Lecture 9:
HNN, BM, RBM, and DBM

Topics of this lecture

- Basic idea of generative model
- A brief review of Hopfield neural network (HNN)
- Boltzmann machine (BM)
- Boltzmann machine with hidden units
- Restricted Boltzmann machine (RBM)
- Deep Boltzmann machine (DBM)

Basic idea of generative model

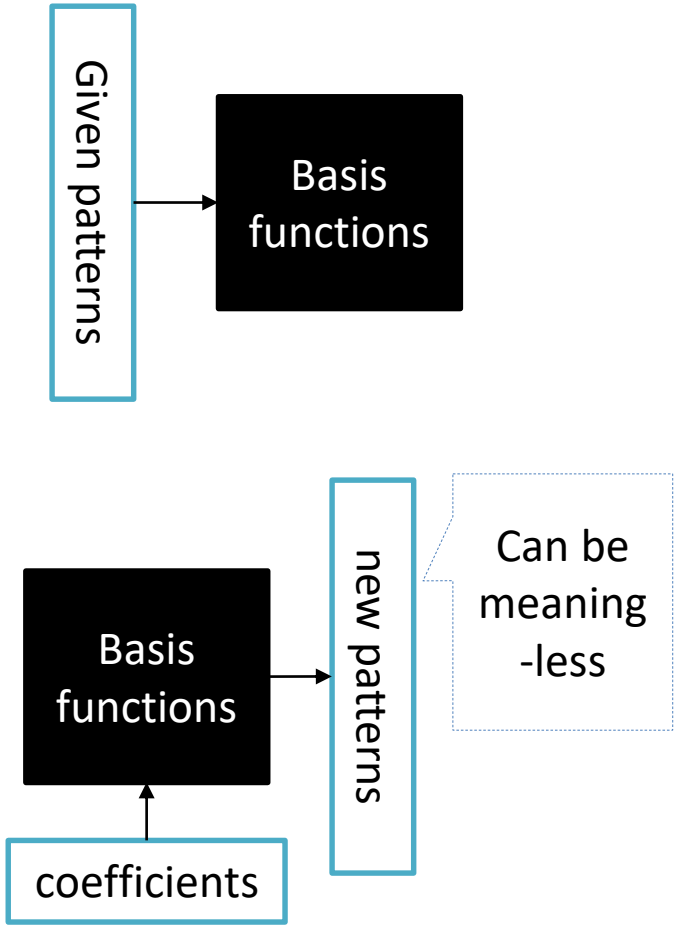
- To solve a pattern classification problem, the most important thing is to extract and select "*discriminative features*" useful for distinguishing patterns in different classes more easily.
- In this sense, *linear discriminant analysis* (LDA) is better than *principal component analysis* (PCA).
- On the other hand, features extracted by PCA are useful for reconstructing the original data more efficiently, although many features so obtained might be common to all data.
- Auto-encoders are similar to PCA, and are useful to extract latent factors that are important for reconstructing the data, rather than for distinguishing the data from different classes.

Basic idea of generative model

- *Representative approaches* like PCA and auto-encoder are useful for extracting *intrinsic features* of patterns in a given class.
- These features can be extracted based ONLY on given data, and do not need teacher signals.
- In this sense, representative approaches can be combined with discriminant approaches, so that better discriminative features can be extracted more efficiently and more effectively.
- For example, the *Fisherface* approach, which is very useful for image recognition, is a combination of PCA and LDA.
- Similarly, corresponding to auto-encoder, we can use viriational autoencoder (VAE) to obtain more discriminant information.
- The main idea of VAE is to make the encoded data more compact in the feature space, if they belong to the target (normal) class. Data belonging to different classes (abnormal) will be mapped to somewhere away from the center of a hyper-ball.

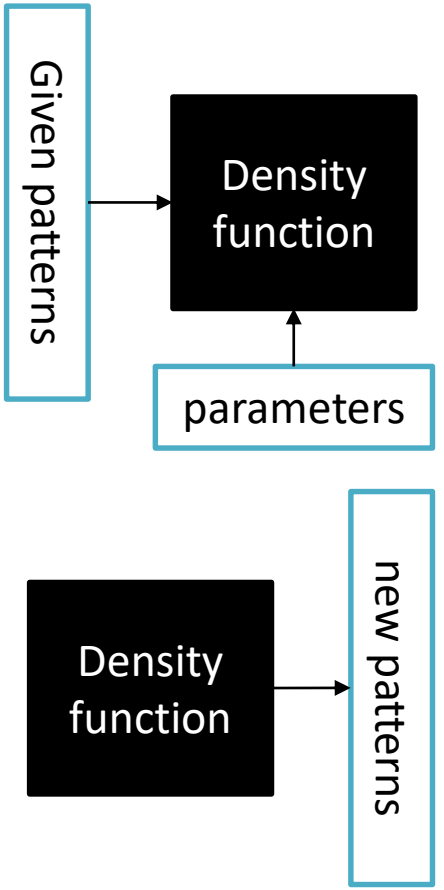
Basic idea of generative model

- The basic idea of representative approaches is to find a set of "*basis functions*" so that given patterns can be reconstructed via linear combinations of these basis functions.
- However, given the basis functions, we cannot "generate" patterns in an interested domain, because we do not know exactly the distribution of the patterns from the basis functions.
- For example, if we generate patterns uniformly based on the basis functions, the patterns may not have any physical meaning.



