

10707

Deep Learning: Spring 2020

Andrej Risteski

Machine Learning Department

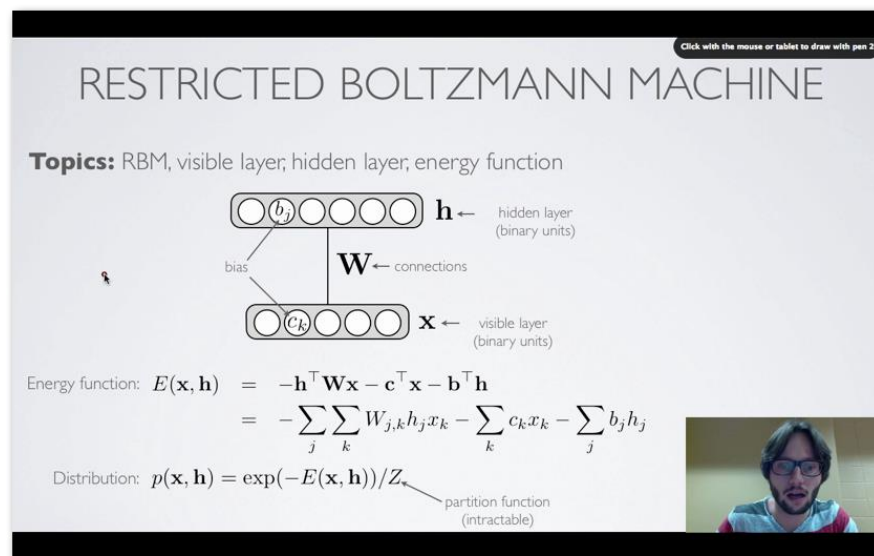
Lecture 13:

Learning energy models,
RBMs and DBNs

Disclaimer: Some of the material/slides for this lecture were borrowed from Hugo Larochelle's class on Neural Networks:

<https://sites.google.com/site/deeplearningsummerschool2016/>

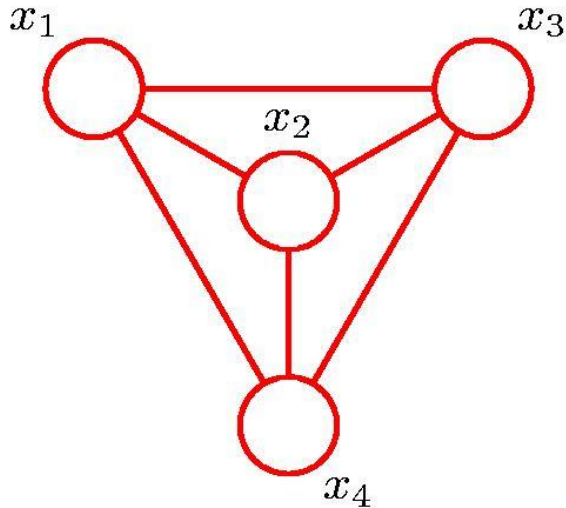
http://info.usherbrooke.ca/hlarochelle/neural_networks



Some are borrowed from Russ Salakhutdinov's offering of 10-707:
<https://deeplearning-cmu-10707.github.io/syllabus.html>

Graphical Models

Recall: **graph** contains a set of nodes connected by edges.



In a **probabilistic graphical model**, each node represents a random variable, links represent “probabilistic dependencies” between random variables.



Graph specifies how joint distribution over all random variables **decomposes** into a **product** of factors, each factor depending on a subset of the variables.

Two types of graphical models:



- **Bayesian networks**, also known as **Directed Graphical Models** (the links have a particular directionality indicated by the arrows)
- **Markov Random Fields**, also known as **Undirected Graphical Models** (the links do not carry arrows and have no directional significance).

Algorithmic pros/cons of latent-variable models (so far)

RBM's

- ⌘ Hard to draw samples 
(In fact, #P-hard provably, even in Ising models)
- ⌘ Easy to sample posterior distribution over latents 

Directed models

- ⌘ Easy to draw samples 
- ⌘ Hard to sample posterior distribution over latents 
(In fact, #P-hard even in mixtures)

Canonical tasks with graphical models

Inference

Given values for the parameters θ of the model, *sample/calculate* marginals (e.g. sample $p_\theta(x_1)$, $p_\theta(x_4, x_5)$, $p_\theta(z|x)$, etc.)

Learning

Find values for the parameters θ of the model, that give a *high likelihood* for the observed data. (e.g. canonical way is solving maximum likelihood optimization

$$\max_{\theta \in \Theta} \sum_{i=1}^n \log p(x_i)$$

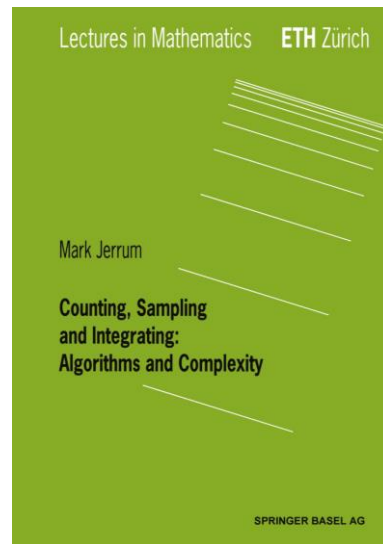
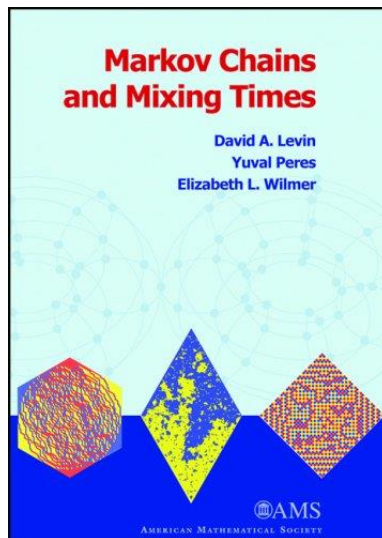
Other methods exist, e.g. method of moments (matching moments of model), but less used in deep learning practice.

Algorithmic approaches

When faced with a difficult to calculate probabilistic quantity (partition function, difficult posterior), there are two families of approaches:

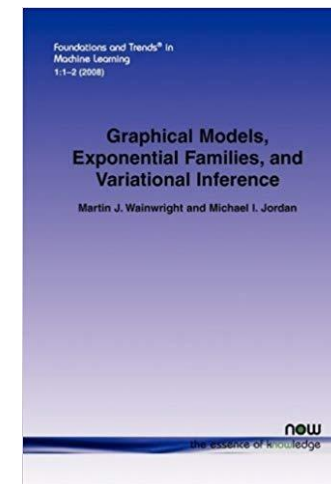
MARKOV CHAIN MONTE CARLO

- ❖ **Random walk** w/ equilibrium distribution the one we are trying to sample from.
- ❖ Well studied in TCS.



VARIATIONAL METHODS

- ❖ Based on solving an **optimization** problem.
- ❖ Very popular in practice.
- ❖ Comparatively poorly understood



Part I: Learning completely observable energy models

Learning energy models

Goal: Sample from distribution given up to constant of proportionality:

$$p_{\theta}(x) \propto \exp(-E_{\theta}(x))$$

Recall our basic approach: maximum likelihood

Given data x_1, x_2, \dots, x_n , solve the optimization problem

$$\max_{\theta \in \Theta} \sum_{i=1}^n \log p_{\theta}(x_i)$$

Expanding likelihoods: $\log p_{\theta}(x) = -E_{\theta}(x) - \log Z_{\theta}$

Our basic algorithm: gradient descent. Can we take gradients?

$\nabla_{\theta} E(\theta)$ is typically easy (e.g. $E(\theta)$ is a neural net, Ising model, etc.)

Learning energy models

Goal: Sample from distribution given up to constant of proportionality:

$$p_{\theta}(x) \propto \exp(-E_{\theta}(x))$$

Recall our basic approach: maximum likelihood

Given data x_1, x_2, \dots, x_n , solve the optimization problem

$$\max_{\theta \in \Theta} \sum_{i=1}^n \log p_{\theta}(x_i)$$

$$\begin{aligned} \nabla_{\theta} \log Z_{\theta} &= \frac{1}{Z_{\theta}} \nabla_{\theta} Z_{\theta} = \frac{1}{Z_{\theta}} \nabla_{\theta} \left(\int_x \exp(-E_{\theta}(x)) \right) \\ &= \frac{1}{Z_{\theta}} \int_x \exp(-E_{\theta}(x)) \nabla_{\theta} (-E_{\theta}(x)) = \mathbb{E}_{p_{\theta}}[-\nabla_{\theta} E_{\theta}(x)] \end{aligned}$$

Learning energy models

Goal: Sample from distribution given up to constant of proportionality:

$$p_{\theta}(x) \propto \exp(-E_{\theta}(x))$$

$$\nabla_{\theta} \left(\frac{1}{n} \sum_{i=1}^n \log p_{\theta}(x_i) \right) = \frac{1}{n} \left(\sum_i -\nabla_{\theta} E_{\theta}(x_i) \right) - \mathbb{E}_{p_{\theta}}[-\nabla_{\theta} E_{\theta}(x)]$$

$$\approx \underbrace{\mathbb{E}_{p_{data}}[-\nabla_{\theta} E_{\theta}(x)] - \mathbb{E}_{p_{\theta}}[-\nabla_{\theta} E_{\theta}(x)]}_{\text{Goal of the algorithm}}$$

Goal of the algorithm: Try to make the expectation of the energy match

Learning energy models

Let's assume $x \in \mathbb{R}^d$. An obvious way to produce the estimates $\mathbb{E}_{p_\theta}[-\nabla_\theta E_\theta(x)]$: sample approximately from p_θ using Langevin dynamics.

This was done recently: (Du, Mordatch '19), (Song, Ermon '19):

What's the difficulty?

Data is multimodal: Langevin might take long to mix.

Learning energy models

The algorithm from (Du, Mordatch '19)

Algorithm 1 Energy training algorithm

Input: data dist. $p_D(\mathbf{x})$, step size λ , number of steps

K

$\mathcal{B} \leftarrow \emptyset$

while not converged **do**

$\mathbf{x}_i^+ \sim p_D$

$\mathbf{x}_i^0 \sim \mathcal{B}$ with 95% probability and \mathcal{U} otherwise

▷ Generate sample from q_θ via Langevin dynamics:

for sample step $k = 1$ to K **do**

$\tilde{\mathbf{x}}^k \leftarrow \tilde{\mathbf{x}}^{k-1} - \nabla_{\mathbf{x}} E_\theta(\tilde{\mathbf{x}}^{k-1}) + \omega, \quad \omega \sim \mathcal{N}(0, \sigma)$

end for

$\mathbf{x}_i^- = \Omega(\tilde{\mathbf{x}}_i^k)$

▷ Optimize objective $\alpha \mathcal{L}_2 + \mathcal{L}_{ML}$ wrt θ :

$\Delta\theta \leftarrow \nabla_\theta \frac{1}{N} \sum_i \alpha (E_\theta(\mathbf{x}_i^+)^2 + E_\theta(\mathbf{x}_i^-)^2) + E_\theta(\mathbf{x}_i^+) - E_\theta(\mathbf{x}_i^-)$

Update θ based on $\Delta\theta$ using Adam optimizer

$\mathcal{B} \leftarrow \mathcal{B} \cup \tilde{\mathbf{x}}_i$

end while

Maintain buffer
of previous
samples to reduce
mixing time

W/ some probability,
start from random pt to
encourage mode
exploration

Langevin
sampler

A bit of l2
regularization to
ensure energy is
somewhat smooth

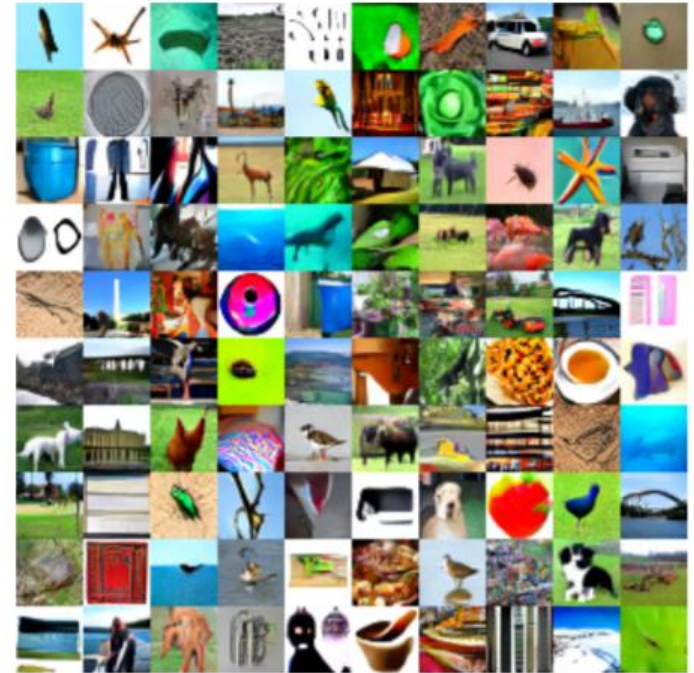


Figure 2: Conditional ImageNet32x32 EBM samples

Learning energy models

The algorithm from (Song, Ermon '19)

Not likelihood based, instead minimize:

$$\mathbb{E}_{p_{data}} ||\nabla_x \log p_{data}(x) - (-E_{\theta}(x))||^2$$

(Turns out to be writeable in a form friendly for taking gradients)

To alleviate multimodality, they use a variant of *simulated tempering*, which we saw last time.

They also use *smoothing by convolving* with Gaussians to account for points with bad estimates of $E_{\theta}(x)$.

