Energy-based Models
--Boltzmann Machine

Hao Dong

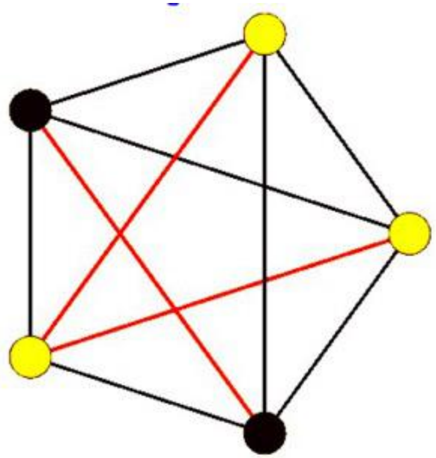
Peking University

Content

- Boltzmann Machine
  - Introduction
  - Training without hidden neurons
  - Training with hidden neurons
  - Summary
- Restricted Boltzmann Machine
- Deep Boltzmann Machine

- Boltzmann Machine
  - Introduction
  - Training without hidden neurons
  - Training with hidden neurons
  - Summary
- Restricted Boltzmann Machine
- Deep Boltzmann Machine

Recap: The Stochastic Hopfield Net



$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ij} y_j + b_i$$

$$P(y_i = 1) = \sigma(z_i)$$

$$P(y_i = -1) = 1 - \sigma(z_i)$$

$$E(Y) = \sum_{i<j} -w_{ij} y_i y_j - b_i y_i$$

$$P(Y) = \frac{\exp(-E(Y))}{\sum_{S'} \exp(-E(Y'))}$$

- The stochastic Hopfield net models a probability distribution over states
  - The state Y is a binary sequence
  - It models a Boltzmann distribution

- The probability that the network will be in any state is P(Y)
  - Generative model: generates states according to P(Y)

Recap: The Stochastic Hopfield Net

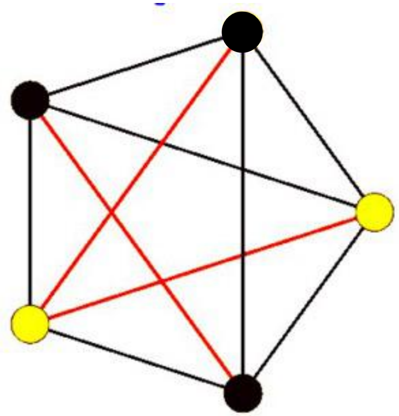
- $P(Y) = P(y_i = 1 \,|y_{j\neq i})\, P(y_{j\neq i})$

- Consider two states Y and Y' with the i-th bit in the +1 and -1

- $\log P(Y) - \log P(Y') = \log P\left(y_i = 1 \middle| y_{j\neq i}\right) - \log P\left(y_i = -1 \middle| y_{j\neq i}\right)$

$$= \log \frac{P\left(y_i = 1 \middle| y_{j\neq i}\right)}{1 - P\left(y_i = 1 \middle| y_{j\neq i}\right)}$$

Recap: The Stochastic Hopfield Net

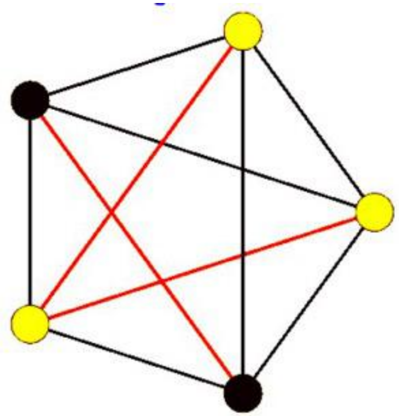


- Consider two states Y and Y' with the i-th bit in the +1 and -1

$$P(Y) = \frac{\exp(-E(Y))}{\sum_{s'} \exp(-E(Y'))}$$

- $\log P(Y) = -E(Y) + C$
- $E(Y) = -\frac{1}{2}(E_{without\ i} + \sum_{j \neq i} w_{ij} y_j + b_i\ )$
- $E(Y') = -\frac{1}{2}(E_{without\ i} - \sum_{j \neq i} w_{ij} y_j - b_i\ )$

- $\log P(Y) - \log P(Y') = E(Y') - E(Y)$
$$= \sum_{j \neq i} w_{ij} y_j + b_i$$

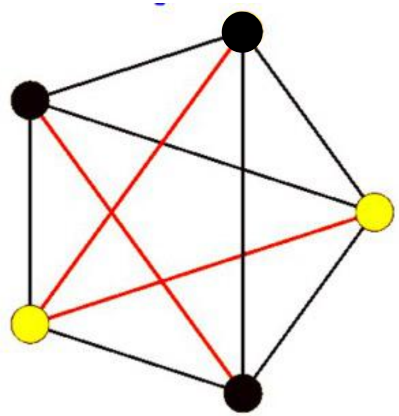


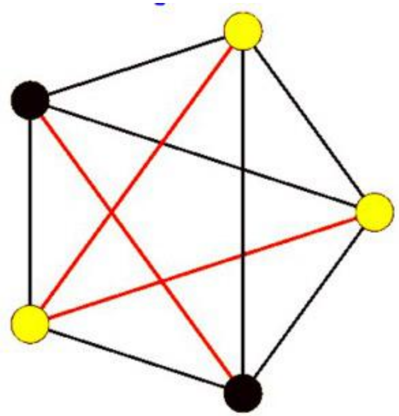
Recap: The Stochastic Hopfield Net

- $\log \frac{P(y_i = 1 | y_{j \neq i})}{1 - P(y_i = 1 | y_{j \neq i})} = \sum_{j \neq i} w_{ij} y_j + b_i$

- We get:
  - $P(y_i = 1 | y_{j \neq i}) = \frac{1}{1 + e^{-(\sum_{j \neq i} w_{ij} s_j + b_i)}}$

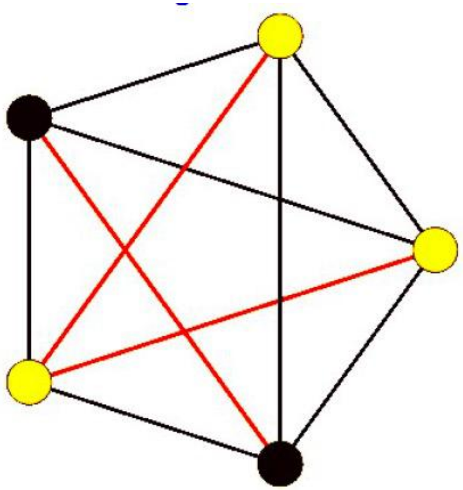
- It's a logistic!