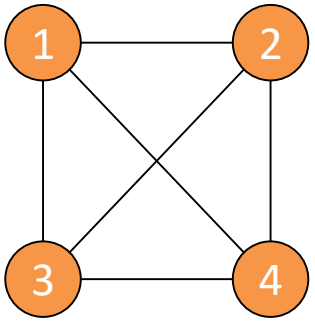
Basic idea of generative model

- The basic idea of *generative approaches* is to find the (joint) *probability density function* (pdf) based on given data, and then generate meaningful patterns based on the pdf.
- This is the most difficult problem. It is much more difficult than discriminant approaches, and more difficult than representative approaches.
- Roughly speaking, LDA, MLP and CNN are discriminant; PCA and auto-encoder are representative; and Bayesian network is generative (special case).



The Hopfield neural network

- Hopfield neural network (HNN) is a model of *auto-associative memory*.
- By auto-association we mean that when a "corrupted" or "broken" pattern is given as the input, the original pattern can be reconstructed by the network.
- The structure of an HNN is shown in the right figure. It is a single layer neural network with feedbacks.



Parameters include connection weights w_{ij} and bias: b_i

- $w_{ij} = w_{ji}$
- $w_{ij} = 0$

The Hopfield neural network

- The output of each neuron is a binary number in {0,1}. All outputs together form a ***state vector.***
- Starting from an initial state vector, the state of the network transits from one to another like an automaton.
- If the state converges, the point to which it converges is called the ***attractor.***
- We can store some interested patterns in an HNN, and recall them later using some observations.
- Precisely speaking, HNN is not a generative model because it can generate memorized patterns only.

The Hopfield neural network

- Suppose that the current state of the network is $v^k = [v_1^k, \ldots, v_n^k]$, the next state is calculated by

$$v_i^{k+1} = sgn(u_i) = sgn\left(\sum_{j \neq i} w_{ij} v_j^k + b_i\right) \quad (1)$$

- where w_{ij} is the weight, and b_i is the bias or threshold.
- Here we assume that
  - State transition is conducted asynchronously;
  - The weight matrix is symmetric (i.e. $w_{ij} = w_{ji}$); and
  - Any neuron does not feedback to itself (i.e. $w_{ii} = 0$)

The Hopfield neural network

- To investigate the behavior of an HNN, we define an energy function as follows:

$$\Phi(\boldsymbol{v}) = -\sum_i b_i v_i - \sum_i \sum_{j \neq i} w_{ij} v_i v_j \quad (2)$$

- The energy function Φ is a function of the state vector $\boldsymbol{v}$.
- For an HNN, its energy function never increases during state transition.
- That is, the value of Φ at the initial state is always larger than that at the attractor.

The Hopfield neural network

- In fact, if we assume that the state transition is asynchronous, at each time point, the state increment takes the following form:

$$\Delta v = (0, \dots, \Delta v_i, \dots 0)^t \quad (3)$$

- and the increment of the energy function is

$$\Delta\Phi = -\left(\sum_{j\neq i} w_{ij} v_j^k + b_i\right)\Delta v_i = -u_i \cdot \Delta v_i \quad (4)$$

- Thus, if we update the neuron output based on Eq. (1), the increment of Φ is always non-positive, or in other word, the energy function always decreases.

