Deep Learning lecture 4 Energy-Based Model

Yi Wu, IIIS

Spring 2025

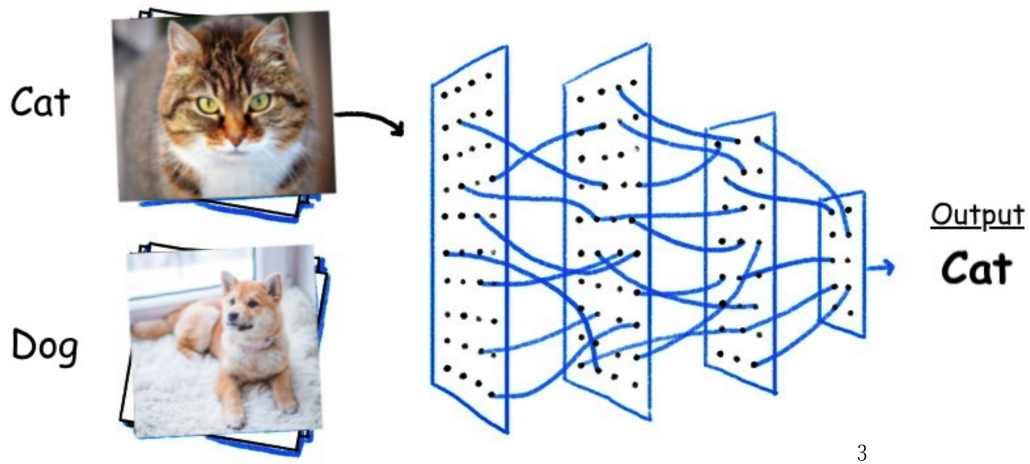
Mar-10

Logistics

- Coding Project 2 due in 1 week
  - Use local compute for coding & Colab for testing
  - Cloud for long-term training
  - Any questions can be posted in Dingding channel
  - Be aware of your model size and computation (flops)!
  - Check out those famous models and works!

Story So Far

- History
  - Lecture 1
    - first neural network (1943) to recent advances in deep learning
- Supervised Learning (Classification)
  - Lecture 2
    - MLP and basic components; Backpropagation
  - Lecture 3
    - Algorithms, Tricks and Architecture
- Discriminative Model
  - $P(y|X)$
  - Labeled data; $X \to y$

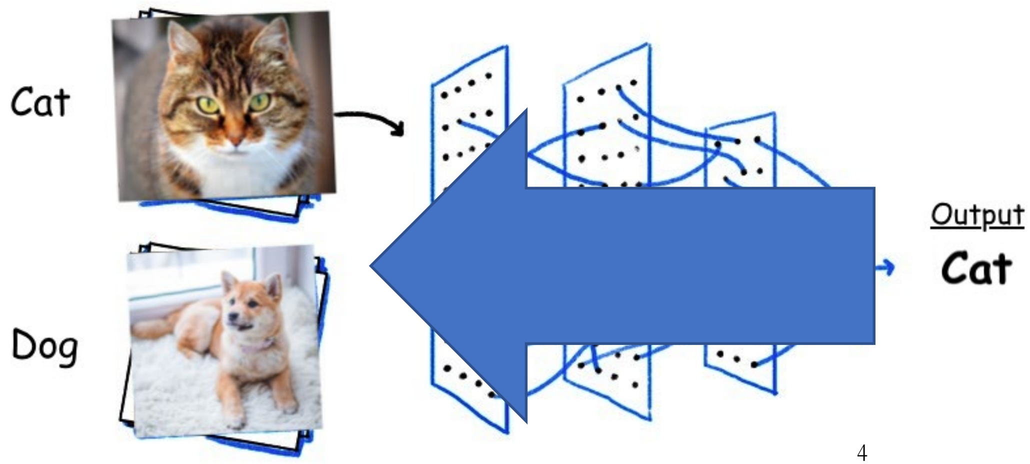


Afterwards

- What if we want to generate X?
  - E.g., Ask the neural network to generate a cat!

- Generative Model
  - $P(X, y) = P(y) * P(X|y)$
  - Or just $P(X)$

- Lecture 4~7
  - Deep Generative Models
  - Different approaches to model $P(X)$



Today's Lecture: Energy-Based Models

- A particularly flexible and general form of ***generative model***
- Part 1: Hopfield Network
  - The simplest model that can memorize and generate patterns
- Part 2: Boltzmann Machine
  - The first deep generative model
- Part 3: General Energy-Based Models & Sampling Methods

Today’s Lecture: Energy-Based Models

- A particularly flexible and general form of ***generative model***

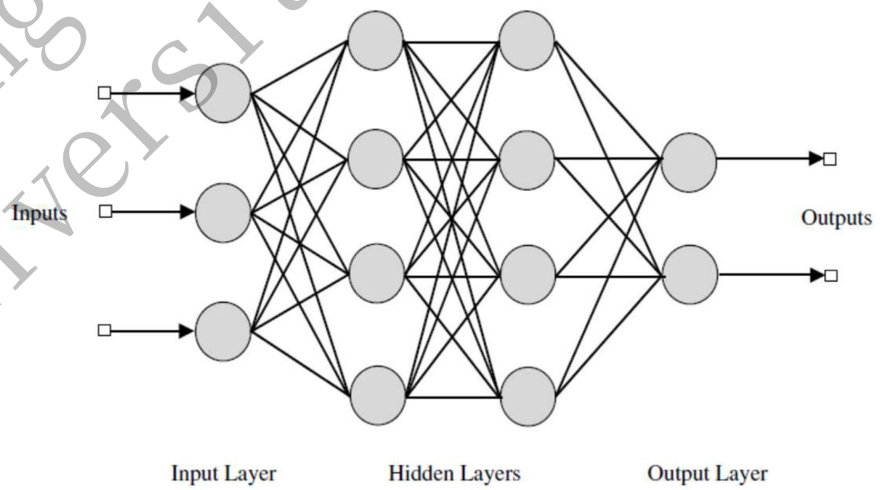
- Part 1: Hopfield Network
  - The simplest model that can memorize and generate patterns