Recap: The Stochastic Hopfield Net

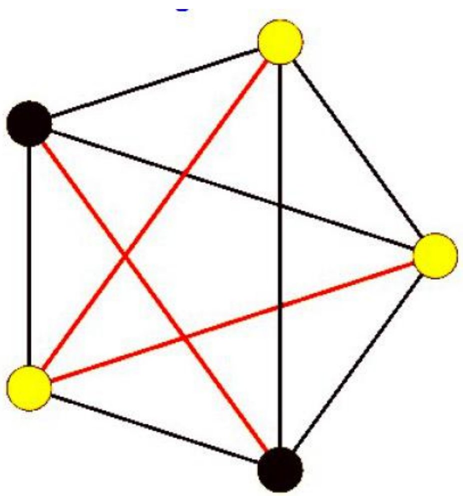
- We can make Hopfield net stochastic
  - Each neuron responds probabilistically
  - More in accord with Thermodynamic models
  - More likely to escape spurious "weak" memories



$$z_i = \frac{1}{T}\sum_{j\neq i} w_{ij}y_j + b_i$$
$$P(y_i = 1) = \sigma(z_i)$$
$$P(y_i = -1) = 1 - \sigma(z_i)$$

Running the network

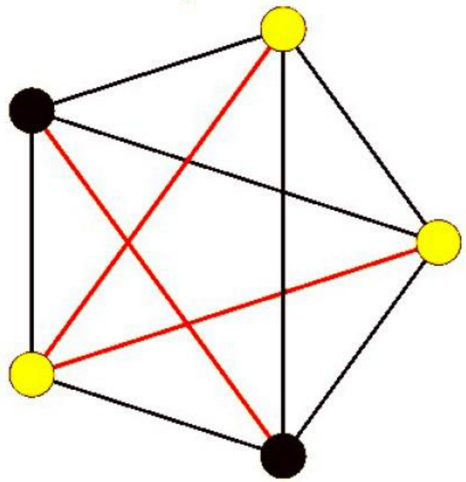
- Initialise the neurons
- Cycle through the neurons and set the neurons to 1/-1 according to the probability
- Until convergence, sample the individual neurons



$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ij} y_j + b_i$$
$$P(y_i = 1) = \sigma(z_i)$$
$$P(y_i = -1) = 1 - \sigma(z_i)$$

The overall probability

- The probability of any state y can be shown to be given by the Boltzmann distribution
  - $E(y) = -\frac{1}{2}y^T W y$
  - $P(\mathrm{y}) = Cexp(-\frac{E(y)}{T})$
- Minimising energy maximises log likelihood
- The parameter of the distribution is the weights matrix W



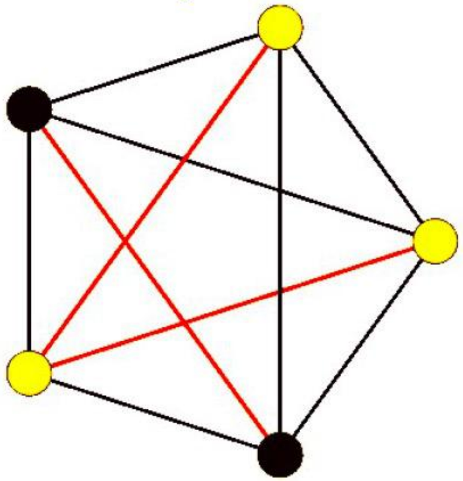
$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ij} y_j + b_i$$

$$P\left(y_i = 1 | y_{j \neq i}\right) = \sigma(z_i)$$

$$P\left(y_i = -1 | y_{j \neq i}\right) = 1 - \sigma(z_i)$$

The overall probability

- The probability of any state y can be shown to be given by the Boltzmann distribution
  - $E(y) = -\frac{1}{2}y^T W y$
  - $P(y) = Cexp(-\frac{E(y)}{T})$
- The conditional distribution of individual bits in the sequence is a logistic
- We call this Boltzmann Machine



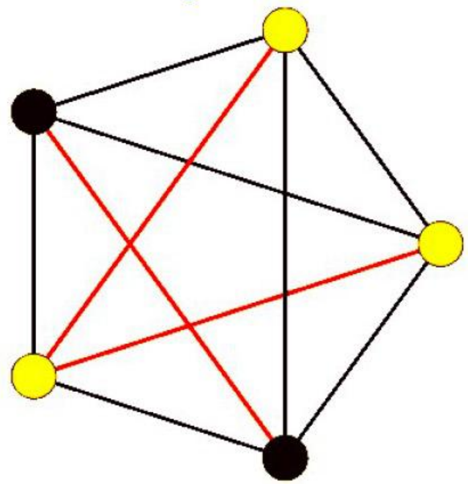
$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ij} y_j + b_i$$

$$P\left(y_i = 1 | y_{j \neq i}\right) = \sigma(z_i)$$

$$P\left(y_i = -1 | y_{j \neq i}\right) = 1 - \sigma(z_i)$$

Boltzmann Machine

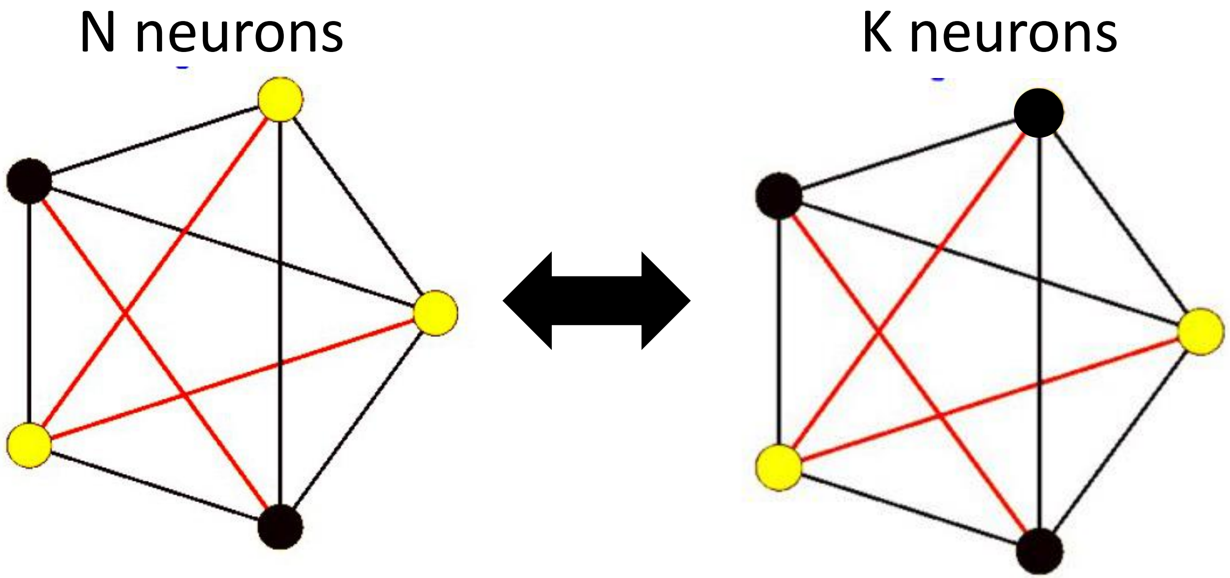
- It can be viewed as a generative model
- Probability of producing any binary vector y:
  - $E(y) = -\frac{1}{2} y^T W y$
  - $P(y) = Cexp(-\frac{E(y)}{T})$



$$z_i = \frac{1}{T} \sum_{j \neq i} w_{ij} y_j + b_i$$
$$P\left(y_i = 1 | y_{j \neq i}\right) = \sigma(z_i)$$
$$P\left(y_i = -1 | y_{j \neq i}\right) = 1 - \sigma(z_i)$$

