RBM, DBN, and DBM

M. Soleymani
Sharif University of Technology
Fall 2017

Slides are based on Salakhutdinov lectures, CMU 2017
and Hugo Larochelle's class on Neural Networks:
https://sites.google.com/site/deeplearningsummerschool2016/.

Energy based models

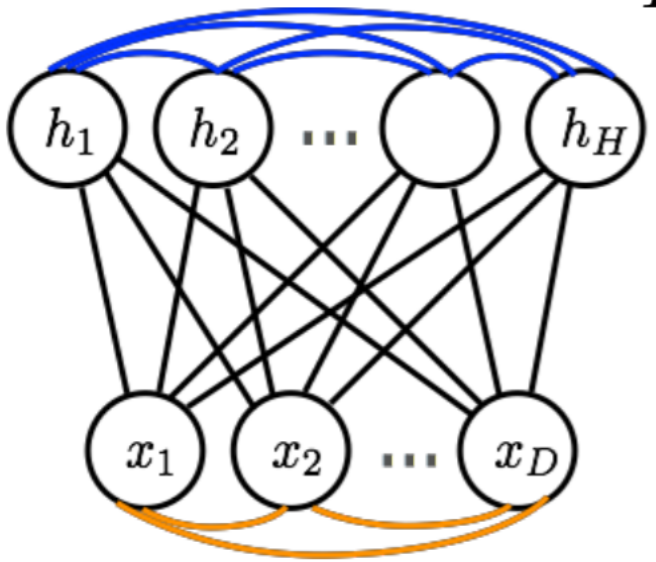
- Gibbs distribution

$$p(x_1, \ldots, x_N) = \frac{1}{Z} \exp\{-E(x_1, \ldots, x_N)\}$$

$$Z = \sum_{x_1, \ldots, x_N} \exp\{-E(x_1, \ldots, x_N)\}$$

Boltzmann Machine

- The original Boltzmann machine has lateral connections in each layer



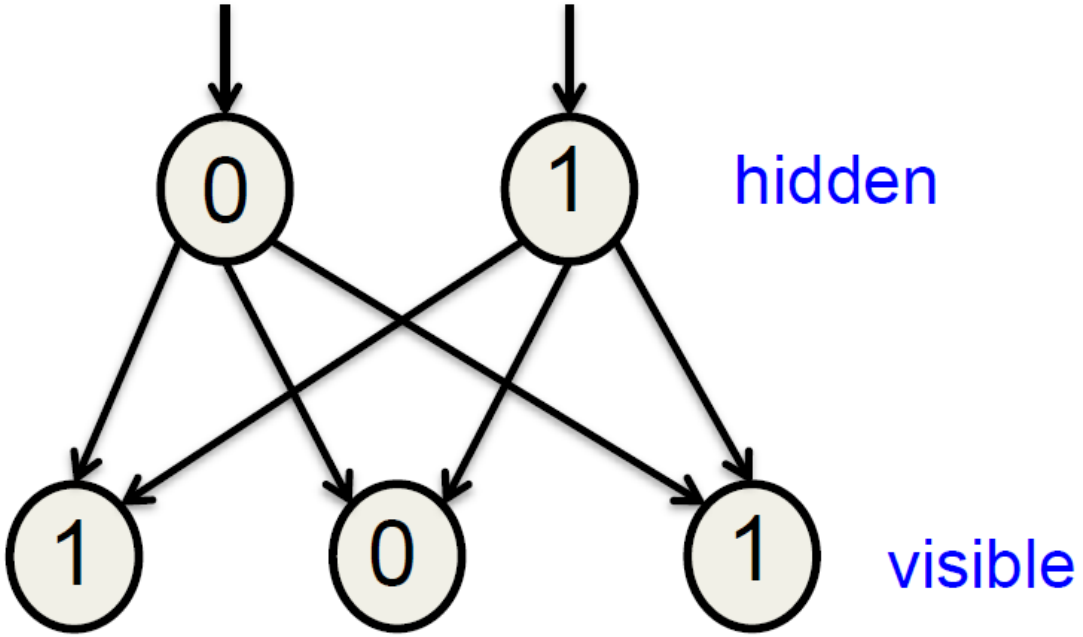
$$E(\mathbf{x}, \mathbf{h}) = -\mathbf{h}^\top \mathbf{W}\mathbf{x} - \mathbf{c}^\top \mathbf{x} - \mathbf{b}^\top \mathbf{h} - \frac{1}{2}\mathbf{x}^\top \mathbf{V}\mathbf{x} + \frac{1}{2}\mathbf{h}^\top \mathbf{U}\mathbf{h}$$

- when only one layer has lateral connection, the model is called a semi-restricted Boltzmann machine

How a causal model generates data

- In a causal model we generate data in two sequential steps:
  - First pick the hidden states from p(h).
  - Then pick the visible states from p(v|h)

- The probability of generating a visible vector, v, is computed by summing over all possible hidden states.



$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{h}) p(\mathbf{v} \mid \mathbf{h})$$

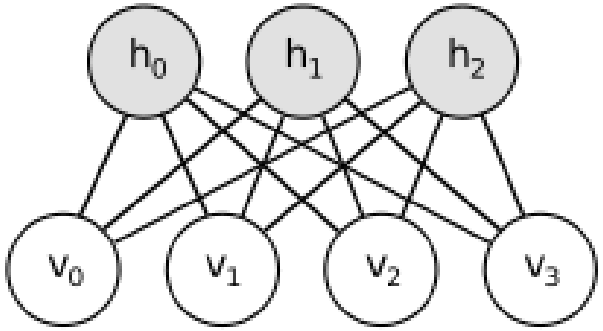
This slide has been adopted from Hinton lectures, “Neural Networks for Machine Learning”, coursera, 2015.

How a Boltzmann Machine generates data

- It is not a causal generative model.
- Instead, everything is defined in terms of the energies of joint configurations of the visible and hidden units.
- The energies of joint configurations are related to their probabilities
  - We can simply define the probability to be $p(v, h) = e^{-E(v,h)}$

Restricted Boltzmann Machines

A Restricted Boltzmann Machine (RBM) is an undirected graphical model with hidden and visible layers.



$$E(v,h) = -v^T W h - c^T v - b^T h$$

$$= -\sum_{i,j} w_{ij} v_i h_j - \sum_i c_i v_i - \sum_j b_j h_j$$

Learnable parameters are b, c which are linear weight vectors for v, h and W which models interaction between them.

Restricted Boltzmann Machines

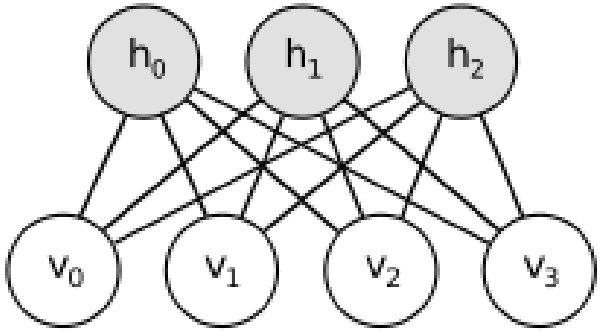
All hidden units are conditionally independent given the visible units and vice versa.

$$p(v_i = 1|\mathbf{h}) = \sigma(\sum_j w_{ij} h_j + b_i)$$

$$p(h_j = 1|\mathbf{v}) = \sigma(\sum_i w_{ij} v_i + c_j)$$

Restricted Boltzmann Machines

RBM probabilities:



$$p(v|h) = \prod_i p(v_i|h)$$

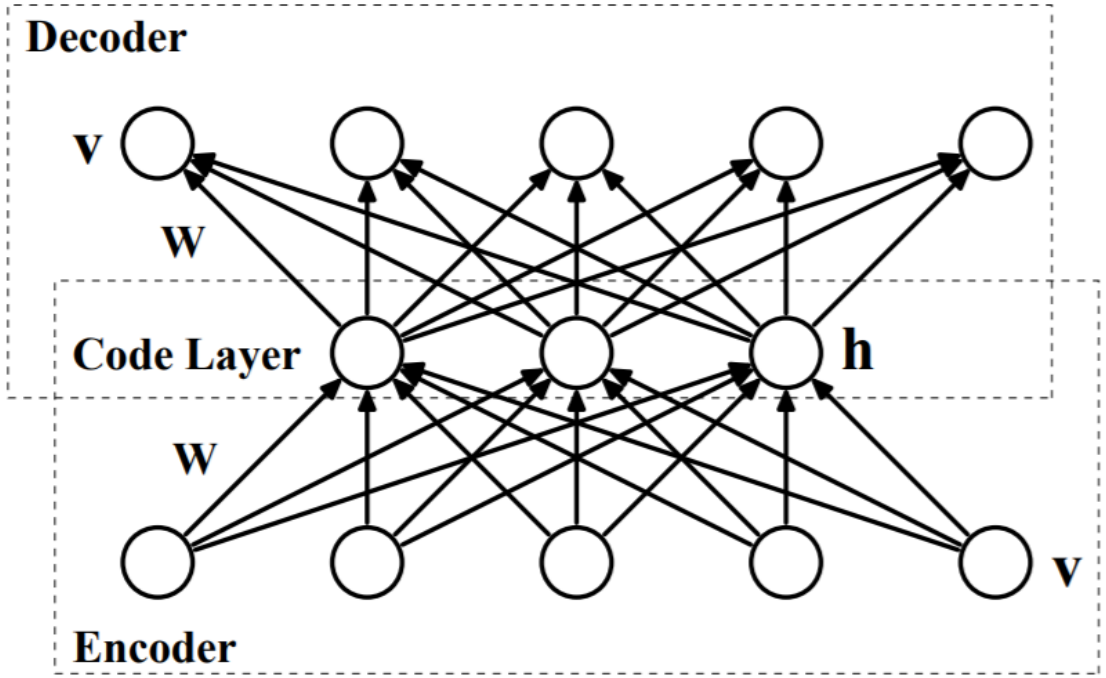
$$p(h|v) = \prod_j p(h_j|v)$$

$$p(v_i = 1|h) = \sigma\left(W_i^T h + b_i\right)$$

$$p\left(h_j = 1|v\right) = \sigma\left(W_j v + c_j\right)$$

Probabilistic Analog of Autoencoder

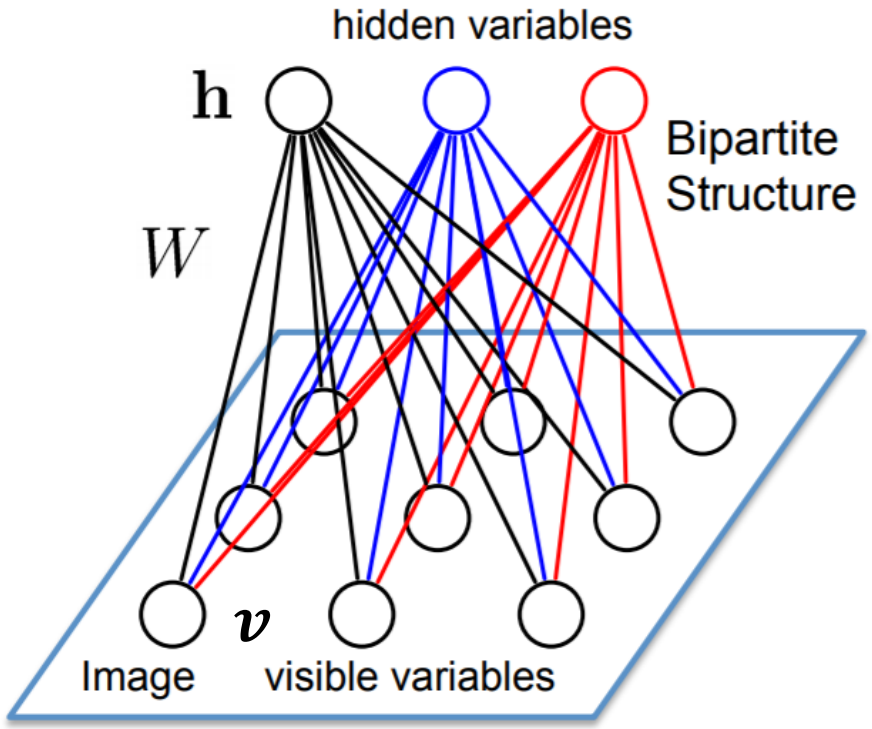
- Autoencoder



$$\text{Encoder: } h_j = \frac{1}{1+\exp(-\sum_i v_i W_{ij})},\ j = 1, ..., K.$$

$$\text{Decoder: } \hat{v}_i = \frac{1}{1+\exp(-\sum_j h_j W_{ij})},\ i = 1, ..., D.$$

RBM: Image input



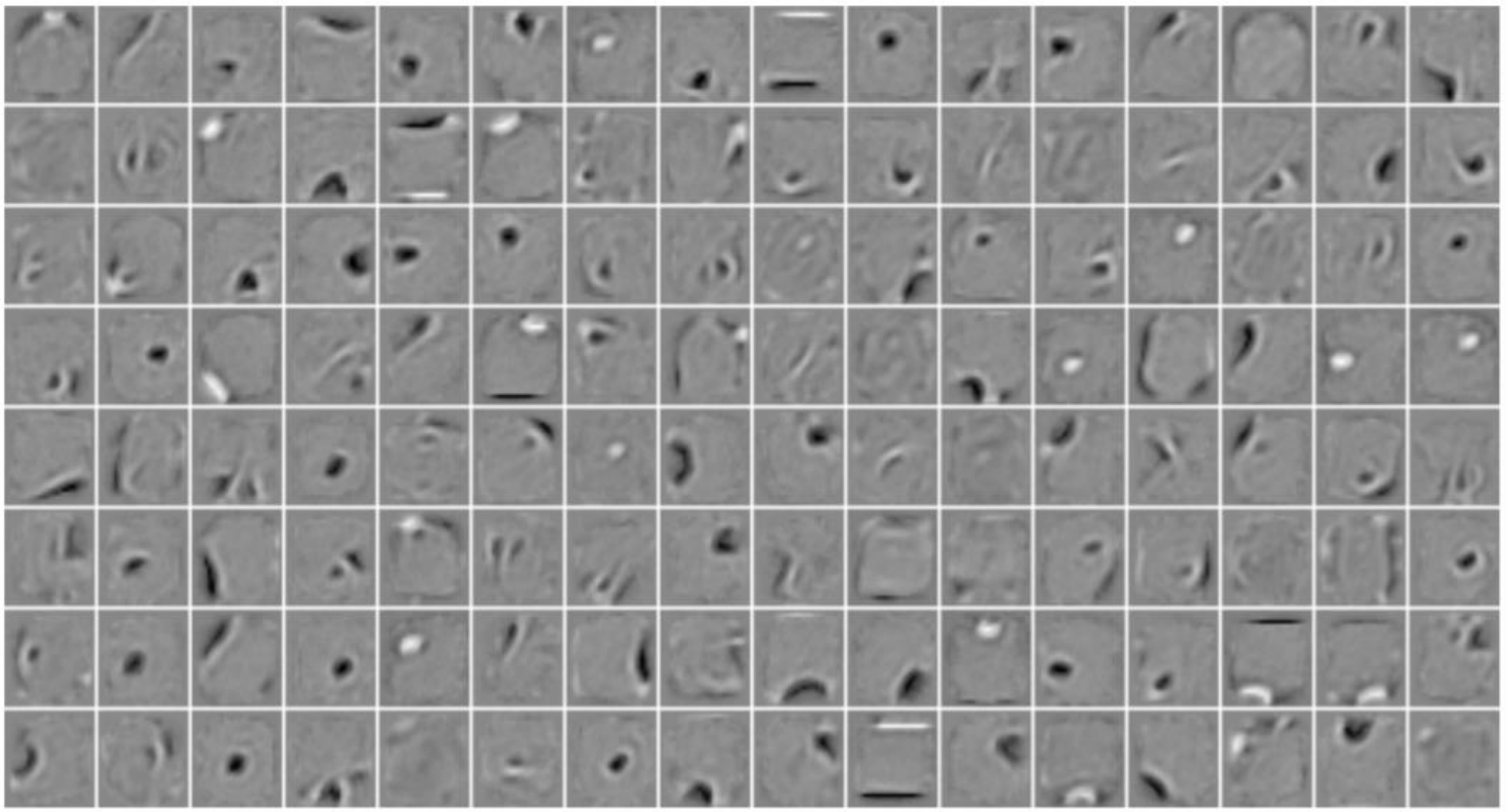
- Undirected bipartite graphical model
- Stochastic binary visible variables:

$$\boldsymbol{v} \in \{\mathbf{0}, \mathbf{1}\}^{\mathbf{D}}$$

- Stochastic binary hidden variables:

$$\mathbf{h} \in \{0, 1\}^{F}$$

MNIST: Learned features



Larochelle et al., JMLR 2009

Restricted Boltzmann Machines

$$E(v,h) = -v^T W h - c^T v - b^T h$$

$$= -\sum_{i,j} w_{ij} v_i h_j - \sum_i c_i v_i - \sum_j b_j h_j$$

The effect of the latent variables can be appreciated by considering the marginal distribution over the visible units:

$$p(\mathbf{v};\Theta) = \sum_{\mathbf{h}} \frac{1}{Z(\Theta)} \exp\{-E(\mathbf{v},\mathbf{h};\Theta)\}$$

Marginal distribution

- What about computing marginal $p(\mathbf{x})$?