Learning energy models

The algorithm from (Song, Ermon '19)

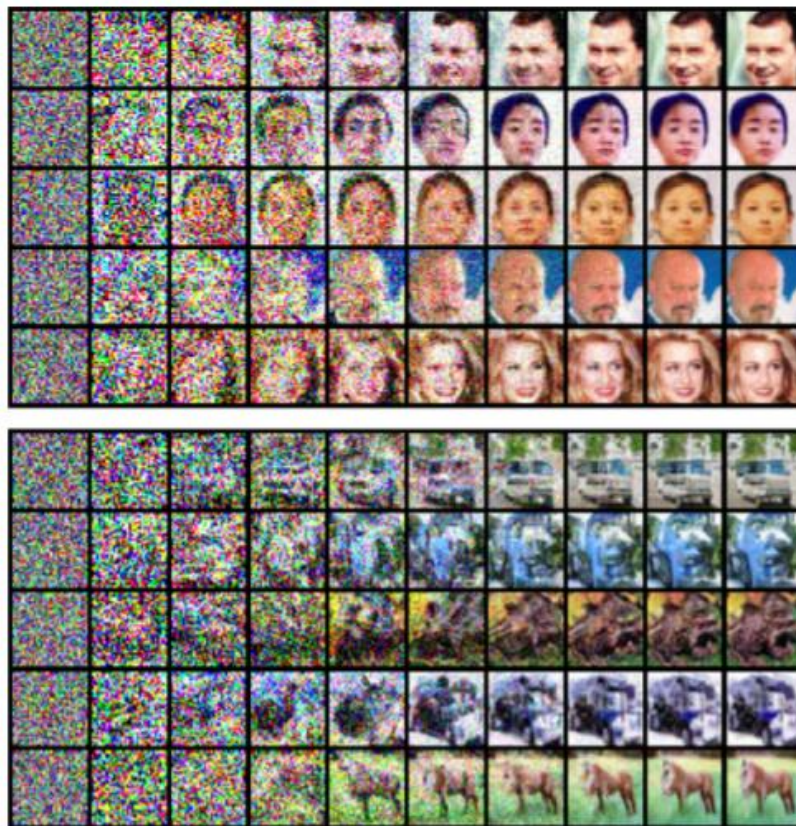


Figure 4: Intermediate samples of annealed Langevin dynamics.

Learning energy models

The algorithm from (Song, Ermon '19)

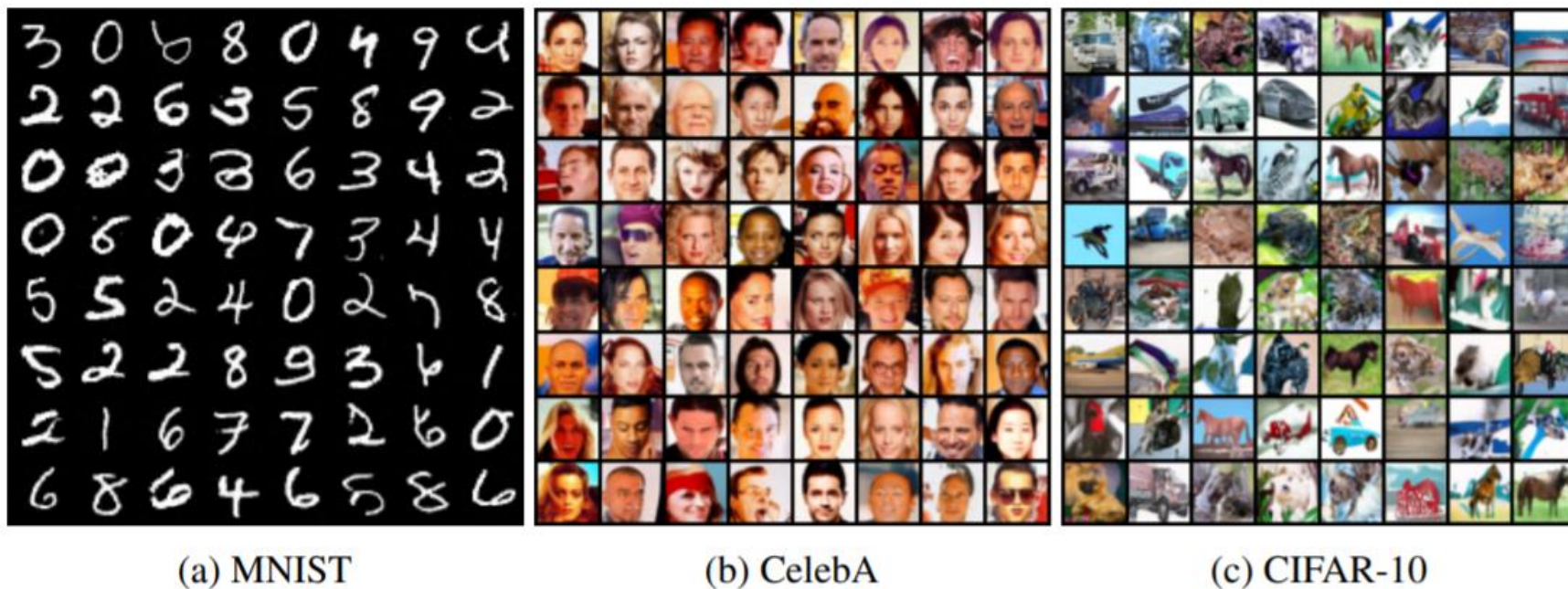


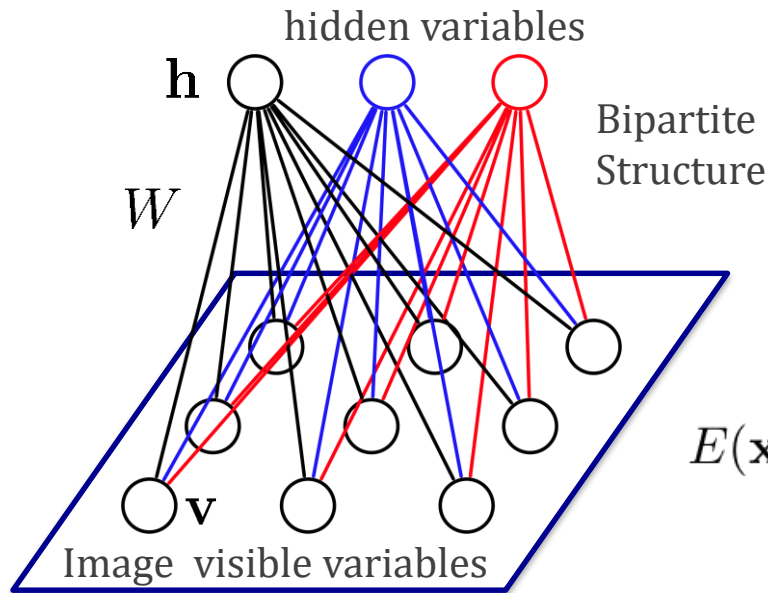
Figure 5: Uncurated samples on MNIST, CelebA, and CIFAR-10 datasets.

Part II: Learning Restricted Boltzmann Machines (RBMs)

Restricted Boltzmann Machines

An **undirected** latent-variable model

We denote visible and hidden variables with vectors \mathbf{v} , \mathbf{h} respectively:



Visible variables $\mathbf{x} \in \{0, 1\}^D$
are connected to hidden variables $\mathbf{h} \in \{0, 1\}^F$

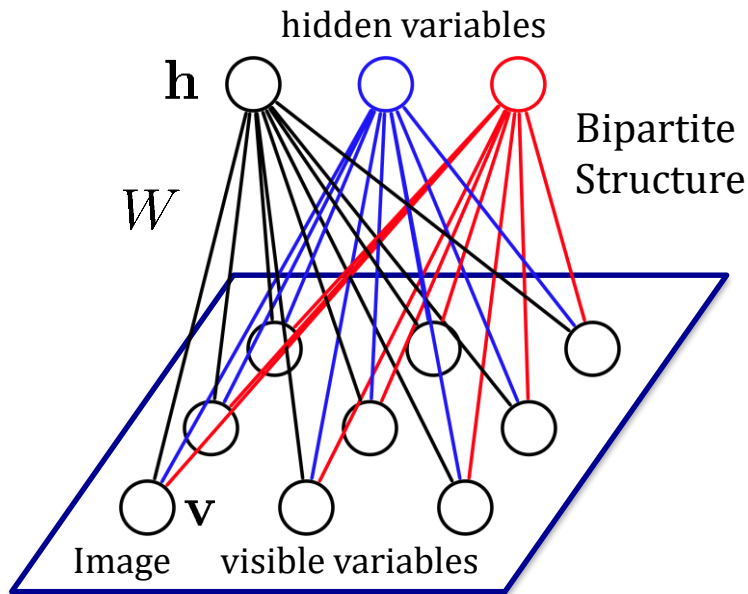
The energy of the joint configuration:

$$\begin{aligned} E(\mathbf{x}, \mathbf{h}) &= -\mathbf{h}^\top \mathbf{W} \mathbf{x} - \mathbf{c}^\top \mathbf{x} - \mathbf{b}^\top \mathbf{h} \\ &= -\sum_j \sum_k W_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j \end{aligned}$$

Probability of the joint configuration:

$$p(\mathbf{x}, \mathbf{h}) = \exp(-E(\mathbf{x}, \mathbf{h}))/Z$$

Restricted Boltzmann Machines



The **posterior** over the hidden variables is easy to sample from!
(Conditional independence!)

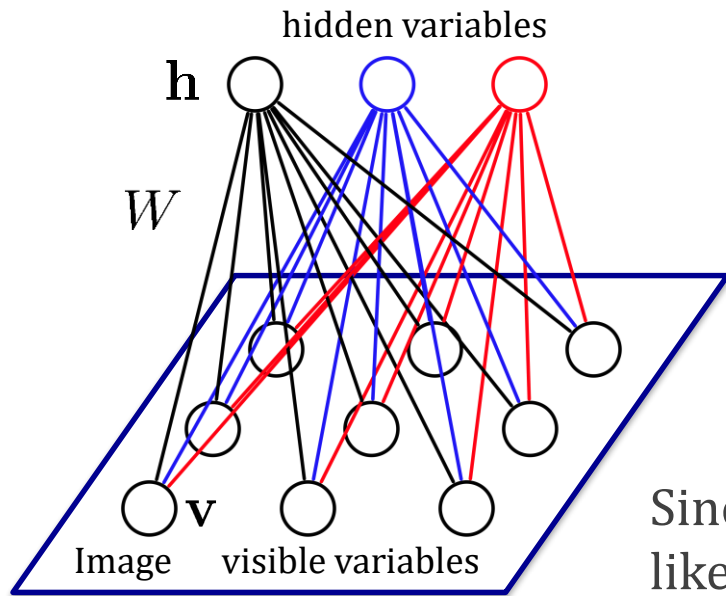
$$p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x}) \quad p(h_j = 1|\mathbf{x}) = \frac{1}{1 + \exp(-(b_j + \mathbf{W}_{j \cdot} \cdot \mathbf{x}))}$$

Factorizes

Similarly:

$$p(\mathbf{x}|\mathbf{h}) = \prod_k p(x_k|\mathbf{h}) \quad p(x_k = 1|\mathbf{h}) = \frac{1}{1 + \exp(-(c_k + \mathbf{h}^\top \mathbf{W}_{\cdot k}))}$$

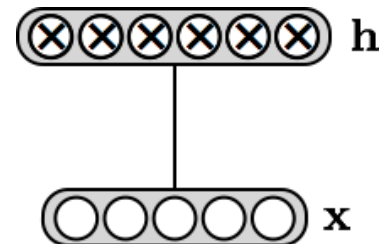
How to learn RBM's



Given data x_1, x_2, \dots, x_n , solve

$$\max_{\theta \in \Theta} \sum_{i=1}^n \log p_{\theta}(x_i)$$

Since we have latent variables, we need to express the likelihood when we marginalize out the latents:



How to learn RBM's

$$p(\mathbf{x}) = \sum_{\mathbf{h} \in \{0,1\}^H} \exp(\mathbf{h}^\top \mathbf{W} \mathbf{x} + \mathbf{c}^\top \mathbf{x} + \mathbf{b}^\top \mathbf{h}) / Z$$

How to learn RBM's

$$\begin{aligned} p(\mathbf{x}) &= \sum_{\mathbf{h} \in \{0,1\}^H} \exp(\mathbf{h}^\top \mathbf{W} \mathbf{x} + \mathbf{c}^\top \mathbf{x} + \mathbf{b}^\top \mathbf{h}) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_H \in \{0,1\}} \exp \left(\sum_j h_j \mathbf{W}_{j \cdot} \mathbf{x} + b_j h_j \right) / Z \end{aligned}$$

How to learn RBM's

$$\begin{aligned} p(\mathbf{x}) &= \sum_{\mathbf{h} \in \{0,1\}^H} \exp(\mathbf{h}^\top \mathbf{W} \mathbf{x} + \mathbf{c}^\top \mathbf{x} + \mathbf{b}^\top \mathbf{h}) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_H \in \{0,1\}} \exp \left(\sum_j h_j \mathbf{W}_{j \cdot} \mathbf{x} + b_j h_j \right) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \left(\sum_{h_1 \in \{0,1\}} \exp(h_1 \mathbf{W}_{1 \cdot} \mathbf{x} + b_1 h_1) \right) \cdots \left(\sum_{h_H \in \{0,1\}} \exp(h_H \mathbf{W}_{H \cdot} \mathbf{x} + b_H h_H) \right) / Z \end{aligned}$$