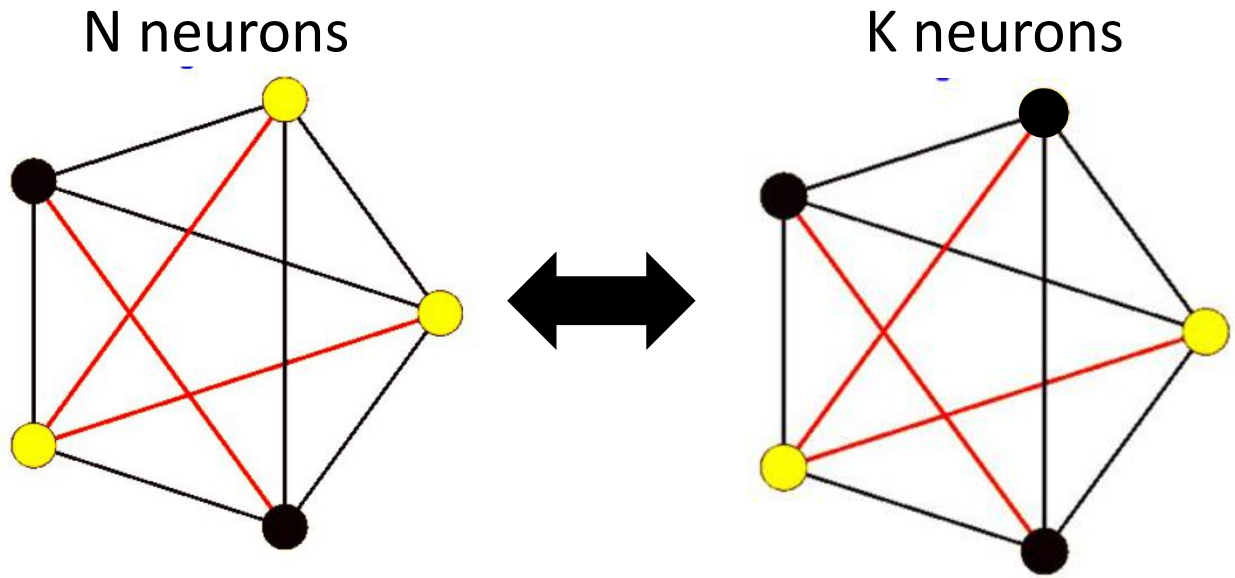
The capacity of Boltzmann Machine

- The network can store up to N N-bit patterns
- How to increase the capacity?

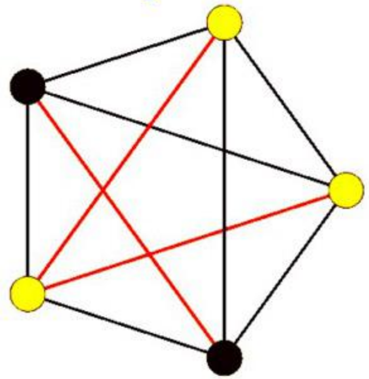


The capacity of Boltzmann Machine

- Add some nodes
  - We don't care the value of these nodes
  - Only serve to increase the capacity
  - Termed Hidden Neurons
- The neurons whose values are important: Visible Neurons



Hopfield Net v.s. Boltzmann machine



$$E(Y) = \sum_{i<j} -w_{ij}y_iy_j - b_iy_i$$

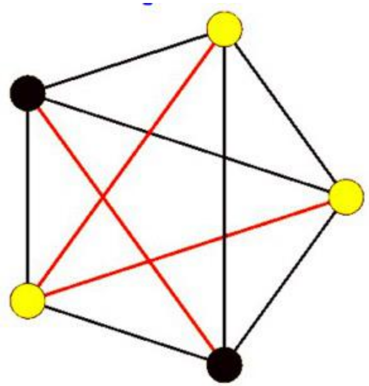
$$P(Y) = \frac{\exp\big(-E(Y)\big)}{\sum_{S'} \exp(-E(Y'))}$$

- Hopfield net
  - Learn weights to “remember” target states and “dislike” other states
  - State: binary pattern of all the neurons
- Boltzmann machine
  - Learn weights to assign more probability to patterns we “like” and less to other patterns

Content

- Boltzmann Machine
  - Introduction
  - Training without hidden neurons
  - Training with hidden neurons
  - Summary
- Restricted Boltzmann Machine
- Deep Boltzmann Machine

Training the network



$$E(Y) = \sum_{i<j} -w_{ij}y_iy_j - b_iy_i$$

$$P(Y) = \frac{\exp\left(-E(Y)\right)}{\sum_{S'} \exp(-E(Y'))}$$

- First we consider the setting without hidden neurons
- Boltzmann machine
  - Given a set of training inputs $Y_1, Y_2, \dots, Y_N$
  - Assign higher probability to patterns seen more frequently
  - Assign lower probability to patterns that are not seen at all

Maximum likelihood training

$$\log\big(P(Y)\big) = \left(\sum_{i<j} w_{ij}y_iy_j\right) - \log\left(\sum_{Y'} \exp\left(\sum_{i<j} w_{ij}y_i'y_j'\right)\right)$$

$$\mathcal{L} = \frac{1}{N}\sum_{Y\in S} \log(P(Y))$$
$$= \frac{1}{N}\textstyle\sum_Y\big(\sum_{i<j} w_{ij}y_iy_j\big) - \log\big(\sum_{Y'} \exp\big(\sum_{i<j} w_{ij}y_i'y_j'\big)\big)$$

- The loss function is average log likelihood of training vectors $S = \{Y_1, Y_2, \dots, Y_N\}$
  - should be maximised
  - In the first summation, y_i and y_j are bits of Y
  - In the second summation, , y_i' and y_j' are bits of Y’ (vectors outside S)

Maximum likelihood training

$$\mathcal{L} = \frac{1}{N}\sum_{Y}\left(\sum_{i<j} w_{ij}y_i y_j\right) - \log\left(\sum_{Y'}\exp\left(\sum_{i<j} w_{ij}y'_i y'_j\right)\right)$$

$$\frac{d\mathcal{L}}{dw_{ij}} = \frac{1}{N}\sum_{Y} y_i y_j \ -?$$

- Use gradient ascent

- The first term is easy to calculate
  - The average $y_i y_j$ over all training vectors

- But the second term is the sum of almost all states
  - exponential number!

PEKING UNIVERSITY 北京大學

The second term

$$\frac{d \log\left(\sum_{Y'} \exp\left(\sum_{i<j} w_{ij} y_i' y_j'\right)\right)}{d w_{ij}} = \sum_{Y'} \frac{\exp\left(\sum_{i<j} w_{ij} y_i' y_j'\right)}{\sum_{Y''} \exp\left(\sum_{i<j} w_{ij} y_i'' y_j''\right)} y_i' y_j'$$

$$= \sum_{Y'} P(Y') y_i' y_j'$$

- The second term is the expected value of y_i', y_j' over all possible values of the state
- We cannot compute it exhaustively, then how?
- Sampling!