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

Marginal distribution

- What about computing marginal $p(\mathbf{x})$?

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

- Also known as Product of Experts model.

Marginal distribution

$$p(\boldsymbol{v}) = \exp\left(\mathbf{c}^\top \boldsymbol{v} + \sum_{j=1}^{H} \log(1 + \exp(b_j + \mathbf{W}_j . \boldsymbol{v}))\right) / Z$$

Marginal distribution

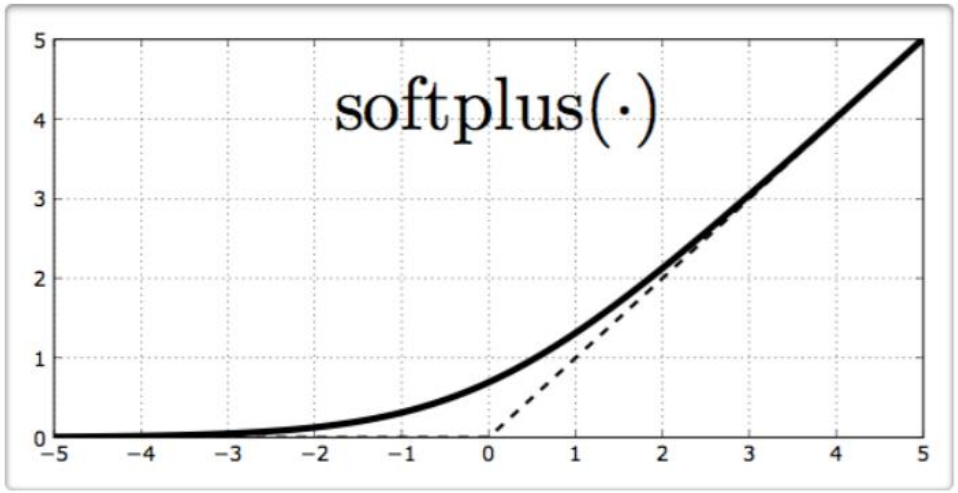
$$p(\boldsymbol{v}) = \exp\left(\mathbf{c}^\top \boldsymbol{v} + \sum_{j=1}^{H} \log(1 + \exp(b_j + \mathbf{W}_j . \boldsymbol{v}))\right) / Z$$

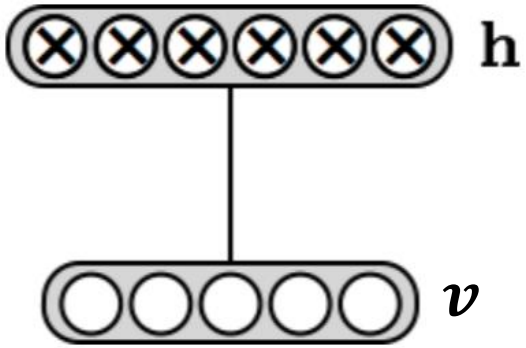
$$= \exp\left(\mathbf{c}^\top \boldsymbol{v} + \sum_{j=1}^{H} \text{softplus}(b_j + \mathbf{W}_j . \boldsymbol{v})\right) / Z$$

bias the probability of each x_i

bias of each feature

feature expected in x





Model Learning

$$P_\theta(\mathbf{v}) = \frac{P^*(\mathbf{v})}{\mathcal{Z}(\theta)} = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}} \exp\left[\mathbf{v}^\top W \mathbf{h} + \mathbf{a}^\top \mathbf{h} + \mathbf{b}^\top \mathbf{v}\right]$$

Given a set of *i.i.d.* training examples $\mathcal{D} = \{\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, ..., \mathbf{v}^{(N)}\}$, we want to learn model parameters $\theta = \{W, a, b\}$.

Maximize log-likelihood objective:

$$L(\theta) = \frac{1}{N} \sum_{n=1}^{N} \log P_\theta(\mathbf{v}^{(n)})$$

Derivative of the log-likelihood:

$$\frac{\partial L(\theta)}{\partial W_{ij}} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial}{\partial W_{ij}} \log\left(\sum_{\mathbf{h}} \exp\left[\mathbf{v}^{(n)\top} W \mathbf{h} + \mathbf{a}^\top \mathbf{h} + \mathbf{b}^\top \mathbf{v}^{(n)}\right]\right) - \frac{\partial}{\partial W_{ij}} \log \mathcal{Z}(\theta)$$

RBM learning: Stochastic gradient descent

$$\frac{\partial}{\partial\theta}\log P(\boldsymbol{v}^{(n)}) = \frac{\partial}{\partial\theta}\log\sum_{h}\exp\left(v^{(n)^T}Wh + c^T v^{(n)} + b^T h\right) - \frac{\partial}{\partial\theta}\log Z$$

Positive phase

Negative phase

- Second term: intractable due to exponential number of configurations.

$$Z = \sum_{v}\sum_{h}\exp(v^T Wh + c^T v + b^T h)$$

Positive phase

$$\frac{\partial}{\partial W}\log\sum_{h}\exp\left(v^{(n)^T}Wh + c^T v^{(n)} + b^T h\right)$$

$$= \frac{\frac{\partial}{\partial W}\sum_h \exp\left(v^{(n)^T}Wh + c^T v^{(n)} + b^T h\right)}{\sum_h \exp\left(v^{(n)^T}Wh + c^T v^{(n)} + b^T h\right)}$$

$$= \frac{\sum_h h v^{(n)^T}\exp\left(v^{(n)^T}Wh + c^T v^{(n)} + b^T h\right)}{\sum_h \exp\left(v^{(n)^T}Wh + c^T v^{(n)} + b^T h\right)}$$

$$= E_{h\sim p(v^n,h)}\left[h v^{(n)^T}\right]$$

RBM learning: Stochastic gradient descent

Maximize $\log p(\mathbf{v};\Theta)$ with respect to $\Theta = \{\mathbf{b}, W, \mathbf{c}\}$

$$\frac{\partial}{\partial W_{ij}} \log P(v^{(n)}) = E_{h_j}\left[v_i h_j | v = v^{(n)}\right] - E_{v_i,h_j}\left[v_i h_j\right]$$

$$\frac{\partial}{\partial b_j} \log P(v^{(n)}) = E_{h_j}\left[h_j | v = v^{(n)}\right] - E_{h_j}\left[h_j\right]$$

$$\frac{\partial}{\partial c_i} \log P(v^{(n)}) = E_{v_i}\left[v_i | v = v^{(n)}\right] - E_{v_i}\left[v_i\right]$$

RBM learning: Stochastic gradient descent

$$\frac{\partial}{\partial W_{ij}} \log P(v^{(n)}) = \boxed{E\left[v_i h_j | v = v^{(n)}\right]} - E\left[v_i h_j\right]$$

Positive statistic

$$E\left[v_i h_j | v = v^{(n)}\right]$$

$$= E\left[h_j | v = v^{(n)}\right] v_i^{(n)}$$

$$= \frac{v_i^{(n)}}{1 + \exp\left(-\left(\sum_i W_{ij} v_i^{(n)} + b_j\right)\right)}$$

- Note that to compute $E\left[v_i h_j\right]$ (negative statistic) we ideally need to integrate (however, a sampler over time can be used to get an estimate of gradients).

Approximate Learning

- Replace the average over all possible input configurations by samples.

$$E_{\boldsymbol{v},\boldsymbol{h}}\left[-\frac{\partial E(\boldsymbol{h},\boldsymbol{v})}{\partial\theta}\right]=\sum_{\boldsymbol{h},\boldsymbol{v}}p(\boldsymbol{h},\boldsymbol{v})\boldsymbol{h}\boldsymbol{v}^{T}$$

- Run MCMC chain (Gibbs sampling) starting from the observed examples.

Model Learning

$$P_\theta(\mathbf{v}) = \frac{P^*(\mathbf{v})}{\mathcal{Z}(\theta)} = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}} \exp\left[\mathbf{v}^\top W \mathbf{h} + \mathbf{a}^\top \mathbf{h} + \mathbf{b}^\top \mathbf{v}\right]$$

Given a set of *i.i.d.* training examples $\mathcal{D} = \{\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, ..., \mathbf{v}^{(N)}\}$, we want to learn model parameters $\theta = \{W, a, b\}$.

Maximize log-likelihood objective:

$$L(\theta) = \frac{1}{N} \sum_{n=1}^{N} \log P_\theta(\mathbf{v}^{(n)})$$

Derivative of the log-likelihood:

$$\frac{\partial L(\theta)}{\partial W_{ij}} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial}{\partial W_{ij}} \log\left(\sum_{\mathbf{h}} \exp\left[\mathbf{v}^{(n)\top} W \mathbf{h} + \mathbf{a}^\top \mathbf{h} + \mathbf{b}^\top \mathbf{v}^{(n)}\right]\right) - \frac{\partial}{\partial W_{ij}} \log \mathcal{Z}(\theta)$$

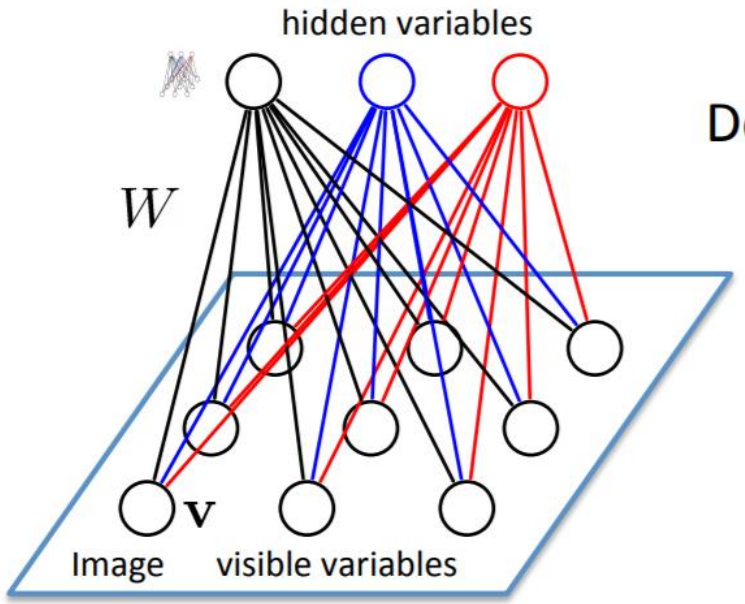
$$= \mathrm{E}_{P_{data}}[v_i h_j] - \mathrm{E}_{P_\theta}[v_i h_j]$$

$$P_{data}(\mathbf{v}, \mathbf{h}; \theta) = P(\mathbf{h}|\mathbf{v}; \theta) P_{data}(\mathbf{v})$$

$$P_{data}(\mathbf{v}) = \frac{1}{N} \sum_{n} \delta(\mathbf{v} - \mathbf{v}^{(n)})$$

Difficult to compute: exponentially many configurations

Model Learning



Derivative of the log-likelihood:

$$\frac{\partial L(\theta)}{\partial W_{ij}} = \mathbb{E}_{P_{data}}[v_i h_j] - \mathbb{E}_{P_\theta}[v_i h_j]$$

$$\sum_{\mathbf{v},\mathbf{h}} v_i h_j P_\theta(\mathbf{v},\mathbf{h})$$

Easy to compute exactly

Difficult to compute: exponentially many configurations.

Use MCMC

$$P_{data}(\mathbf{v},\mathbf{h};\theta) = P(\mathbf{h}|\mathbf{v};\theta)P_{data}(\mathbf{v})$$

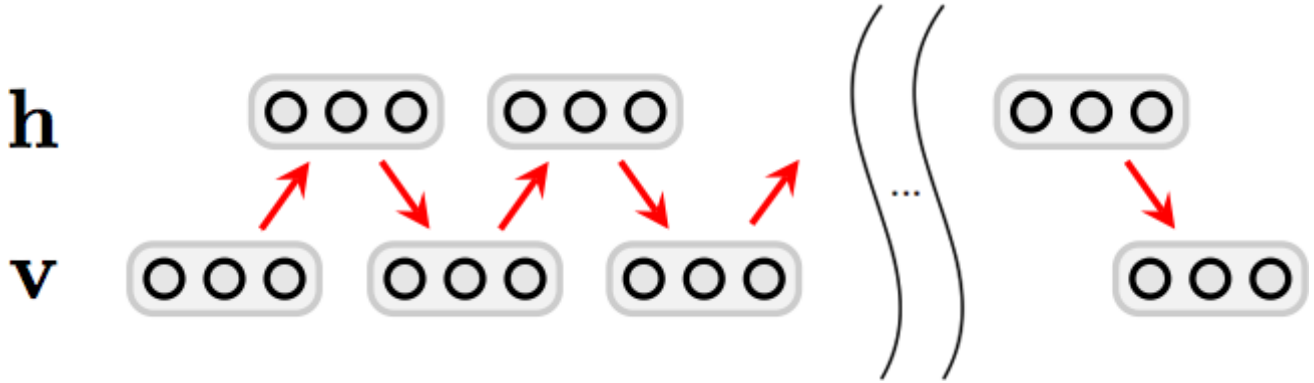
$$P_{data}(\mathbf{v}) = \frac{1}{N}\sum_n \delta(\mathbf{v}-\mathbf{v}^{(n)})$$

RBM learning: Contrastive divergence

Getting an unbiased sample of the second term is very difficult.

It can be done by starting at any random state of the visible units and performing Gibbs sampling for a very long time.

