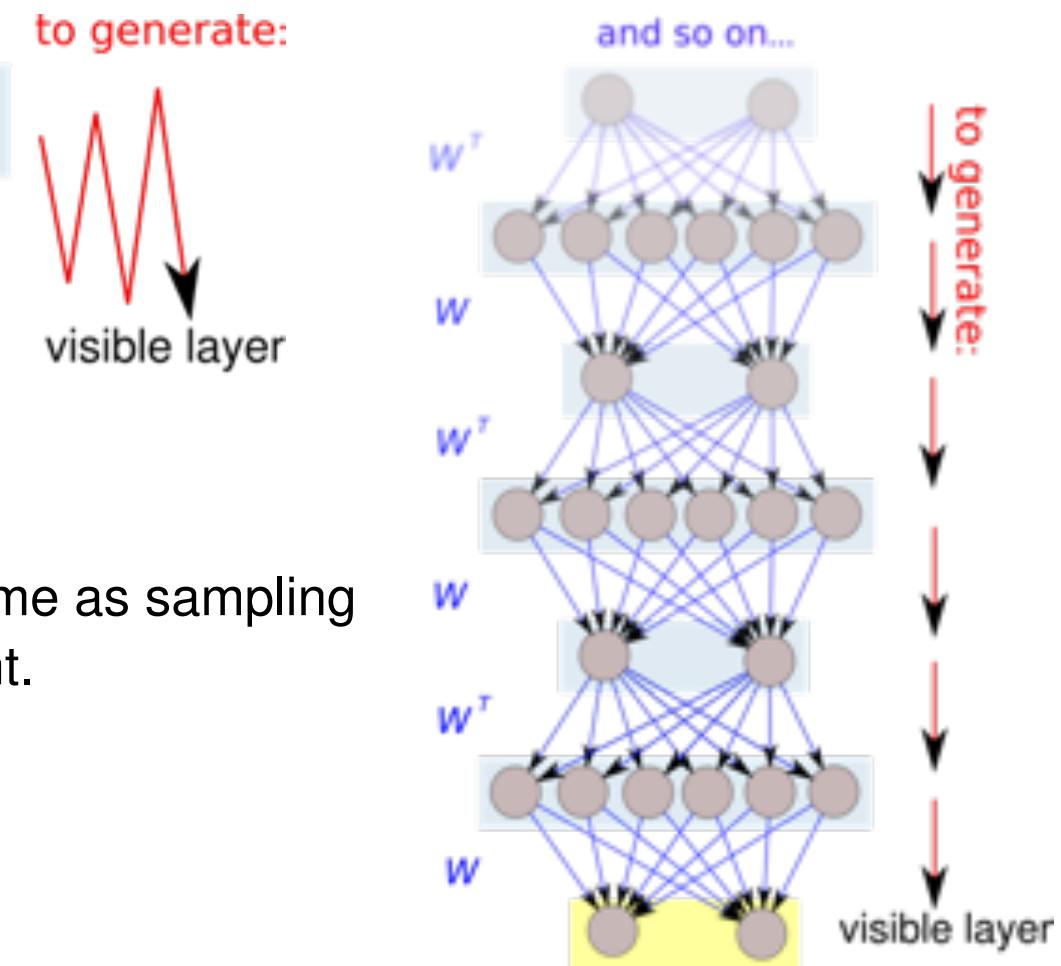
DEEP BELIEF NETWORKS (DBNS)

Deep Belief Networks (DBNs)