How to learn RBM's

$$\begin{aligned} p(\mathbf{x}) &= \sum_{\mathbf{h} \in \{0,1\}^H} \exp(\mathbf{h}^\top \mathbf{W} \mathbf{x} + \mathbf{c}^\top \mathbf{x} + \mathbf{b}^\top \mathbf{h}) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_H \in \{0,1\}} \exp \left(\sum_j h_j \mathbf{W}_{j \cdot} \mathbf{x} + b_j h_j \right) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \left(\sum_{h_1 \in \{0,1\}} \exp(h_1 \mathbf{W}_{1 \cdot} \mathbf{x} + b_1 h_1) \right) \cdots \left(\sum_{h_H \in \{0,1\}} \exp(h_H \mathbf{W}_{H \cdot} \mathbf{x} + b_H h_H) \right) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) (1 + \exp(b_1 + \mathbf{W}_{1 \cdot} \mathbf{x})) \cdots (1 + \exp(b_H + \mathbf{W}_{H \cdot} \mathbf{x})) / Z \end{aligned}$$

How to learn RBM's

$$\begin{aligned} p(\mathbf{x}) &= \sum_{\mathbf{h} \in \{0,1\}^H} \exp(\mathbf{h}^\top \mathbf{W} \mathbf{x} + \mathbf{c}^\top \mathbf{x} + \mathbf{b}^\top \mathbf{h}) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_H \in \{0,1\}} \exp \left(\sum_j h_j \mathbf{W}_{j \cdot} \mathbf{x} + b_j h_j \right) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \left(\sum_{h_1 \in \{0,1\}} \exp(h_1 \mathbf{W}_{1 \cdot} \mathbf{x} + b_1 h_1) \right) \cdots \left(\sum_{h_H \in \{0,1\}} \exp(h_H \mathbf{W}_{H \cdot} \mathbf{x} + b_H h_H) \right) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) (1 + \exp(b_1 + \mathbf{W}_{1 \cdot} \mathbf{x})) \cdots (1 + \exp(b_H + \mathbf{W}_{H \cdot} \mathbf{x})) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \exp(\log(1 + \exp(b_1 + \mathbf{W}_{1 \cdot} \mathbf{x}))) \cdots \exp(\log(1 + \exp(b_H + \mathbf{W}_{H \cdot} \mathbf{x}))) / Z \end{aligned}$$

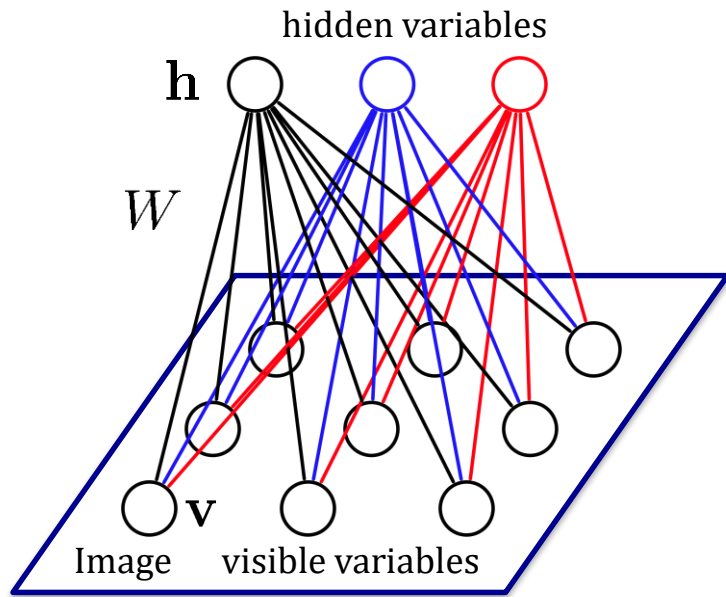
How to learn RBM's

$$\begin{aligned} p(\mathbf{x}) &= \sum_{\mathbf{h} \in \{0,1\}^H} \exp(\mathbf{h}^\top \mathbf{W} \mathbf{x} + \mathbf{c}^\top \mathbf{x} + \mathbf{b}^\top \mathbf{h}) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_H \in \{0,1\}} \exp \left(\sum_j h_j \mathbf{W}_{j \cdot} \mathbf{x} + b_j h_j \right) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \left(\sum_{h_1 \in \{0,1\}} \exp(h_1 \mathbf{W}_{1 \cdot} \mathbf{x} + b_1 h_1) \right) \cdots \left(\sum_{h_H \in \{0,1\}} \exp(h_H \mathbf{W}_{H \cdot} \mathbf{x} + b_H h_H) \right) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) (1 + \exp(b_1 + \mathbf{W}_{1 \cdot} \mathbf{x})) \cdots (1 + \exp(b_H + \mathbf{W}_{H \cdot} \mathbf{x})) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \exp(\log(1 + \exp(b_1 + \mathbf{W}_{1 \cdot} \mathbf{x}))) \cdots \exp(\log(1 + \exp(b_H + \mathbf{W}_{H \cdot} \mathbf{x}))) / Z \\ &= \exp \left(\underbrace{\mathbf{c}^\top \mathbf{x} + \sum_{j=1}^H \log(1 + \exp(b_j + \mathbf{W}_{j \cdot} \mathbf{x}))}_{= F(\mathbf{x})} \right) / Z \end{aligned}$$

How to learn RBM's

$$\begin{aligned} p(\mathbf{x}) &= \sum_{\mathbf{h} \in \{0,1\}^H} \exp(\mathbf{h}^\top \mathbf{W} \mathbf{x} + \mathbf{c}^\top \mathbf{x} + \mathbf{b}^\top \mathbf{h}) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_H \in \{0,1\}} \exp \left(\sum_j h_j \mathbf{W}_{j \cdot} \mathbf{x} + b_j h_j \right) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \left(\sum_{h_1 \in \{0,1\}} \exp(h_1 \mathbf{W}_{1 \cdot} \mathbf{x} + b_1 h_1) \right) \cdots \left(\sum_{h_H \in \{0,1\}} \exp(h_H \mathbf{W}_{H \cdot} \mathbf{x} + b_H h_H) \right) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) (1 + \exp(b_1 + \mathbf{W}_{1 \cdot} \mathbf{x})) \cdots (1 + \exp(b_H + \mathbf{W}_{H \cdot} \mathbf{x})) / Z \\ &= \exp(\mathbf{c}^\top \mathbf{x}) \exp(\log(1 + \exp(b_1 + \mathbf{W}_{1 \cdot} \mathbf{x}))) \cdots \exp(\log(1 + \exp(b_H + \mathbf{W}_{H \cdot} \mathbf{x}))) / Z \\ &= \exp \left(\underbrace{\mathbf{c}^\top \mathbf{x} + \sum_{j=1}^H \log(1 + \exp(b_j + \mathbf{W}_{j \cdot} \mathbf{x}))}_{= F(\mathbf{x})} \right) / Z \\ &= \exp(F(\mathbf{x})) / Z \end{aligned}$$

How to learn RBM's



Given data x_1, x_2, \dots, x_n , solve

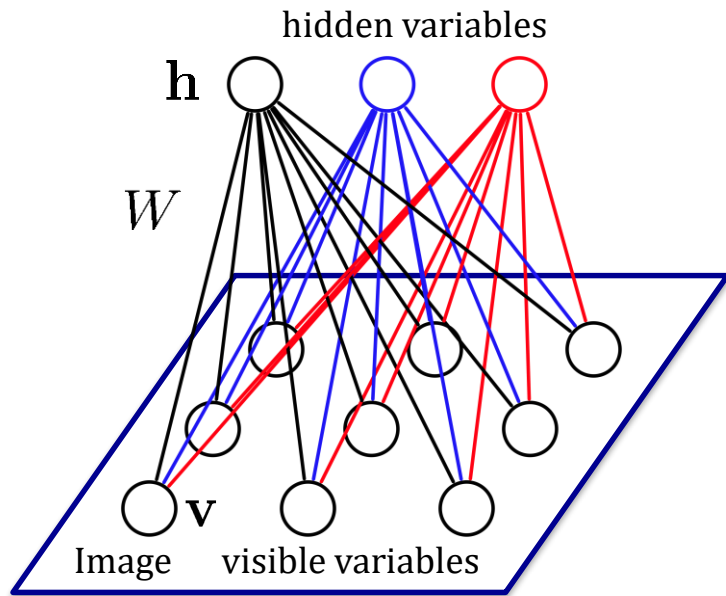
$$\max_{\theta \in \Theta} \sum_{i=1}^n \log p_{\theta}(x_i)$$

With this reduction, the undirected model calculations imply:

$$\nabla_{\theta} \left(\frac{1}{n} \sum_{i=1}^n \log p_{\theta}(x_i) \right) = \frac{1}{n} \left(\sum_i -\nabla_{\theta} F_{\theta}(x_i) \right) - \mathbb{E}_{p_{\theta}}[-\nabla_{\theta} F_{\theta}(x)]$$

$$\begin{aligned} \nabla_{\mathbf{w}_{ij}} F_{\theta}(\mathbf{x}) &= \nabla_{\mathbf{w}_{ij}} (\mathbf{c}^T \mathbf{x} + \sum_{j=1}^H \log(1 + \exp(\mathbf{b}_j + \mathbf{w}_j \cdot \mathbf{x}))) = \frac{\exp(\mathbf{b}_j + \mathbf{w}_j \cdot \mathbf{x})}{1 + \exp(\mathbf{b}_j + \mathbf{w}_j \cdot \mathbf{x})} \mathbf{x}_i \\ &= \frac{1}{1 + \exp(-(b_j + \mathbf{w}_j \cdot \mathbf{x}))} \mathbf{x}_i = P(\mathbf{h}_j = 1 | \mathbf{x}) \mathbf{x}_i \end{aligned}$$

How to learn RBM's



Given data x_1, x_2, \dots, x_n , solve

$$\max_{\theta \in \Theta} \sum_{i=1}^n \log p_{\theta}(x_i)$$

With this reduction, the undirected model calculations imply:

$$\nabla_{\theta} \left(\frac{1}{n} \sum_{i=1}^n \log p_{\theta}(x_i) \right) = \frac{1}{n} \left(\sum_i -\nabla_{\theta} F_{\theta}(x_i) \right) - \mathbb{E}_{p_{\theta}}[-\nabla_{\theta} F_{\theta}(x)]$$

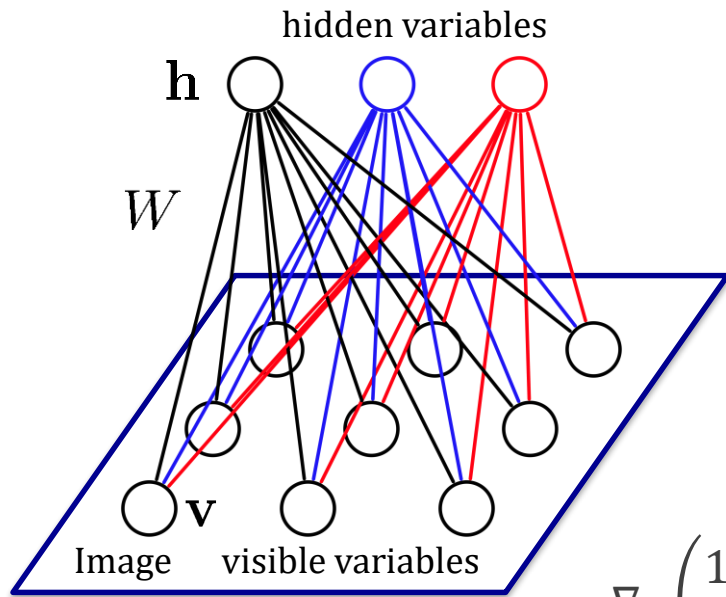
$$\nabla_{W_{ij}} F_{\theta}(\mathbf{x}) = P(\mathbf{h}_j = 1 | \mathbf{x}) \mathbf{x}_i \Rightarrow \nabla_{\mathbf{W}} F_{\theta}(\mathbf{x}) = \mathbf{h}(\mathbf{x}) \mathbf{x}^T$$

$$\nabla_{\mathbf{b}} F_{\theta}(\mathbf{x}) = \mathbf{h}(\mathbf{x})$$

$$\nabla_{\mathbf{c}} F_{\theta}(\mathbf{x}) = \mathbf{x}$$

$$\begin{aligned} \mathbf{h}(\mathbf{x}) &\stackrel{\text{def}}{=} \begin{pmatrix} p(h_1=1|\mathbf{x}) \\ \vdots \\ p(h_H=1|\mathbf{x}) \end{pmatrix} \\ &= \text{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x}) \end{aligned}$$

How to learn RBM's



Given data x_1, x_2, \dots, x_n , solve

$$\max_{\theta \in \Theta} \sum_{i=1}^n \log p_{\theta}(x_i)$$

$$\nabla_{\theta} \left(\frac{1}{n} \sum_{i=1}^n \log p_{\theta}(x_i) \right) = \frac{1}{n} \left(\sum_i -\nabla_{\theta} F_{\theta}(x_i) \right) - \mathbb{E}_{p_{\theta}}[-\nabla_{\theta} F_{\theta}(x)]$$

The hard term is again: $\mathbb{E}_{p_{\theta}}[-\nabla_{\theta} E_{\theta}(x)]$ --- we need to draw samples from p_{θ}

We will draw samples using a Markov random walk: **Gibbs sampler!**

Gibbs sampling

Consider sampling a distribution over n variables $\mathbf{x} = (x_1, x_2, \dots, x_n)$, s.t. each of the conditional distributions $P(x_i | \mathbf{x}_{-i})$ is easy to sample. :

A common way to do this is using **Gibbs sampling**:

Repeat:

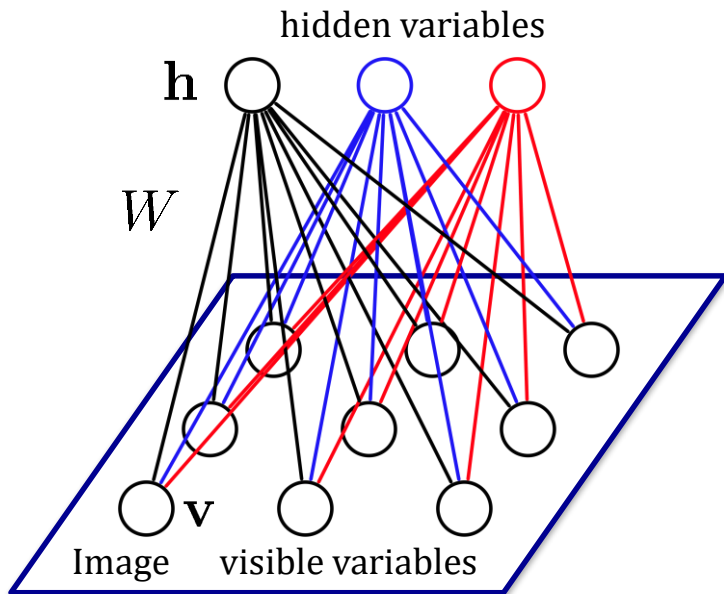
Let current state be $\mathbf{x} = (x_1, x_2, \dots, x_n)$

Pick $i \in [n]$ uniformly at random.