The second term

$$\frac{d \log\left(\sum_{Y'} \exp\left(\sum_{i<j} w_{ij} y'_i y'_j\right)\right)}{d w_{ij}} = \sum_{Y'} P(Y') y'_i y'_j$$

$$= \frac{1}{M} \sum_{Y' \in Y_{sample}} y'_i y'_j$$

- The expectation can be estimated as the average of samples drawn from the distribution

- How to sample?

Gibbs Sampling

- A special Metropolis-Hastings algorithm
- Use the conditional distribution

- Suppose $y_1, y_2, \ldots, y_n$:
  - Randomly set values to them
  - Update y_i based on $P(y_i | y_{j \neq i})$
  - Get a Markov Chain
  - Skip the first several samples and sample at intervals

- The samples are approximately close to the joint distribution

Maximum Likelihood Training

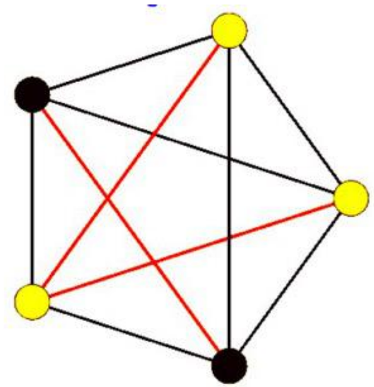
PEKING UNIVERSITY 北京大學

$$\frac{d\mathcal{L}}{dw_{ij}} = \frac{1}{N}\sum_{Y} y_i y_j - \frac{1}{M}\sum_{Y' \in Y_{sample}} y_i' y_j'$$

$$w_{ij} = w_{ij} + \alpha \frac{d\mathcal{L}}{dw_{ij}}$$

- The overall gradient ascent rule

Training Process

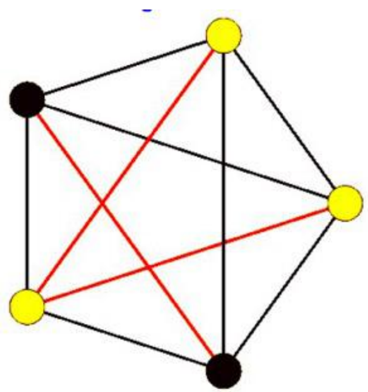


$$\frac{d\mathcal{L}}{dw_{ij}} = \frac{1}{N}\sum_{Y} y_i y_j - \frac{1}{M}\sum_{Y' \in Y_{sample}} y_i' y_j'$$

$$w_{ij} = w_{ij} + \alpha \frac{d\mathcal{L}}{dw_{ij}}$$

- Initialise weights
- Obtain “state samples”
- Compute gradient and update weights
- Iterate until convergence

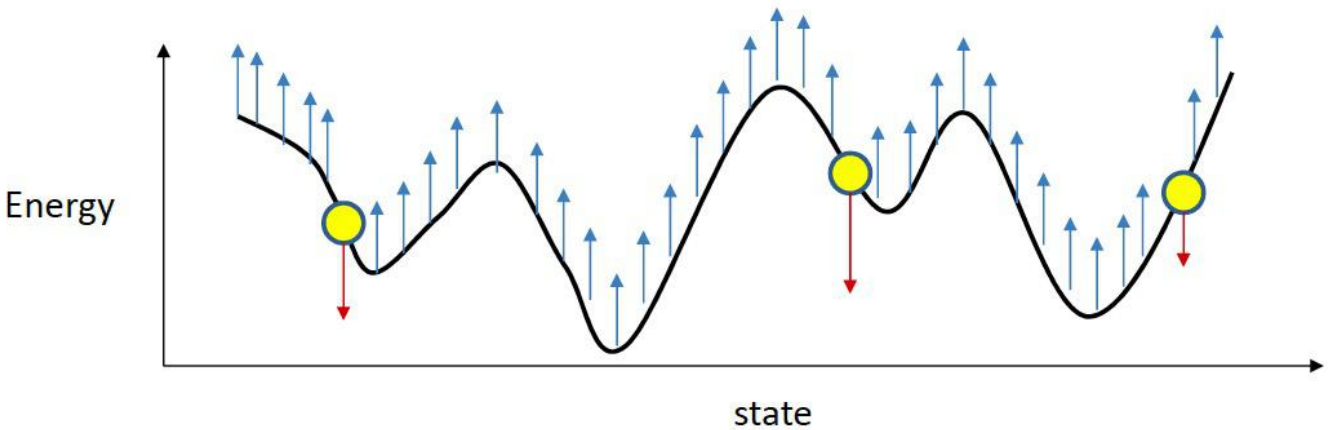
Training Process



$$\frac{d\mathcal{L}}{dw_{ij}} = \frac{1}{N}\sum_{Y} y_i y_j - \frac{1}{M}\sum_{Y' \in Y_{sample}} y_i' y_j'$$

$$w_{ij} = w_{ij} + \alpha \frac{d\mathcal{L}}{dw_{ij}}$$

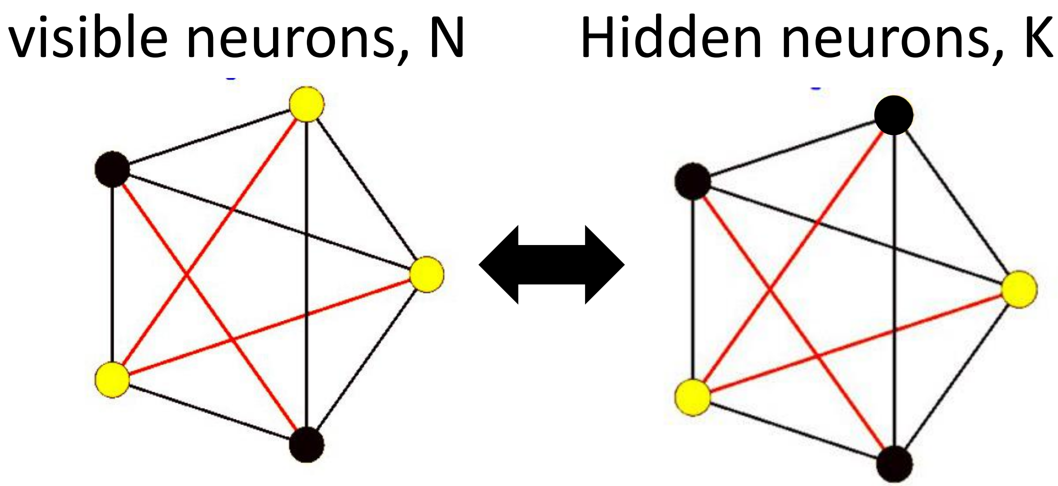
- Similar to the update rule for Hopfield network



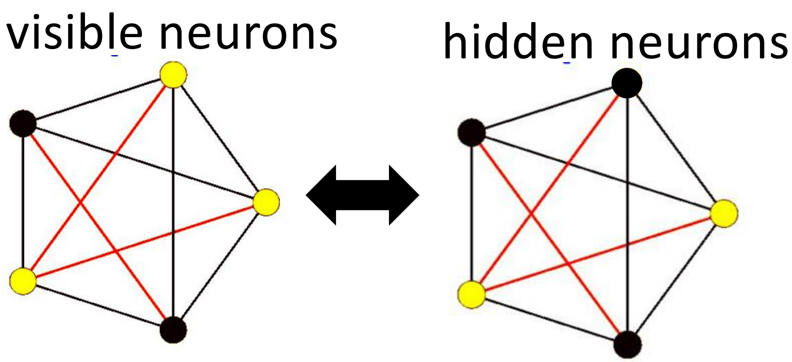
Content

Training with hidden neurons

- For a given pattern of visible neurons, there are many hidden patterns (2^K)
- We want to choose the one with lowest energy
  - But exponential search space is exponential!



