CSE 5526: Introduction to Neural Networks

Deep Belief Networks

Deep circuits can represent logic expressions using exponentially fewer components

- Consider the parity problem (project 1)
- for $\boldsymbol{x} \in \{0,1\}^D$

$$f(\boldsymbol{x}) = \begin{cases} 1, & \sum_i x_i \text{ is even} \\ 0, & \text{otherwise} \end{cases}$$

- The depth-2 circuit to compute $f(\boldsymbol{x})$ uses $O(2^D)$ AND, OR, and NOT elements
- A depth-*D* circuit to compute $f(\boldsymbol{x})$ uses $O(D)$
- In general, a depth-*k* circuit uses $\mathrm{O}\left(D^{\frac{k-2}{k-1}} 2^{D^{\frac{1}{k-1}}}\right)$
  - See (Hastad, 1987, Thm. 2.2)

Backpropagation through deep neural nets leads to the vanishing gradient problem

- Recall, gradient of error WRT weights in layer ℓ

$$\frac{\partial E(\boldsymbol{w})}{\partial w_{ji}^{\ell}} = -\delta_j^{\ell} y_i^{\ell-1} \text{ where } \delta_j^{\ell} = \varphi'\left(v_j^{\ell}\right) \sum_k \delta_k^{\ell+1} w_{jk}^{\ell}$$

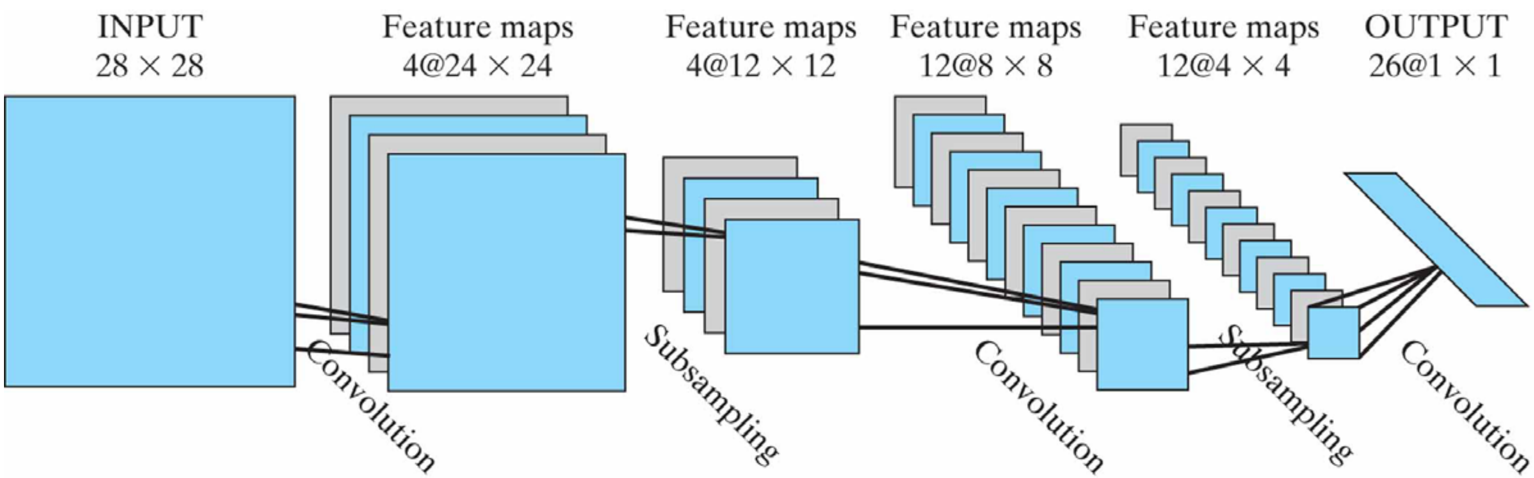
- In matrix notation, define vector $\boldsymbol{\delta}^{\ell}$ and diagonal matrix $\Phi'^{(\ell)}$ with $\varphi'\left(v_j^{\ell}\right)$ on its diagonal, then

$$\begin{aligned} \boldsymbol{\delta}^{\ell} &= \Phi'^{(\ell)} W^{\ell} \boldsymbol{\delta}^{\ell+1} \\ &= \Phi'^{(\ell)} W^{\ell} \Phi'^{(\ell+1)} W^{\ell+1} \dots \boldsymbol{\delta}^{L} \\ &\approx (\Phi' W)^{L-\ell} \boldsymbol{\delta}^{L} \end{aligned}$$