- Part 2: Boltzmann Machine
  - The first deep generative model

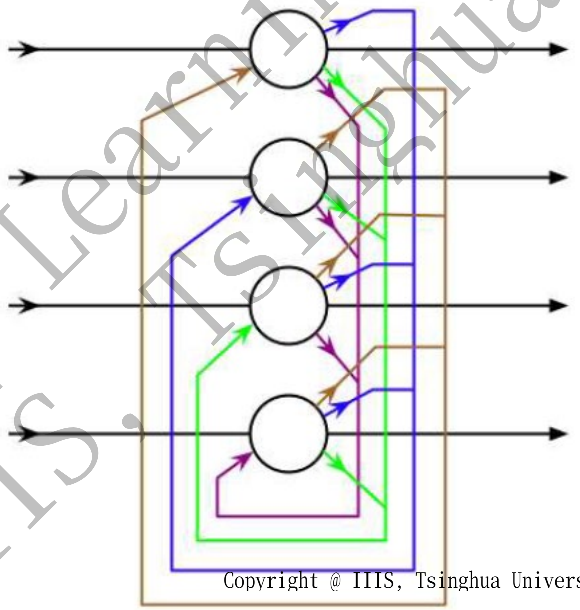
- Part 3: General Energy-Based Models & Sampling Methods

Classification

- Recap: Classification
  - Layer-by-layer computation
  - Computation Graph: Directed Acyclic Graph
  - Feedforward networks



- What about …
  - Loops!

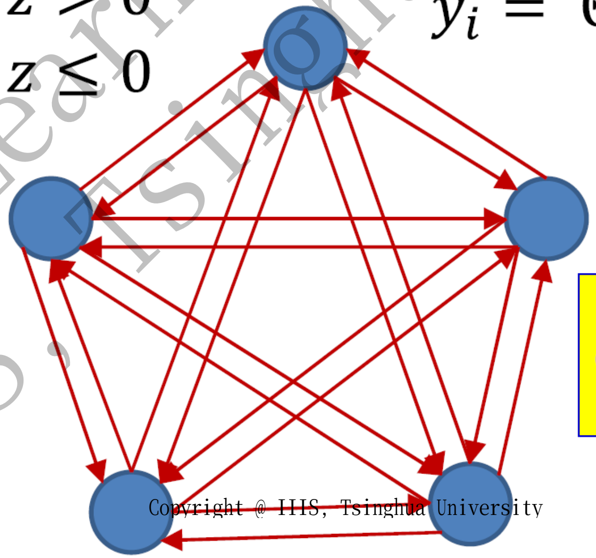


A Loopy Network

- A "fully-connected" network
  - Each neuron receives inputs from all the other neurons
  - $y_i = +1\ or\ -1$ with hard thresholding

$$\Theta(z) = \begin{cases} +1\ if\ z > 0 \\ -1\ if\ z \leq 0 \end{cases}$$

$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$



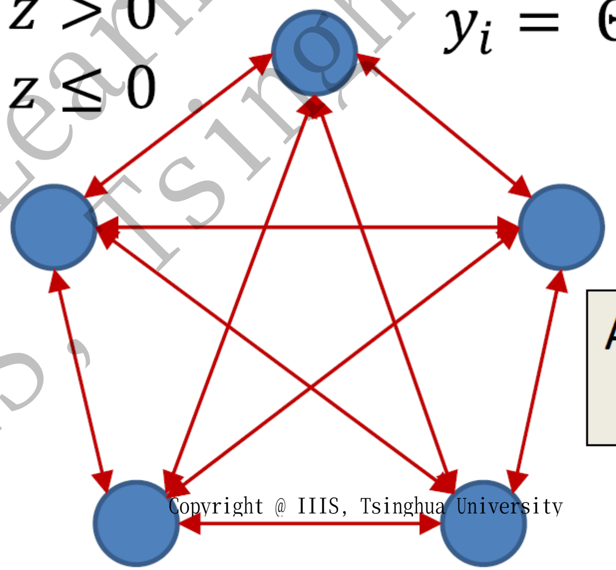
The output of a neuron affects the input to the neuron

Hopfield Network

- A “fully-connected” network
  - Each neuron receives inputs from all the other neurons
  - $y_i = +1\ or\ -1$ with hard thresholding
  - Symmetric weights

$$\Theta(z) = \begin{cases} +1\ if\ z > 0 \\ -1\ if\ z \leq 0 \end{cases}$$

$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$



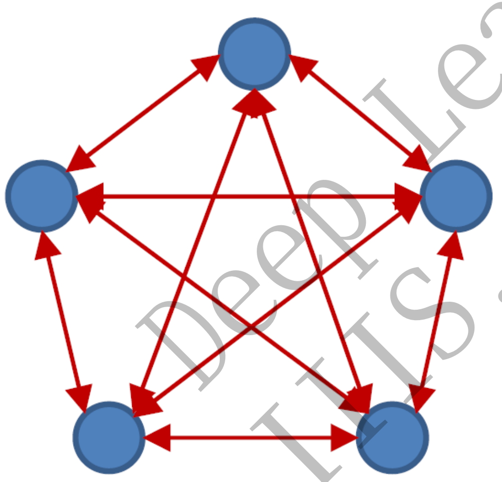
A symmetric network:
$w_{ij} = w_{ji}$

Hopfield Network

- A Hopfield Network may not be stable!
  - At each time each neuron receives a "field" $z_i = \sum_{j \neq i} w_{ji} y_j + b_i$
  - If the sign of neuron matches the sign of the field, it flips

$$y_i \leftarrow -y_i \text{ if } y_i \left(\sum_{j \neq i} w_{ji} y_j + b_i \right) < 0$$

  - This can further cause other neurons to flip!