Training the network



$$P(Y) = \frac{\exp\left(-E(Y)\right)}{\sum_{S'} \exp(-E(Y'))}$$

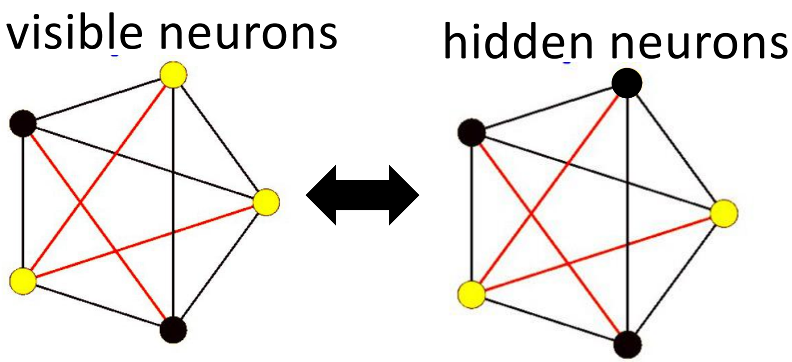
$$P(Y) = P(V, H)$$

$$P(V) = \sum_{H} P(Y)$$

- Y=(V, H)
  - V: output of the visible neurons
  - H: output of the hidden neurons

- The marginal probabilities over visible bits are interested
- The hidden bits are the latent representation learned by the network

Training the network



$$P(Y) = \frac{\exp\big(-E(Y)\big)}{\sum_{S'} \exp(-E(Y'))}$$

$$P(Y) = P(V, H)$$

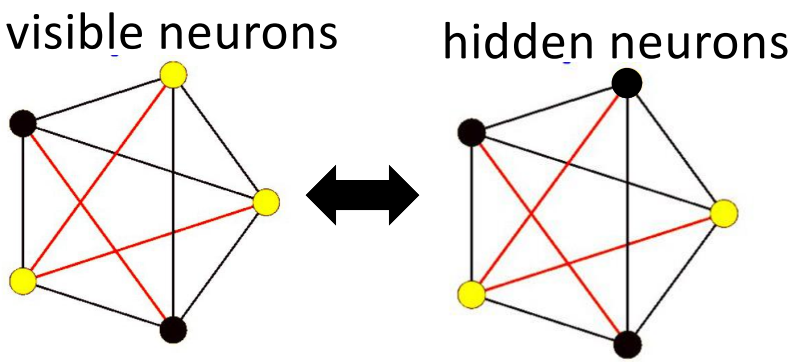
$$P(V) = \sum_{H} P(Y)$$

Maximise this term for training patterns

- Y=(V, H)
  - V: output of the visible neurons
  - H: output of the hidden neurons

- The marginal probabilities over visible bits are interested
- The hidden bits are the latent representation learned by the network

Training the network



$$E(Y) = \sum_{i<j} -w_{ij}y_iy_j - b_iy_i$$

$$P(Y) = \frac{\exp\big(-E(Y)\big)}{\sum_{S'} \exp(-E(Y'))}$$

$$P(V) = \sum_{H} \frac{\exp\big(-E(Y)\big)}{\sum_{S'} \exp(-E(Y'))}$$

- Train the network to assign a desired probability distribution to the visible states
- Probability of visible state sums over all hidden states

Maximum likelihood training

$$\log\big(P(V)\big) = \log\left(\sum_{H} \exp\left(\sum_{i<j} w_{ij} y_i y_j\right)\right) - \log\left(\sum_{Y'} \exp\left(\sum_{i<j} w_{ij} y_i' y_j'\right)\right)$$

$$\mathcal{L} = \frac{1}{N} \sum_{V \in \{V\}} \log(P(V))$$

$$= \frac{1}{N} \sum\nolimits_{V \in \{V\}} \log\big(\sum\nolimits_{H} \exp\big(\sum\nolimits_{i<j} w_{ij} y_i y_j\big)\big) - \log\big(\sum\nolimits_{Y'} \exp\big(\sum\nolimits_{i<j} w_{ij} y_i' y_j'\big)\big)$$

- The loss function is average log likelihood of visible neurons of training vectors {V}= $\{V_1, V_2, \ldots, V_N\}$
  - should be maximised
  - Two terms have the same format

Maximum likelihood training

PEKING UNIVERSITY 北京大學

$$\mathcal{L} = \frac{1}{N} \sum_{V \in \{V\}} \log\left(\sum_{H} \exp\left(\sum_{i<j} w_{ij} y_i y_j\right)\right) - \log\left(\sum_{Y'} \exp\left(\sum_{i<j} w_{ij} y'_i y'_j\right)\right)$$

$$\frac{d\mathcal{L}}{dw_{ij}} = \frac{1}{N} \sum_{V \in \{V\}} \sum_{H} P(Y|V) y_i y_j \; - \sum_{Y'} P(Y') y'_i y'_j$$

- Similar as the setting without hidden neurons

- But both terms are summations over an exponential states
  - Both need sampling

Maximum likelihood training

$$\frac{d\mathcal{L}}{dw_{ij}} = \frac{1}{N}\sum_{V\in\{V\}}\sum_{H} P(Y|V)y_i y_j - \sum_{Y'} P(Y')y'_i y'_j$$

$$\sum_{H} P(Y|V)y_i y_j = \frac{1}{K}\sum_{H\in H_{samples}} y_i y_j$$

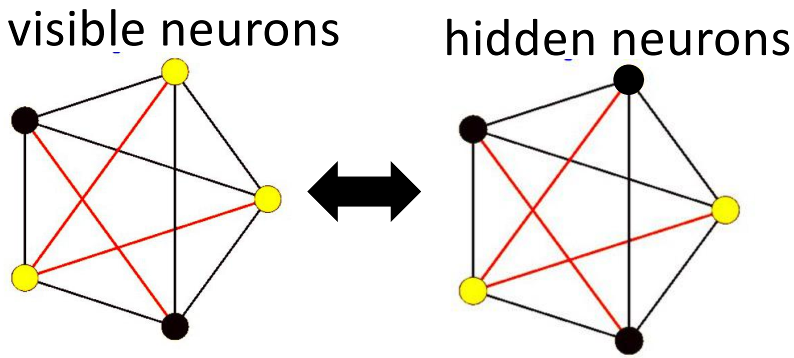
$$\sum_{Y'} P(Y')y'_i y'_j = \frac{1}{M}\sum_{Y'\in S_{samples}} y'_i y'_j$$

- The first term is calculated as the average of sampled hidden state with the visible state fixed

- The second term is calculated as the average of sampled states “freely”