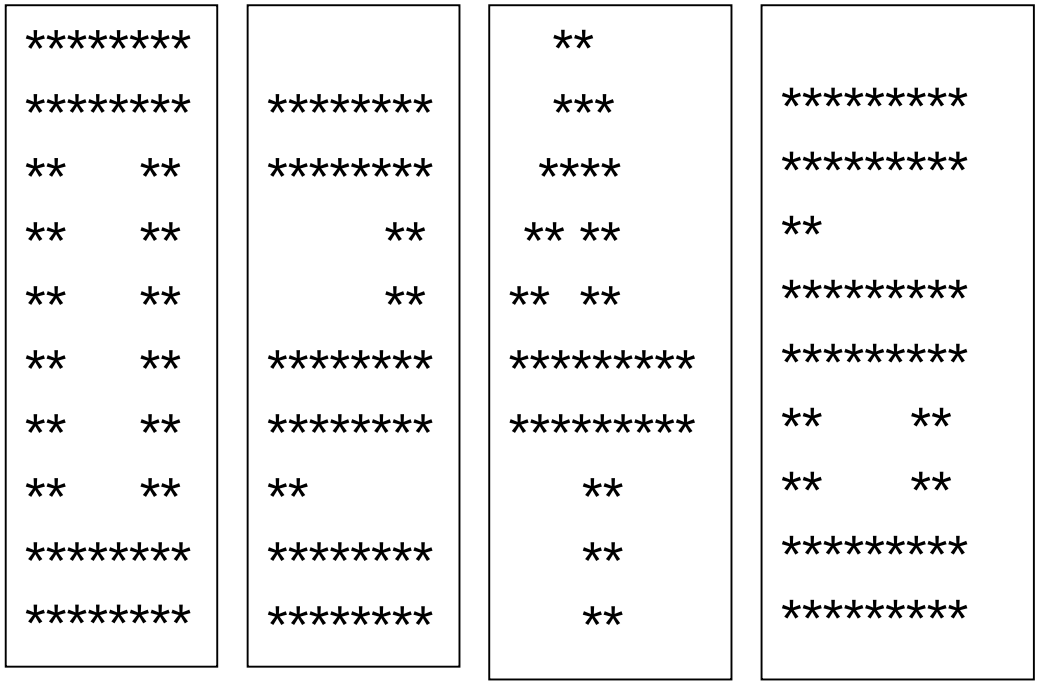
The Hopfield neural network

- Although HNN is not a generative model, it can be used as an associative memory to "re-generate" some memorized pattern if we provide a corrupted version.
- Suppose that we have P patterns to memorize. These patterns can be stored into an HNN simply by defining the weight matrix as follows:

$$W = \sum_{m=1}^{P} s^m (s^m)^t - PI \qquad (5)$$

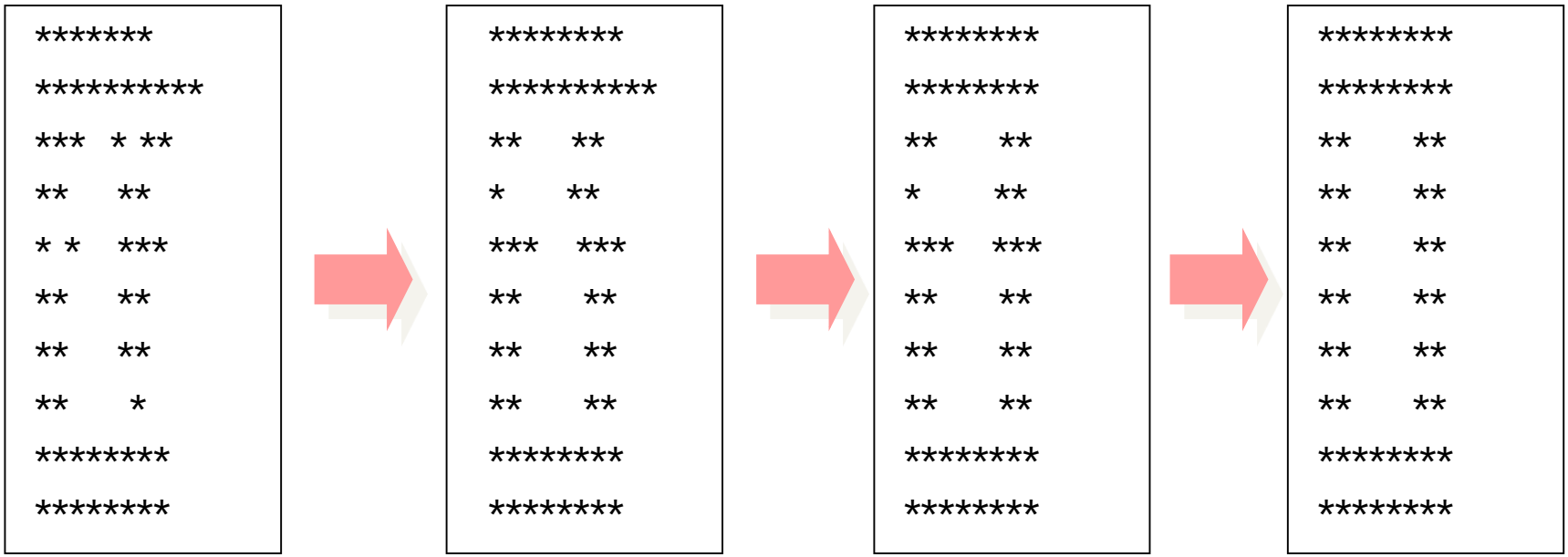
- Where s^m is the m-th pattern, and I is the unit matrix. We also assume that the biases are always 0.