RBM^s are equivalent to infinitely deep belief networks

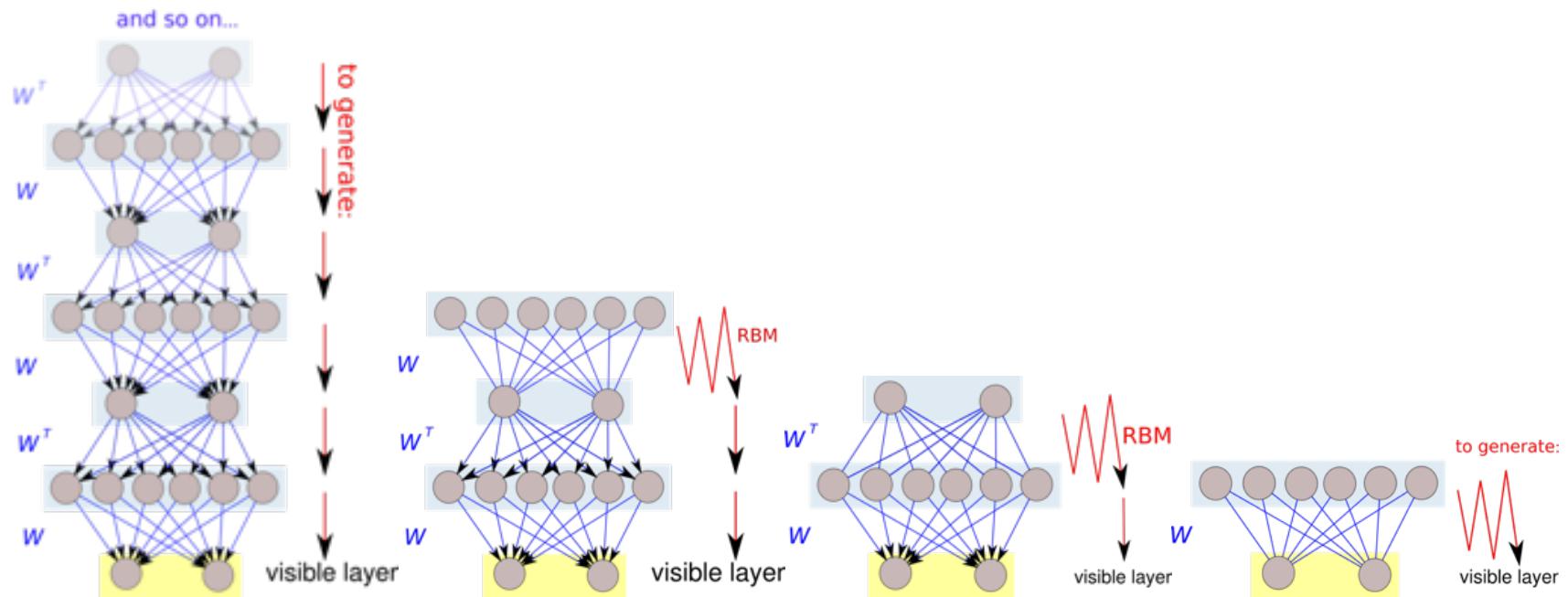


sampling from this is the same as sampling from the network on the right.



Deep Belief Networks (DBNs)

RBM^s are equivalent to infinitely deep belief networks

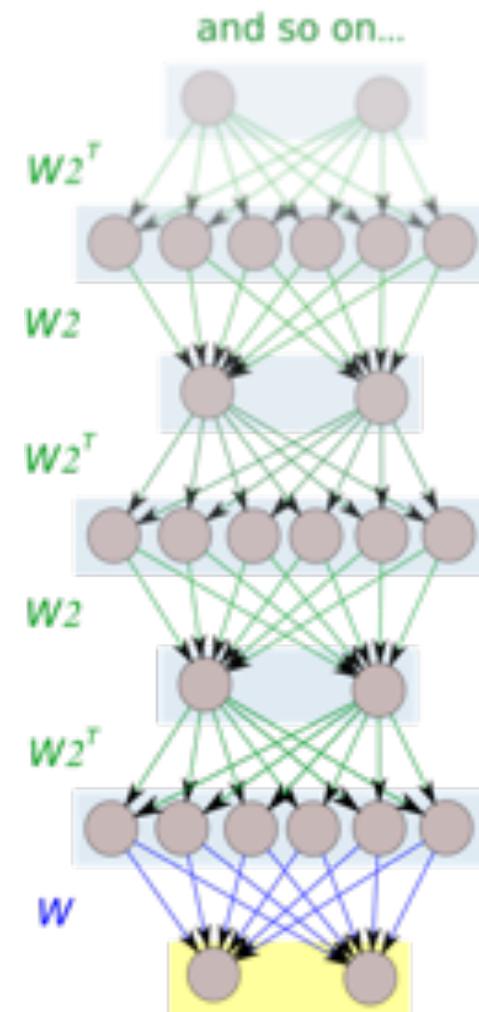
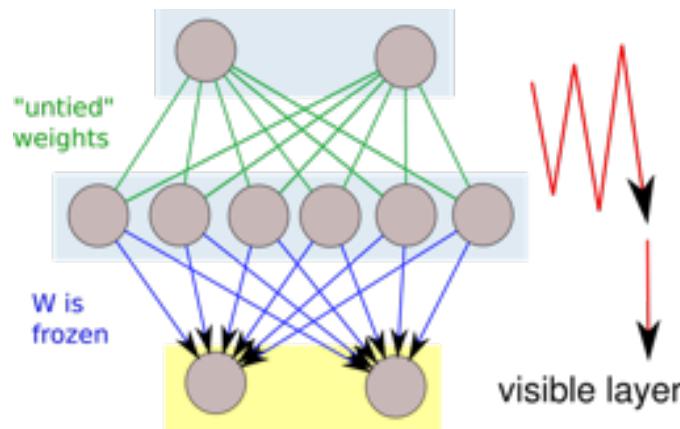