- Generally, $(\Phi' W)^{L-\ell}$ either goes to ∞ or 0

Convolutional networks are deep networks that are feasible to train

- Neural network that learns "receptive fields"
  - And applies them across different spatial positions
- Weight matrices are very constrained
- Train using standard backprop



LeNet-1 zipcode recognizer

- Trained on 7300 digits and tested on 2000 new ones
  - 1% error on training set, 5% error on test set
  - If allowing no decision, 1% error on the test set
  - Difficult task (see examples)
- Remark: constraining network connectivity is a way of incorporating prior knowledge about a problem
  - Backprop applies whether or not the network is constrained

LeNet-1 zipcode recognizer architecture

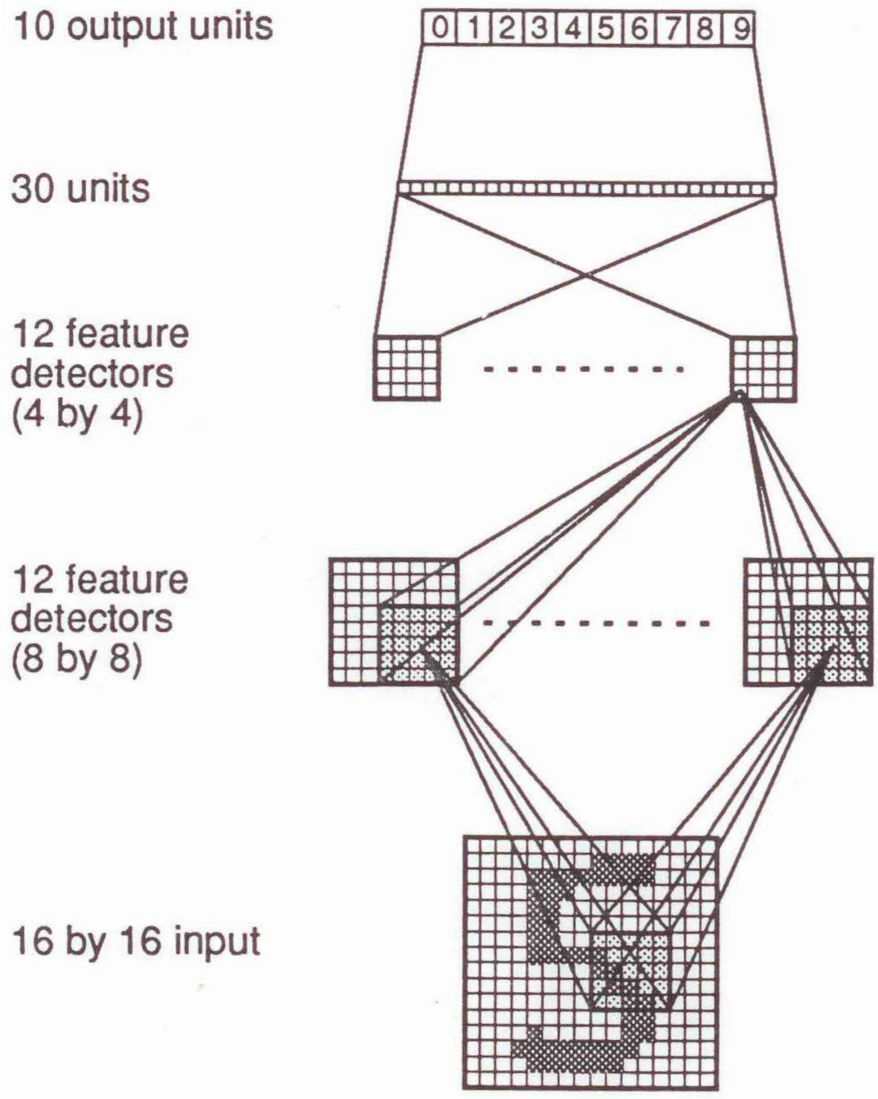


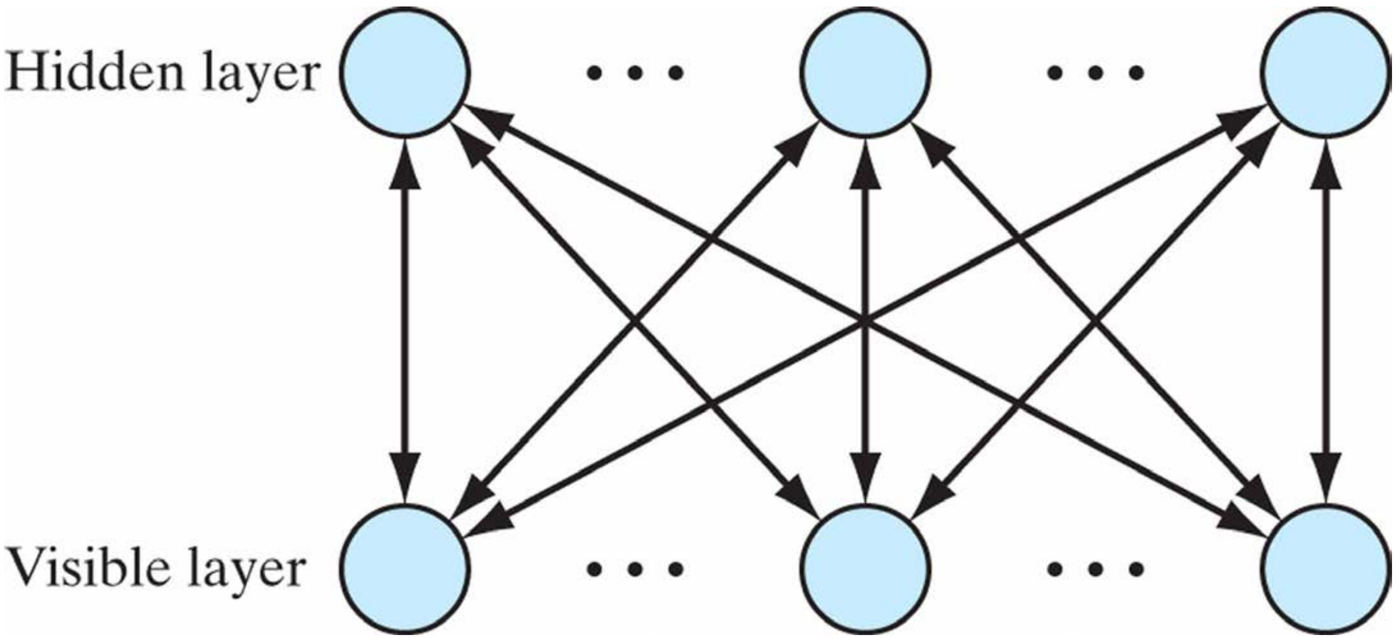
FIGURE 6.10 Archi
of the ZIP-code re:
network.

Another way to train deep neural nets is to use unsupervised pre-training

- Build training up from the bottom
  - Train a shallow model to describe the data
  - Treat that as a fixed transformation
  - Train another shallow model on transformed data
  - Etc.
- No long-distance gradients necessary
- Initialize a deep neural network with these params

Restricted Boltzmann machines can be used as building blocks in this way

- A restricted Boltzmann machine (RBM) is a Boltzmann machine with one visible layer and one hidden layer, and no connection within each layer



RBM conditions are easy to compute

- The energy function is:

$$E(\mathbf{v},\mathbf{h}) = -\frac{1}{2}\sum_i\sum_j w_{ji}v_j\,h_i = -\frac{1}{2}\boldsymbol{v}^T W\boldsymbol{h}$$