$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i \right)$$

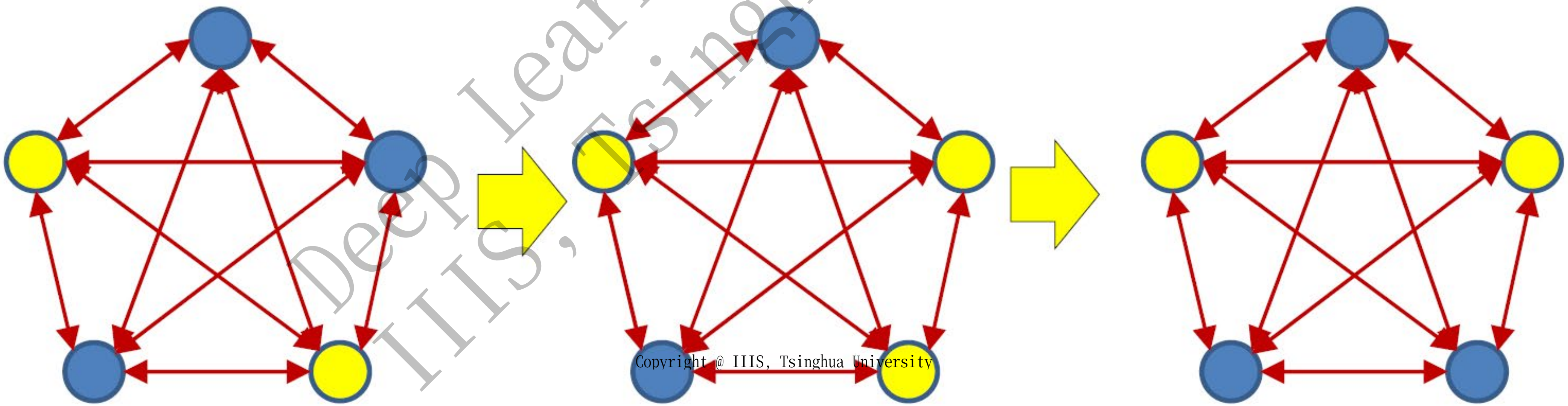
$$\Theta(z) = \begin{cases} +1 \; if \; z > 0 \\ -1 \; if \; z \leq 0 \end{cases}$$

Hopfield Network

- Neurons flip if its sign does not match its local "field"
  - $y_i \leftarrow -y_i$ if $y_i\left(\sum_{j \neq i} w_{ji} y_j + b_i\right) < 0$ for all neurons
  - Repeat until no neuron flips
  - Will this process converge?

$$\Theta(z) = \begin{cases} +1 \; if \; z > 0 \\ -1 \; if \; z \leq 0 \end{cases}$$

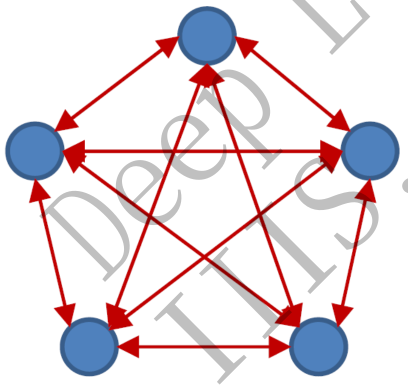
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$



Hopfield Network

- Let y_i^- denote the value of y_i before a "flip"
- Let y_i^+ denote the value of y_i after a "flip"
- If $y_i^-\left(\sum_{j\neq i} w_{ji}y_j + b_i\right) \geq 0$, nothing happen

$$y_i^+\left(\sum_{j\neq i} w_{ji}y_j + b_i\right) - y_i^-\left(\sum_{j\neq i} w_{ji}y_j + b_i\right) = 0$$



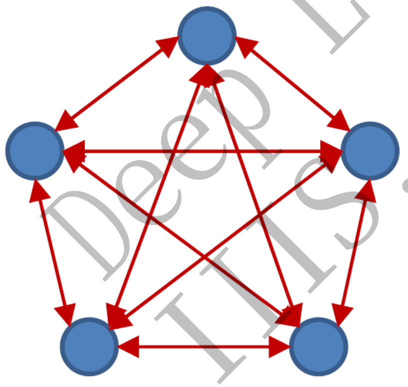
$$y_i = \Theta\left(\sum_{j\neq i} w_{ji}y_j + b_i\right)$$

$$\Theta(z) = \begin{cases} +1\ if\ z > 0 \\ -1\ if\ z \leq 0 \end{cases}$$

Hopfield Network

- Let y_i^- denote the value of y_i before a "flip"
- Let y_i^+ denote the value of y_i after a "flip"
- If $y_i^-\left(\sum_{j\neq i} w_{ji}y_j + b_i\right) \geq 0$, nothing happen
- If $y_i^-\left(\sum_{j\neq i} w_{ji}y_j + b_i\right) < 0$, $y_i^+ = -y_i^-$

$$y_i^+\left(\sum_{j\neq i} w_{ji}y_j + b_i\right) - y_i^-\left(\sum_{j\neq i} w_{ji}y_j + b_i\right) = 2y_i^+\left(\sum_{j\neq i} w_{ji}y_j + b_i\right)$$



$$y_i = \Theta\left(\sum_{j\neq i} w_{ji}y_j + b_i\right)$$

$$\Theta(z) = \begin{cases} +1\ if\ z > 0 \\ -1\ if\ z \leq 0 \end{cases}$$

Hopfield Network

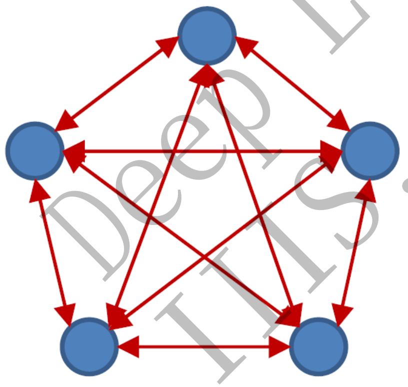
- Let y_i^- denote the value of y_i before a "flip"
- Let y_i^+ denote the value of y_i after a "flip"
- If $y_i^-\left(\sum_{j\neq i} w_{ji} y_j + b_i\right) \geq 0$, nothing happen
- If $y_i^-\left(\sum_{j\neq i} w_{ji} y_j + b_i\right) < 0$, $y_i^+ = -y_i^-$

Every flip increases

$2y_i\left(\sum_{j\neq i} w_{ji}\, y_j + b_i\right)$

$$y_i^+\left(\sum_{j\neq i} w_{ji} y_j + b_i\right) - y_i^-\left(\sum_{j\neq i} w_{ji} y_j + b_i\right) = 2y_i^+\left(\sum_{j\neq i} w_{ji} y_j + b_i\right)$$

Positive!



$$y_i = \Theta\left(\sum_{j\neq i} w_{ji} y_j + b_i\right)$$

$$\Theta(z) = \begin{cases} +1 \; if \; z > 0 \\ -1 \; if \; z \leq 0 \end{cases}$$