The Hopfield neural network



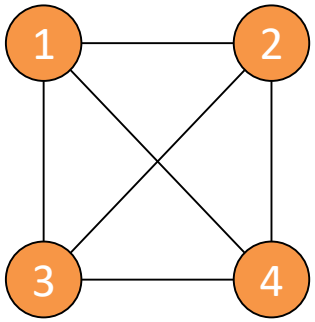
- We have four patterns.
- The purpose is to save them into an HNN, and recall them when they are corrupted by noises.

The Hopfield neural network



Boltzmann machine

- The structure of a BM is the same as an HNN (see right figure).
- The only difference is that the next state is defined statistically based on the current state.
- Again, we can define the energy function using Eq. (2).
- We assume that the state variables follows the Gibbs distribution or Boltzmann distribution.



Boltzmann machine

- That is, the probability density function (pdf) of the state vector is given by

$$p(\boldsymbol{V}=\boldsymbol{v})=\frac{1}{Z}\exp\big(-\Phi(\boldsymbol{v})\big) \qquad (6)$$

- where Z is a normalization constant. Since the pdf depends on the parameter $\boldsymbol{\theta}$ (including the weights and the biases), Eq. (6) should be rewritten as

$$p(\boldsymbol{V}=\boldsymbol{v}|\boldsymbol{\theta})=\frac{1}{Z(\boldsymbol{\theta})}\exp\big(-\Phi(\boldsymbol{v},\boldsymbol{\theta})\big)$$

$$=\frac{1}{Z(\boldsymbol{\theta})}\exp\Big(\sum_i b_i v_i+\sum_i\sum_{j\neq i} w_{ij}v_i v_j\Big) \qquad (7)$$

- Provided that the parameters are properly defined, BM can be used to generate desired patterns. That is, BM is a true generative model.

Boltzmann machine

- Suppose that we have an **independent and identically distributed** (iid) sample containing N patterns

$$\Omega = \{\boldsymbol{v}^1, \boldsymbol{v}^2, \dots, \boldsymbol{v}^N\}$$

- We can find the best parameter set θ based on these patterns by maximizing the following log-likelihood:

$$L_\Omega(\boldsymbol{\theta}) = ln\left(\prod_{k=1}^{N} p(\boldsymbol{V} = \boldsymbol{v}^k|\boldsymbol{\theta})\right) = \sum_{k=1}^{N} \ln(p(\boldsymbol{V} = \boldsymbol{v}^k|\boldsymbol{\theta})) \qquad (8)$$

- We can find the optimal solution by finding the first derivative (the gradient) first, and set it to zero.

Boltzmann machine

- Specifically, we have

$$\frac{\partial L_{\Omega}(\boldsymbol{\theta})}{\partial b_i} = \sum_{k=1}^{N} v_i^k - N \cdot E_{p(\boldsymbol{v}|\boldsymbol{\theta})}[V_i] \qquad (9)$$

$$\frac{\partial L_{\Omega}(\boldsymbol{\theta})}{\partial w_{ij}} = \sum_{k=1}^{N} v_i^k v_j^k - N \cdot E_{p(\boldsymbol{v}|\boldsymbol{\theta})}[V_i V_j] \qquad (10)$$

where $E_{p(\boldsymbol{v}|\boldsymbol{\theta})}[X]$ is the model dependent expectation of X.

- Theoretically, we can find the optimal parameter set $\boldsymbol{\theta}$ by setting the above equations to zero.
- In practice, however, it is extremely difficult to find the model dependent expectations.

Boltzmann machine

- To solve the problem more efficiently, we can use *gradient ascent algorithm*.
- That is, we can update the parameter as follows based on the gradient:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha \nabla L_{\Omega}(\boldsymbol{\theta}) \quad (11)$$

- where α is the learning rate (may not be constant).
- To find the gradient, it is necessary to find the model dependent expectations of V_i and $V_i V_j$.
- A simple way is to use *Gibbs sampling*, which is a *Markov chain Monte Carlo* (MCMC) method.

Boltzmann machine

- The basic steps of Gibbs sampling are as follows:
  - Fix the states of all neurons except the i-th one.
  - Update the state of the i-th neuron based on the current parameters.
- The conditional probability of $V_i = v_i$ with all other neurons fixed is given by

$$p(V_i = v_i | \boldsymbol{V}_{-i} = \boldsymbol{v}_{-i}, \boldsymbol{\theta}) = \frac{p(\boldsymbol{v}|\boldsymbol{\theta})}{\sum_{v_i=0}^{v_i=1} p(\boldsymbol{v}|\boldsymbol{\theta})} \qquad (12)$$

- where $\boldsymbol{V}_{-i}$ (or $\boldsymbol{v}_{-i}$) is a vector of random variables (values) obtained by excluding the i-th element from V (or v).