Block-Gibbs MCMC



Initialize v0 = v
Sample h0 from P(h|v0)
For t=1:T
  Sample vt from P(v|ht-1)
  Sample ht from P(h|vt)

Negative statistic

$$E[v_i h_j] \approx \frac{1}{M} \sum_{m=1}^{M} v_i^{(m)} h_j^{(m)}$$

$$v^{(m)} h^{(m)} \sim P(v, h)$$

- Initializing N independent Markov chain each at a data point and running until convergence:

$$E[v_i h_j] \approx \frac{1}{N} \sum_{n=1}^{N} v_i^{(n),T} h_j^{(n),T}$$

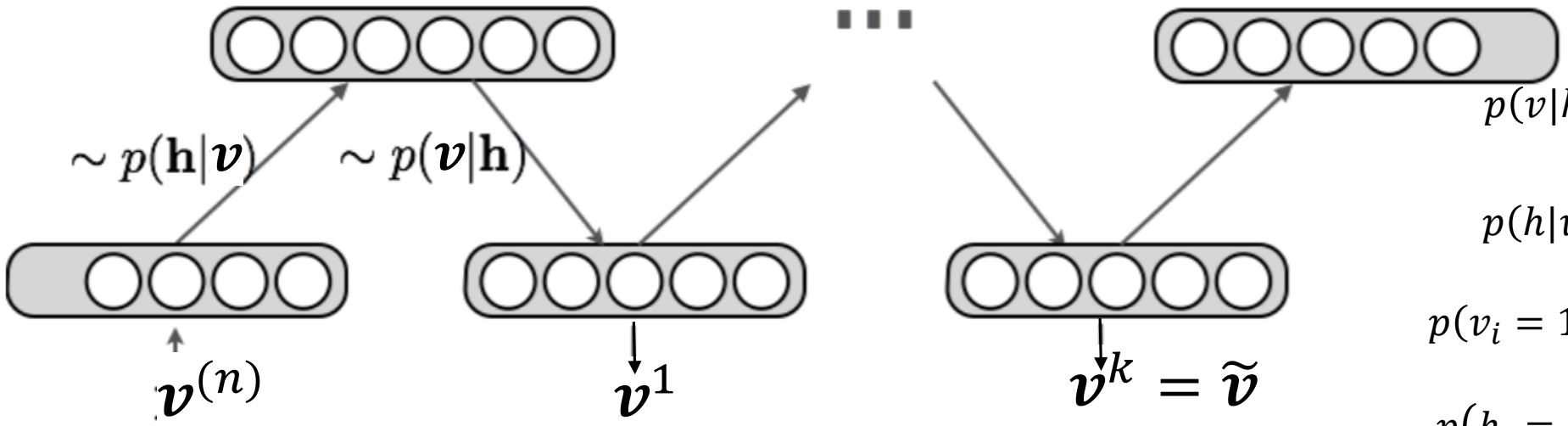
$$v^{(n),0} = v^{(n)}$$

$$h^{(n),k} \sim P(h|v = v^{(n),k}) \quad \text{for } k \geq 0$$

$$v^{(n),k} \sim P(v|h = h^{(n),k-1}) \quad \text{for } k \geq 1$$

Contrastive Divergence

- Key idea behind Contrastive Divergence:
  - Replace the expectation by a point estimate at $\tilde{\boldsymbol{v}}$
  - Obtain the point $\tilde{\boldsymbol{v}}$ by Gibbs sampling
  - Start sampling chain at $\boldsymbol{v}^{(n)}$



$$p(v|h) = \prod_i p(v_i|h)$$

$$p(h|v) = \prod_j p(h_j|v)$$

$$p(v_i = 1|h) = \sigma(W_i^T h + b_i)$$

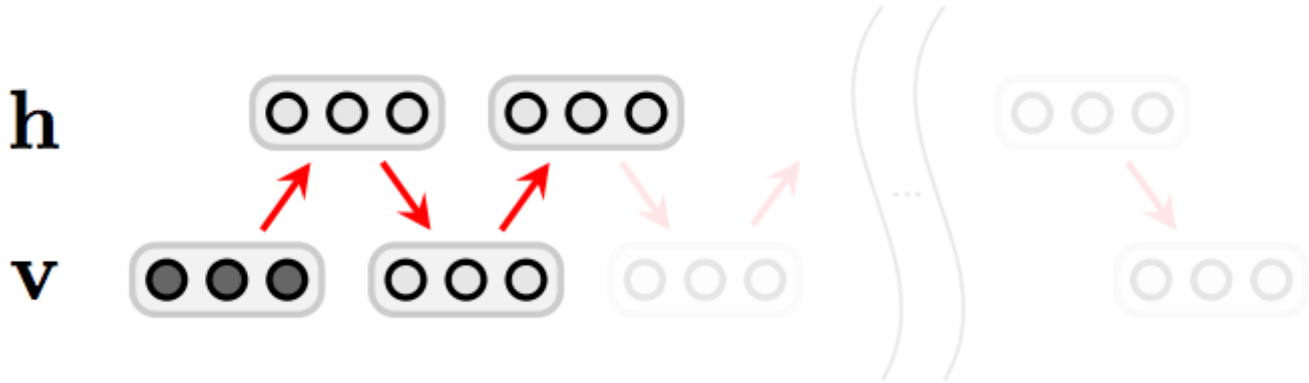
$$p(h_j = 1|v) = \sigma(W_j v + c_j)$$

negative sample

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

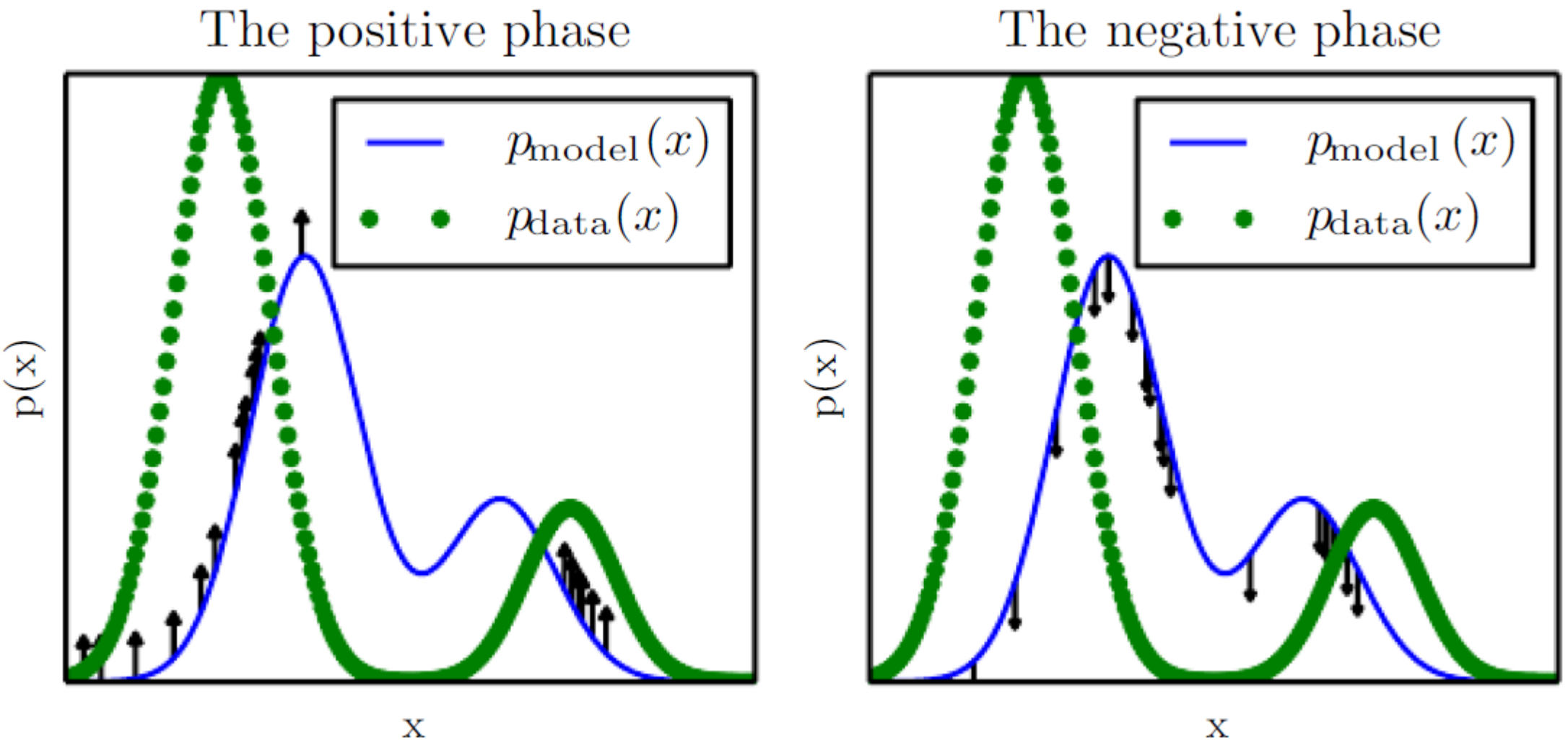
RBM inference: Block-Gibbs MCMC



CD-k Algorithm

- Repeat until stopping criteria
  - For each training sample $v^{(n)}$
    - Generate a negative sample $\tilde{v}$ using k steps of Gibbs sampling starting at the point $v^{(n)}$
    - Update model parameters:
      - $W \leftarrow W + \alpha\left(\frac{\partial \log p(v^{(n)})}{\partial W}\right) = W + \alpha\left(h(v^{(n)}){v^{(n)}}^T - h(\tilde{v})\tilde{v}^T\right)$
      - $b \leftarrow b + \alpha\left(h(v^{(n)}) - h(\tilde{v})\right)$
      - $c \leftarrow c + \alpha(v^{(n)} - \tilde{v})$

Positive phase vs. negative phase



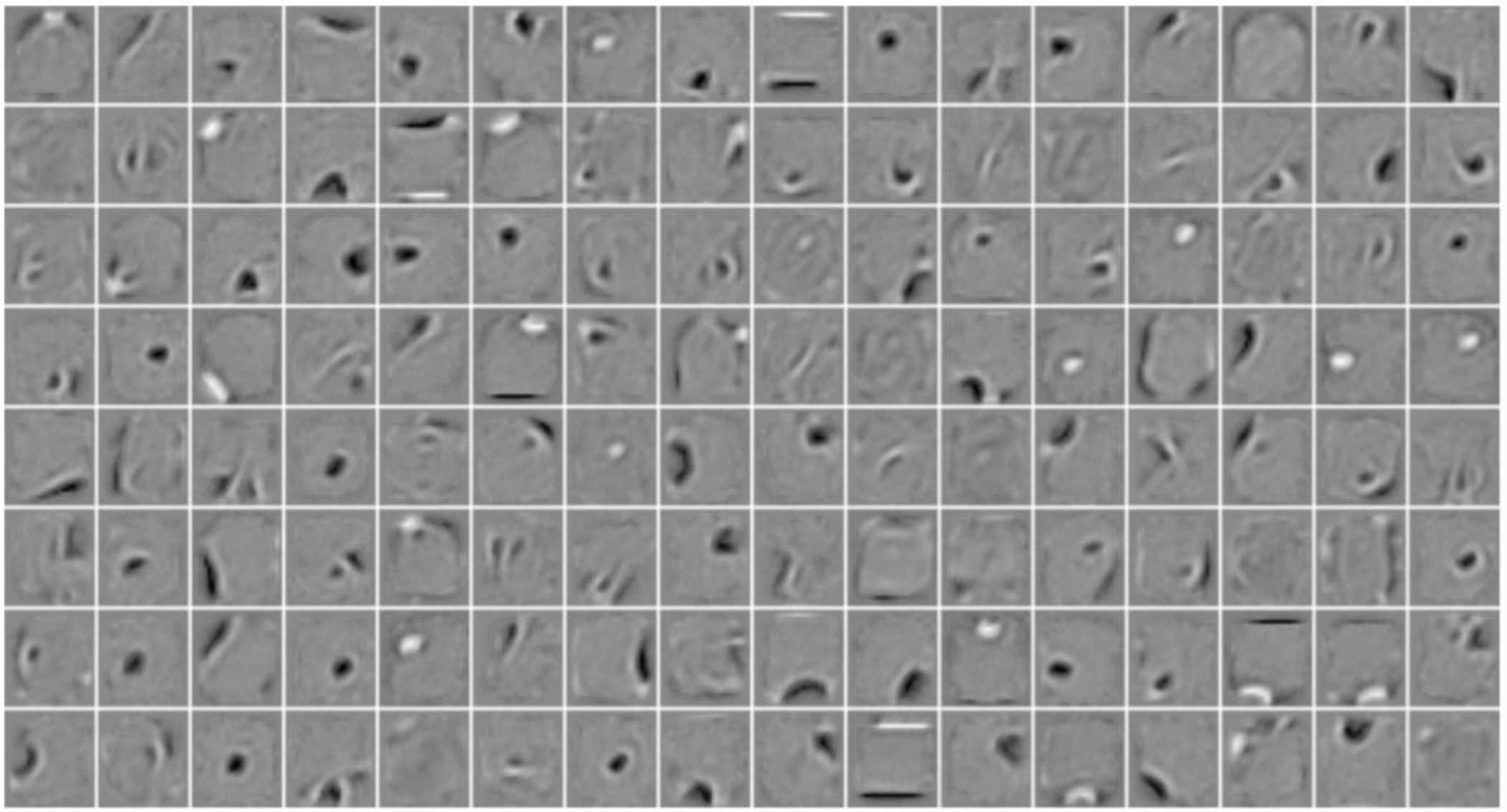
Contrastive Divergence

Since convergence to a final distribution takes time, good initialization can speeds things up dramatically.

Contrastive divergence uses a sample image to initialize the visible weights, then runs Gibbs sampling for a few iterations (even k = 1) – not to “equilibrium.”

This gives acceptable estimates of the expected values in the gradient update formula.

MNIST: Learned features



Larochelle et al., JMLR 2009

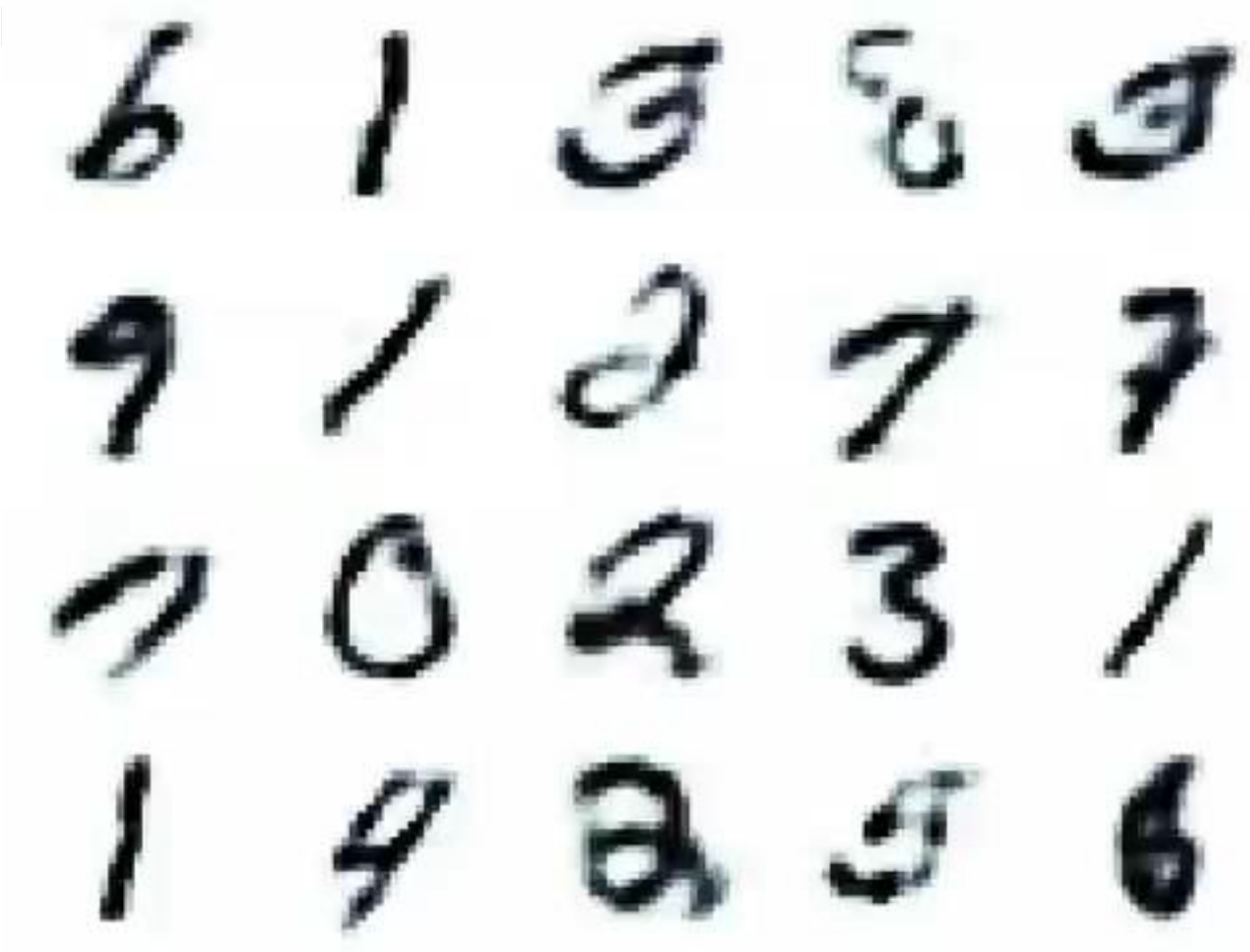
Tricks and Debugging

- 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 $\left\| v^{(n)} - \tilde{v} \right\|$ 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

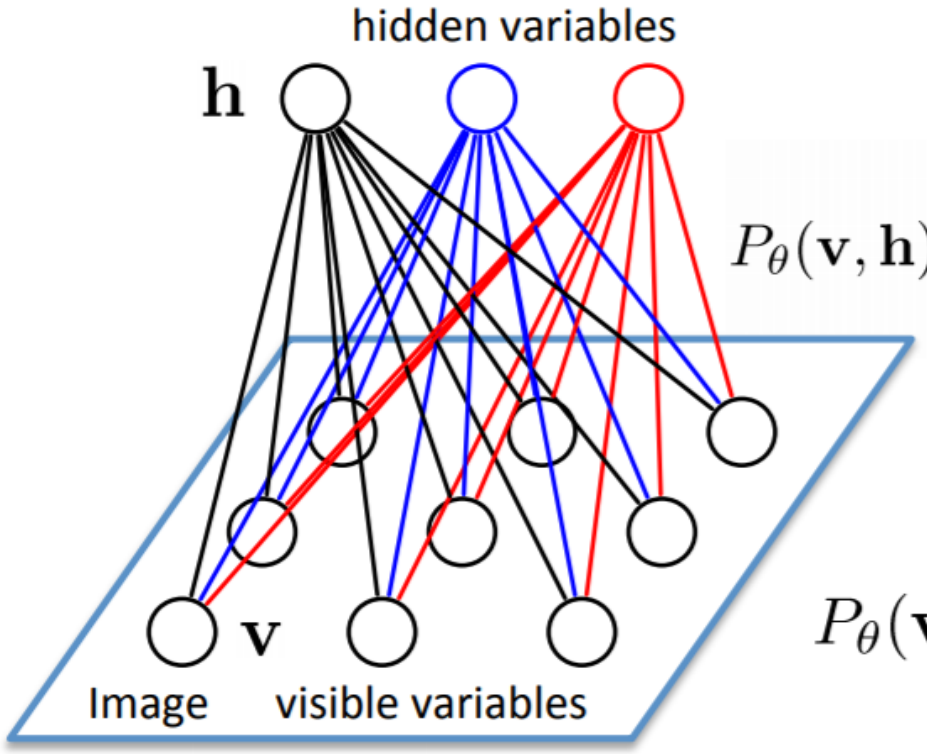
Salakhutdinov, Murray, ICML 2008.