- So when we train an RBM, we're really training an ∞^{ly} deep sigmoid belief net!
- It's just that the weights of all layers are **tied**.

Deep Belief Networks (DBNs)

Un-tie the weights from layers 2 to infinity

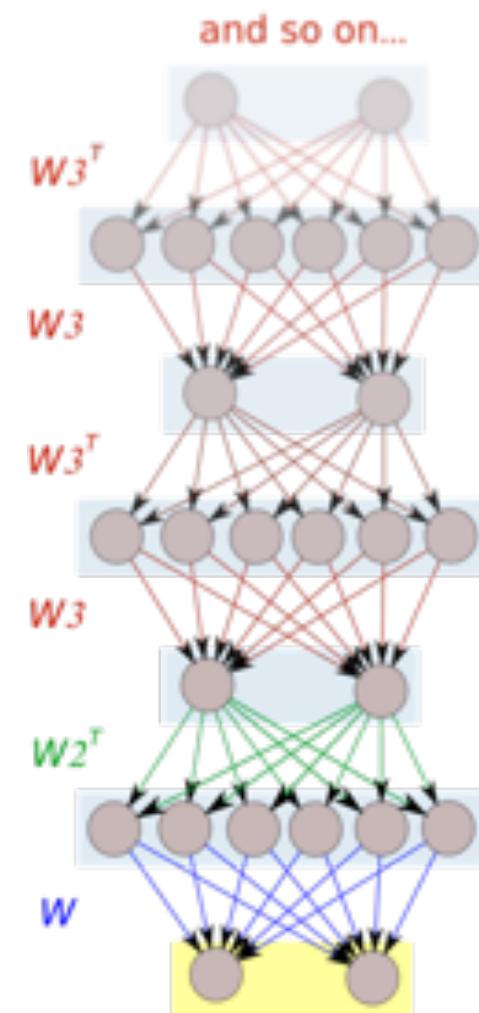
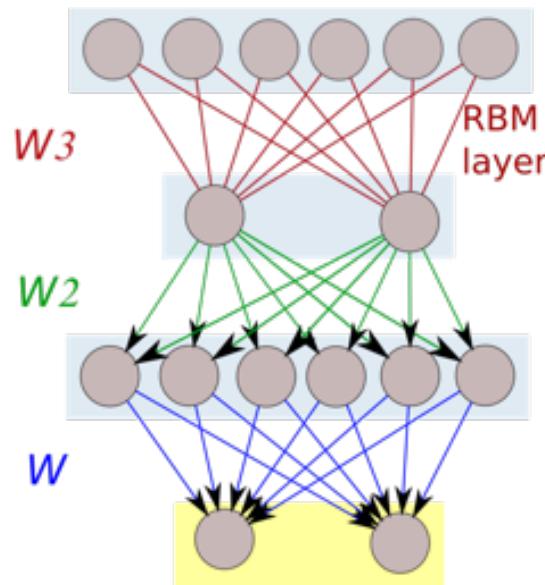
If we freeze the first RBM,
and then train another RBM
atop it, we are **untying** the
weights of layers 2+ in the ∞
net (which remain tied
together).



Deep Belief Networks (DBNs)

Un-tie the weights from layers 3 to infinity

and ditto for the 3rd layer...