Boltzmann machine

- Substituting Eq. (7) to Eq. (12), we can get

$$p(V_i = v_i | \boldsymbol{V}_{-i} = \boldsymbol{v}_{-i}, \boldsymbol{\theta}) = p(V_i = v_i | u_i) = \frac{\exp(u_i v_i)}{1+\exp(u_i)} \qquad (13)$$

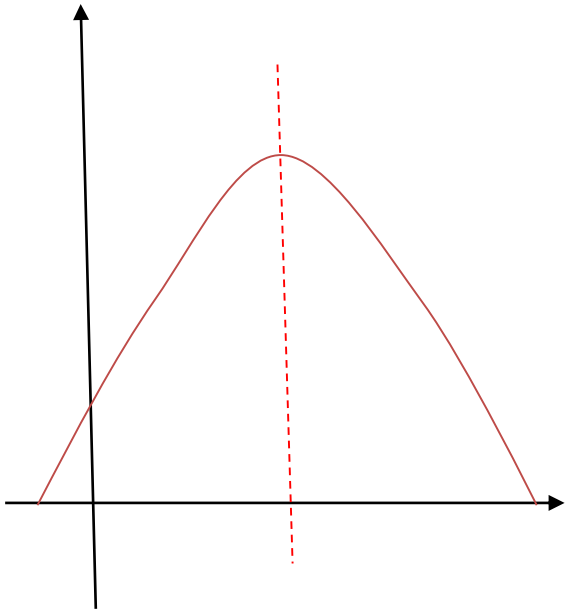
- Note that the neuron output 0 or 1 only. We have the probabilities:

$$p(V_i = 0|u_i) = \frac{1}{1+\exp(u_i)}, \text{ and } p(V_i = 1|u_i) = \frac{\exp(u_i)}{1+\exp(u_i)}$$

- Starting from an initial state vector, we can update the state of each neuron in turn based on the above probabilities.
- Based on the generated patterns, we can find the model dependent expectations of V_i and $V_i V_j$, and update the parameters using Eq. (11).