Hopfield Network

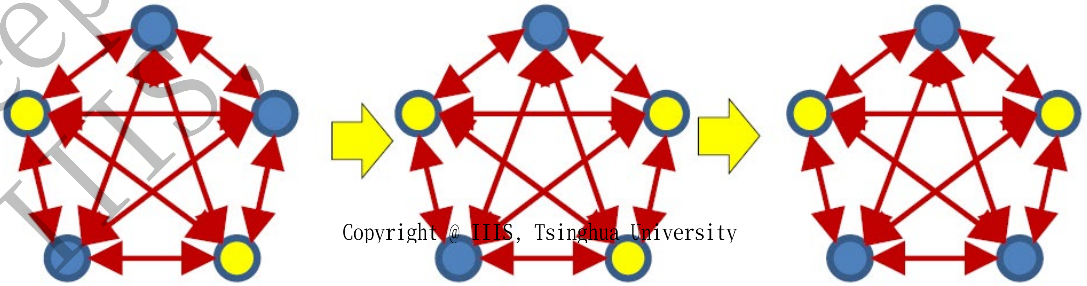
- Consider the sum over every pair of neurons (assume $w_{ii} = 0$)

$$D(y_1, \dots, y_N) = \sum_{i<j} y_i w_{ij} y_j + y_i b_i$$

- Any flip that changes y_i^- to y_i^+ increases $D(y_1, \dots, y_N)$

$$\Delta D = D(\dots, y_i^+, \dots) - D(\dots, y_i^-, \dots) = 2y_i^+ \left(\sum_{j \neq i} w_{ji} y_j + b_i \right) > 0$$

- Convergence?



Hopfield Network

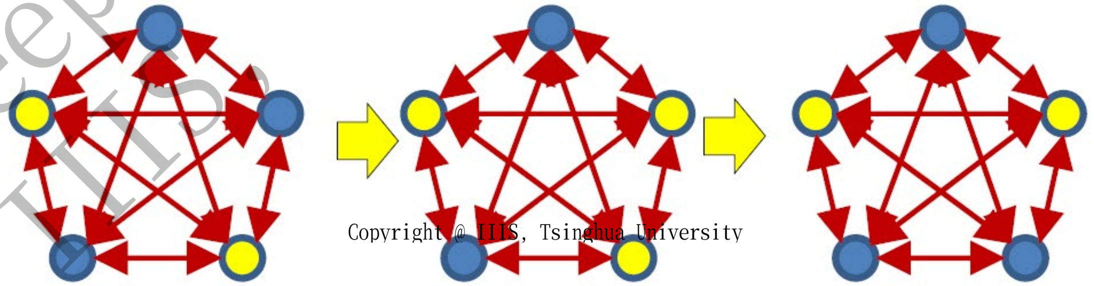
- D is upper-bounded (we only change y_i)

$$D(y_1, \dots, y_N) = \sum_{i<j} w_{ij} y_i y_j + \sum_i b_i y_i \le \sum_{i<j} |w_{ij}| + \sum_i |b_i|$$

- ΔD is lower-bounded

$$\Delta D_{\min} = \min_{i,\{y_j\}} 2 \left| \sum_j w_{ij} y_j + b_i \right| > 0$$

- $\{y_i\}$ converges with a finite number of iterations!
  - $\{y_i\}$: *state*



Hopfield Network

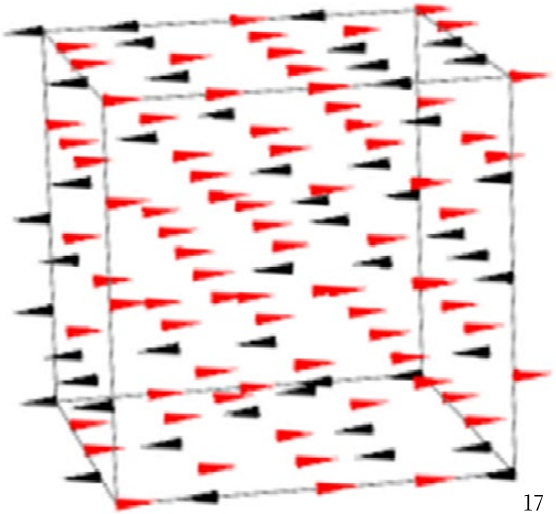
- The ***Energy*** of Hopfield Network

$$E = -D = -\sum_{i<j} w_{ij} y_i y_j - \sum_i b_i y_i$$

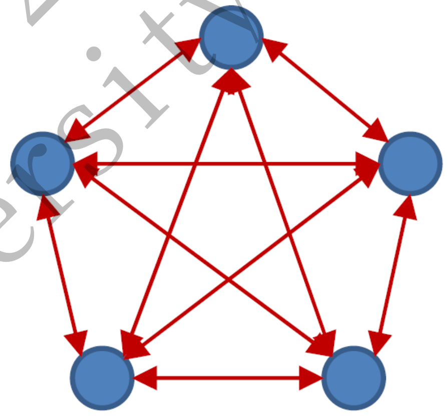
  - The evolution of Hopfield network always decreases its energy!
- The concept of ***Energy***
  - Magnetic dipoles in a disordered magnetic material
  - Each dipole tries to align itself to the local field
  - Field at a particular dipole $f(p_i)$, p_i is the position of x_i

$$f(p_i) = \sum_{j \neq i} J_j x_j + b_i$$

  - ***Ising model*** of magnetic materials (Ising and Lenz, 1924)