Sample $x \sim P(X_i = x | \mathbf{x}_{-i})$

Update state to $\mathbf{y} = (x_1, x_2, \dots, x_{i-1}, x, x_{i+1}, \dots, x_n)$

Gibbs sampling for RBM's

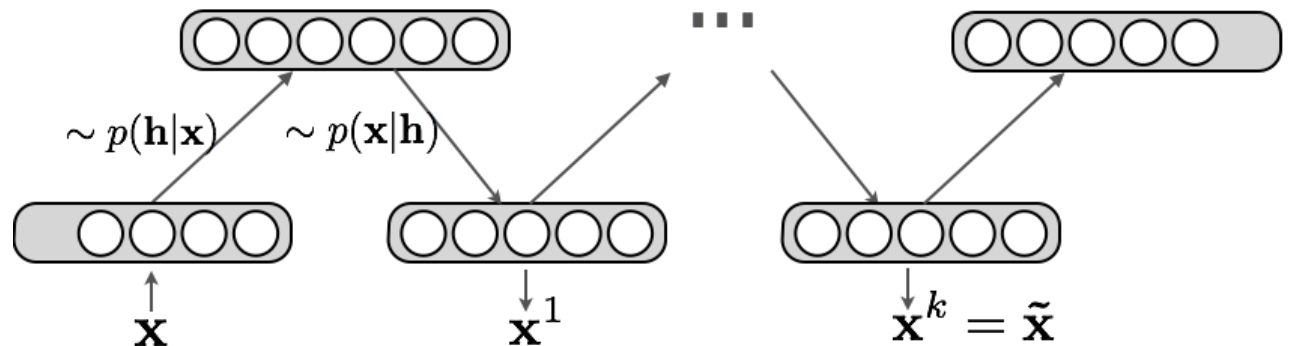


Repeat:

Sample $\mathbf{h} \sim P(\mathbf{h}|\mathbf{v})$

Sample $\mathbf{v} \sim P(\mathbf{v}|\mathbf{h})$

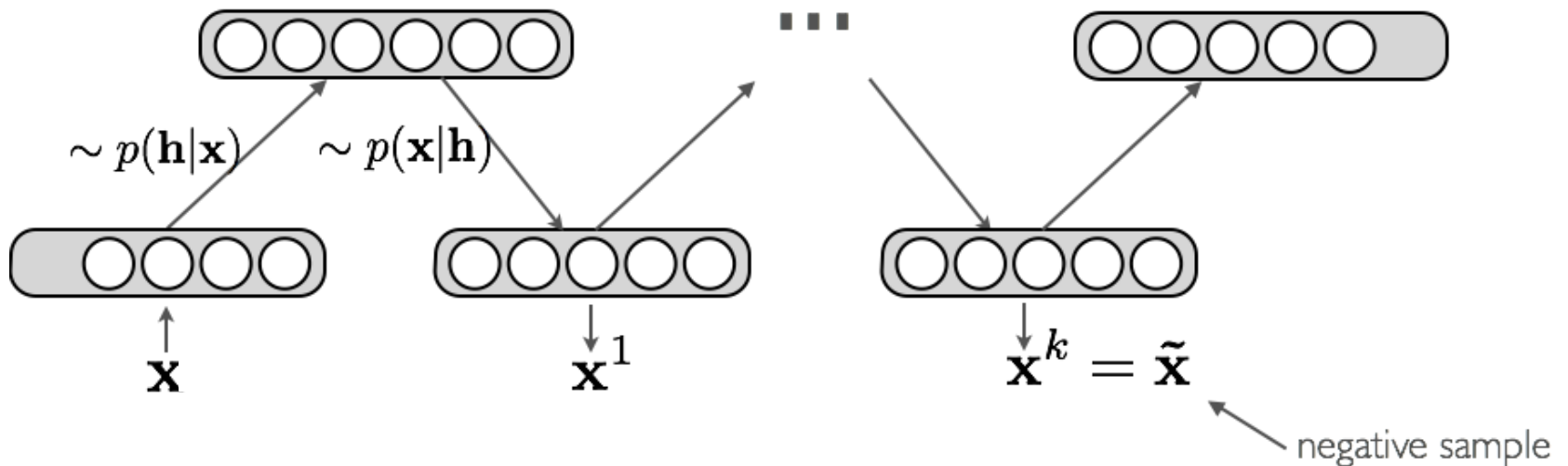
Pictorially:



Contrastive Divergence

Key idea behind Contrastive Divergence:

- Replace the expectation by a point estimate at $\tilde{\mathbf{x}}$
- Obtain the point $\tilde{\mathbf{x}}$ by Gibbs sampling
- Start sampling chain at \mathbf{x}



k is often taken to be just 1.

CD-k Algorithm

For each training example \mathbf{x}

- Generate a negative sample $\tilde{\mathbf{x}}$ using k steps of Gibbs sampling, starting at the data point \mathbf{x}
- Update model parameters:

$$\begin{aligned}\mathbf{W} &\Leftarrow \mathbf{W} + \alpha \left(\mathbf{h}(\mathbf{x}) \mathbf{x}^\top - \mathbf{h}(\tilde{\mathbf{x}}) \tilde{\mathbf{x}}^\top \right) \\ \mathbf{b} &\Leftarrow \mathbf{b} + \alpha \left(\mathbf{h}(\mathbf{x}) - \mathbf{h}(\tilde{\mathbf{x}}) \right) \\ \mathbf{c} &\Leftarrow \mathbf{c} + \alpha \left(\mathbf{x} - \tilde{\mathbf{x}} \right)\end{aligned}$$

Gradients we derived before

- Go back to 1 until stopping criteria

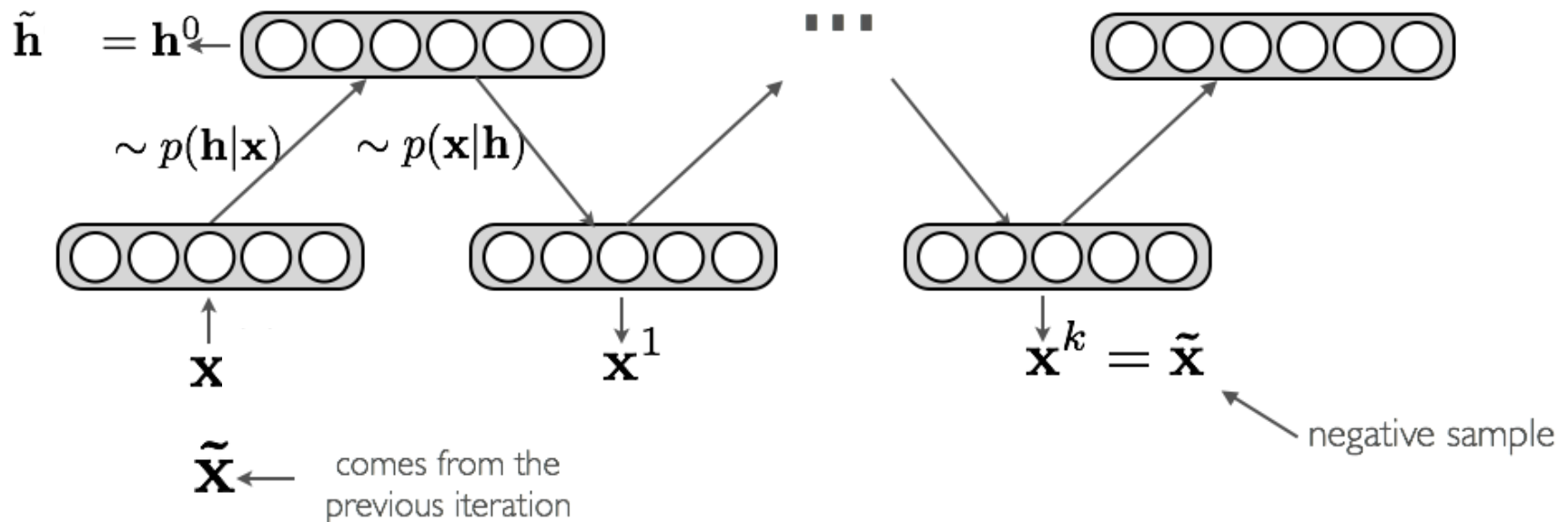
Step size

CD-k Algorithm

- CD-k: contrastive divergence with k iterations of Gibbs sampling
- In general, the bigger k is, the less biased the estimate of the gradient will be
- In practice, $k=1$ works well for learning good features and for pre-training

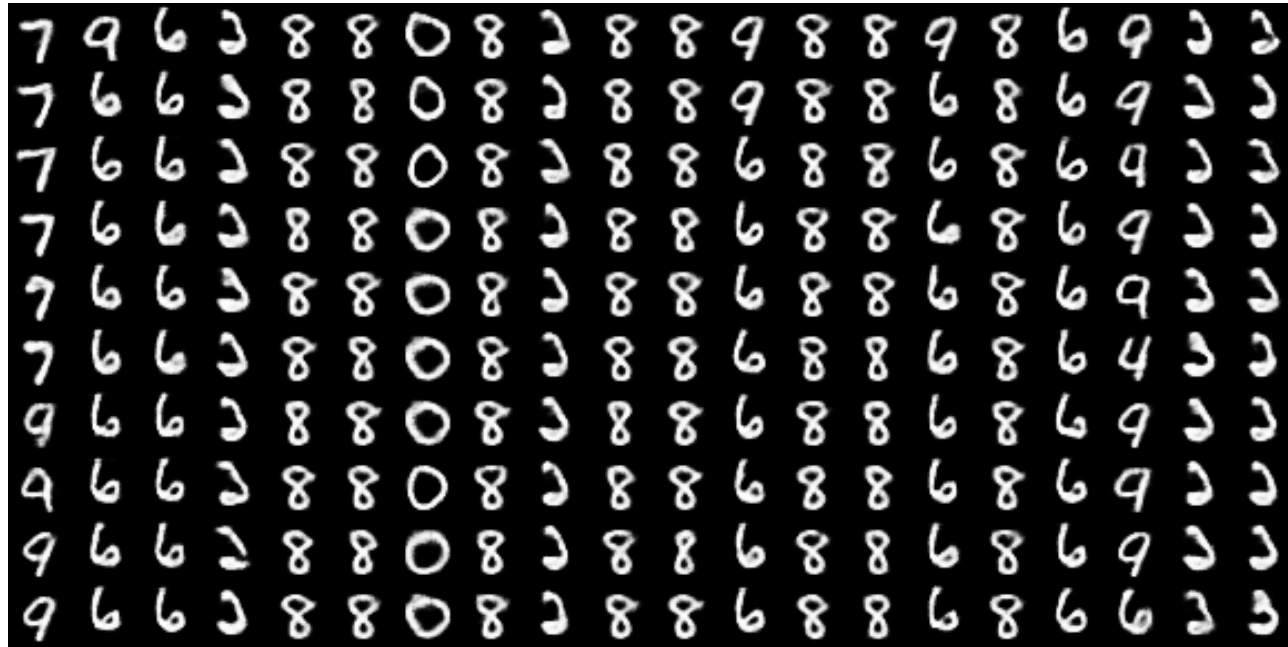
Persistent CD

Idea: instead of initializing the chain to \mathbf{x} , initialize the chain to the negative sample of the last iteration



Example: MNIST

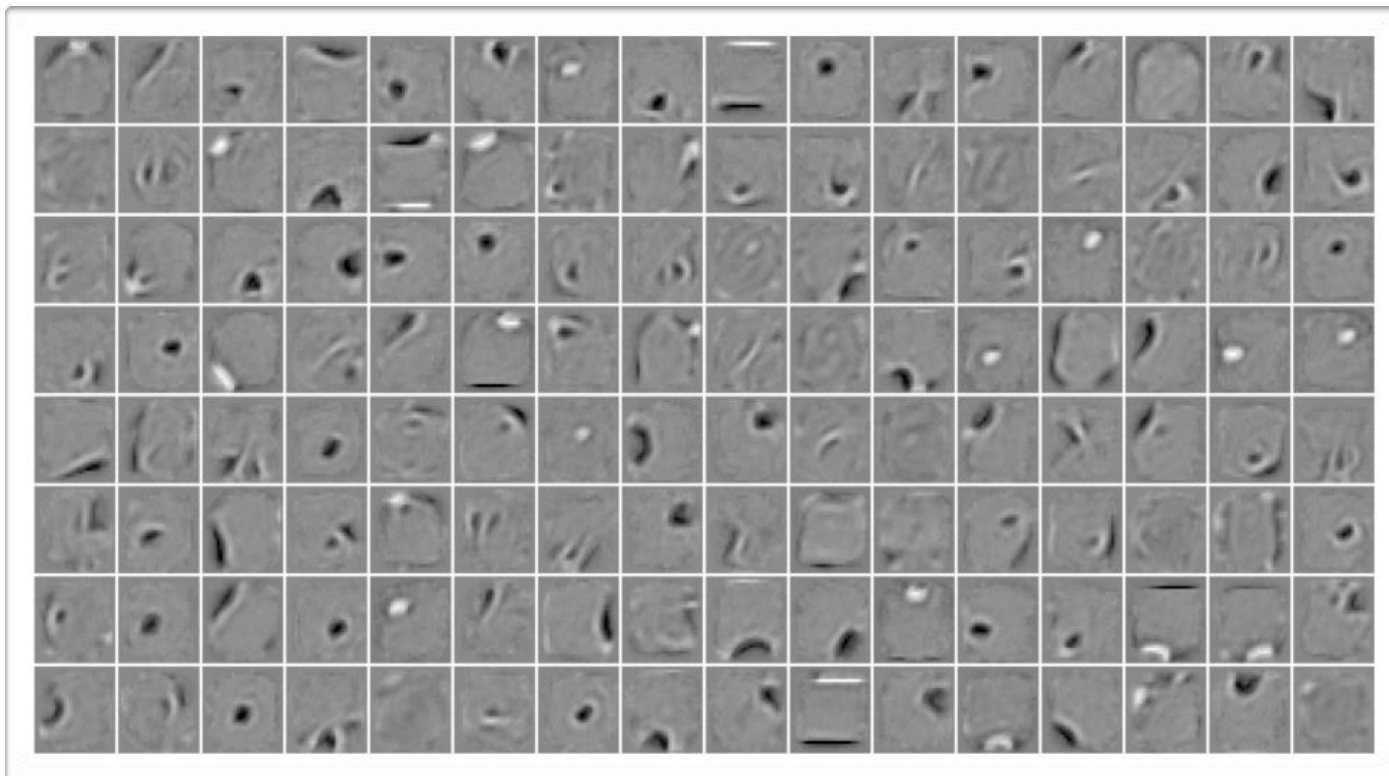
MNIST dataset:



Each row is small set of “initial points”, after which next row is gotten by running 1000 Gibbs steps.