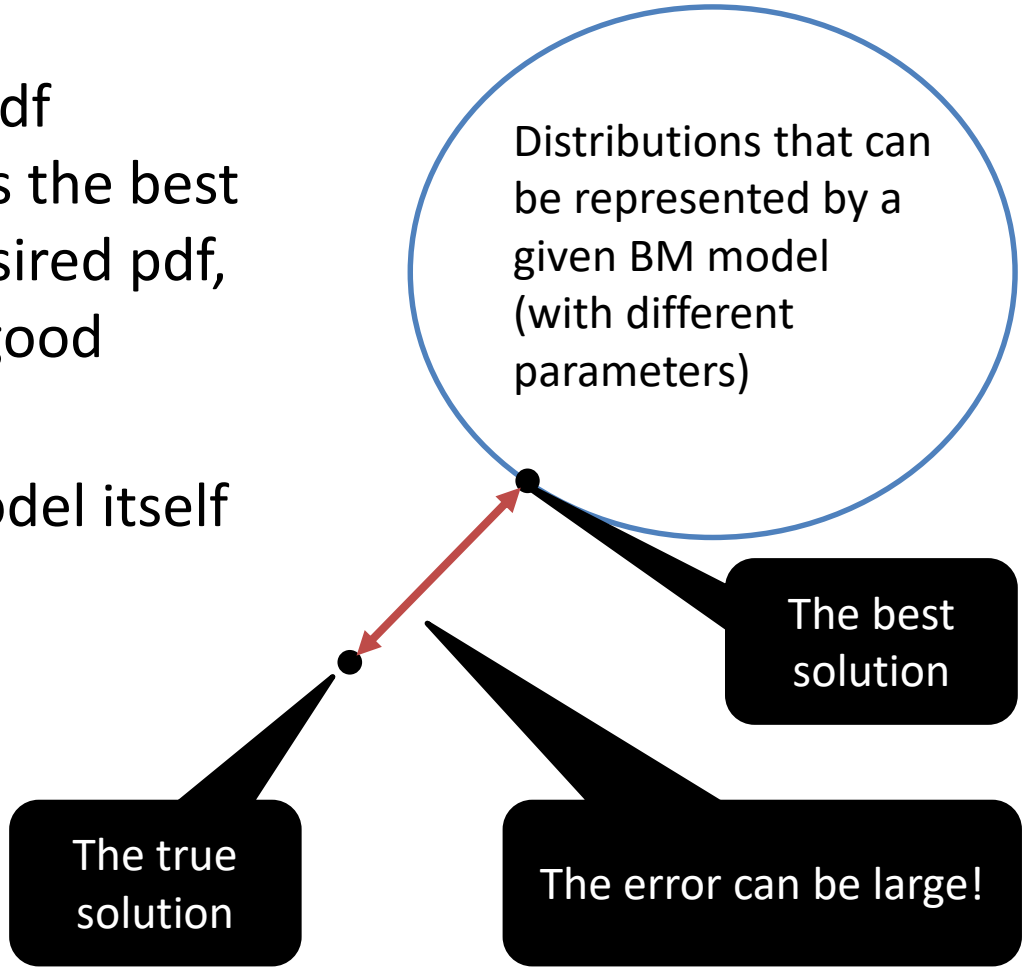
Boltzmann machine

- Since the log likelihood function is concave, the global maximum value can be found by using the gradient ascent algorithm, provided that Gibbs sampling is performed properly.
- Also, it is known that maximizing the likelihood is equivalent to minimizing the *Kullback-Leibler (KL) divergence*. Therefore, the pdf so obtained can approximate the original pdf well, provided that enough data are observed.



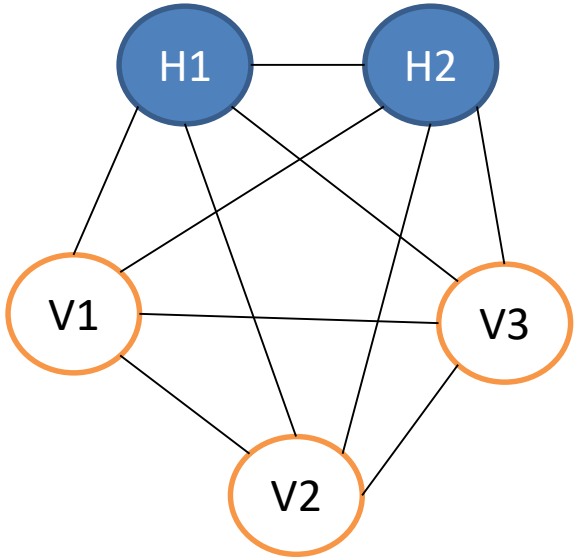
BM with hidden units

- However, even if the pdf represented by a BM is the best approximation of a desired pdf, the BM may not be a good model.
- This happens if the model itself does not have enough *representative ability*.



BM with hidden units

- It is known that adding hidden units in the BM can increase the representative ability. If the number of hidden units is large enough, the BM can approximate any distribution well.
- An example of BM with hidden unit is shown in the right figure.
- The hidden units are not visible, and their states must be estimated using information of the visible units.



BM with hidden units

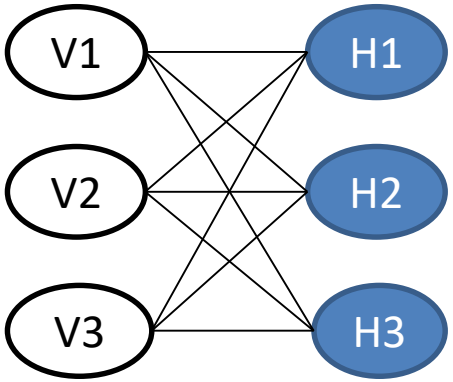
- We can define the state vector as follows:

$$x = (x_1, x_2, \dots, x_n)^t$$
$$= (v_1, \dots, v_{n_v}, h_1, \dots, h_{n_h})^t \qquad (14)$$

- where n_v and n_h are the numbers of visible units and hidden units, respectively, and $n_v + n_h = n$.
- We can find the BM based on given data by maximizing the log likelihood function, using the same method as we used for designing a BM without hidden units.
- However, the process will be more time consuming, and global optimal solution is NOT guaranteed because the likelihood is no longer concave.

Restricted BM

- Instead of using general BM, we can use BM with some restrictions. Proper restrictions can make the BM very efficient.
- The most popular restricted BM (RBM) is the one shown in the right figure.
- In this model, the BM is a *bipartite graph* consisting of two layers, the visible layer and the hidden layer.
- There is no connection between units in the same layer.



- RBMs were initially invented by Paul Smolensky in 1986.
- Geoffrey Hinton proposed a fast learning algorithm in 2006.
- This triggered the boom of deep learning.