RBM inference

- Block-Gibbs



Gaussian Bernoulli RBMs



Pair-wise **Unary**

$$P_\theta(\mathbf{v},\mathbf{h}) = \frac{1}{\mathcal{Z}(\theta)} \exp\left(\sum_{i=1}^{D}\sum_{j=1}^{F} W_{ij} h_j \frac{v_i}{\sigma_i} + \sum_{i=1}^{D} \frac{(v_i - b_i)^2}{2\sigma_i^2} + \sum_{j=1}^{F} a_j h_j\right)$$

$$\theta = \{W, a, b\}$$

$$P_\theta(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{D} P_\theta(v_i|\mathbf{h}) = \prod_{i=1}^{D} \mathcal{N}\left(b_i + \sum_{j=1}^{F} W_{ij} h_j, \sigma_i^2\right)$$

Gaussian Bernoulli RBMs

- Let x represent a real-valued (unbounded) input, add a quadratic term to the energy function

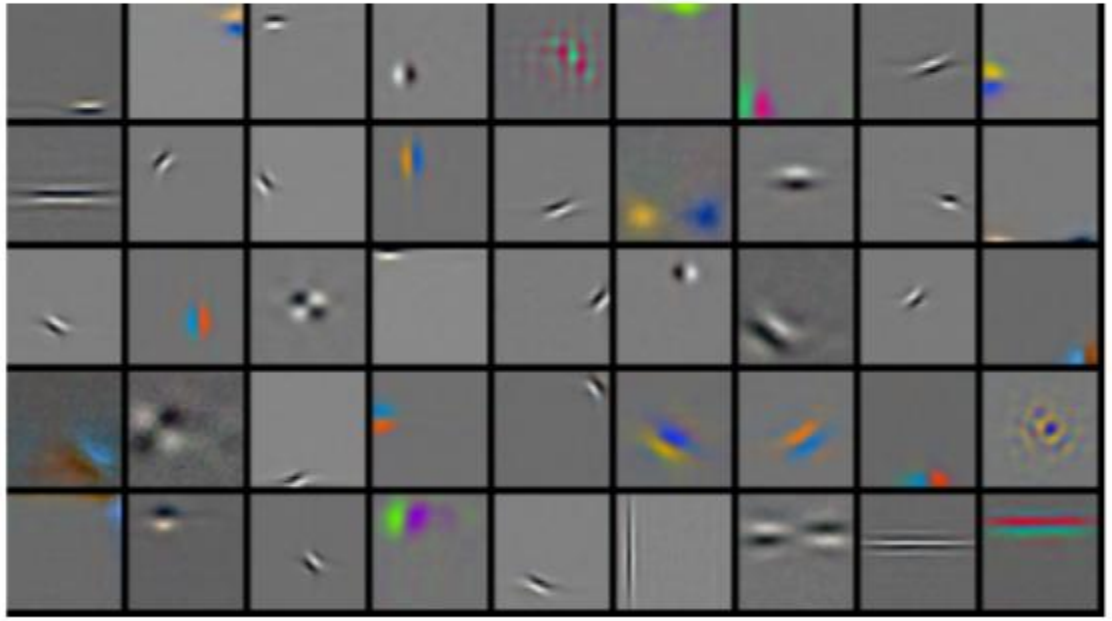
$$E(v,h) = v^T W h + c^T v + b^T h + \frac{1}{2} v^T v$$

- In this case $p(v|h)$ becomes a Gaussian distribution with mean $\mu = c + W^T h$ and identity covariance matrix
- recommend to normalize the training set by:
  - subtracting the mean of each input
  - dividing each input by the training set standard deviation
- should use a smaller learning rate than in the regular RBM

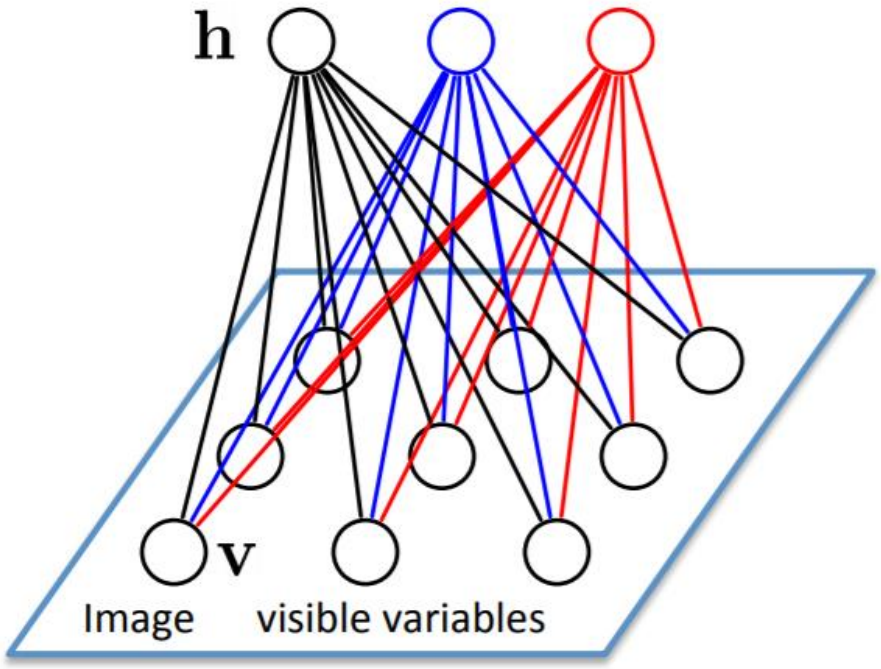
Gaussian Bernoulli RBMs

4 million **unlabelled** images

Learned features (out of 10,000)



Gaussian Bernoulli RBMs



Interpretation: Mixture of exponential number of Gaussians

$$P_\theta(\mathbf{v}) = \sum_{\mathbf{h}} P_\theta(\mathbf{v}|\mathbf{h}) P_\theta(\mathbf{h}),$$

where

$$P_\theta(\mathbf{h}) = \int_{\mathbf{v}} P_\theta(\mathbf{v}, \mathbf{h}) d\mathbf{v}$$ is an implicit prior, and

$$P(v_i = x|\mathbf{h}) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x - b_i - \sigma_i \sum_j W_{ij} h_j)^2}{2\sigma_i^2}\right)$$ Gaussian

Deep Belief Networks

- one of the first non-convolutional models to successfully admit training of deep architectures (2007)

Pre-training

- We will use a greedy, layer-wise procedure
- Train one layer at a time with unsupervised criterion
- Fix the parameters of previous hidden layers
- Previous layers viewed as feature extraction

Pre-training

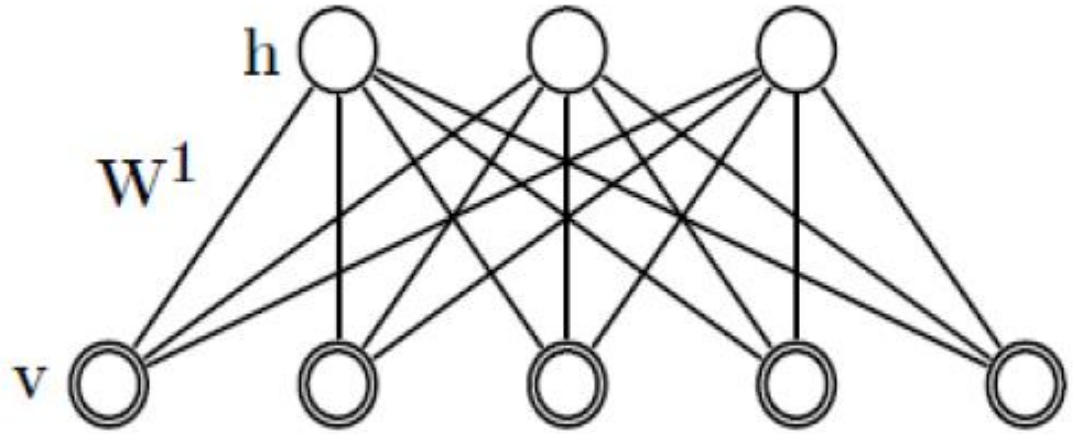
- Unsupervised Pre-training
  - first layer: find hidden unit features that are more common in training inputs than in random inputs
  - second layer: find combinations of hidden unit features that are more common than random hidden unit features
  - third layer: find combinations of combinations of ...

- Pre-training initializes the parameters in a region from which we can reach a better parameters

Deep Belief Networks

- First construct a standard RBM with visible layer v and hidden layer h

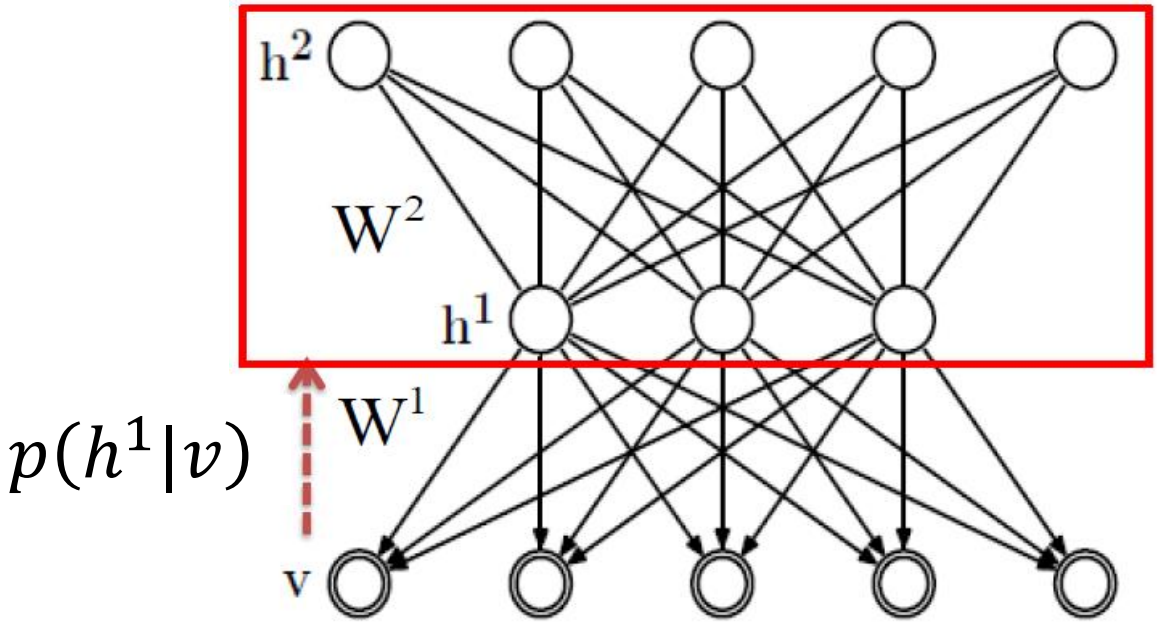
- Train this RBM



Tutorial on Deep Learning and Applications, Honglak Lee, NIPS 2010

Deep Belief Networks

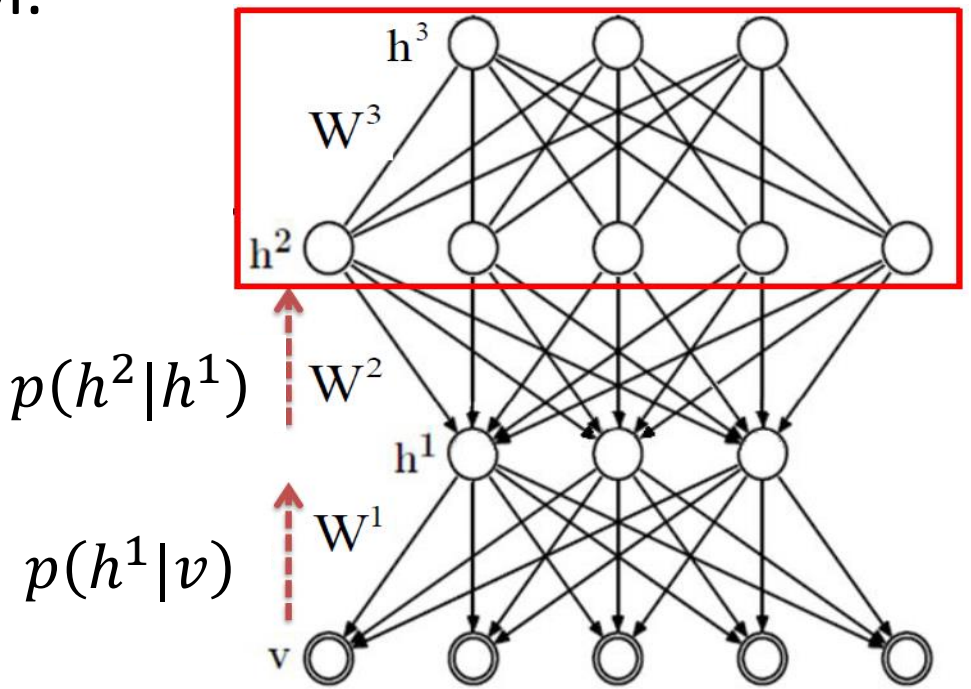
- Stack another hidden layer on top of the RBM to form a new RBM
- Fix W^1 , sample h^1 from $p(h^1|v)$ as input.
- Train W^2 as an RBM.