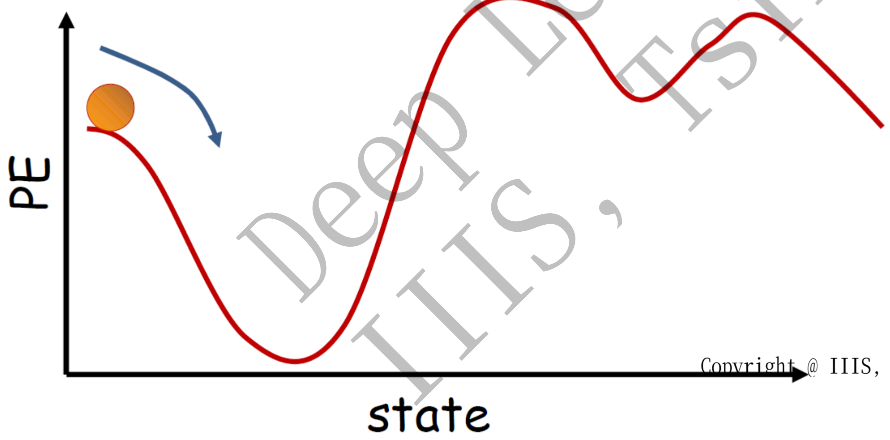
Hopfield Network: Pattern Generation

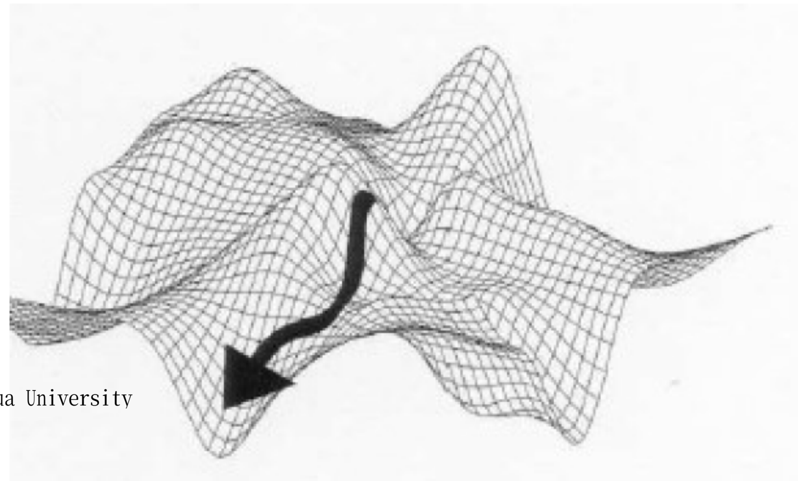


- The Hopfield network (simplified)

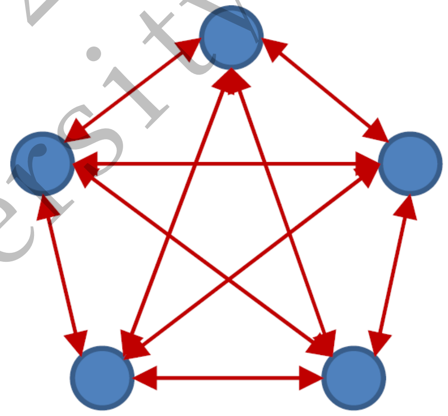
$$E = -\sum_{i<j} w_{ij} y_i y_j$$

  - Network evolution arrives at a local optimum in the energy contour
    - Every change in the network state Y decreases the energy E
  - ***Any small jitter from this stable state returns it to the stable state***





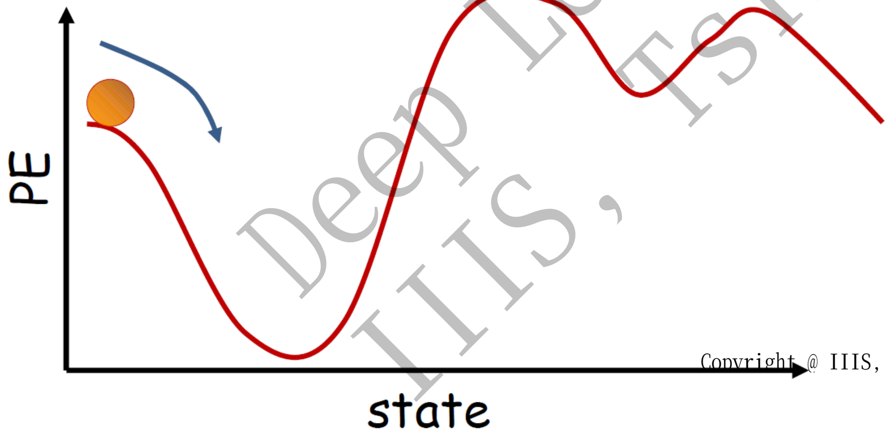
Hopfield Network: Pattern Generation

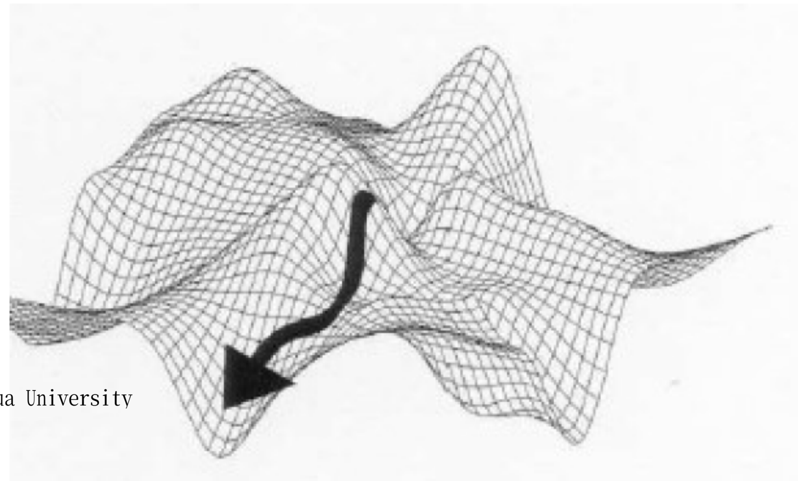


- The Hopfield network (simplified)