Learned Features

MNIST dataset:



Tricks and Debugging

Unfortunately, it is not easy to debug training RBMs (e.g. using gradient checks)

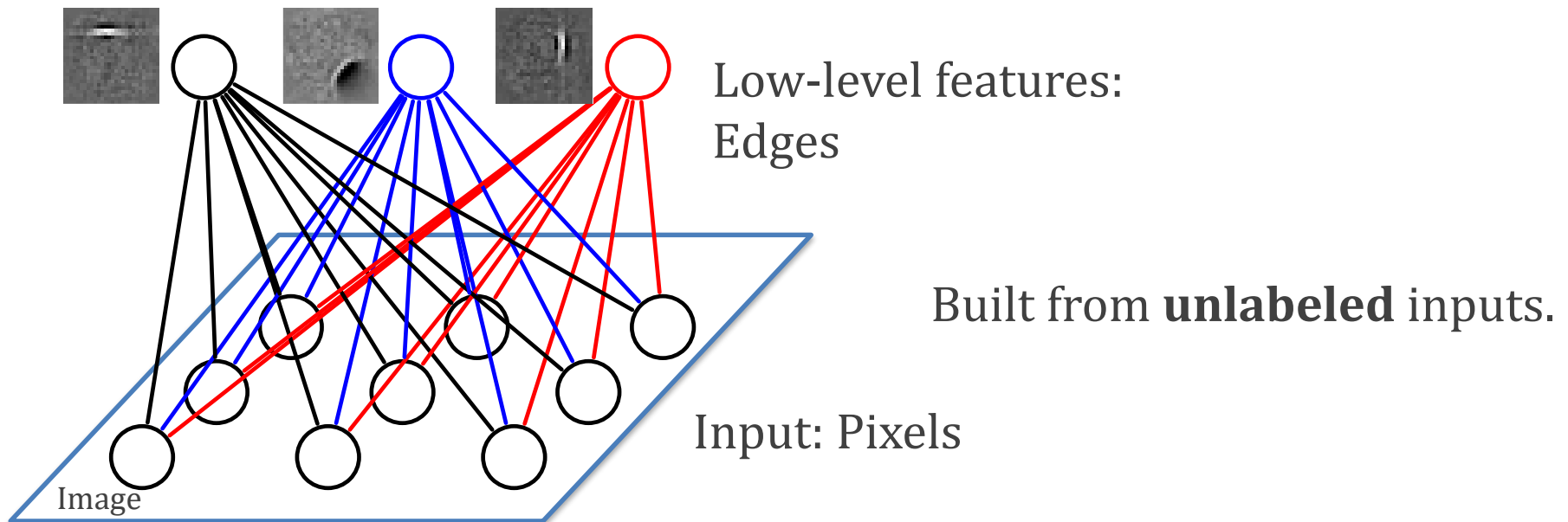
We instead rely on approximate “tricks”

- we plot the average stochastic reconstruction $\|\mathbf{x}^{(t)} - \tilde{\mathbf{x}}\|^2$ and see if it tends to decrease
- for inputs that correspond to image, we visualize the connection coming into each hidden unit as if it was an image
- gives an idea of the type of visual feature each hidden unit detects
- we can also try to approximate the partition function Z and see whether the (approximated) NLL decreases

(Salakhutdinov, Murray, ICML 2008)

Part II: Learning Deep Belief Networks (DBNs)

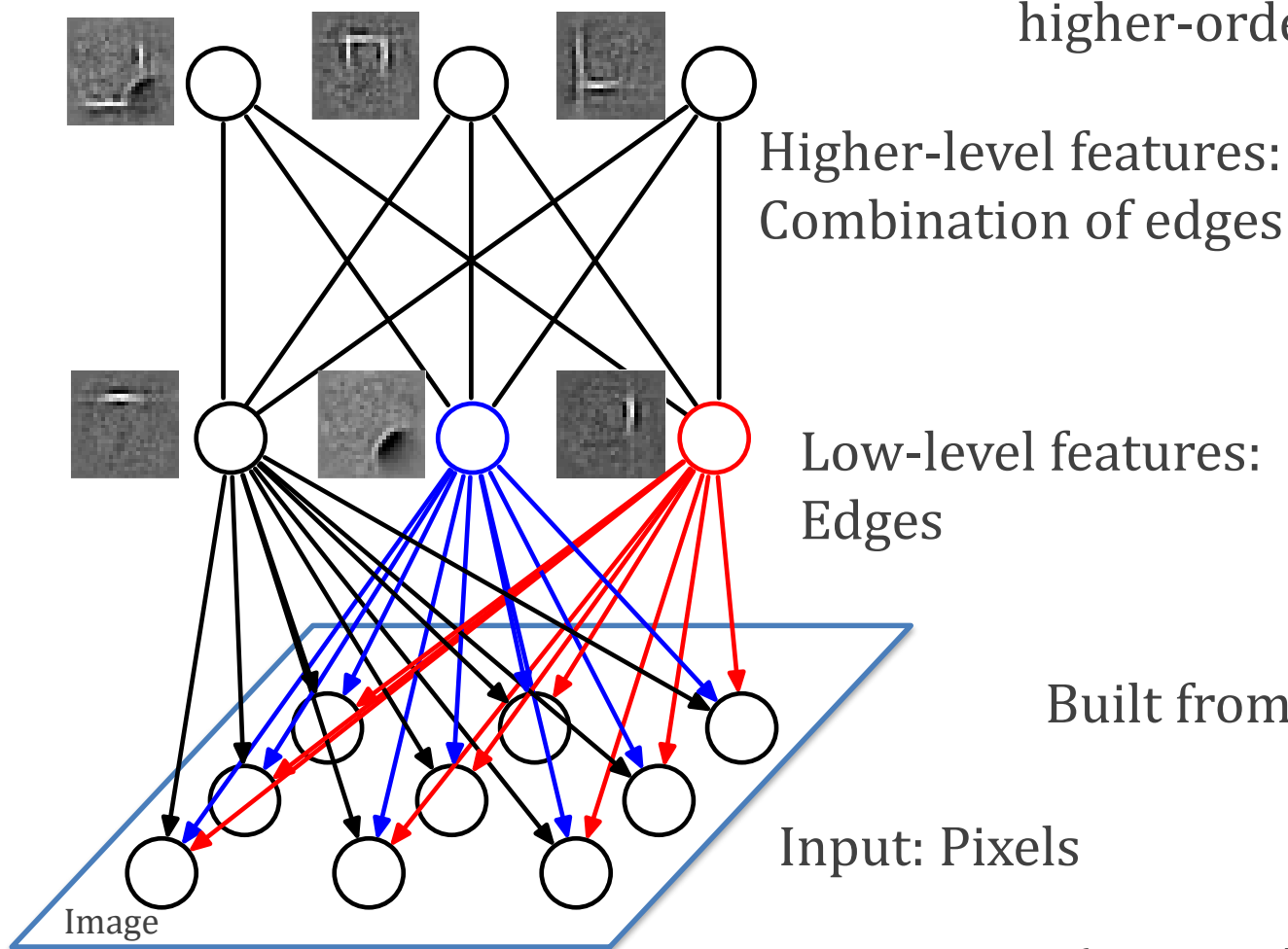
Deep Belief Network



(Hinton et.al. Neural Computation 2006)

Deep Belief Network

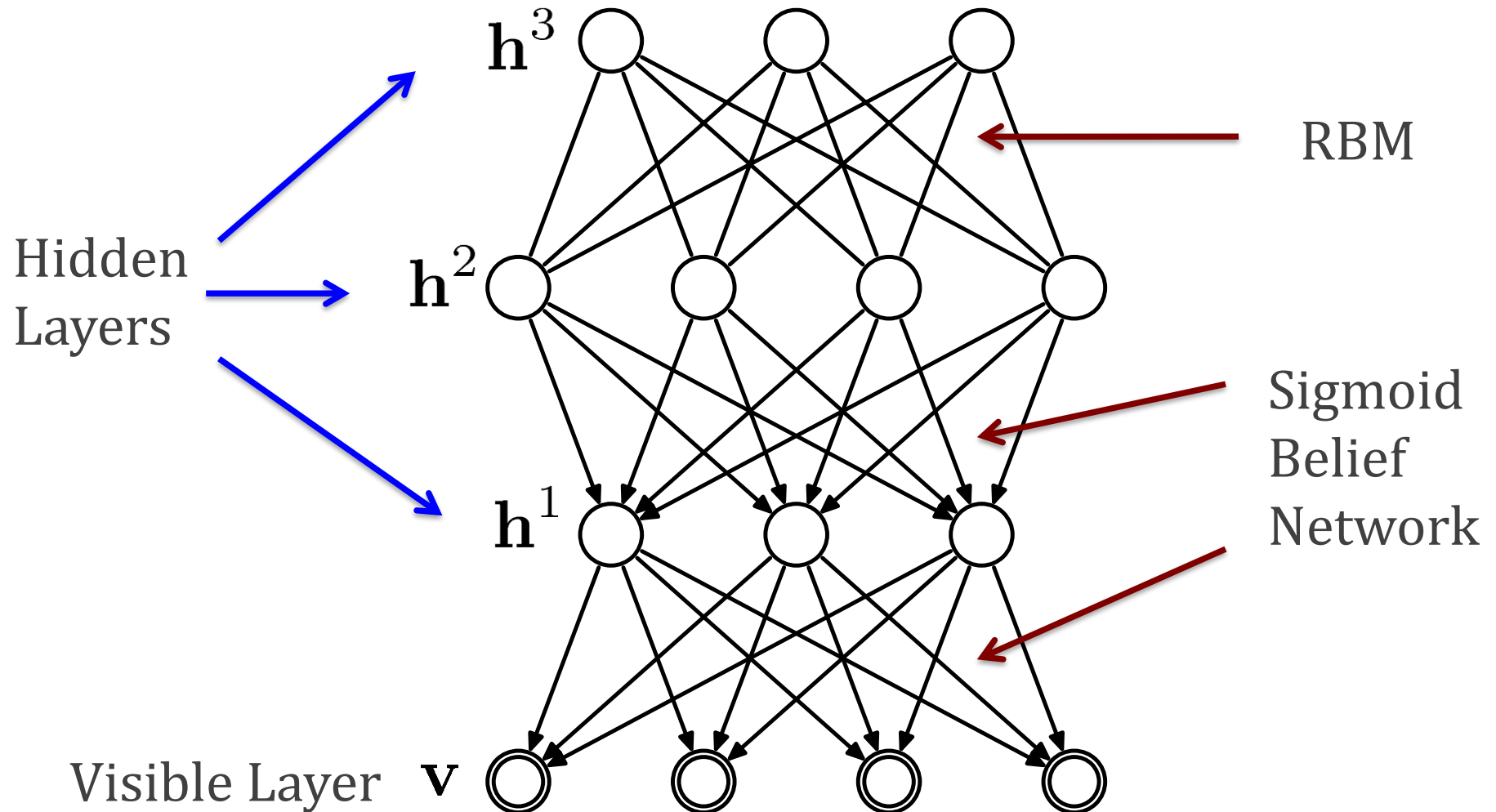
Internal representations capture higher-order statistical structure



Built from **unlabeled** inputs.