- So $p(\boldsymbol{v}|\boldsymbol{h})$, $p(\boldsymbol{h}|\boldsymbol{v})$ are now easy to compute
  - No Gibbs sampling necessary

$$p(\boldsymbol{h}|\boldsymbol{v}) = \exp\left(\frac{1}{2}\boldsymbol{v}^T W\boldsymbol{h}\right)\left(\sum_{\boldsymbol{h}}\exp\left(\frac{1}{2}\boldsymbol{v}^T W\boldsymbol{h}\right)\right)^{-1}$$

$$\sum_{\boldsymbol{h}}\exp\left(\frac{1}{2}\boldsymbol{v}^T W\boldsymbol{h}\right) = \prod_i\sum_{h_i}\exp\left(\frac{1}{2}\boldsymbol{v}^T W_{i.}h_i\right)$$

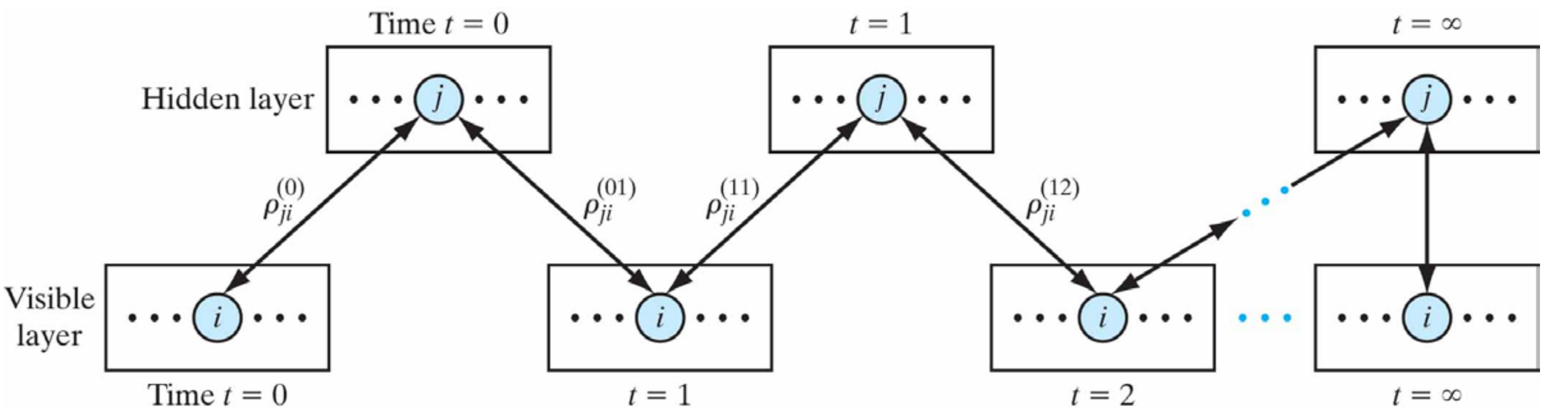
RBM training still needs Gibbs sampling

- Setting $T = 1$, we have

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \rho_{ji}{}^{+} - \rho_{ji}{}^{-}$$

$$= \left\langle h_i{}^{(0)} v_j{}^{(0)} \right\rangle - \left\langle h_i{}^{(\infty)} v_j{}^{(\infty)} \right\rangle$$

  - The second correlation is computed using alternating Gibbs sampling until thermal equilibrium