$$E = -\sum_{i<j} w_{ij} y_i y_j$$

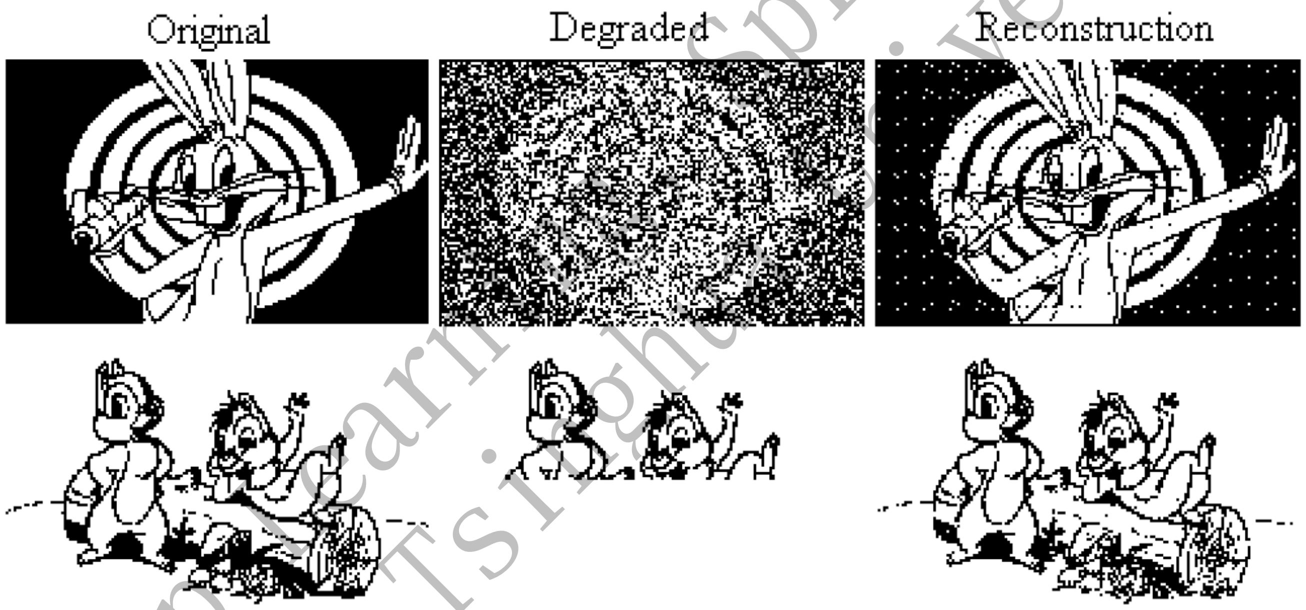
  - Each local optimum state is a “stored” pattern
    - If the network is initialized close to a stored pattern, it evolves to the pattern
  - *Associated Memory (content addressable memory)*





Hopfield Network: Pattern Generation

- Image Reconstruction by Hopfield Network (1982)



Hopfield network reconstructing degraded images
from noisy (top) or partial (bottom) cues.

- *How can we store the desired patterns?*

Hopfield Network: Training

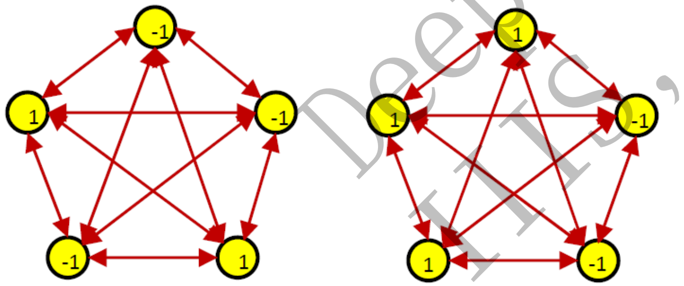
- Let's teach the network to store this image
  - N pixels → N neurons
  - Symmetric weights → $\frac{1}{2}N(N-1)$ parameters to learn
    - We omit bias terms for simplicity

- Design $\{w_{ij}\}$ such that the energy is at a local minimum for a desired pattern y
  - Hebbian Learning Rule $w_{ij} \leftarrow y_i y_j$ (1949)
  - $E = -\sum_{i<j} w_{ij} y_i y_j = -\frac{1}{2}N(N-1)$ → lowest possible energy!

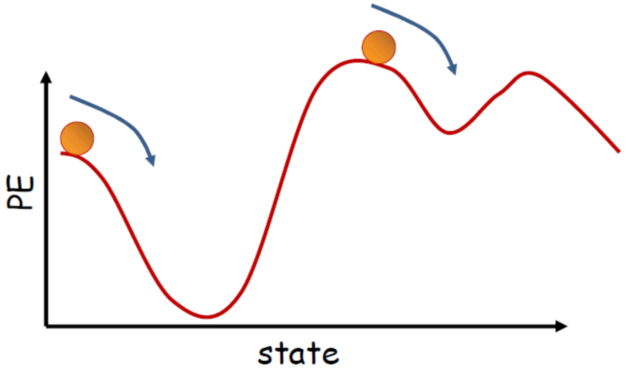
Hopfield Network: Training

- Let’s teach the network to store this image
  - N pixels → N neurons
  - Symmetric weights → $\frac{1}{2}N(N-1)$ parameters to learn
    - We omit bias terms for simplicity

- Design $\{w_{ij}\}$ such that the energy is at a local minimum for a desired pattern y
  - Redundancy! y & $-y$ will be both stored



$$E = -\sum_{i}\sum_{j<i} w_{ji} y_j y_i$$



Hopfield Network: Training

- What if we want to store ***multiple*** patterns?
  - $P = \{y^p\}$ N_p patterns
  - Hebbian Learning Rule

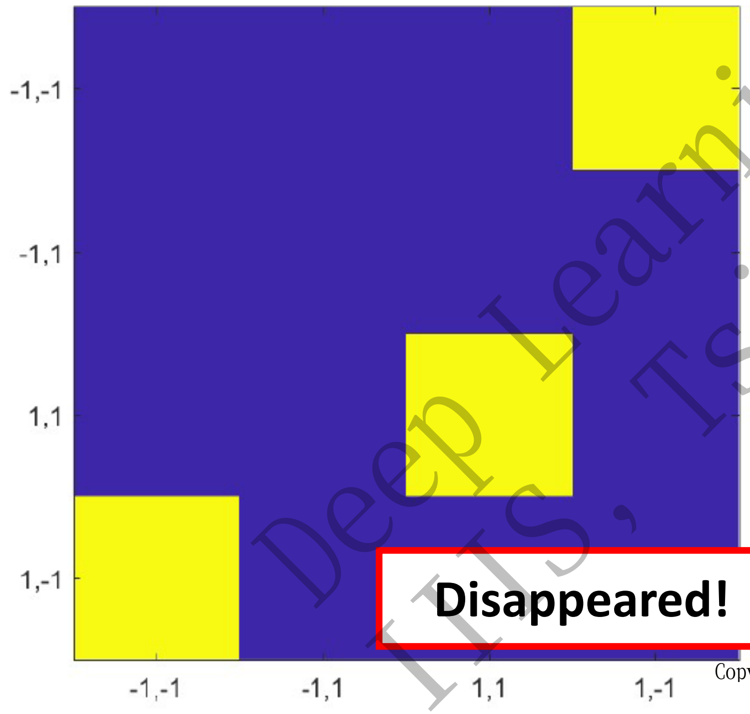
$$w_{ij} = \frac{1}{N_p}\sum_p y_i^p y_j^p$$