(Hinton et.al. Neural Computation 2006)

Deep Belief Network



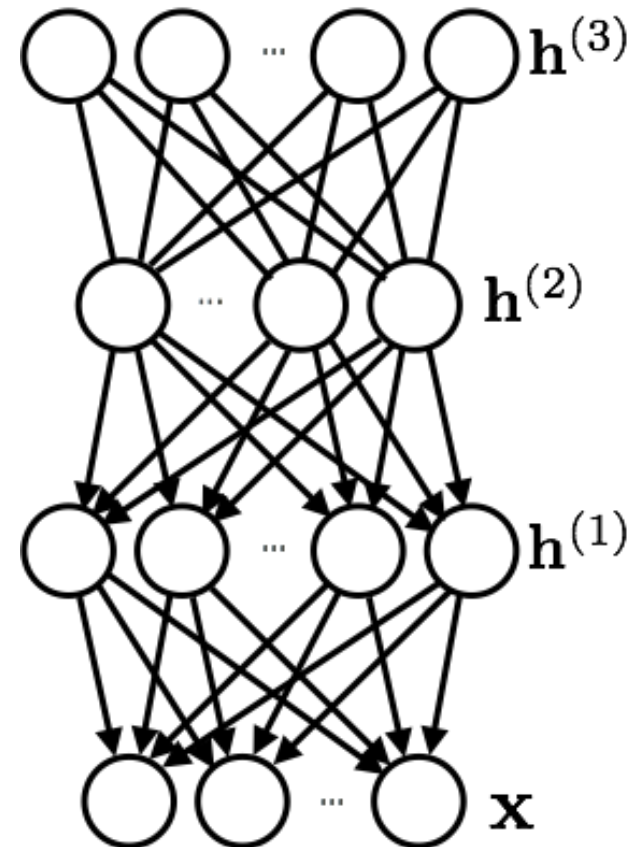
Deep Belief Network

Deep Belief Networks:

- it is a generative model that mixes undirected and directed connections between variables
- top 2 layers' distribution $p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)})$ is an RBM!
- other layers form a Bayesian network with conditional distributions:

$$p(h_j^{(1)} = 1 | \mathbf{h}^{(2)}) = \text{sigm}(\mathbf{b}^{(1)} + \mathbf{W}^{(2)\top} \mathbf{h}^{(2)})$$

$$p(x_i = 1 | \mathbf{h}^{(1)}) = \text{sigm}(\mathbf{b}^{(0)} + \mathbf{W}^{(1)\top} \mathbf{h}^{(1)})$$



Deep Belief Network

The **joint distribution** of a DBN is as follows

$$p(\mathbf{x}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}) = p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)}) p(\mathbf{h}^{(1)} | \mathbf{h}^{(2)}) p(\mathbf{x} | \mathbf{h}^{(1)})$$

where

$$p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)}) = \exp \left(\mathbf{h}^{(2)\top} \mathbf{W}^{(3)} \mathbf{h}^{(3)} + \mathbf{b}^{(2)\top} \mathbf{h}^{(2)} + \mathbf{b}^{(3)\top} \mathbf{h}^{(3)} \right) / Z$$

$$p(\mathbf{h}^{(1)} | \mathbf{h}^{(2)}) = \prod_j p(h_j^{(1)} | \mathbf{h}^{(2)})$$

$$p(\mathbf{x} | \mathbf{h}^{(1)}) = \prod_i p(x_i | \mathbf{h}^{(1)})$$

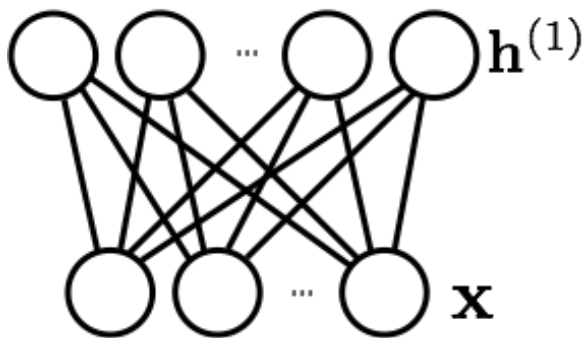
Why this odd parametrization?

Variational intuitions

Consider an RBM.

Let's write the marginal $p(\mathbf{x})$ in terms of the **Gibbs variational principle**.

As $p(\mathbf{x}) = \sum_{\mathbf{h}^{(1)}} p(\mathbf{x}, \mathbf{h}^{(1)})$ (i.e. a partition function of the model $p(\mathbf{h}^{(1)}) \propto p(\mathbf{x}, \mathbf{h}^{(1)})$), we have:



For every distribution $q(\mathbf{h}^{(1)}|\mathbf{x})$:

$$\begin{aligned} \log p(\mathbf{x}) &\geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{h}^{(1)}) \\ &\quad - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x}) \end{aligned}$$

Equality is attained if $q(\mathbf{h}^{(1)}|\mathbf{x}) = p(\mathbf{h}^{(1)}|\mathbf{x})$.

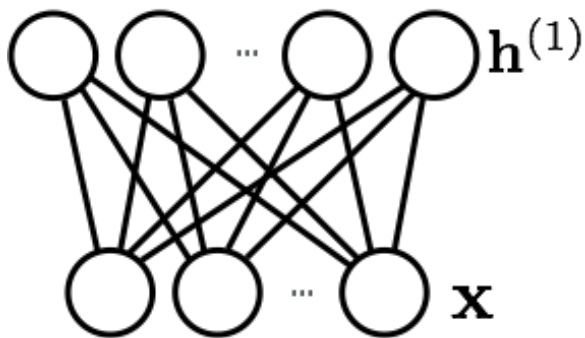
In fact, $\text{LHS} - \text{RHS} = \text{KL}(q(\mathbf{h}^{(1)}|\mathbf{x}) || p(\mathbf{h}^{(1)}|\mathbf{x}))$

Variational intuitions

Consider an RBM.

Let's write the marginal $p(\mathbf{x})$ in terms of the Gibbs variational principle.

As $p(\mathbf{x}) = \sum_{\mathbf{h}^{(1)}} p(\mathbf{x}, \mathbf{h}^{(1)})$ (i.e. a partition function of the model $p(\mathbf{h}^{(1)}) \propto p(\mathbf{x}, \mathbf{h}^{(1)})$), we have:



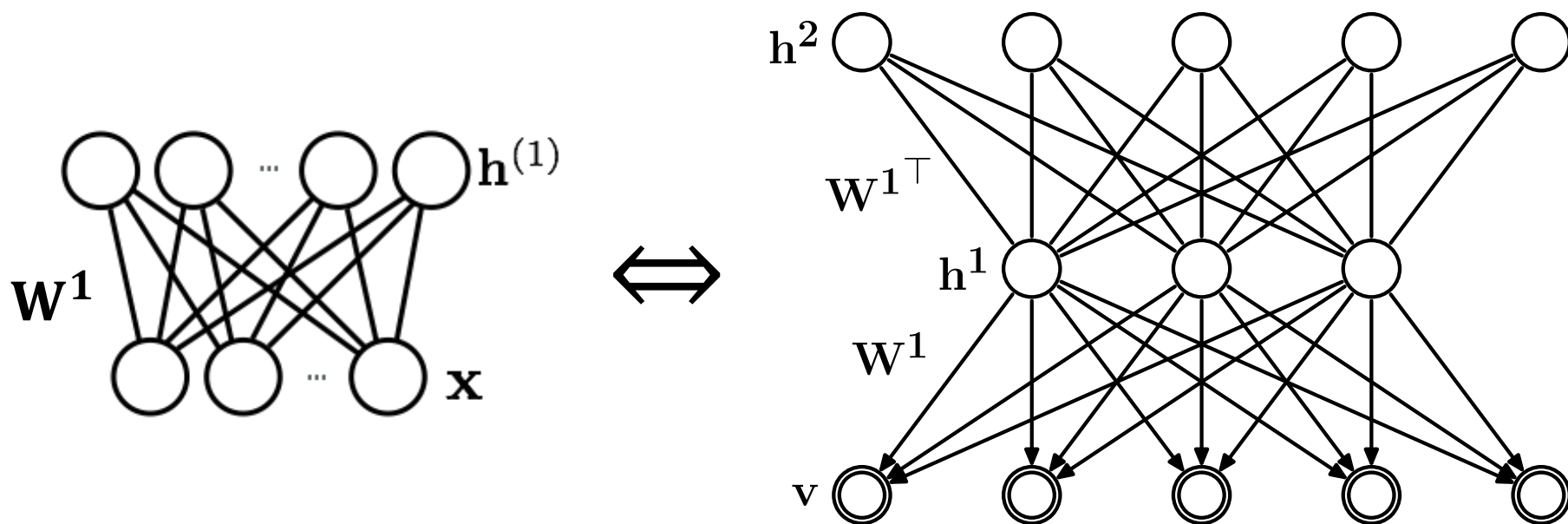
For every distribution $q(\mathbf{h}^{(1)}|\mathbf{x})$:

$$\begin{aligned} \log p(\mathbf{x}) &\geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{h}^{(1)}) \\ &\quad - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x}) \end{aligned}$$

The idea will be to add layers, s.t. we improve the variational bound on the RHS.


Adding one layer

Observation: a two-layer DBN with appropriately tied weights is equivalent to an RBM:



Variational intuitions

adding 2nd layer means
untying the parameters



$$\begin{aligned}\log p(\mathbf{x}) &\geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \left(\log p(\mathbf{x}|\mathbf{h}^{(1)}) + \log p(\mathbf{h}^{(1)}) \right) \\ &\quad - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})\end{aligned}$$

- When adding a second layer, we model $p(\mathbf{h}^{(1)})$ using a separate set of parameters

- they are the parameters of the RBM involving $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$
- $p(\mathbf{h}^{(1)})$ is now the marginalization of the second hidden layer

$$p(\mathbf{h}^{(1)}) = \sum_{\mathbf{h}^{(2)}} p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)})$$

Variational intuitions

adding 2nd layer means
untying the parameters

$$\log p(\mathbf{x}) \geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \left(\log p(\mathbf{x}|\mathbf{h}^{(1)}) + \log p(\mathbf{h}^{(1)}) \right) - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})$$

- we can train the parameters of the **bound**. This is equivalent to maximizing the terms that are constant:


$$- \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{h}^{(1)})$$

- this is like training an RBM on data **generated** from $q(\mathbf{h}^{(1)}|\mathbf{x})$!

Layerwise pretraining
improves variational
lower bound

Variational intuitions

adding 2nd layer means
untying the parameters


$$\begin{aligned}\log p(\mathbf{x}) \quad &\geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \left(\log p(\mathbf{x}|\mathbf{h}^{(1)}) + \log p(\mathbf{h}^{(1)}) \right) \\ &\quad - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})\end{aligned}$$

- for $q(\mathbf{h}^{(1)}|\mathbf{x})$ we use **the posterior of the first layer RBM**. This is equivalent to a feed-forward (sigmoidal) layer, followed by sampling
- by initializing the weights of the second layer RBM as the transpose of the first layer weights, **the bound is initially tight!**
- a 2-layer DBN with tied weights is equivalent to a 1-layer RBM

Layer-wise Training

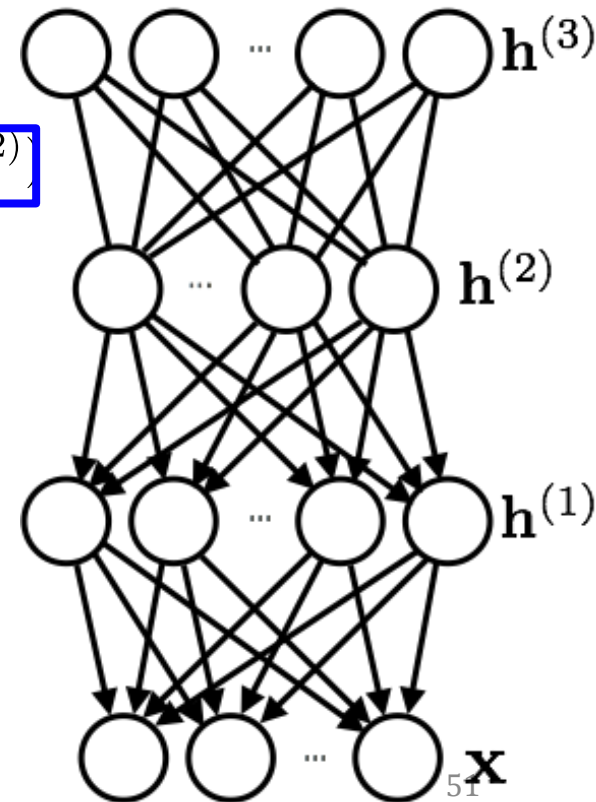
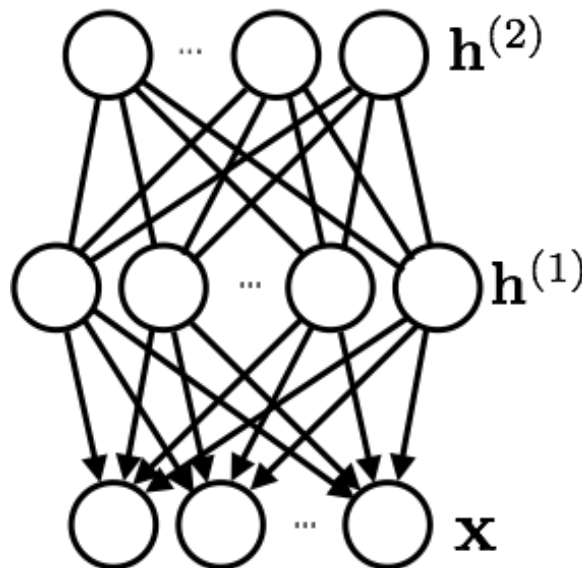
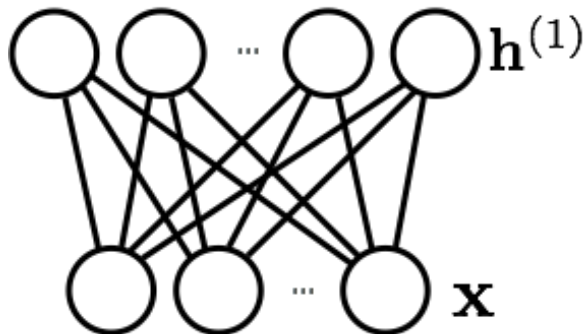
This is where the RBM stacking procedure comes from:

- **idea:** improve prior on last layer by adding another hidden layer

$$p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)}) = p(\mathbf{h}^{(1)} | \mathbf{h}^{(2)}) \sum_{\mathbf{h}^{(3)}} p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)})$$

$$p(\mathbf{x}, \mathbf{h}^{(1)}) = p(\mathbf{x} | \mathbf{h}^{(1)}) \sum_{\mathbf{h}^{(2)}} p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)})$$

$$p(\mathbf{x}) = \sum_{\mathbf{h}^{(1)}} p(\mathbf{x}, \mathbf{h}^{(1)})$$



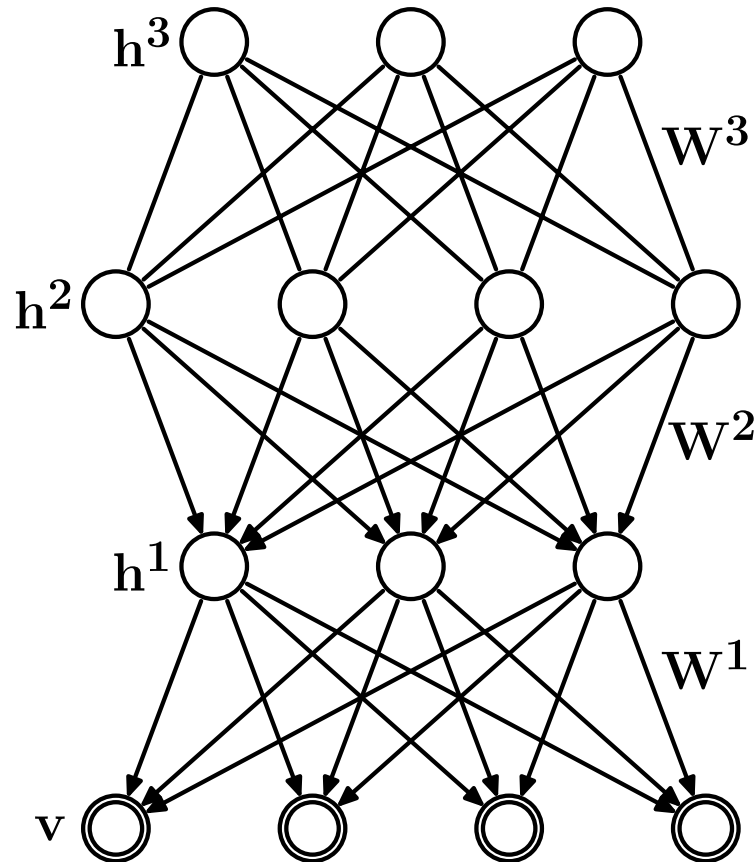
Deep Belief Network

Approximate
Inference

$$Q(\mathbf{h}^3 | \mathbf{h}^2)$$

$$Q(\mathbf{h}^2 | \mathbf{h}^1)$$

$$Q(\mathbf{h}^1 | \mathbf{v})$$



Generative
Process

$$P(\mathbf{h}^2, \mathbf{h}^3)$$

$$P(\mathbf{h}^1 | \mathbf{h}^2)$$

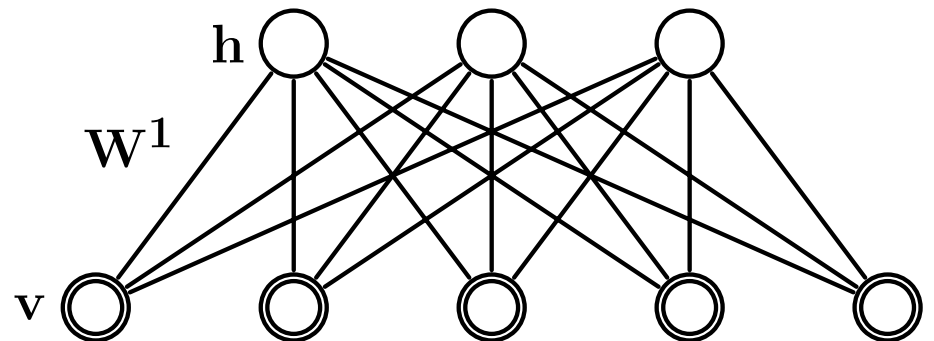
$$P(\mathbf{v} | \mathbf{h}^1)$$

$$Q(\mathbf{h}^t | \mathbf{h}^{t-1}) = \prod_j \sigma \left(\sum_i W^t h_i^{t-1} \right)$$

$$P(\mathbf{h}^{t-1} | \mathbf{h}^t) = \prod_j \sigma \left(\sum_i W^t h_i^t \right)$$

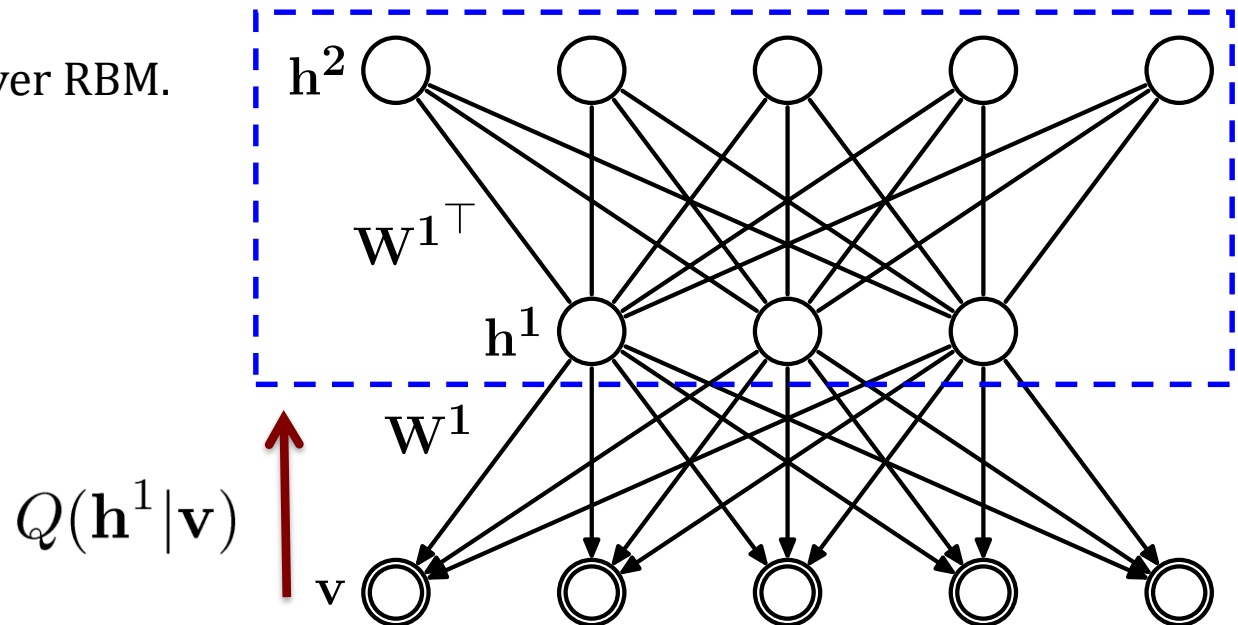
DBN Layer-wise Training

- Learn an RBM with an input layer $v=x$ and a hidden layer h .



DBN Layer-wise Training

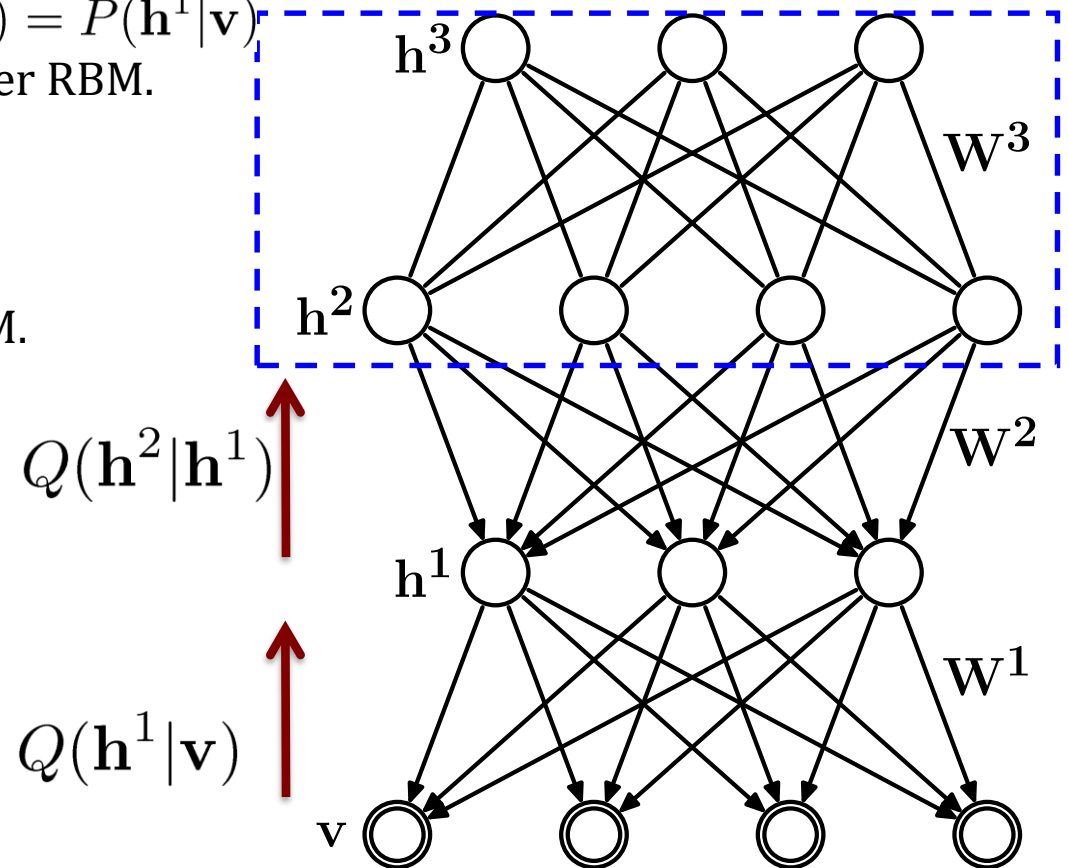
- Learn an RBM with an input layer $\mathbf{v}=\mathbf{x}$ and a hidden layer \mathbf{h} .
- Treat inferred values $Q(\mathbf{h}^1|\mathbf{v}) = P(\mathbf{h}^1|\mathbf{v})$ as the data for training 2nd-layer RBM.
- Learn and freeze 2nd layer RBM.



DBN Layer-wise Training

- Learn an RBM with an input layer $\mathbf{v}=\mathbf{x}$ and a hidden layer \mathbf{h} .
- Treat inferred values $Q(\mathbf{h}^1|\mathbf{v}) = P(\mathbf{h}^1|\mathbf{v})$ as the data for training 2nd-layer RBM.
- Learn and freeze 2nd layer RBM.
- Proceed to the next layer.

Unsupervised Feature Learning.



DBN Layer-wise Training

- Learn an RBM with an input layer $\mathbf{v}=\mathbf{x}$ and a hidden layer \mathbf{h} .
- Treat inferred values $Q(\mathbf{h}^1|\mathbf{v}) = P(\mathbf{h}^1|\mathbf{v})$ as the data for training 2nd-layer RBM.

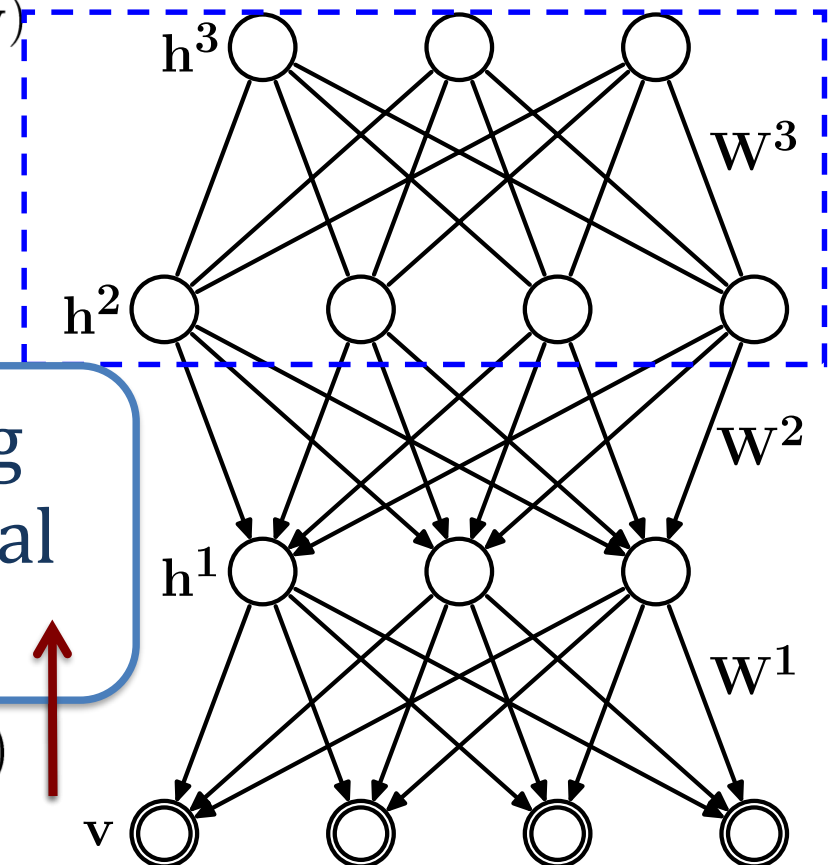
Unsupervised Feature Learning.

- Learn and freeze 2nd layer RBM.

- Proce

Layerwise training
improves variational
lower bound

$$Q(\mathbf{h}^1|\mathbf{v})$$



Deep Belief Networks

This process of adding layers can be repeated recursively

- we obtain the greedy layer-wise pre-training procedure for neural networks

We now see that this procedure corresponds to maximizing a bound on the likelihood of the data in a DBN

- in theory, if our approximation $q(\mathbf{h}^{(1)}|\mathbf{x})$ is very far from the true posterior, the bound might be very loose
- this only means we might not be improving the true likelihood
- we might still be extracting better features!