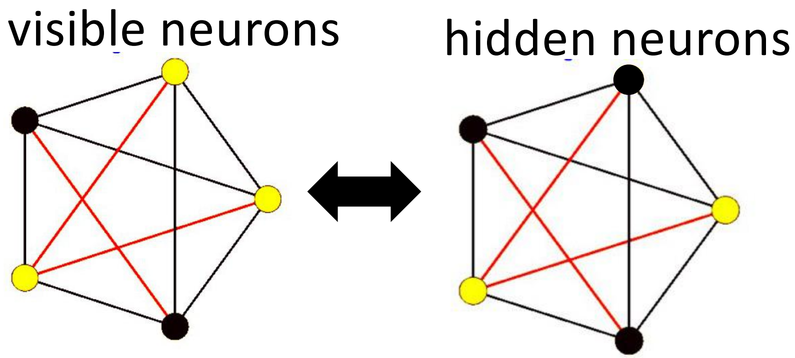
Training Process– Sample 1

- For each training pattern V_i:
  - Fix visible neurons according to V_i
  - Let the hidden neurons evolve from a random initial point to generate H_i
  - Get $Y_i = [V_i, H_i]$
- Repeat K times to generate synthetic training
  - $Y = \{Y_{1,1}, Y_{1,2}, \dots, Y_{1,K}, Y_{2,1}, \dots, Y_{N,K}\}$



Training Process – Sample 2

- Unclamp the visible units and let the entire network evolve several times to generate
  - $Y_{samples} = \{Y_{sample,1}, Y_{sample,2}, \dots, Y_{sample,M}\}$

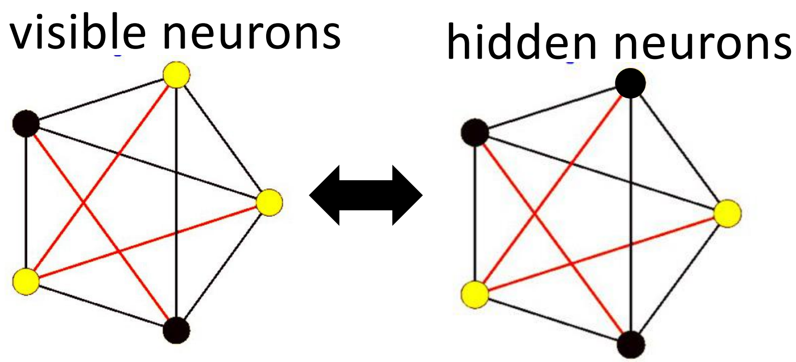


Training Process

$$\frac{d\mathcal{L}}{dw_{ij}} = \frac{1}{NK}\sum_{Y} y_i y_j - \frac{1}{M}\sum_{Y' \in Y_{sample}} y_i' y_j'$$

$$w_{ij} = w_{ij} + \alpha \frac{d\mathcal{L}}{dw_{ij}}$$

- Initialise weights
- Get training samples
- Compute gradient and update weights
- Iterate until convergence



Content

- Boltzmann Machine
  - Introduction
  - Training without hidden neurons
  - Training with hidden neurons
  - Summary
- Restricted Boltzmann Machine
- Deep Boltzmann Machine

Boltzmann Machine

- Stochastic extension of Hopfield network
- Store more patterns than Hopfield network through hidden neurons

- Application:
  - Pattern completion
  - Pattern denoising
  - Computing conditional probabilities of patterns
  - Classification
    - Add more bits representing class
    - $[y_1, \ldots, y_N, class]$

Boltzmann Machine

- Training process takes a long time…
- Can’t work for large problems
- How to solve these problems?

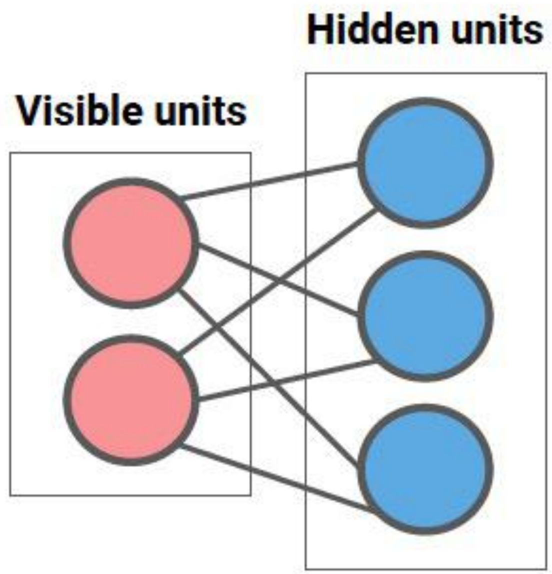
Content

- Boltzmann Machine
  - Introduction
  - Training without hidden neurons
  - Training with hidden neurons
  - Summary
- Restricted Boltzmann Machine
- Deep Boltzmann Machine

Restricted Boltzmann machine (RBM)

- Restricted
  - There are no visible-visible and hidden-hidden connections.
  - Proposed as “Harmonium Models” by Paul Smolensky

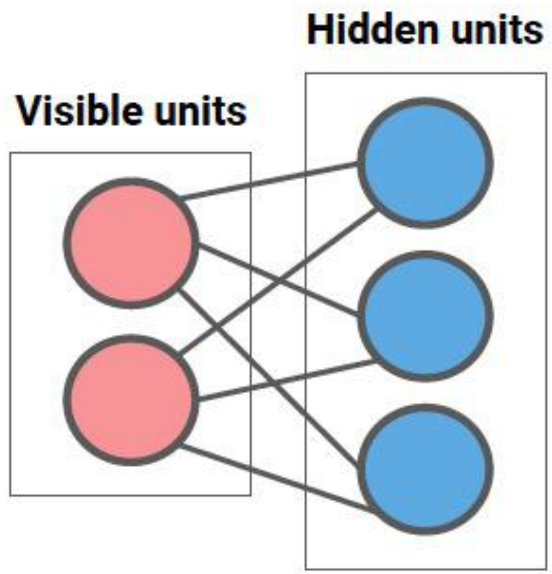
- Joint Distribution:
  - $P(V,H) = \frac{\exp(V^T WH + bV + cH)}{\sum_{v'h'} \exp(V'^T WH' + bV' + cH')}$



Restricted Boltzmann machine (RBM)

Hidden: $z_i = \sum_j w_{ji} v_i + b_i$ $\qquad P(h_i = 1) = \frac{1}{1+e^{-z_i}}$

Visible: $y_i = \sum_j w_{ji} h_i + b_i$ $\qquad P(v_i = 1) = \frac{1}{1+e^{-y_i}}$