Tutorial on Deep Learning and Applications, Honglak Lee, NIPS 2010

Deep Belief Networks

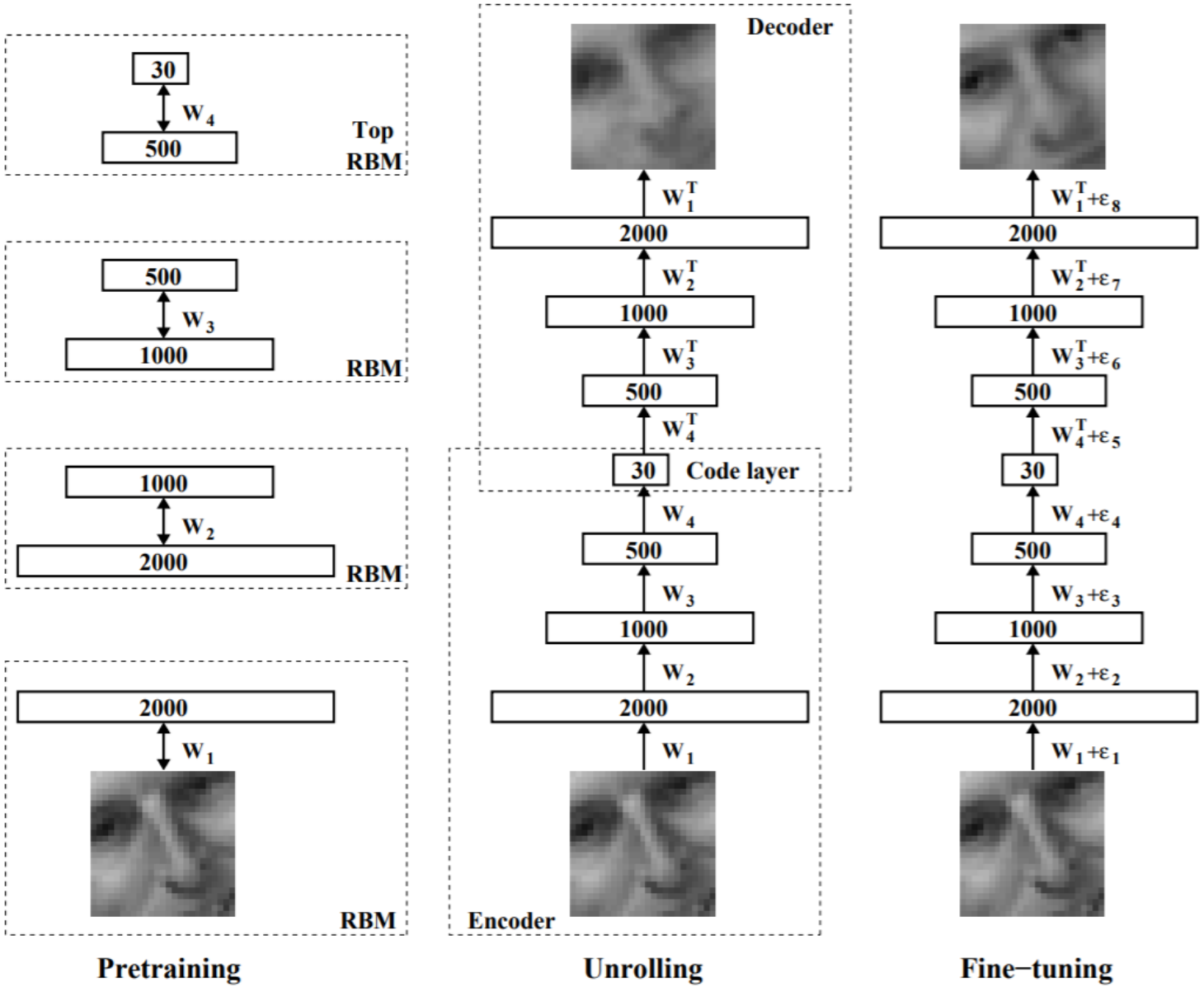
- Stack a third hidden layer on top of the RBM to form a new RBM
- Fix W^1 , W^2 sample h^2 from $p(h^2|h^1)$ as input.
- Train W^3 as RBM.
- Continue…



Tutorial on Deep Learning and Applications, Honglak Lee, NIPS 2010

Learning Deep Autoencoders

Important in the history of deep learning



The global fine-tuning uses backpropagation.

Initially encoder and decoder networks use the same weights.

Fine-tuning

- Once all layers are pre-trained
  - add output layer
  - train the whole network using supervised learning
- Supervised learning is performed as in a regular network
  - forward propagation, backpropagation and update
  - We call this last phase fine-tuning
    - all parameters are ‘‘tuned’’ for the supervised task at hand
    - representation is adjusted to be more discriminative

Fine-tuning

- Discriminative fine-tuning:
  - Use DBN to initialize a multi-layer neural network.
  - Maximize the conditional distribution:

$$\log P(\mathbf{y}|\mathbf{v})$$

Learning Deep Autoencoders

Important in the history of deep learning

- Learn probabilistic model on unlableled data
- Use learned parameters to initialize a discriminative model for a specific task
- Slightly adjust discriminative model for a specific task using a labeled set.

Stacking RBMs, Autoencoders

- Stacked Restricted Boltzmann Machines:
  - Hinton, Teh and Osindero suggested this procedure with RBMs:
    - A fast learning algorithm for deep belief nets.
  - To recognize shapes, first learn to generate images (Hinton, 2006).

- Stacked autoencoders, sparse-coding models, etc.
  - Bengio, Lamblin, Popovici and Larochelle (stacked autoencoders)
  - Ranzato, Poultney, Chopra and LeCun (stacked sparse coding models)

- Lots of others started stacking models together.

Impact of Initialization

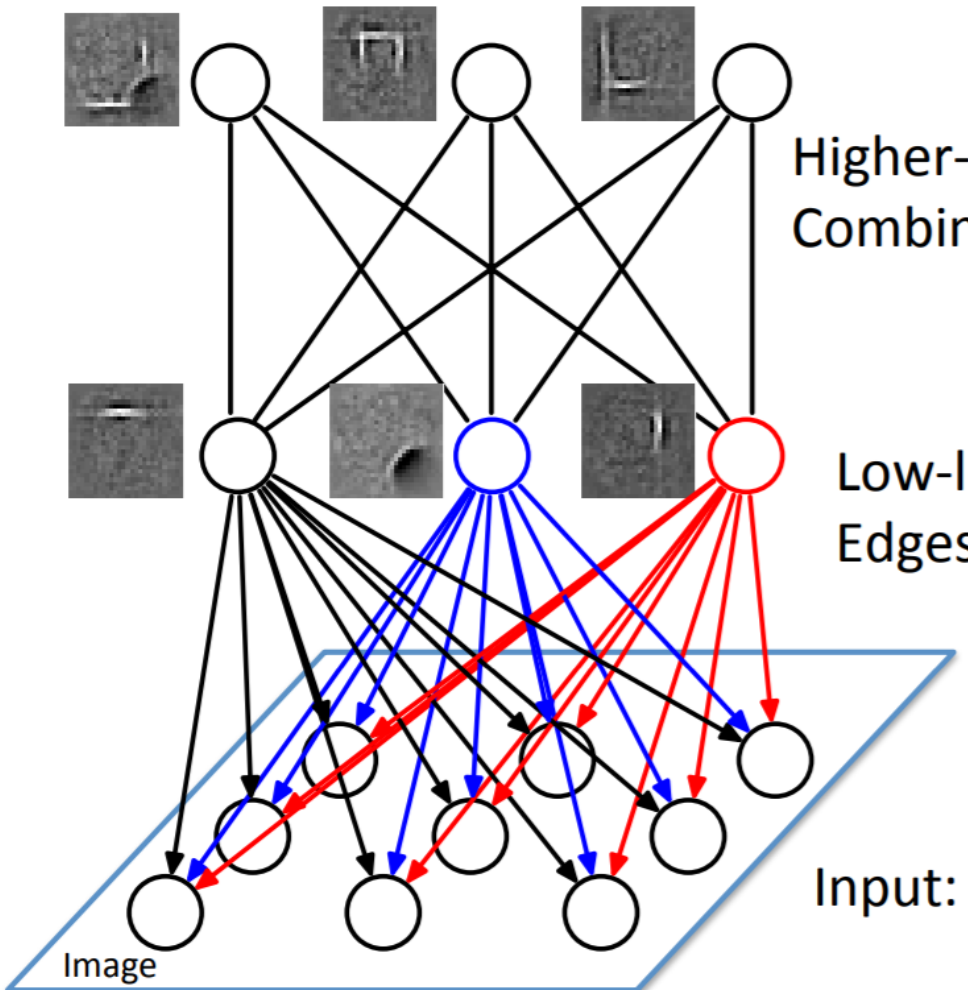
Network		MNIST-small	MNIST-rotation
Type	Depth	classif. test error	classif. test error
Neural network	1	**4.14** % ± 0.17	15.22 % ± 0.31
Deep net	2	**4.03** % ± 0.17	**10.63** % ± 0.27
	3	**4.24** % ± 0.18	11.98 % ± 0.28
	4	4.47 % ± 0.18	11.73 % ± 0.29
Deep net + autoencoder	1	3.87 % ± 0.17	11.43% ± 0.28
	2	**3.38** % ± 0.16	9.88 % ± 0.26
	3	**3.37** % ± 0.16	**9.22** % ± 0.25
	4	**3.39** % ± 0.16	**9.20** % ± 0.25
Deep net + RBM	1	3.17 % ± 0.15	10.47 % ± 0.27
	2	**2.74** % ± 0.14	9.54 % ± 0.26
	3	**2.71** % ± 0.14	**8.80** % ± 0.25
	4	**2.72** % ± 0.14	**8.83** % ± 0.24

Stacked denoising autoencoder

Stacked Autoencoders	Stacked RBMS	Stacked Denoising Autoencoders
SAA-3	**DBN-3**	**SdA-3** (ν)
3.46±0.16	3.11±0.15	**2.80±0.14** (10%)
10.30±0.27	**10.30±0.27**	**10.29±0.27** (10%)
11.28±0.28	**6.73±0.22**	10.38±0.27 (40%)
23.00±0.37	**16.31±0.32**	**16.68±0.33** (25%)
51.93±0.44	47.39±0.44	**44.49±0.44** (25%)
2.41±0.13	2.60±0.14	**1.99±0.12** (10%)
24.05±0.37	22.50±0.37	**21.59±0.36** (25%)
18.41±0.34	**18.63±0.34**	**19.06±0.34** (10%)

Vincent et al., Extracting and Composing Robust Features with Denoising Autoencoders, 2008.

Deep Belief Network



Internal representations capture higher-order statistical structure

Higher-level features:
Combination of edges

Low-level features:
Edges

Built from **unlabeled** inputs.

Input: Pixels

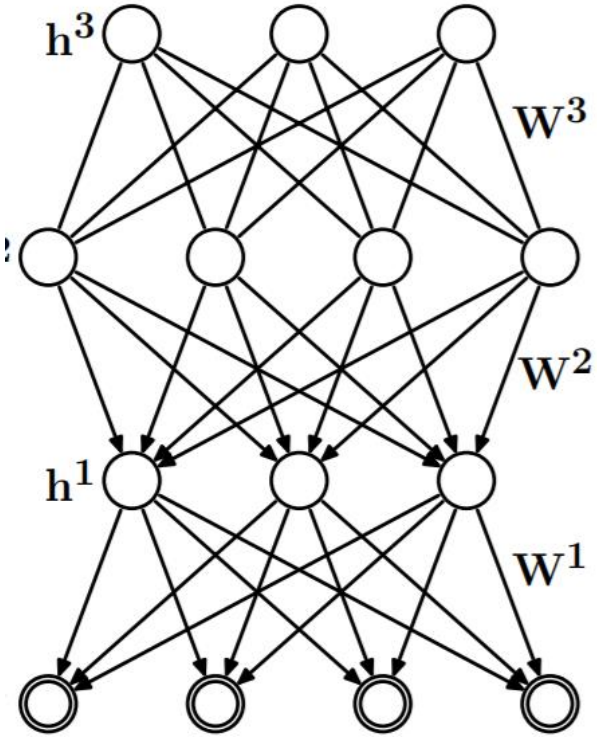
Hinton et.al. Neural Computation 2006.

DBN Training

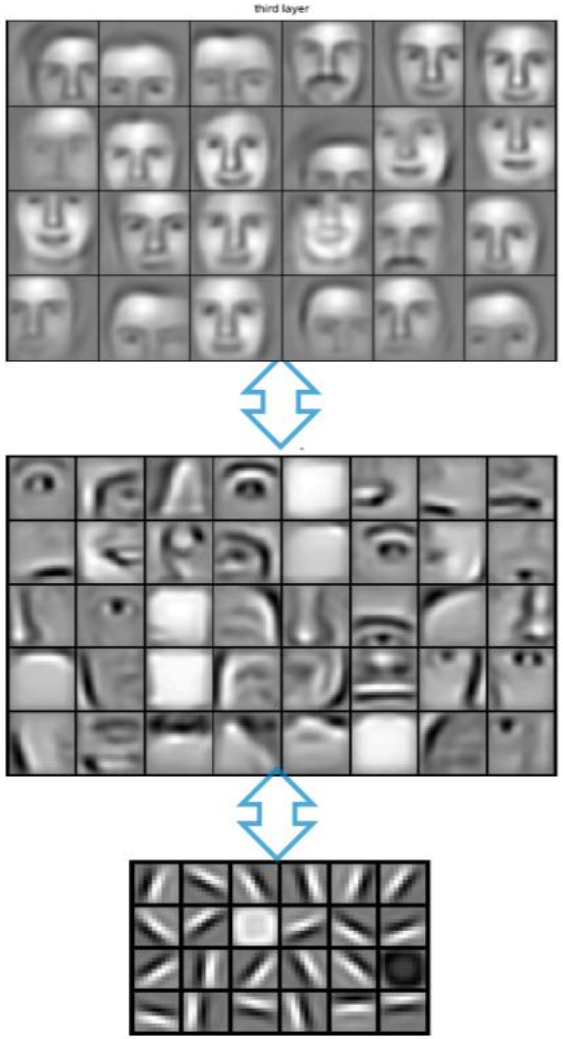
- This process of adding layers can be repeated recursively
  - we obtain the greedy layer-wise pre-training procedure for neural networks
- This procedure corresponds to maximizing a bound on the likelihood of the data in a DBN
  - in theory, if our approximation $q(h^{(1)}|v)$ is very far from the 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).

Learning Part-based Representation

Convolutional DBN



Faces



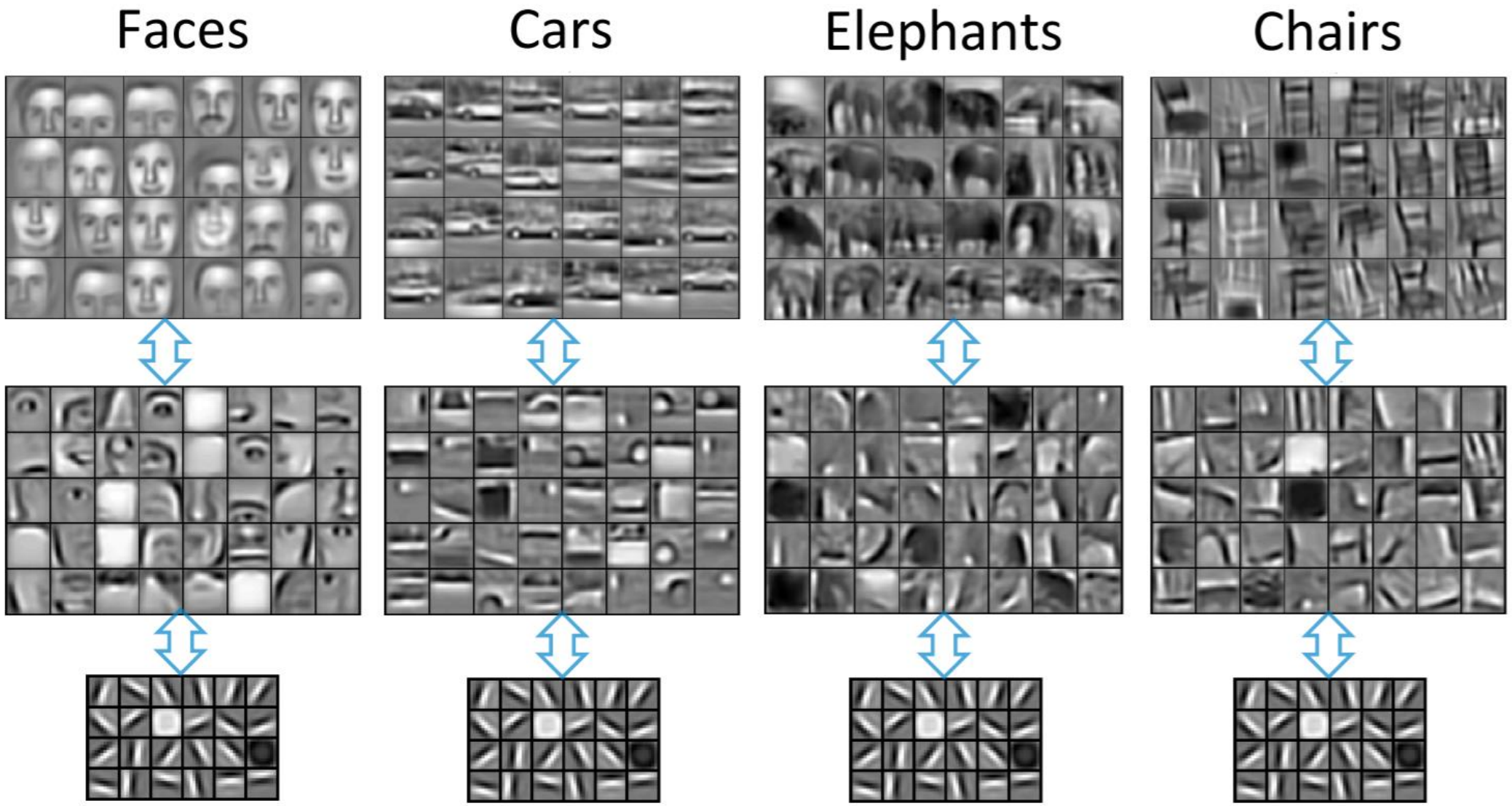
Groups of parts.

Object Parts

Trained on face images.