Fine-tuning is done by the Up-Down algorithm

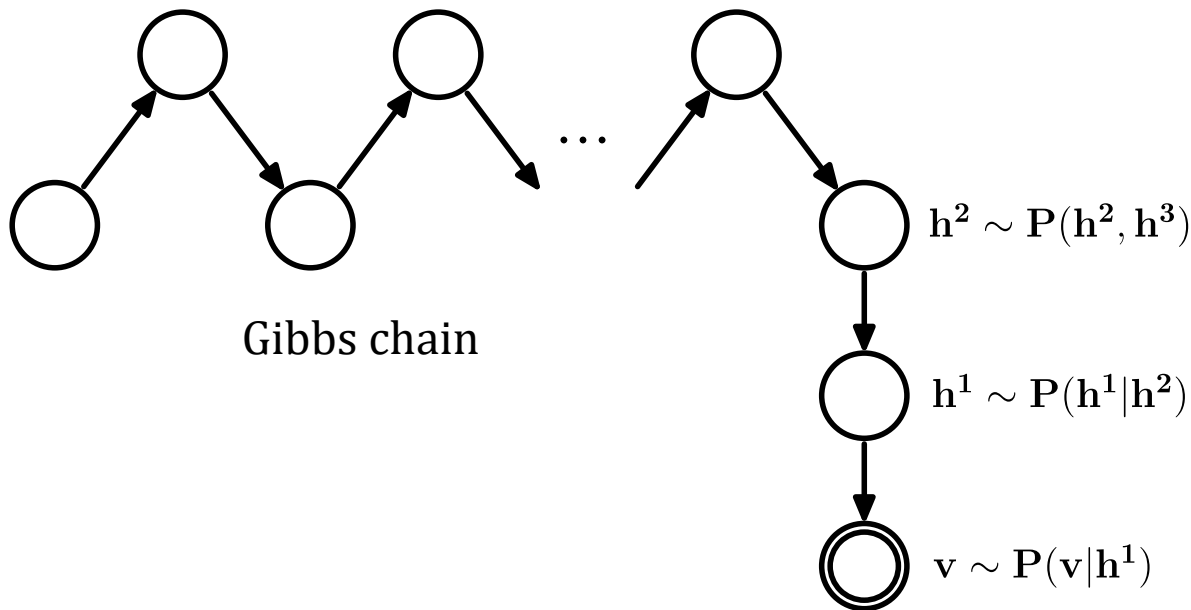
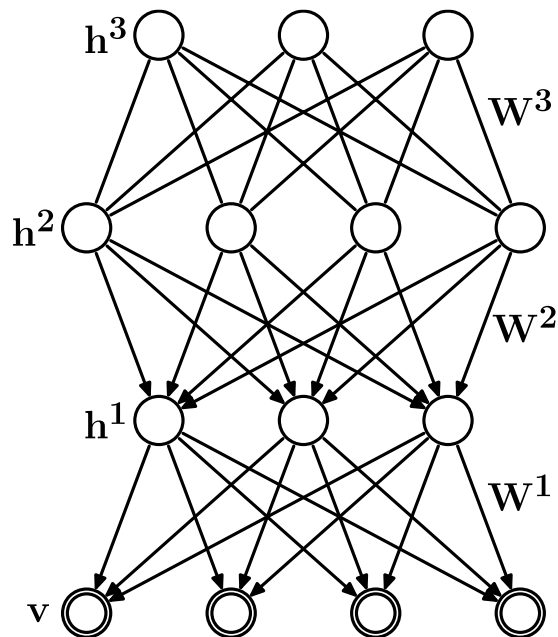
- A fast learning algorithm for deep belief nets. Hinton, Teh, Osindero, 2006.

Sampling from DBNs

- To sample from the DBN model:

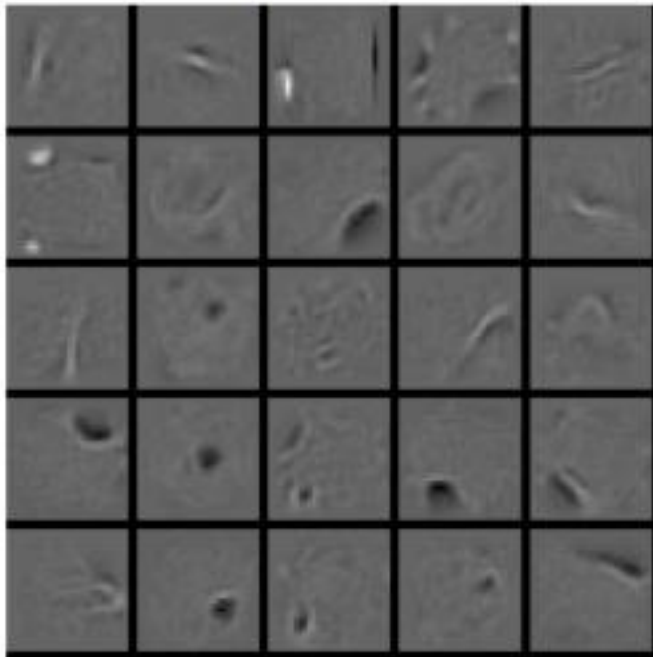
$$P(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3) = P(\mathbf{v}|\mathbf{h}^1)P(\mathbf{h}^1|\mathbf{h}^2)P(\mathbf{h}^2, \mathbf{h}^3)$$

- Sample \mathbf{h}^2 using alternating Gibbs sampling from RBM.
- Sample lower layers using sigmoid belief network.

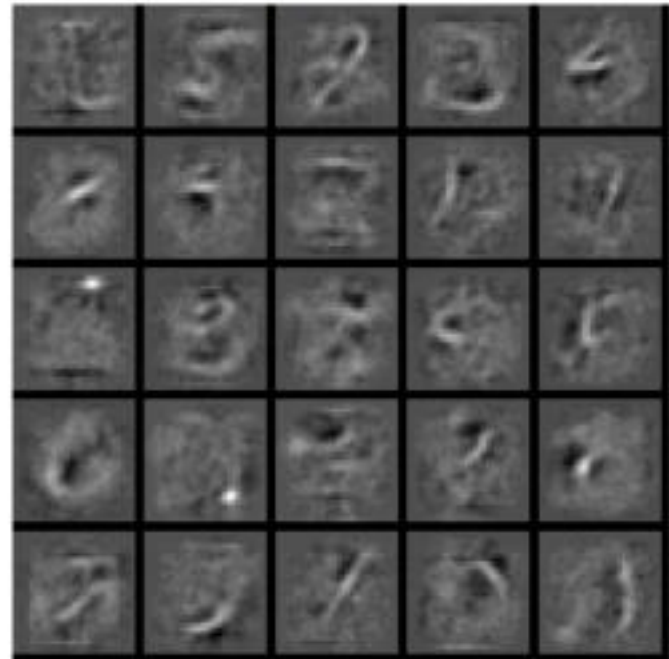


Learned Features

1st-layer features

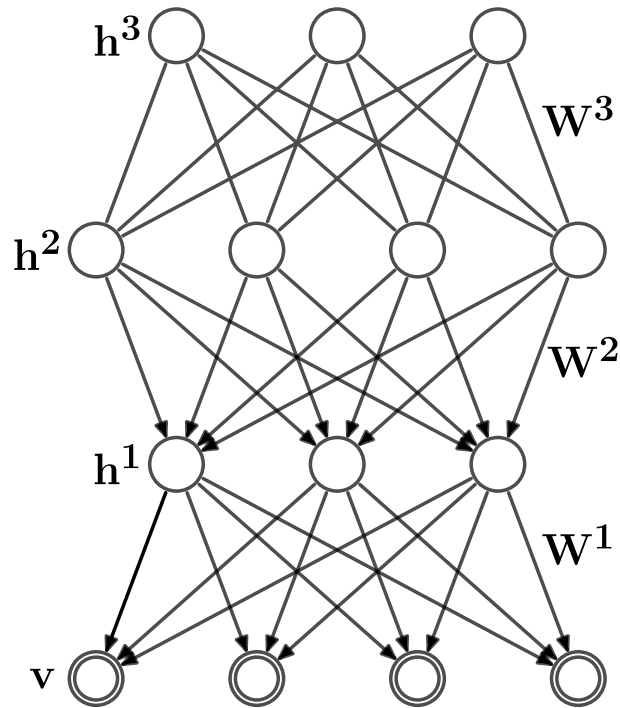


2nd-layer features

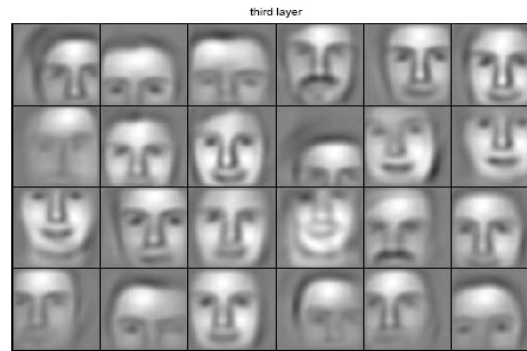


Learning Part-based Representation

Convolutional DBN



Faces



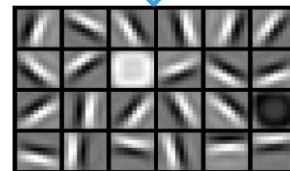
Groups of parts.



second layer



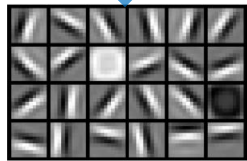
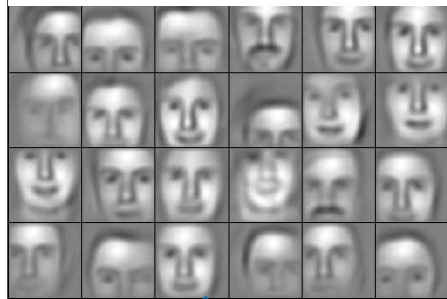
Object Parts



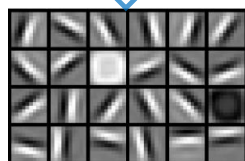
Trained on face images.

Learning Part-based Representation

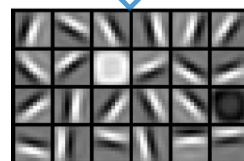
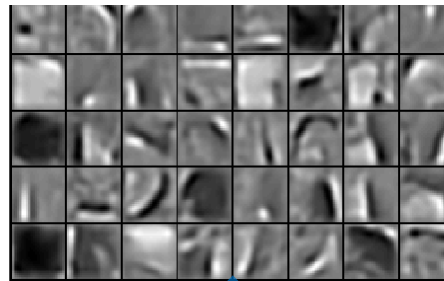
Faces



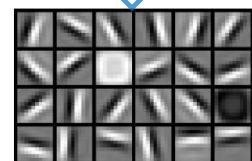
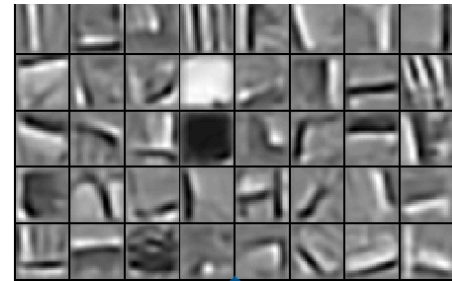
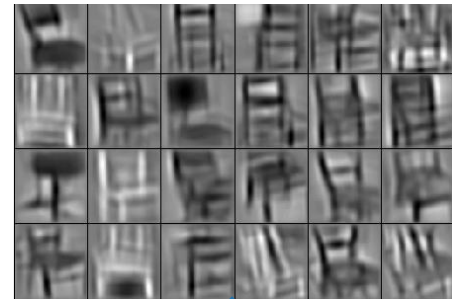
Cars



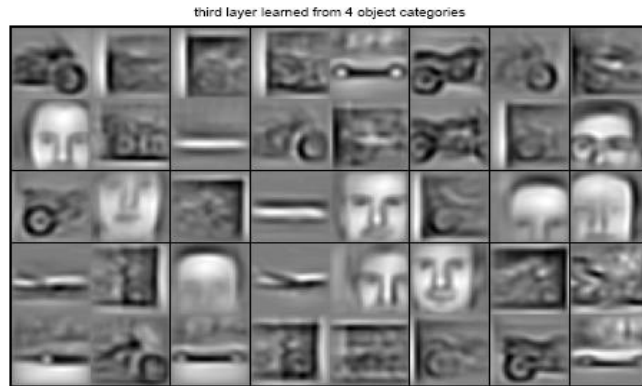
Elephants



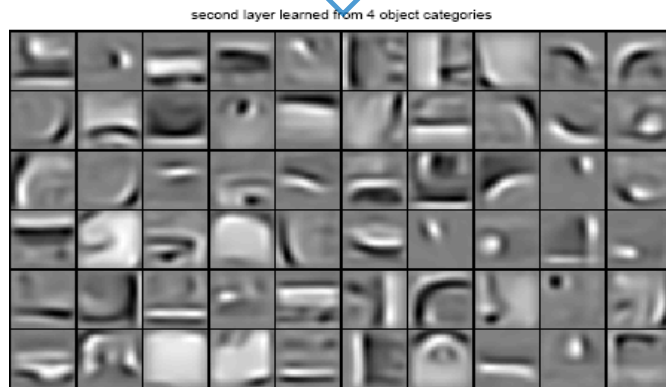
Chairs



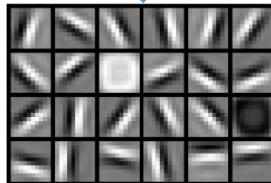
Learning Part-based Representation



Groups of parts.



Class-specific object parts



Trained from multiple classes (cars, faces, motorbikes, airplanes).