Contrastive divergence is a quick way to train an RBM

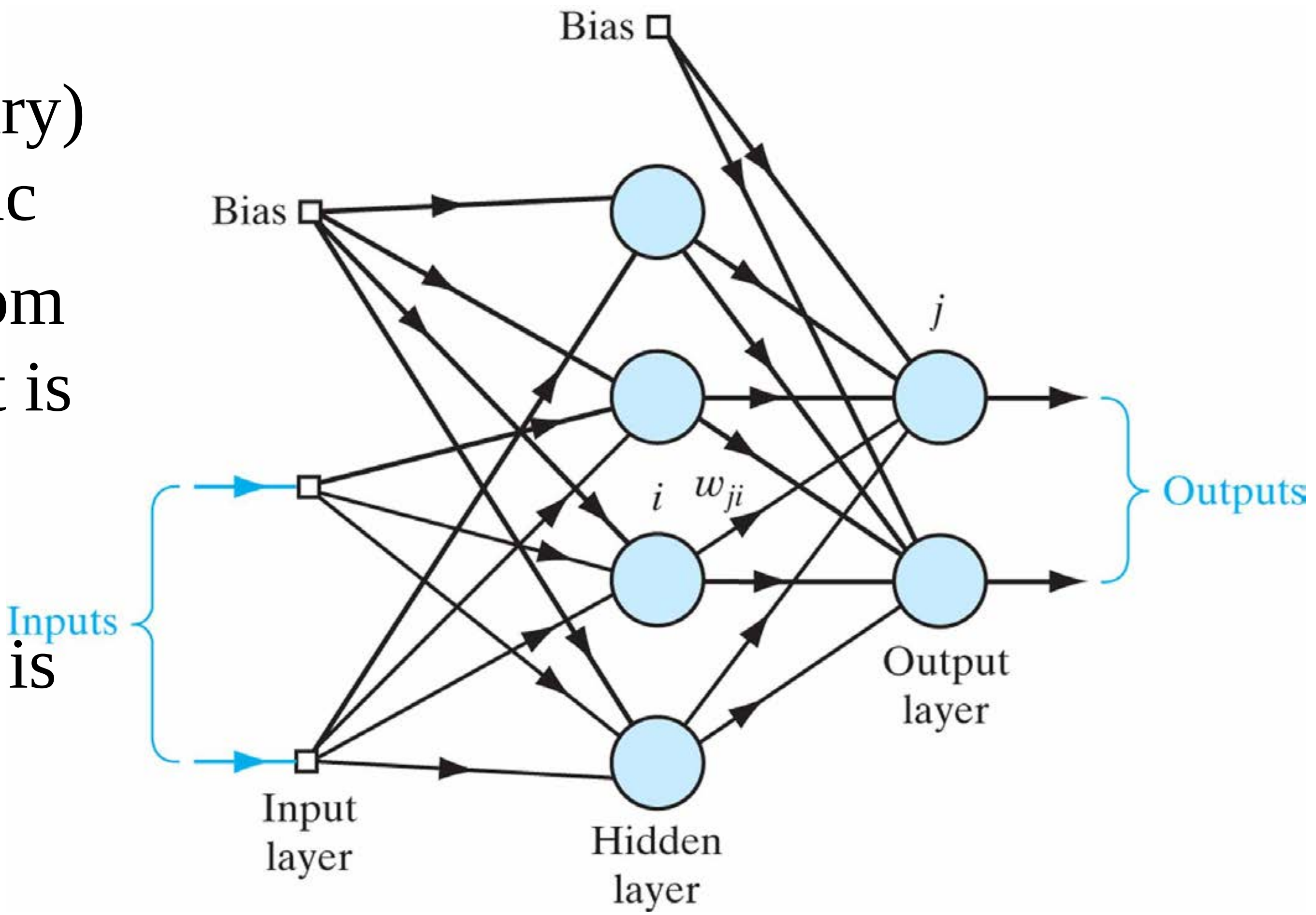
- Contrastive divergence training
  - Start at observed data, sample $\boldsymbol{h}$, then $\boldsymbol{v}$, then $\boldsymbol{h}$

$$\triangle\, w_{ji} = \eta\left(\left\langle {h_i}^{(0)} {v_j}^{(0)} \right\rangle - \left\langle {h_i}^{(1)} {v_j}^{(1)} \right\rangle\right)$$

- First term is exact
- Second term approximates a sample from the unclamped joint distribution
  - Assuming that $p(\boldsymbol{v}, \boldsymbol{h})$ is close to the data distribution
  - Then $(\boldsymbol{v}^{(1)}, \boldsymbol{h}^{(1)})$ is a reasonable sample from $p(\boldsymbol{v}, \boldsymbol{h})$

Logistic belief nets are directed Boltzmann machines

- Each unit is bipolar (binary) and stochastic
- Sampling from the belief net is easy
- Computing probabilities is still hard