Lee et al., ICML 2009

Learning Part-based Representation



Lee et al., ICML 2009

Deep Autoencoders

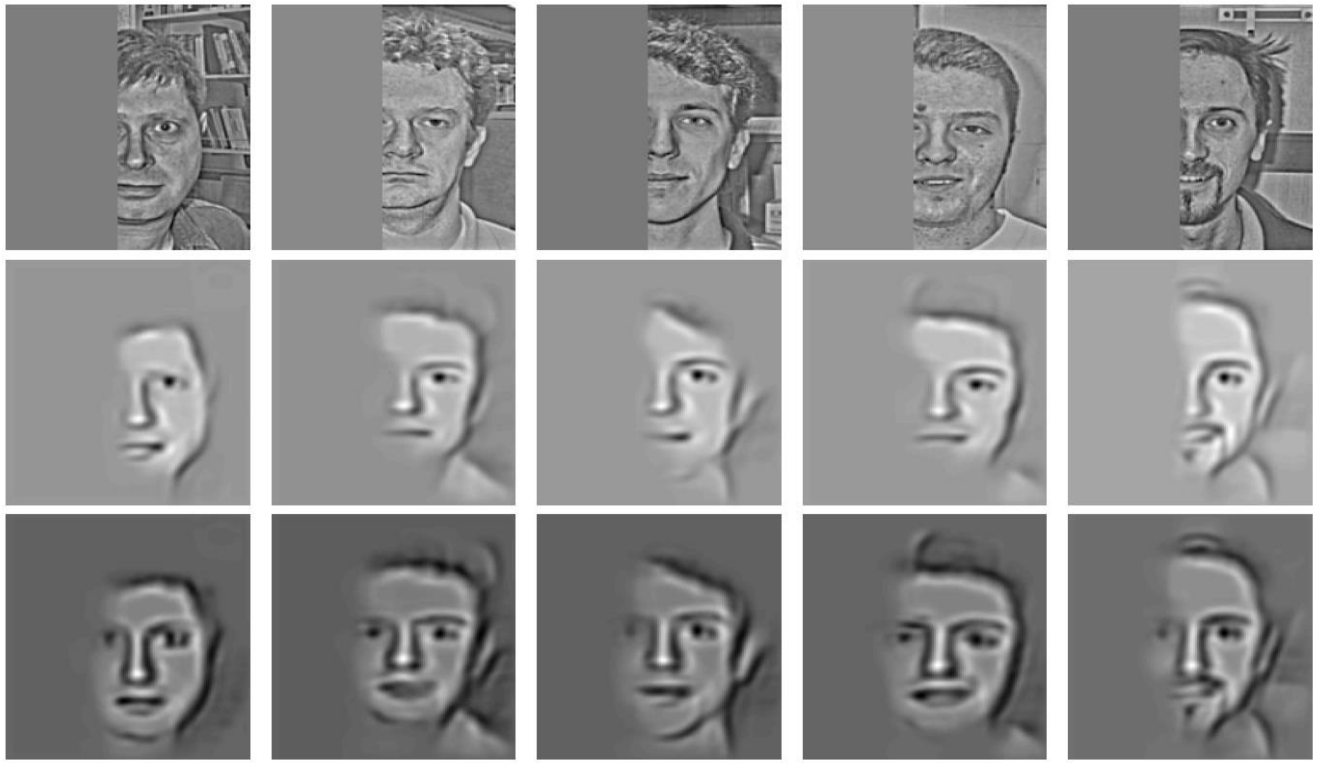
• We used 25x25 – 2000 – 1000 – 500 – 30 autoencoder to extract 30-D real-valued codes for Olivetti face patches.

• **Top**: Random samples from the test dataset.

• **Middle**: Reconstructions by the 30-dimensional deep autoencoder.

• **Bottom**: Reconstructions by the 30-dimentinoal PCA.

Deep Belief Networks: Examples



- (top) input image.
- (middle) reconstruction from the second layer units after single bottom-up pass, by projecting the second layer activations into the image space.
- (bottom) reconstruction from second layer after 20 iterations of block Gibbs sampling.

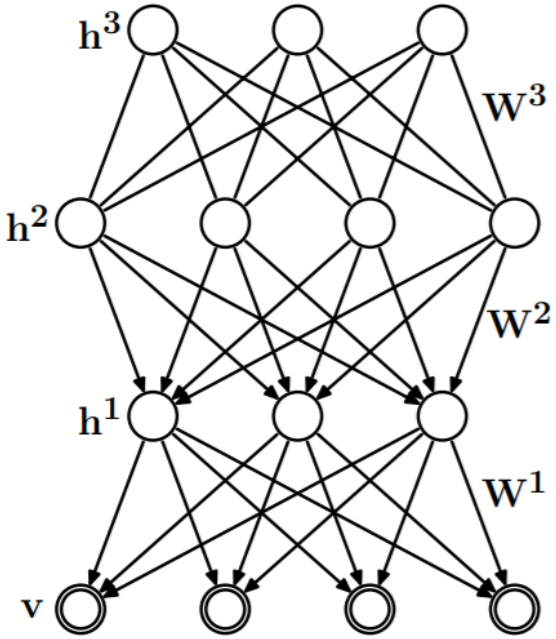
Lee et al., Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations, ICML 2009.

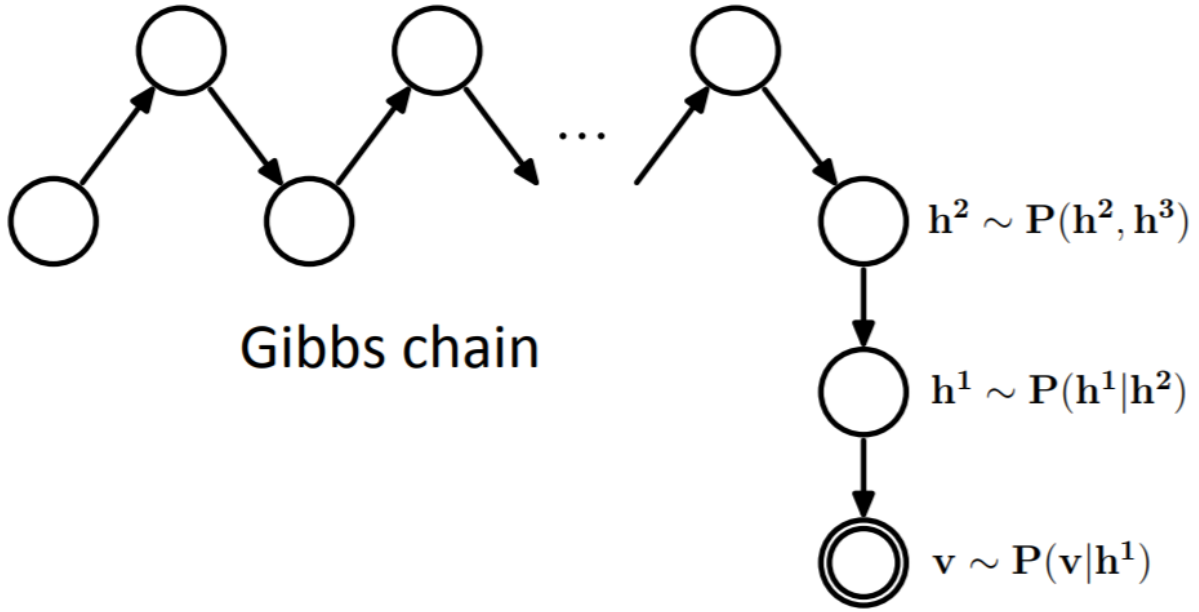
Sampling from DBN

- To sample from the DBN model:

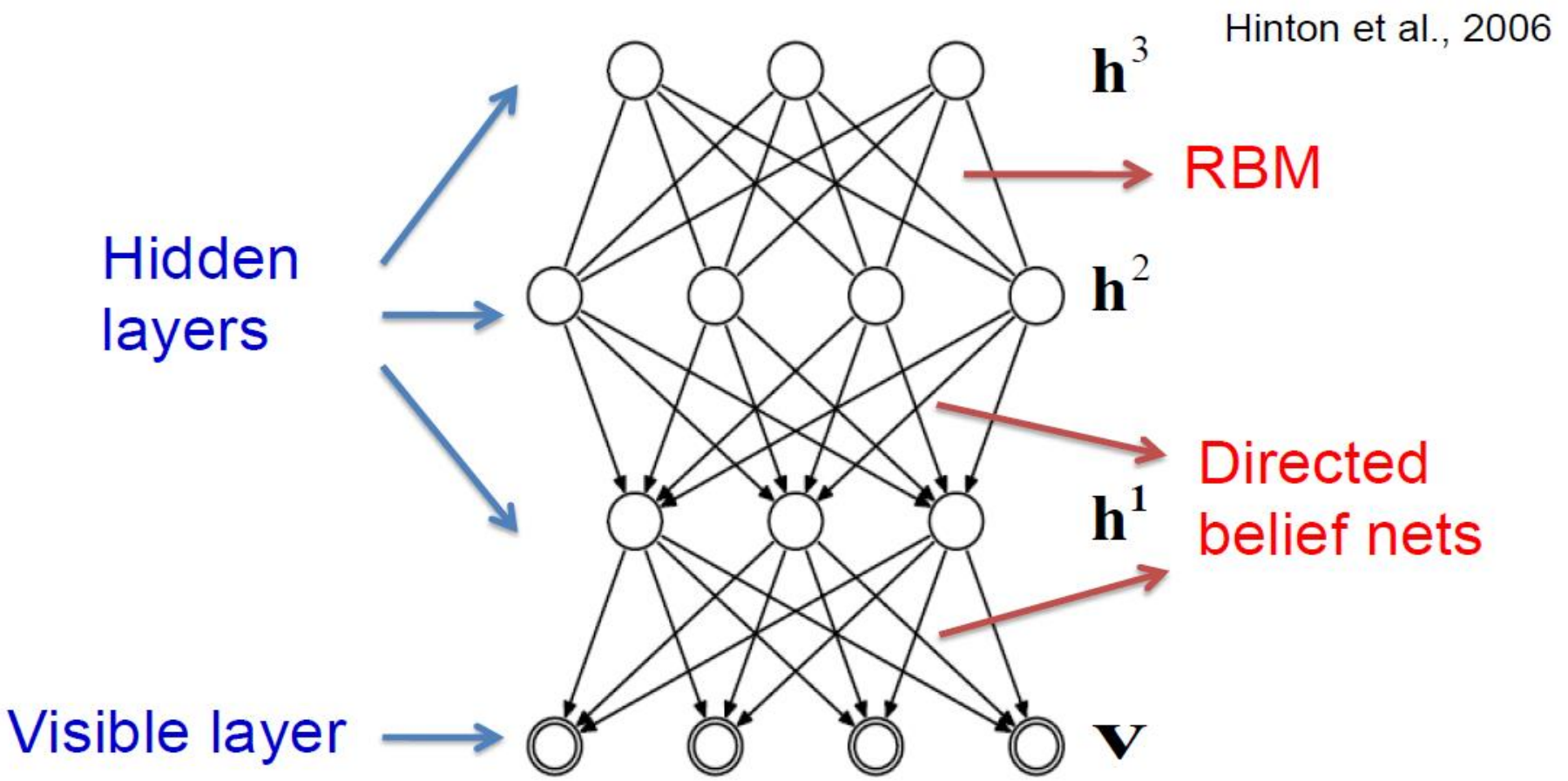
$$P(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3) = P(\mathbf{v}|\mathbf{h}^1)P(\mathbf{h}^1|\mathbf{h}^2)P(\mathbf{h}^2, \mathbf{h}^3)$$

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





Deep Belief Networks

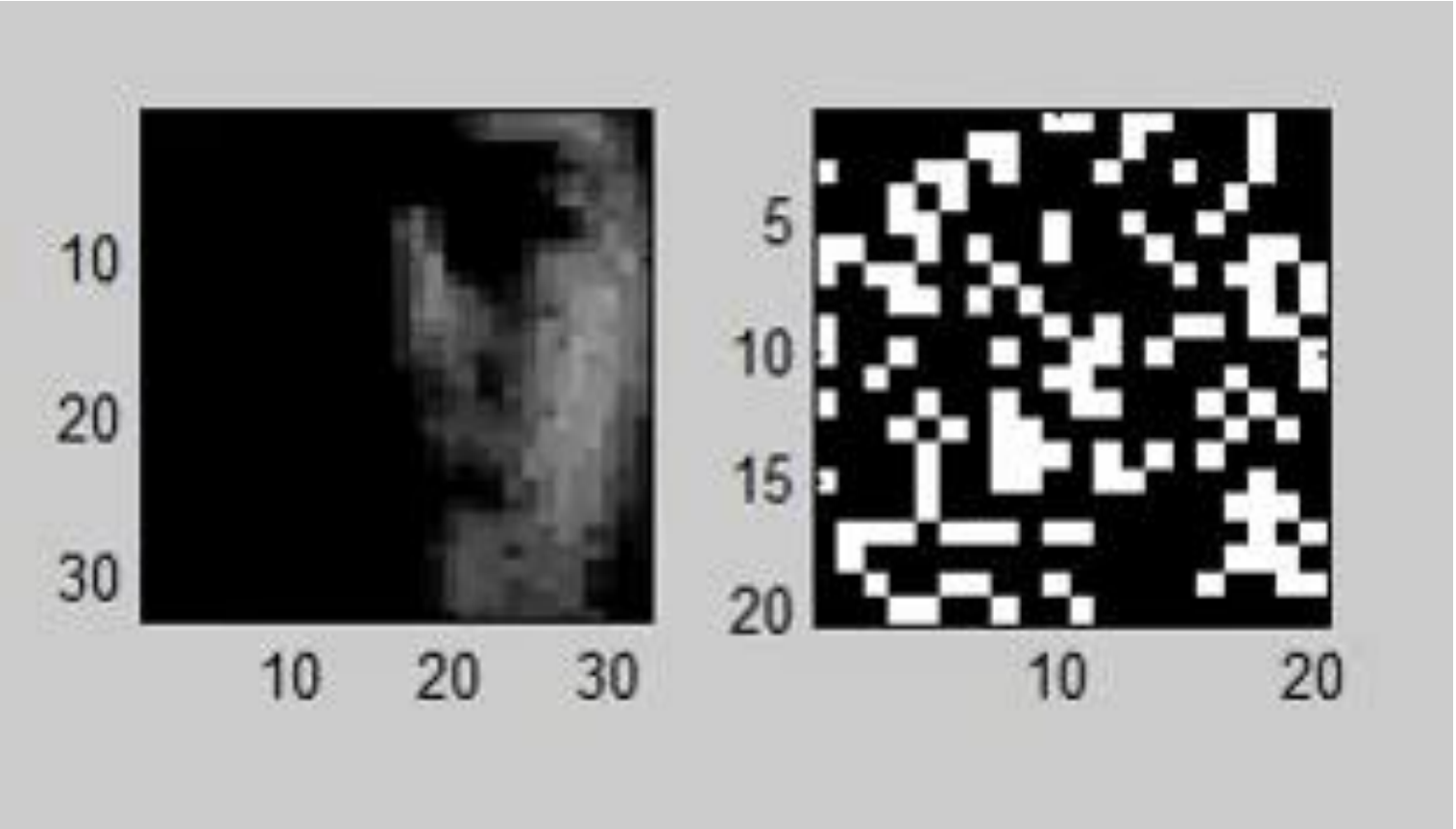


Tutorial on Deep Learning and Applications, Honglak Lee, NIPS 2010

Deep Belief Networks Examples

You can also run a DBN generator unsupervised:

https://www.youtube.com/watch?v=RSF5PbwKU3I

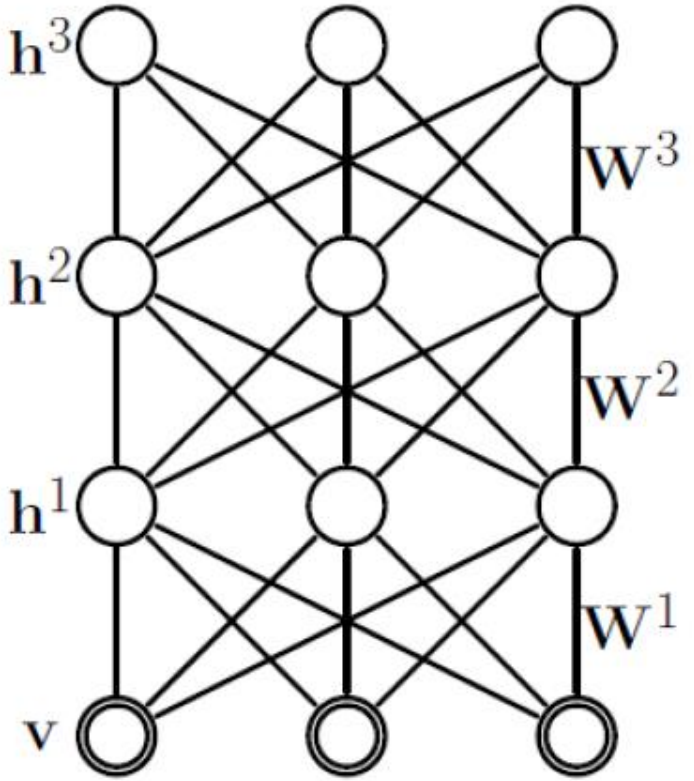


Deep Boltzmann Machines (DBM)

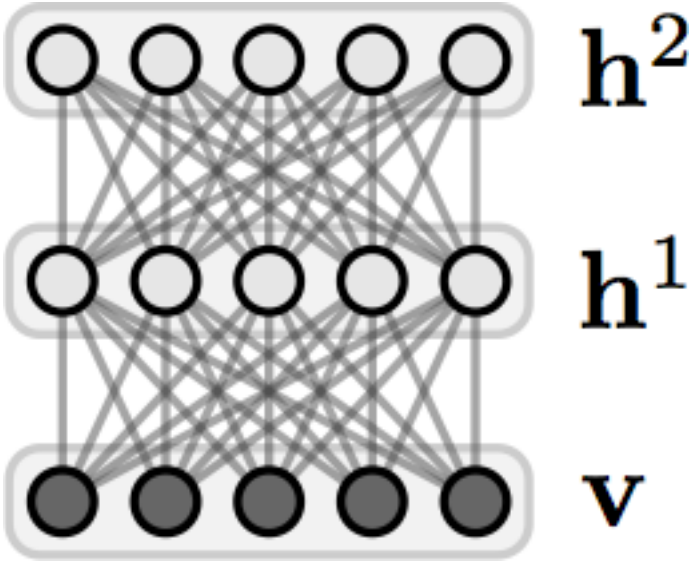
There are limitations on the types of structure that can be represented efficiently by a single layer of hidden variables.