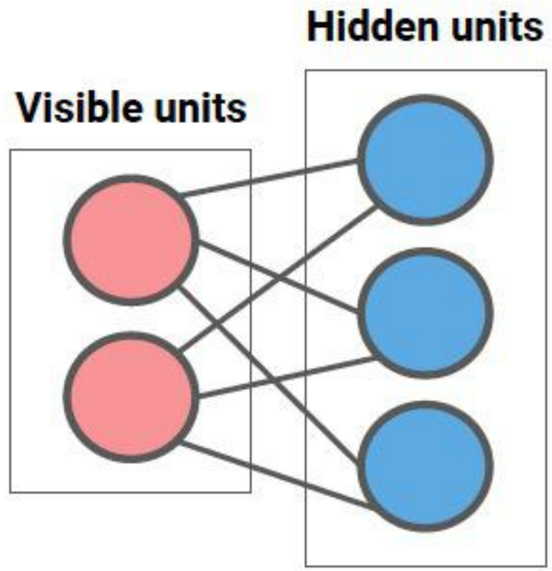


- Pros:
  - Sample for hidden neurons: no looping
  - Sample for all neurons: bigraph

Restricted Boltzmann machine (RBM)

Hidden: $z_i = \sum_j w_{ji} v_i + b_i$ $\quad P(h_i = 1) = \frac{1}{1+e^{-z_i}}$

Visible: $y_i = \sum_j w_{ji} h_i + b_i$ $\quad P(v_i = 1) = \frac{1}{1+e^{-y_i}}$

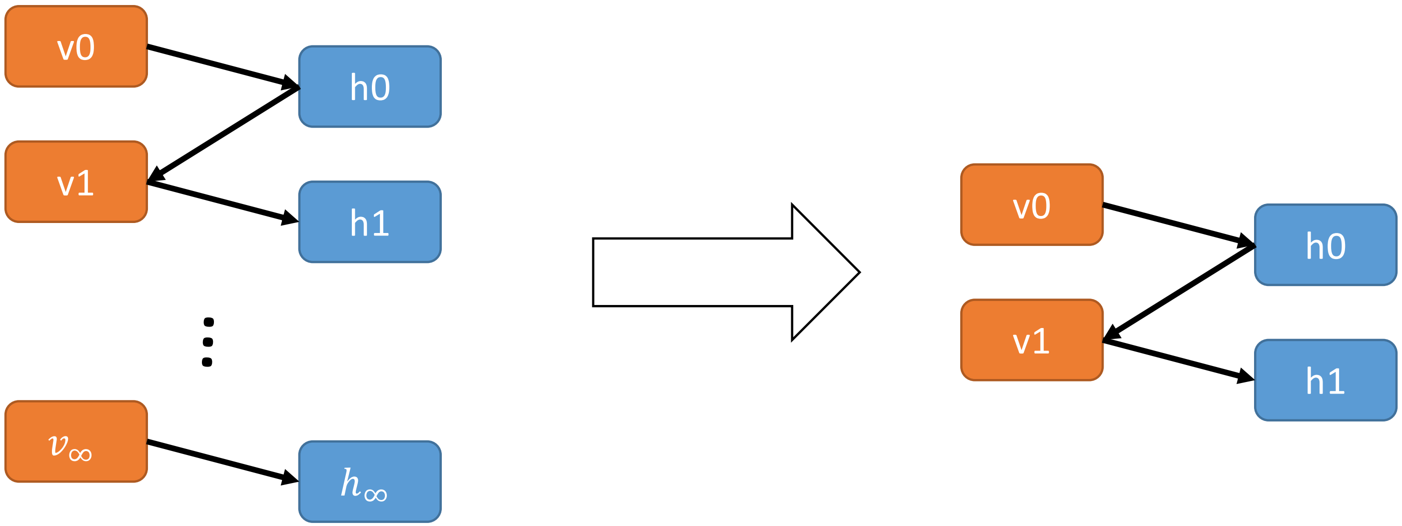


- For each sample:
  - Initialize visible neurons
  - Iteratively generate hidden and visible units
  - $\frac{d \log p}{d w_{ij}} = < v, h >^0 - < v, h >^\infty$

Contrastive Divergence

- Recall in Hopfield Network:
  - No need to raise the entire surface, just the neighborhood

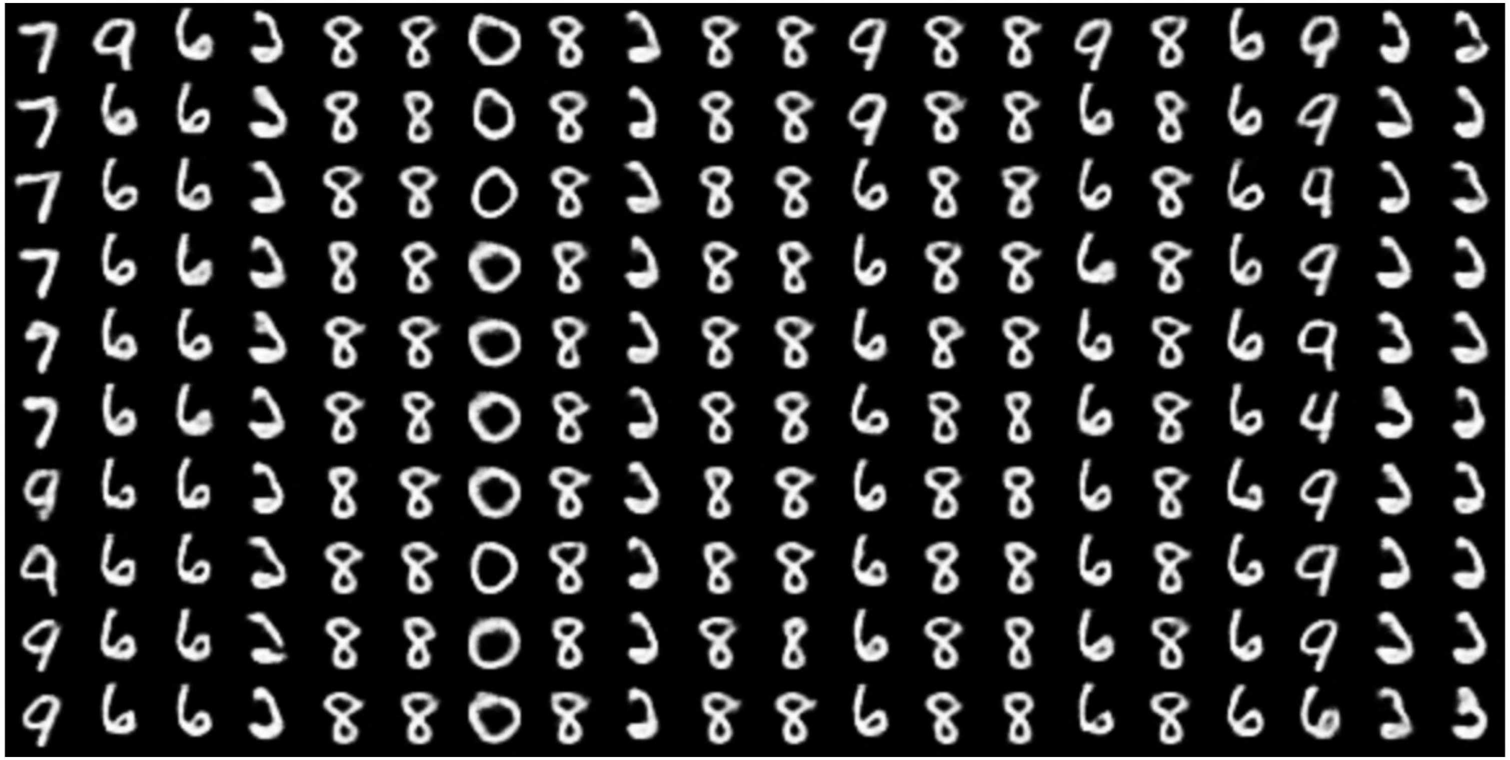
- One iteration is enough in RBM
  - $\frac{d \log p}{dw_{ij}} = < v, h >^0 - < v, h >^1$