Sampling from a logistic belief net

- Given the bipolar states of the units in layer k, we generate the state of each unit in layer $k-1$:

$$P\left({h_j}^{(k-1)} = 1\right) = \varphi\left(\sum_i {w_{ji}}^{(k)} {h_i}^{(k)}\right)$$

where superscript indicates layer number and

$$\varphi(x) = \frac{1}{1 + \exp(-x)}$$

is a logistic activation function

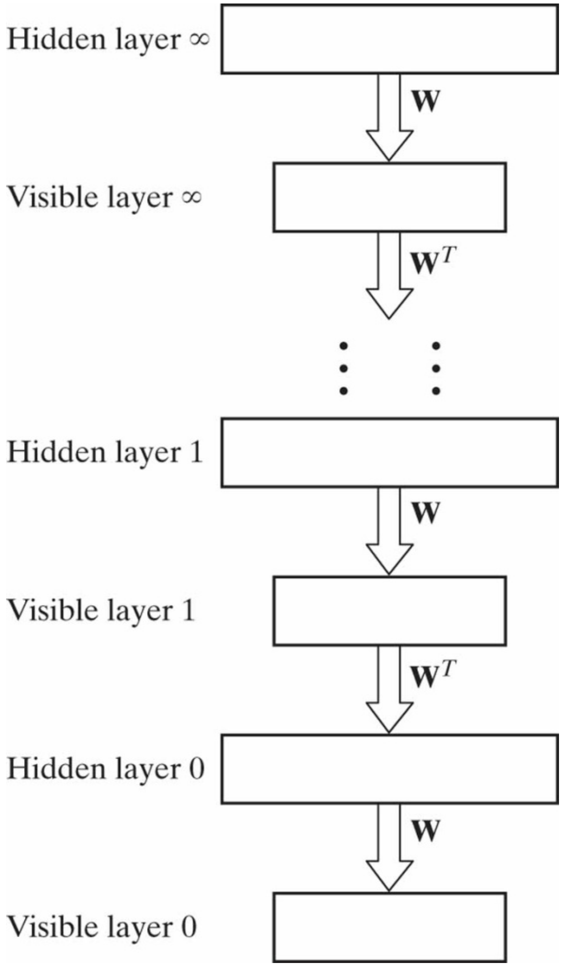
Learning rule

- The bottom layer $\mathbf{h}^{(0)}$ is equal to the visible layer $\mathbf{v}$
- Learning in a belief net maximizes the likelihood of generating the input patterns applied to $\mathbf{v}$, we have
$$\triangle\, w_{ji} = \left\langle {h_i}^{(k)}\left[{h_j}^{(k-1)} - P\left({h_j}^{(k-1)} = 1\right)\right]\right\rangle$$
- The difference term in the above equation includes an evaluation of the posterior probability given the training data
  - Computing posteriors is, unfortunately, very difficult

A special belief net

- However, for a special kind of belief net, computing posteriors is easy
- Consider a logistic belief net with an infinite number of layers and tied weights
  - That is, a deep belief net (DBN)

Sampling from an infinite belief net produces samples from the posterior



Learning in this infinite belief net is now easy

- Because of the tied weights, all but two terms cancel each other out

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \left\langle h_i^{(0)}\left(v_j^{(0)} - v_j^{(1)}\right)\right\rangle + \left\langle v_j^{(1)}\left(h_i^{(0)} - h_i^{(1)}\right)\right\rangle + \left\langle h_i^{(1)}\left(v_j^{(1)} - v_j^{(2)}\right)\right\rangle + \cdots$$

$$= \left\langle h_i^{(0)} v_j^{(0)}\right\rangle - \left\langle h_i^{(\infty)} v_j^{(\infty)}\right\rangle$$

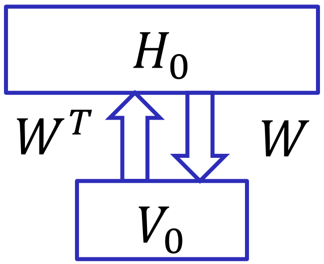
Thus learning in this infinite belief net is equivalent to learning in an RBM

- This rule is exactly the same as the one for the RBM
  - Hence the equivalence between learning an infinite belief net and an RBM
- Infinite belief nets are also known as deep belief nets (DBNs)