Similar idea, but more layers.

Training more complicated…



Deep architectures

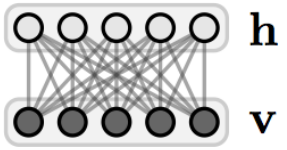


DBM

$$E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2) = \sum_i b_i v_i + \sum_{i,j} w_{ij}^1 v_i h_j^1 + \sum_j c_j^1 h_j^1 + \sum_{j,k} w_{jk}^2 h_j^1 h_k^2 + \sum_k c_k^2 h_k^2.$$

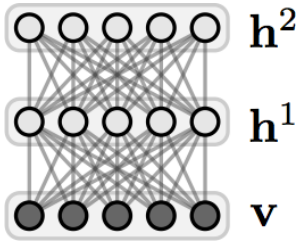
Shallow and Deep architectures

- Modeling high-order and long-range interactions



RBM

$$E(\mathbf{v}, \mathbf{h}) = \sum_i b_i v_i + \sum_{i,j} w_{ij} v_i h_j + \sum_j c_j h_j$$



DBM

$$E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2) = \sum_i b_i v_i + \sum_{i,j} w^1_{ij} v_i h^1_j + \sum_j c^1_j h^1_j + \sum_{j,k} w^2_{jk} h^1_j h^2_k + \sum_k c^2_k h^2_k.$$

Deep Boltzmann Machines

$$
\begin{aligned}
E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2) &= \sum_i b_i v_i + \sum_{i,j} w_{ij}^1 v_i h_j^1 + \sum_j c_j^1 h_j^1 \\
&+ \sum_{j,k} w_{jk}^2 h_j^1 h_k^2 + \sum_k c_k^2 h_k^2.
\end{aligned}
$$

$$
p(\mathbf{v}; \Theta) = \sum_{\mathbf{h}^1, \mathbf{h}^2} \frac{1}{Z(\Theta)} \exp\{-E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2; \Theta)\}
$$

Deep Boltzmann Machines

Conditional distributions remain factorized due to layering.

$$p(v_i = 1|\mathbf{h}^1) = \sigma(\sum_j w_{ij}^1 h_j^1 + b_i)$$

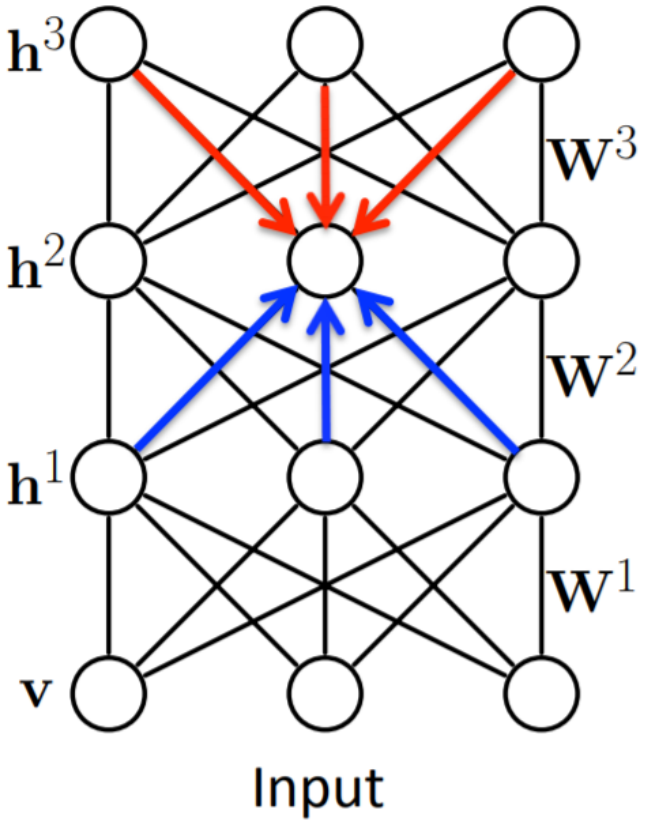
$$p(h_j^1 = 1|\mathbf{v}, \mathbf{h}^2) = \sigma(\sum_i w_{ij}^1 v_i + \sum_k w_{jk}^2 h_k^2 + c_j^1)$$

$$p(h_k^2 = 1|\mathbf{h}^1) = \sigma(\sum_j w_{jk}^2 h_j^1 + c_k^2)$$

Deep Boltzmann Machines

$$P_\theta(\mathbf{v}) = \frac{P^*(\mathbf{v})}{\mathcal{Z}(\theta)} = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}^1,\mathbf{h}^2,\mathbf{h}^3} \exp\left[\mathbf{v}^\top W^1 \mathbf{h}^1 + \mathbf{h}^{1\top} W^2 \mathbf{h}^2 + \mathbf{h}^{2\top} W^3 \mathbf{h}^3\right]$$

Deep Boltzmann Machine



All connections are undirected.

Bottom-up and Top-down:

$$P(h_k^2 = 1|\mathbf{h}^1, \mathbf{h}^3) = \sigma\left(\sum_j W_{jk}^2 h_j^1 + \sum_m W_{km}^3 h_m^3\right)$$

Bottom-up

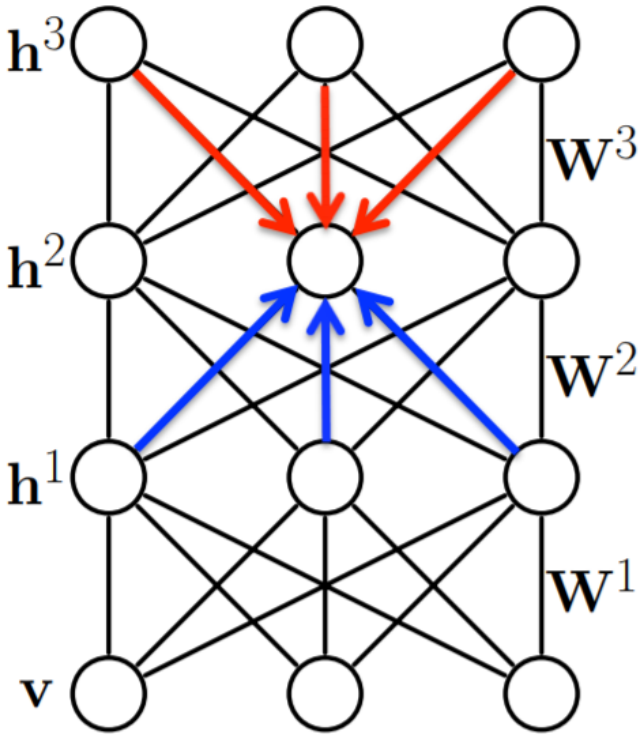
Top-Down

Unlike many existing feed-forward models: ConvNet (LeCun), HMAX (Poggio et.al.), Deep Belief Nets (Hinton et.al.)

Deep Boltzmann Machines

$$P_\theta(\mathbf{v}) = \frac{P^*(\mathbf{v})}{\mathcal{Z}(\theta)} = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}^1,\mathbf{h}^2,\mathbf{h}^3} \exp\left[\mathbf{v}^\top W^1 \mathbf{h}^1 + \mathbf{h}^{1\top} W^2 \mathbf{h}^2 + \mathbf{h}^{2\top} W^3 \mathbf{h}^3\right]$$

Deep Boltzmann Machine



Conditional distributions:

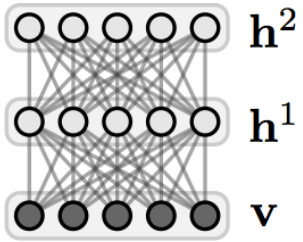
$$P(h_j^1 = 1|\mathbf{v}, \mathbf{h}^2) = \sigma\left(\sum_i W_{ij}^1 v_i + \sum_k W_{jk}^2 h_k^2\right)$$

$$P(h_k^2 = 1|\mathbf{h}^1, \mathbf{h}^3) = \sigma\left(\sum_j W_{jk}^2 h_j^1 + \sum_m W_{km}^3 h_m^3\right)$$

$$P(h_m^3 = 1|\mathbf{h}^2) = \sigma\left(\sum_k W_{km}^3 h_k^2\right)$$

- Note that exact computation of $P(\mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3|\mathbf{v})$ is intractable.

Deep Boltzmann Machines



DBM

$$E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2) = \sum_i b_i v_i + \sum_{i,j} w_{ij}^1 v_i h_j^1 + \sum_j c_j^1 h_j^1 + \sum_{j,k} w_{jk}^2 h_j^1 h_k^2 + \sum_k c_k^2 h_k^2.$$

- Probabilistic
- Generative
- Powerful

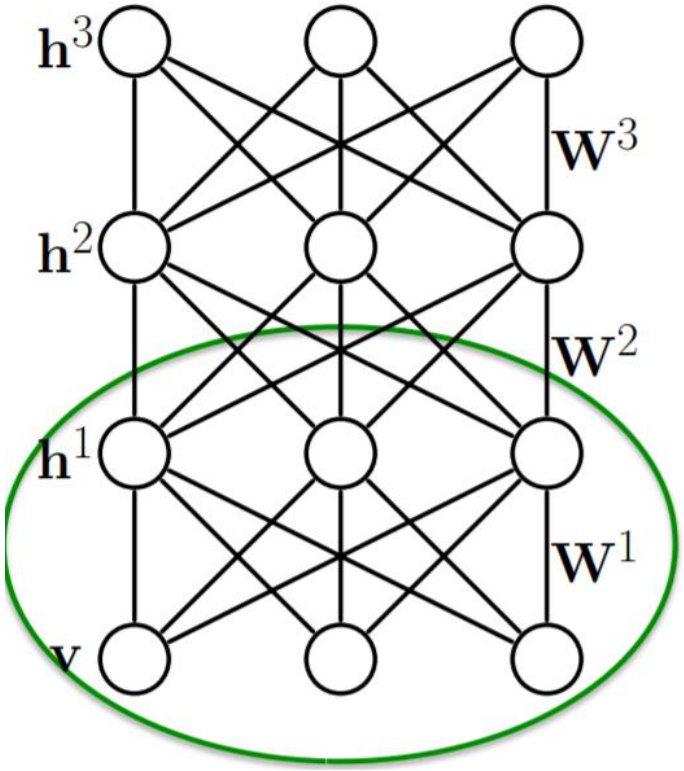
DBM's have the potential of learning internal representations that become increasingly complex at higher layers

Typically trained with many examples.

DBM: Learning

$$P_\theta(\mathbf{v}) = \frac{P^*(\mathbf{v})}{\mathcal{Z}(\theta)} = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}^1,\mathbf{h}^2,\mathbf{h}^3} \exp\left[\mathbf{v}^\top W^1 \mathbf{h}^1 + \mathbf{h}^{1\top} W^2 \mathbf{h}^2 + \mathbf{h}^{2\top} W^3 \mathbf{h}^3\right]$$

Deep Boltzmann Machine



$\theta = \{W^1, W^2, W^3\}$ model parameters

- Dependencies between hidden variables.

Maximum likelihood learning:

$$\frac{\partial \log P_\theta(\mathbf{v})}{\partial W^1} = \mathrm{E}_{P_{data}}[\mathbf{v}\mathbf{h}^{1\top}] - \mathrm{E}_{P_\theta}[\mathbf{v}\mathbf{h}^{1\top}]$$

Problem: Both expectations are intractable!

$$P_{data}(\mathbf{v}, \mathbf{h}^1) = P_\theta(\mathbf{h}^1|\mathbf{v}) P_{data}(\mathbf{v})$$

$$P_{data}(\mathbf{v}) = \frac{1}{N} \sum_{n=1}^{N} \delta(\mathbf{v} - \mathbf{v_n})$$

Not factorial any more!

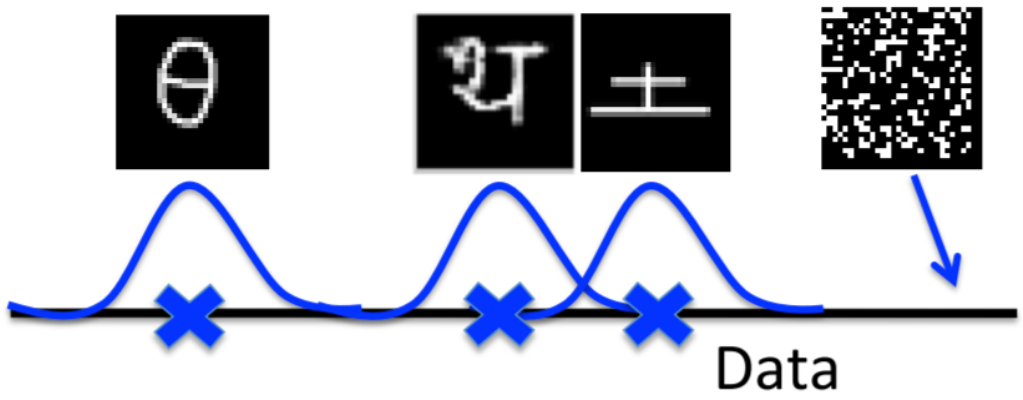
DBM: Learning

- Dependencies between hidden variables.

Maximum likelihood learning:

$$\frac{\partial \log P_\theta(\mathbf{v})}{\partial W^1} = \mathrm{E}_{P_{data}}[\mathbf{v}\mathbf{h}^{1\top}] - \mathrm{E}_{P_\theta}[\mathbf{v}\mathbf{h}^{1\top}]$$

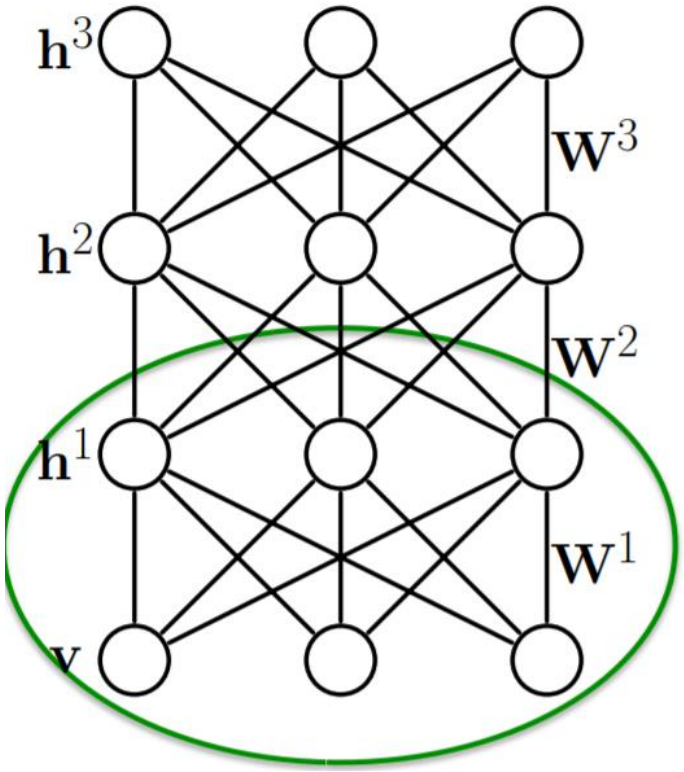
Problem: Both expectations are intractable!