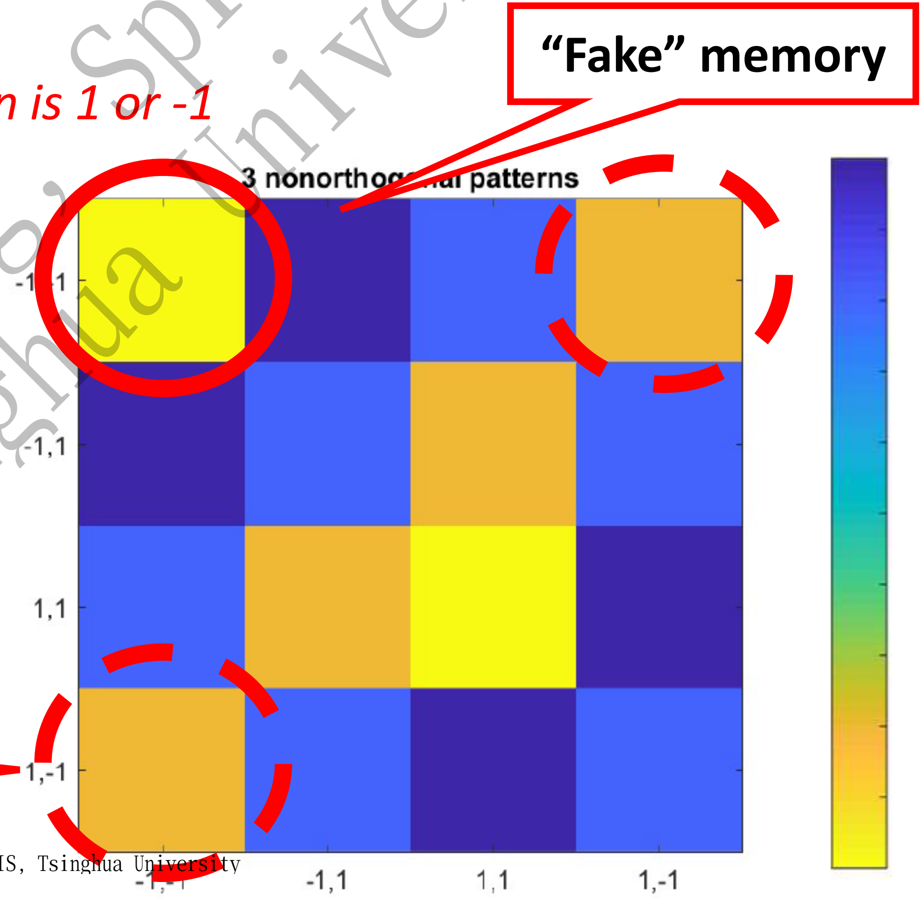
- The issue of Hebbian Learning
  - Spurious local optima

Hopfield Network: Training

- Example: 4-dimensional Hopfield Network with Hebbian Learning
  - Three patterns to store
    - *Let's assume the value of each neuron is 1 or -1*

Left: desired patterns





Right: stored patterns

Hopfield Network: Training

- We want to construct a network with desired ***stable*** local optimum
  - A pattern can be recovered after 1-bit change
- For a specific set of K patterns, we can always build a network for which all patterns are stable provided $K \leq N$
  - Mostafa and St. Jacques (1985)
  - For large N, the upper bound on K is actually $\frac{N}{4 \log N}$
  - McElice et. al. (1987)
  - Still possible with undesired local minimum
- **How can we find the weights?**
  - K patterns to be stored
  - Avoid undesired local minimum as much as we can

Hopfield Network: Optimization

- Problem Formulation
  - Desired patterns $P = \{y^p\}$
  - Energy function $E(y) = -\frac{1}{2} y^T W y$ (we omit bias term for simplicity)
- Objective for W
  - Minimize E for all y^p
  - It should also maximize E for all non-desired patterns!

$$W = \arg\min_W \sum_{y \in P} E(y) - \sum_{y' \notin P} E(y')$$

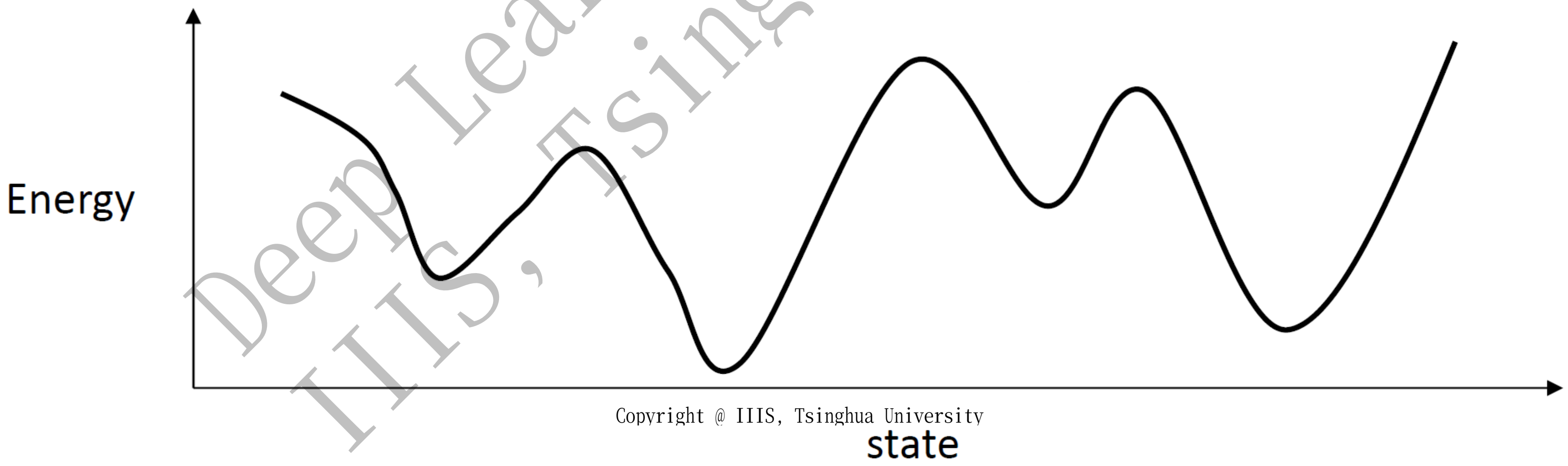
  - Gradient Descent

$$W \leftarrow W - \eta \left(\sum_{y \in P} y y^T - \sum_{y' \notin P} y' y'^T \right)$$