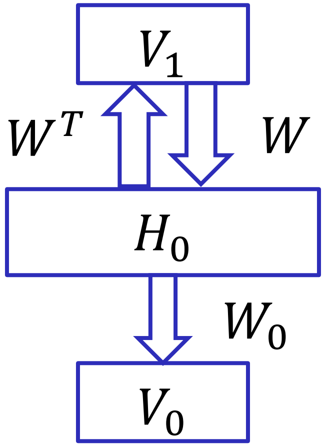
Training a general deep net layer-by-layer

1. First learn W with all weights tied
2. Freeze (fix) W as W^0, which represents the learned weights for the first hidden layer
3. Learn the weights for the second hidden layer by treating responses of the first hidden layer to the training data as “input data”
4. Freeze the weights for the second hidden layer
5. Repeat steps 3-4 as many times as the prescribed number of hidden layers

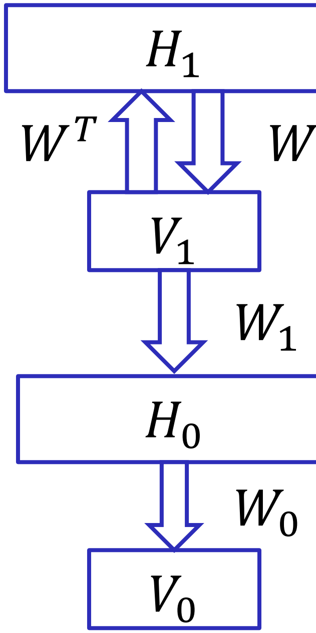
Thus an infinite belief network can be implemented with finite computation



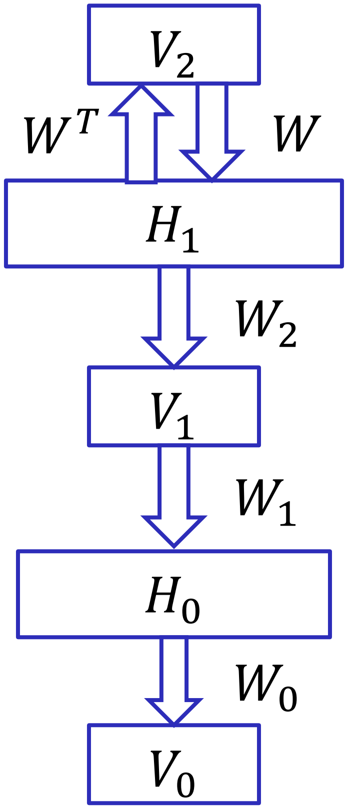
Thus an infinite belief network can be implemented with finite computation



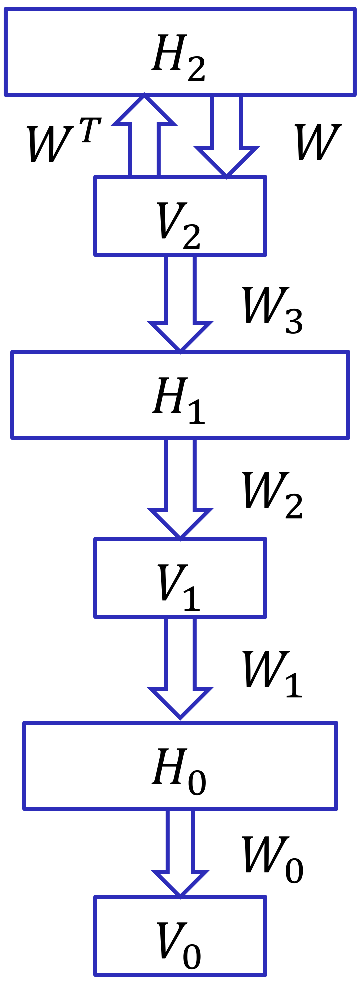
Thus an infinite belief network can be implemented with finite computation