DBM: Learning

$$P_\theta(\mathbf{v}) = \frac{P^*(\mathbf{v})}{\mathcal{Z}(\theta)} = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}^1,\mathbf{h}^2,\mathbf{h}^3} \exp\left[\mathbf{v}^\top W^1 \mathbf{h}^1 + \mathbf{h}^{1\top} W^2 \mathbf{h}^2 + \mathbf{h}^{2\top} W^3 \mathbf{h}^3\right]$$

Deep Boltzmann Machine



$\theta = \{W^1, W^2, W^3\}$ model parameters

(Approximate) Maximum Likelihood:

$$\frac{\partial \log P_\theta(\mathbf{v})}{\partial W^1} = \mathbb{E}_{P_{data}}[\mathbf{v}\mathbf{h}^{1\top}] - \mathbb{E}_{P_\theta}[\mathbf{v}\mathbf{h}^{1\top}]$$

Variational Inference

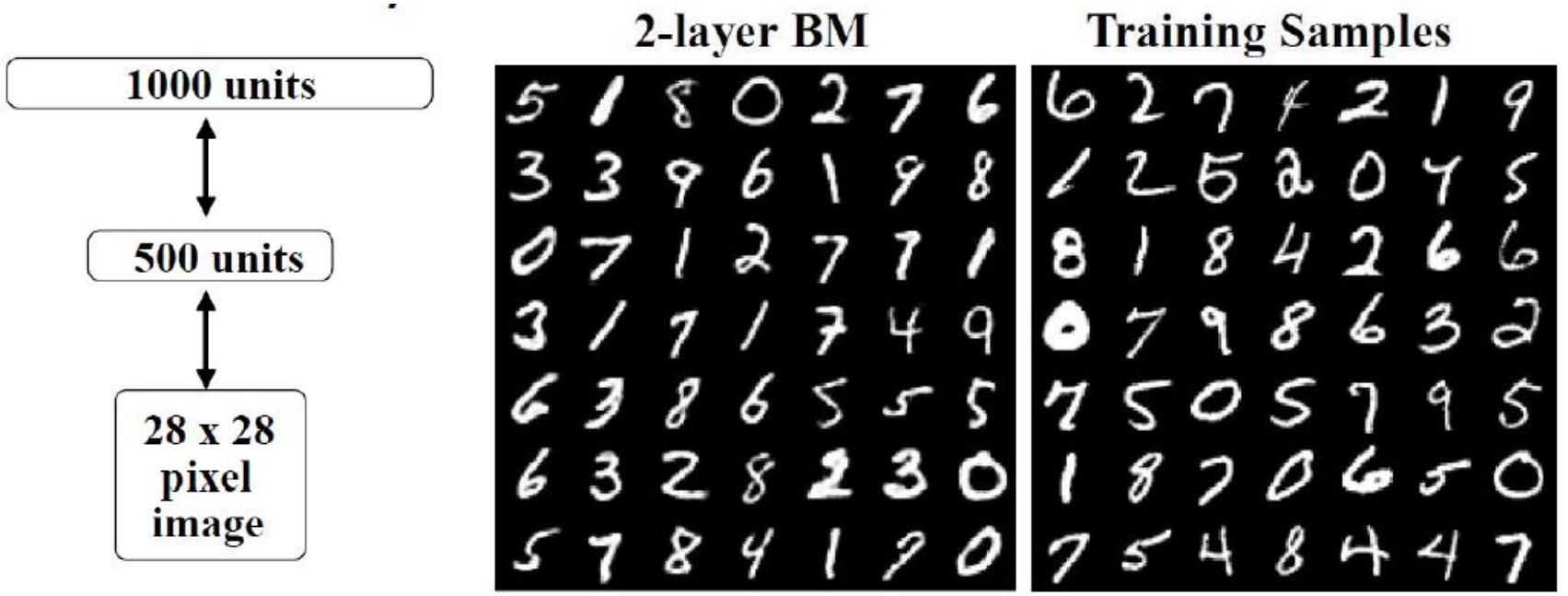
Stochastic Approximation (MCMC-based)

Experiments

60,000 training and 10,000 testing examples

0.9 million parameters

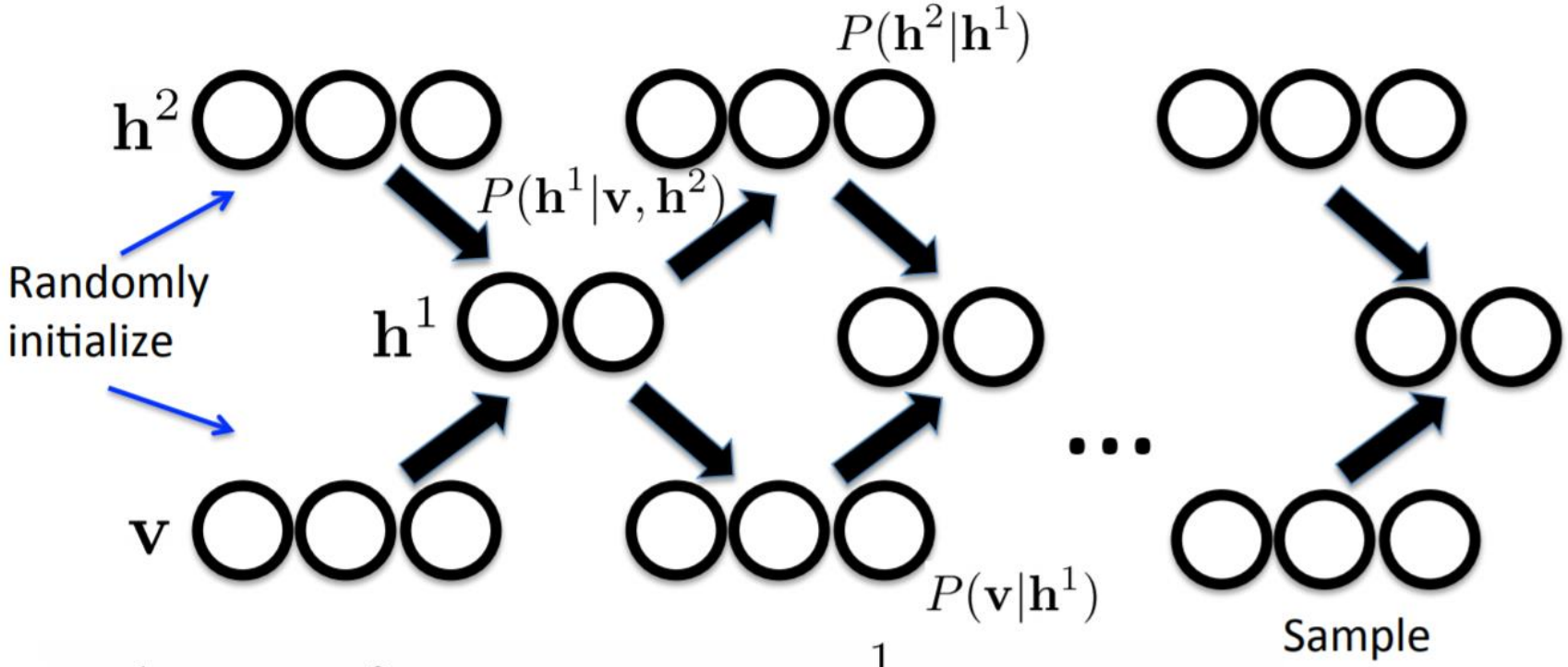
Gibbs sampler for 100,000 steps



Tutorial on Deep Learning and Applications, Honglak Lee, NIPS 2010

Sampling from DBMs

Sampling from two-hidden layer DBM by running a Markov chain:



$$P(h_m^1 = 1|\mathbf{v}, \mathbf{h}^2) = \frac{1}{1 + \exp(-\sum_i W_{im}^1 v_i - \sum_j W_{mj}^2 h_j^2)}$$

$$P(h_j^2 = 1|\mathbf{h}^1) = \frac{1}{1 + \exp(-\sum_m W_{mj}^2 h_m^1)}$$

$$P(v_i = 1|\mathbf{h}^1) = \frac{1}{1 + \exp(-\sum_m W_{im}^1 h_m^1)}$$

HandwriDng Recognition

MNIST Dataset
60,000 examples of 10 digits

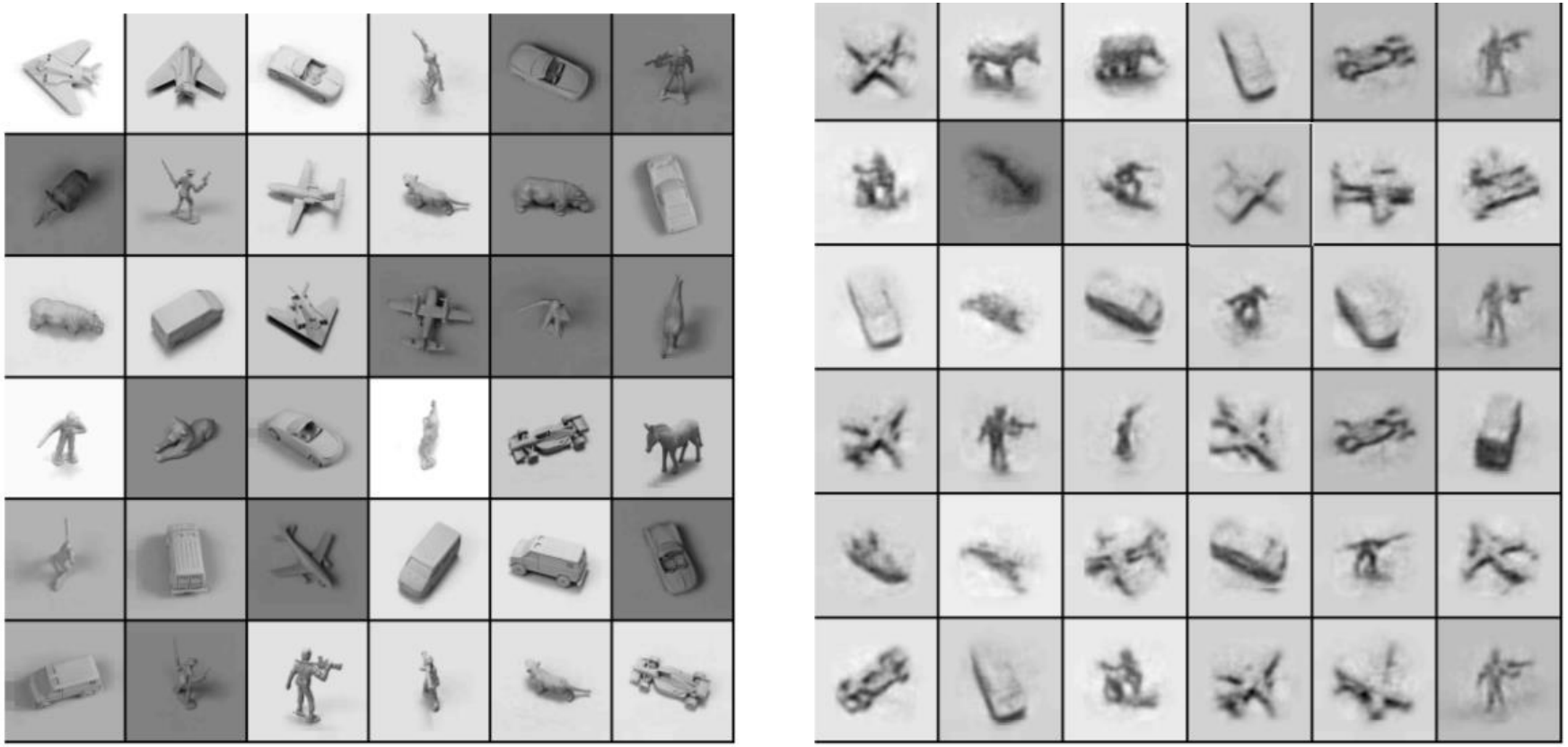
Learning Algorithm	Error
Logistic regression	12.0%
K-NN	3.09%
Neural Net (Platt 2005)	1.53%
SVM (Decoste et.al. 2002)	1.40%
Deep Autoencoder (Bengio et. al. 2007)	1.40%
Deep Belief Net (Hinton et. al. 2006)	1.20%
DBM	**0.95%**

Optical Character Recognition
42,152 examples of 26 English letters

Learning Algorithm	Error
Logistic regression	22.14%
K-NN	18.92%
Neural Net	14.62%
SVM (Larochelle et.al. 2009)	9.70%
Deep Autoencoder (Bengio et. al. 2007)	10.05%
Deep Belief Net (Larochelle et. al. 2009)	9.68%
DBM	**8.40%**

Permutation-invariant version.

Generative Model of 3-D Objects

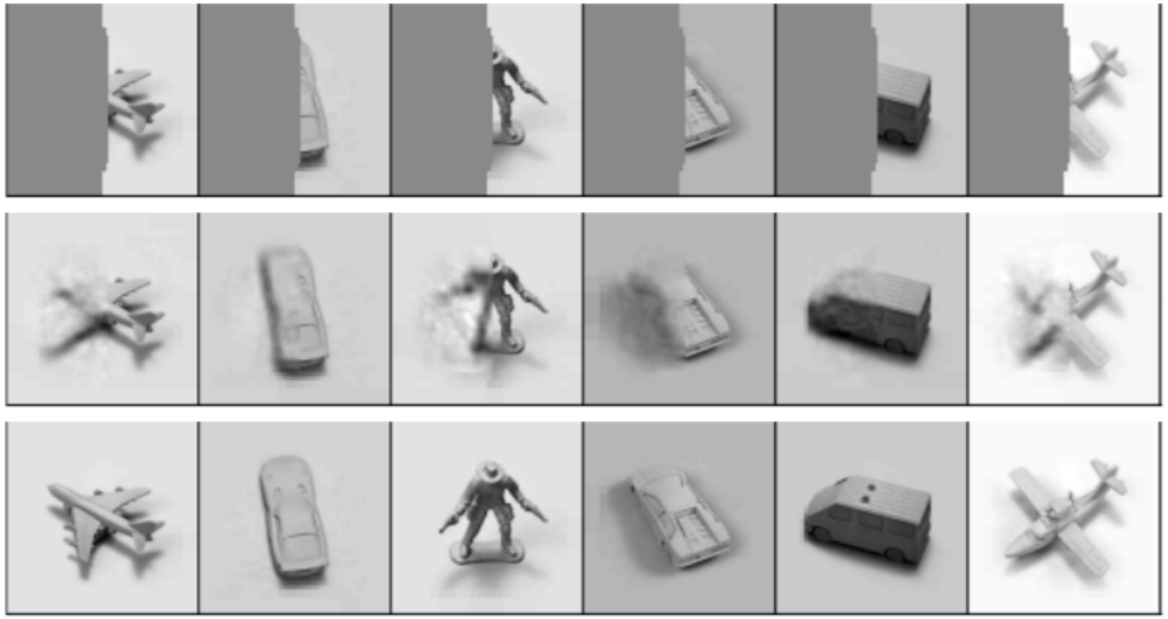


24,000 examples, 5 object categories, 5 different objects within each category, 6 lightning conditions, 9 elevations, 18 azimuths.

Generative Model of 3-D Objects

Learning Algorithm	Error
Logistic regression	22.5%
K-NN (LeCun 2004)	18.92%
SVM (Bengio & LeCun 2007)	11.6%
Deep Belief Net (Nair & Hinton 2009)	9.0%
DBM	**7.2%**

Pattern Completion



Permutation-invariant version.

Experiments

Running a generator open-loop:

https://www.youtube.com/watch?v=-l1QTbgLTyQ

Resources

- Deep Learning Book, Chapter 20 (and Chapter 18).