Restricted Boltzmann machine (RBM)

- Generative models for binary data
- Can be extended to continuous-valued data
  - Change the distribution of visible neurons (or hidden neurons)
  - "Exponential Family Harmoniums with an Application to Information Retrieval", Welling et al., 2004
- Useful for classification and regression

Boltzmann Machines: samples



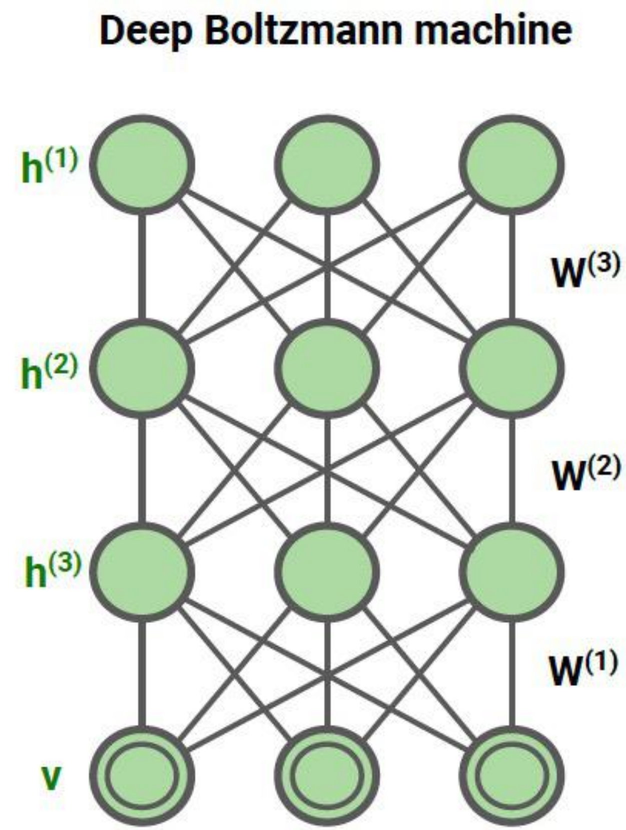
Content

- Boltzmann Machine
  - Introduction
  - Training without hidden neurons
  - Training with hidden neurons
  - Summary
- Restricted Boltzmann Machine
- Deep Boltzmann Machine

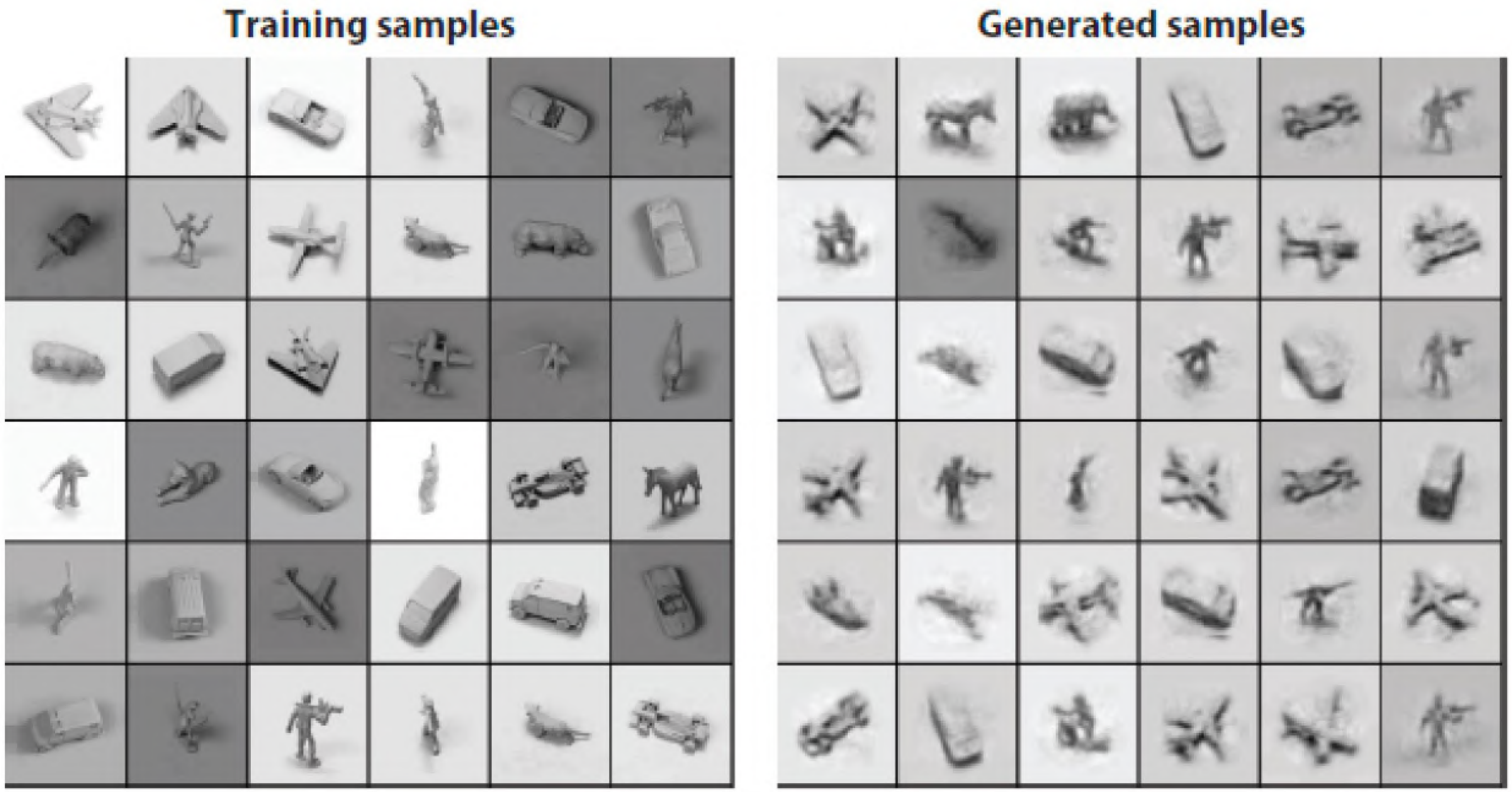
Deep Boltzmann Machines

- Stacked RBMs are one of the first deep generative models

- Bottom layer v are visible neurons
- Multiple hidden layers



Boltzmann Machines: samples



Reference

- CMU 11-785 Lec 19
- Stanford cs236 Lec 11

Summary

- Boltzmann Machine
  - Introduction
  - Training without hidden neurons
  - Training with hidden neurons
  - Summary
- Restricted Boltzmann Machine RBM
- Deep Boltzmann Machine DBM

北京大學 PEKING UNIVERSITY

Thanks