Thus an infinite belief network can be implemented with finite computation



Thus an infinite belief network can be implemented with finite computation



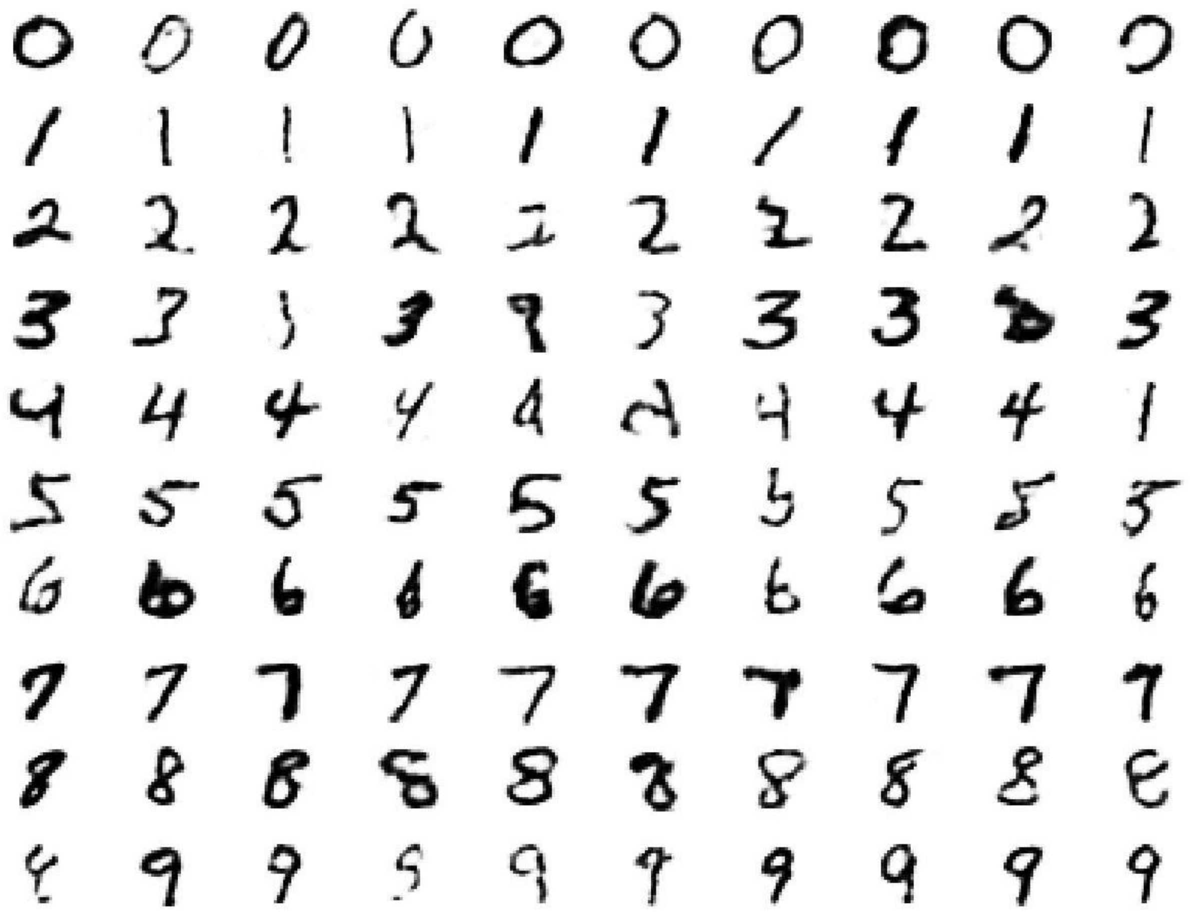
Remarks (Hinton, Osindero, Yeh, 2006)

- As the number of layers increases, the maximum likelihood approximation of the training data improves
- For discriminative training (e.g. for classification) we add an output layer on top of the learned generative model, and train the entire net by a discriminative algorithm
- Although much faster than Boltzmann machines (e.g. no simulated annealing), pretraining is still quite slow, and involves a lot of design as for MLP

DBNs have been successfully applied to an increasing number of tasks

- Ex: MNIST handwritten digit recognition
- A DNN with two hidden layers achieves 1.25% error rate, vs. 1.4% for SVM and 1.5% for MLP
- Great example animations
  - http://www.cs.toronto.edu/~hinton/digits.html

Samples from the learned generative model with one label clamped on



Samples with one label clamped on starting at a randomly initialized image