Restricted BM

- In an RBM, units in the same layer are conditionally independent. That is, when the state of the visible (hidden) layer is given, the hidden (visible) layer units are independent.
- Using this property, patterns can be generated more efficiently.
- The energy function of an RBM is defined by

$$\Phi(\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{\theta}) = -\sum_{i=1}^{N_v} b_i^v v_i - \sum_{j=1}^{N_h} b_j^h h_j - \sum_{i=1}^{N_v} \sum_{j=1}^{N_h} w_{ij} v_i h_j \qquad (15)$$

Restricted BM

- Given the visible layer, the conditional probability of the hidden layer is given by

$$p(\boldsymbol{h}|\boldsymbol{v},\boldsymbol{\theta}) = \frac{p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta})}{\sum_{\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta})} \qquad (16)$$

- Since $p(\boldsymbol{h}|\boldsymbol{v},\boldsymbol{\theta}) = \left(\frac{1}{Z(\boldsymbol{\theta})}\right)\exp(-\Phi(\boldsymbol{v},\boldsymbol{h},\boldsymbol{\theta}))$ and Φ is defined by (15),

$$p(\boldsymbol{h}|\boldsymbol{v},\boldsymbol{\theta}) = \frac{p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta})}{\sum_{\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta})} = \prod_{j=1}^{N_h} p(h_j|\boldsymbol{v},\boldsymbol{\theta}) \qquad (17)$$

$$p\left(h_j\middle|\boldsymbol{v},\boldsymbol{\theta}\right) = \frac{\exp((b_j^h+\sum_i w_{ij}v_i)h_j)}{1+\exp(b_j^h+\sum_i w_{ij}v_i)} \qquad (18)$$

Restricted BM

- From Eq. (18), we have

$$p\left(h_j = 1 \middle| \boldsymbol{v}, \boldsymbol{\theta}\right) = \frac{\exp(b_j^h + \sum_i w_{ij} v_i)}{1 + \exp(b_j^h + \sum_i w_{ij} v_i)} = g(u_j^h) \qquad (19)$$

- Where $g()$ is the logistic sigmoid function, and u_j^h is the effective input of the j-th hidden unit.
- In a similar way, we have

$$p(v_i = 1 | \boldsymbol{h}, \boldsymbol{\theta}) = \frac{\exp(b_i^v + j \sum w_{ij} h_j)}{1 + \exp(b_i^v + \sum_j w_{ij} h_j)} = g(u_i^v) \qquad (20)$$

- We can generate patterns easily based (19) and (20), starting from any initial value of $\boldsymbol{v}$ or $\boldsymbol{h}$.

Restricted BM

- Remember that the purpose of generating patterns is to find the gradient of the log likelihood function, so that we can update the parameters using the gradient ascent algorithm.
- However, if we use Gibbs sampling method directly, it would be very time consuming, because theoretically we need “enough” long time to generate “enough” number of patterns to approximate the model dependent expectations.
- That is, even if we can generate the patterns more easily if we use RBM, the theoretic “time cost” for learning is still very high.

Restricted BM

- Fortunately, we have a very practically useful algorithm for approximating the gradient.
- This algorithm is called contrastive divergence (CD), and was proposed by Hinton in the mid-2000.
- The basic idea of CD is very simple. Instead of generating "many patterns" to approximate the model dependent expectations, we just generate k (usually $k = 1$ is enough) patterns as follows (using (19) and (20)):
  - $v^0, h^0, v^1, h^1, \ldots, v^k, h^k$
- The initial visible pattern is selected randomly from given training data.

Restricted BM

- We can then update the parameters as follows:
  - $\Delta w_{ij} = \alpha(v_i^0 h_j^0 - v_i^k p_j^k)$ (21)
  - $\Delta b_i^v = \alpha\left(v_i^0 - v_i^k\right)$ (22)
  - $\Delta b_j^h = \alpha(h_j^0 - p_j^k)$ (23)
- Where p_j^k is given by Eq. (19).
- It has been proved that the CD algorithm can produce an RBM very close to the optimal solution, although the solution is biased [1].

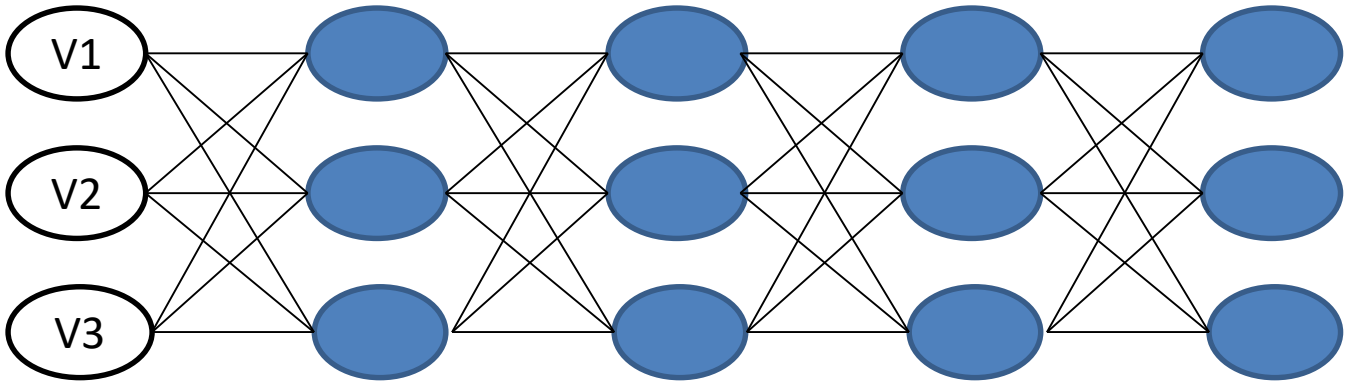
[1] M. A. Carreira-Perpinan and G. E. Hinton, "On contrastive divergence learning". http://www.cs.toronto.edu/~fritz/absps/cdmiguel.pdf