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

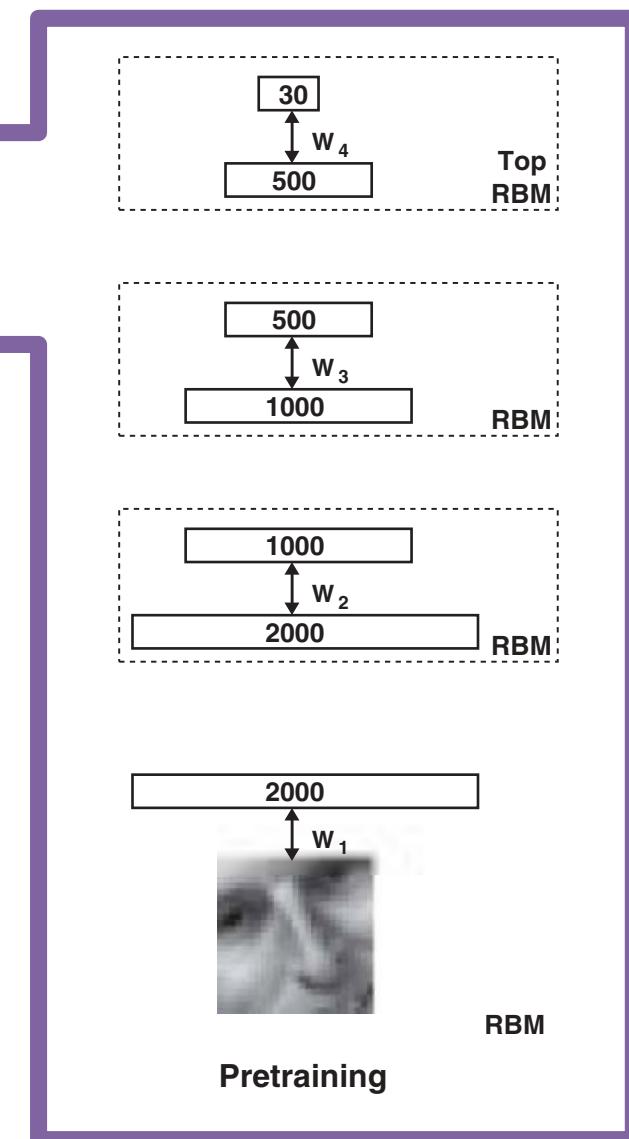
LEARNING DBNS

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

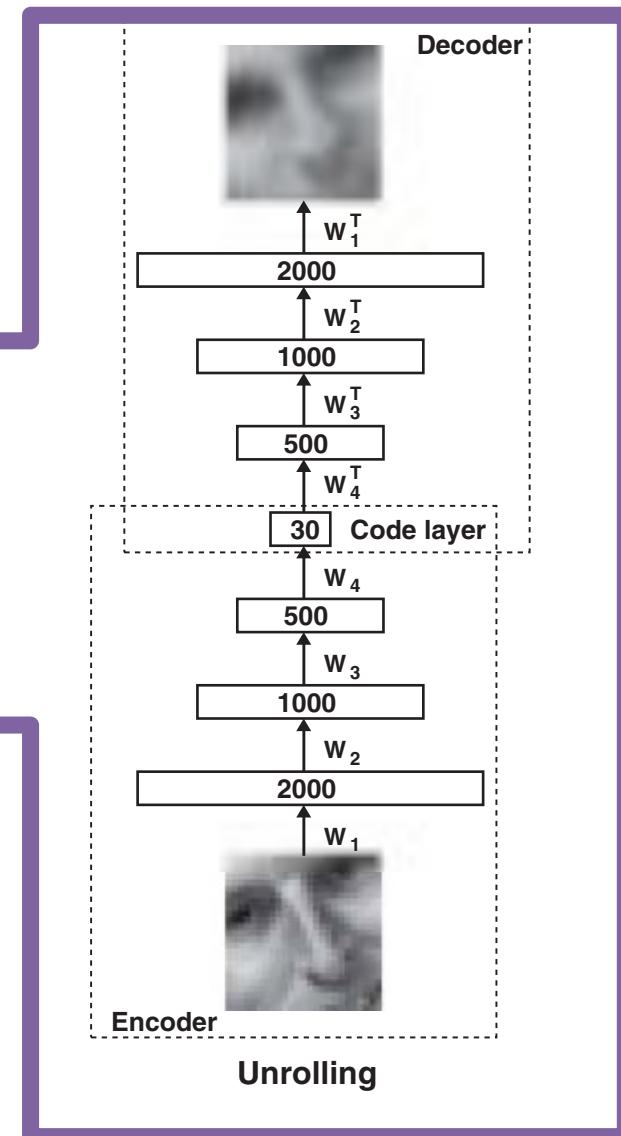
Setting A: DBN Autoencoder

- 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



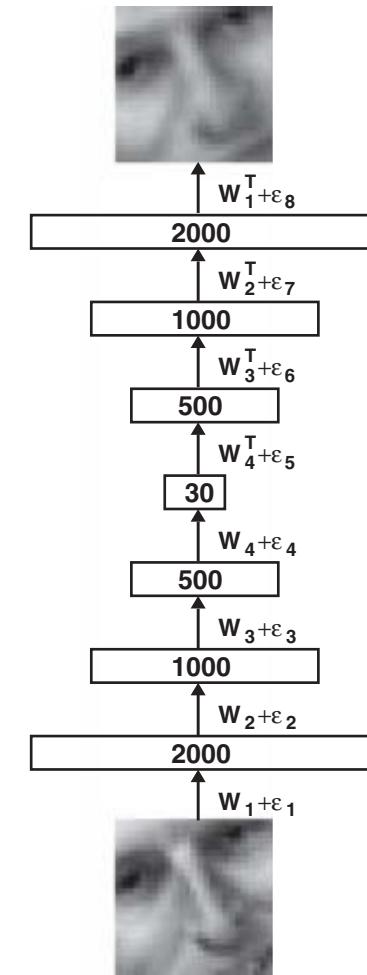