Restricted BM

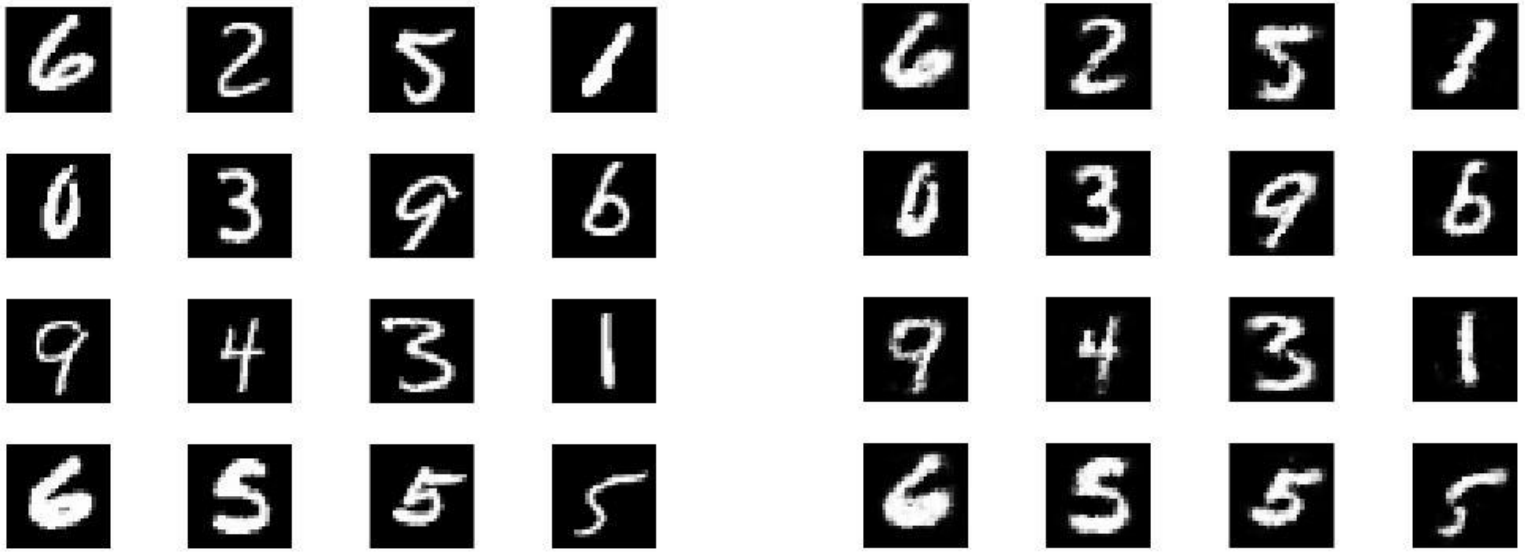
- It is interesting to compare RBM with the auto-encoder.
- Using an auto-encoder, we can find an approximated (usually better) version of the input vector, based on the latent factors obtained through learning.
- On the other hand, using an RBM, we can find various versions of the input vector, based on the probability distribution obtained through learning.
- That is, an RBM can be used to generate many patterns that can be useful for further learning.

Deep Boltzmann machine

- Similar to deep auto-encoder, we can add one more hidden layer to an RBM.
- Once the first hidden layer is learned and fixed, this layer can be considered a visible layer, and the new hidden layer can be learned in the same way.
- Repeat this process, we can obtained deep Boltzmann machine (see the figure given below).
- The main purpose of using more hidden layers is to increase the representative capability of the network.



Deep Boltzmann machine



- Left: Original (training) data
- Right: Reconstructed data using the first hidden layer
- Matlab program downloaded from the following link:

https://www.mathworks.com/matlabcentral/fileexchange/42853-deep-neural-network

Homework of today

- Generate several patterns of the digit “9” using a 5x4 binary matrix (see right), and train a DBM using the same program used in the previous page.
- See what kind of patterns you can reconstruct from the first hidden layer.