Setting A: DBN Autoencoder

- 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



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



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

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

- **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

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



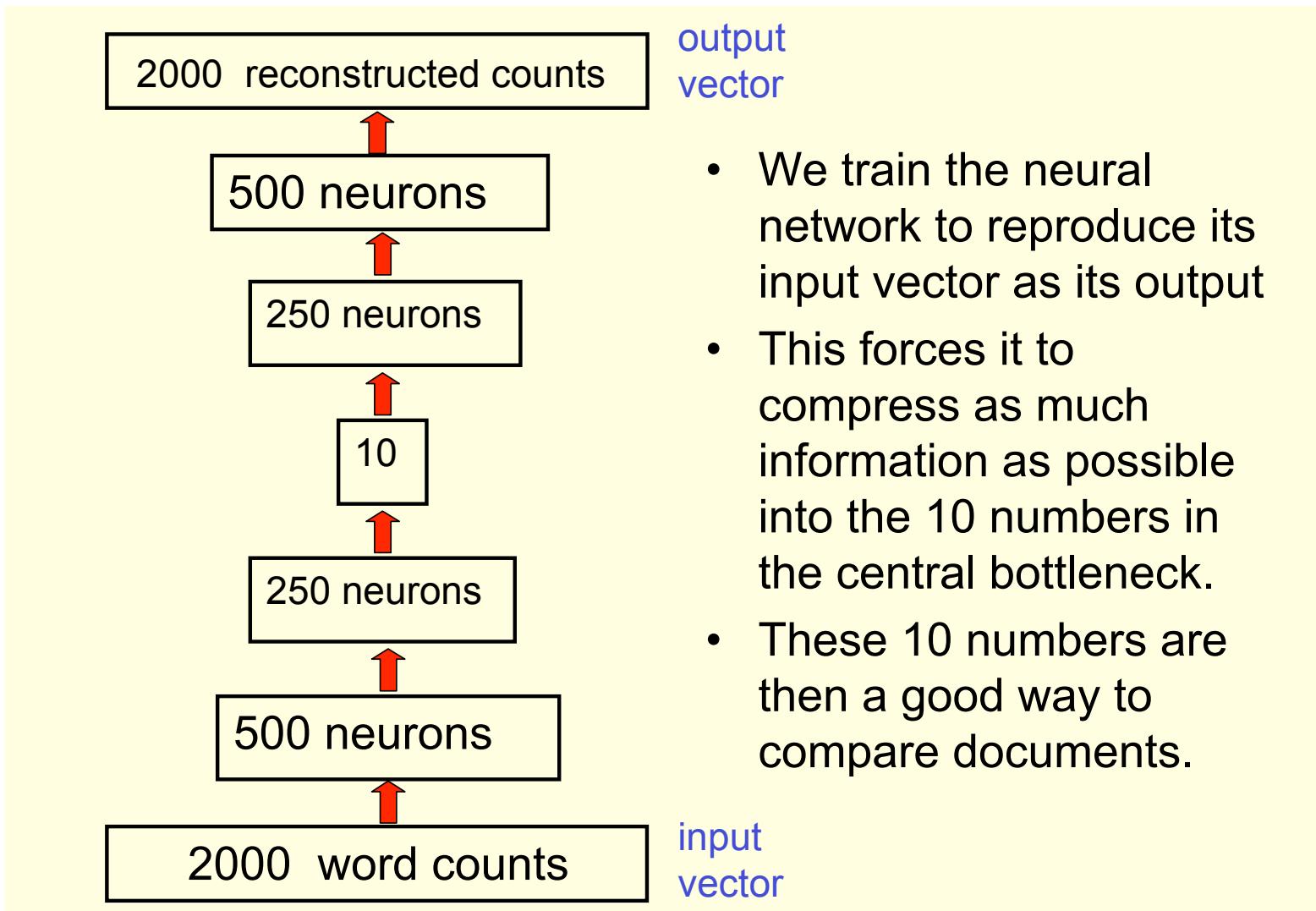
MNIST Digit Recognition

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.

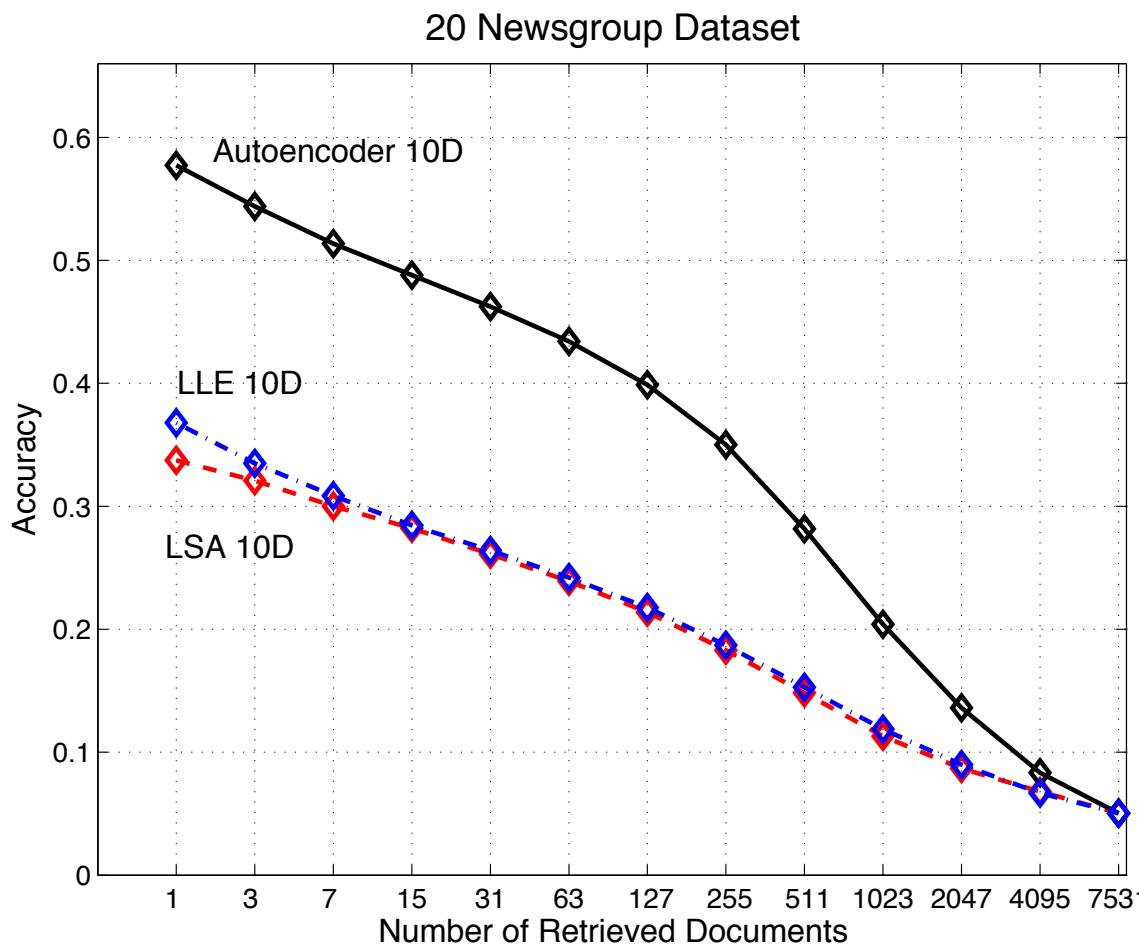
Document Clustering and Retrieval



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.

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