Hopfield Network: Optimization

- Update rule for W

$$W \leftarrow W - \eta\left(\sum_{y\in P} yy^T - \sum_{y'\notin P} y'y'^T\right)$$

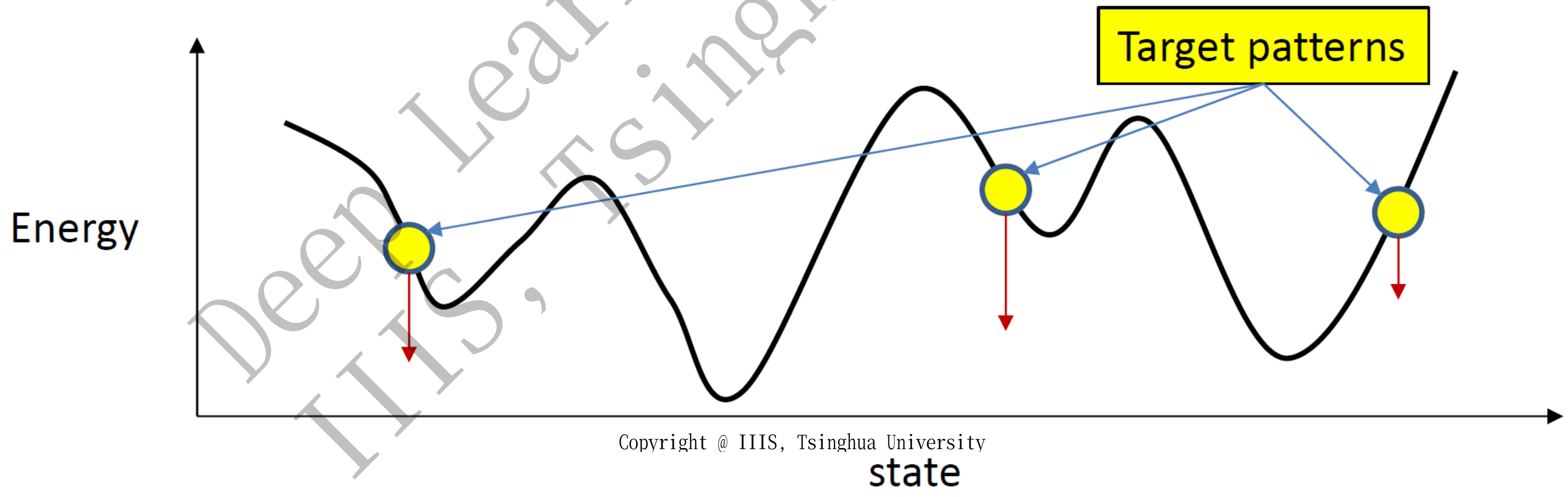


Hopfield Network: Optimization

- Update rule for W

$$W \leftarrow W - \eta\left(\boxed{\sum_{y \in P} yy^T} - \sum_{y' \notin P} y'y'^T\right)$$

  - The first term is minimizing the energy of desired patterns!

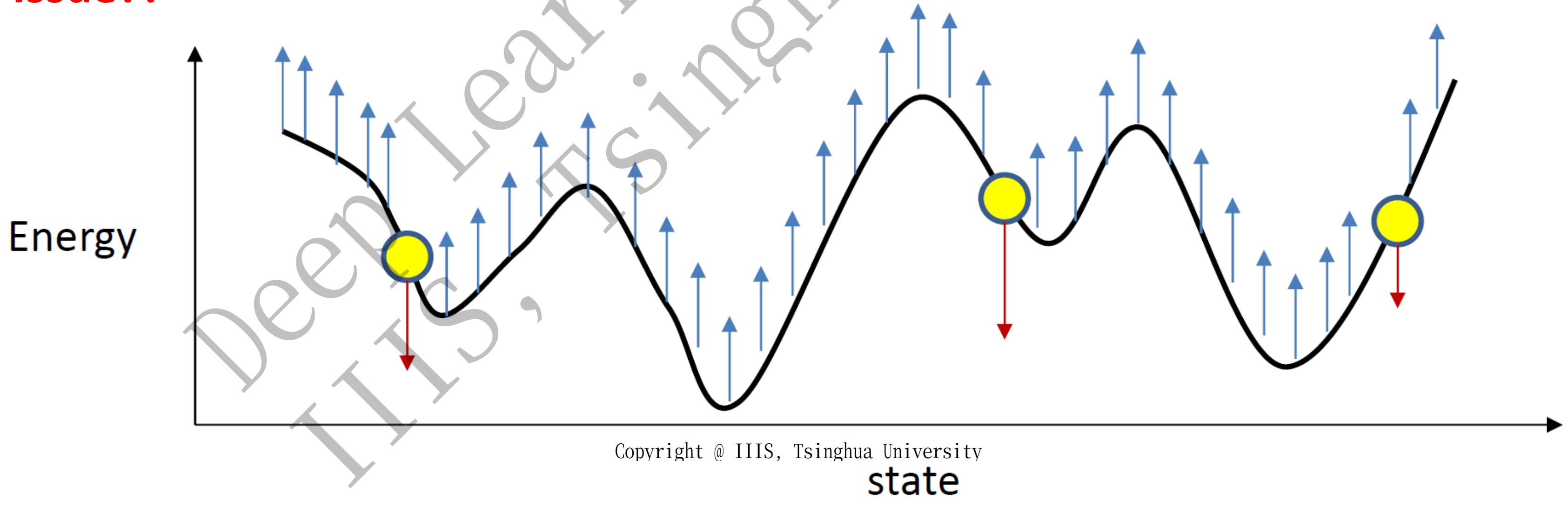


Hopfield Network: Optimization

- Update rule for W

$$W \leftarrow W - \eta\left(\sum_{y \in P} yy^T - \boxed{\sum_{y' \notin P} y'y'^T}\right)$$

- The second term essentially raises all the patterns in the space
  - **Issue??**

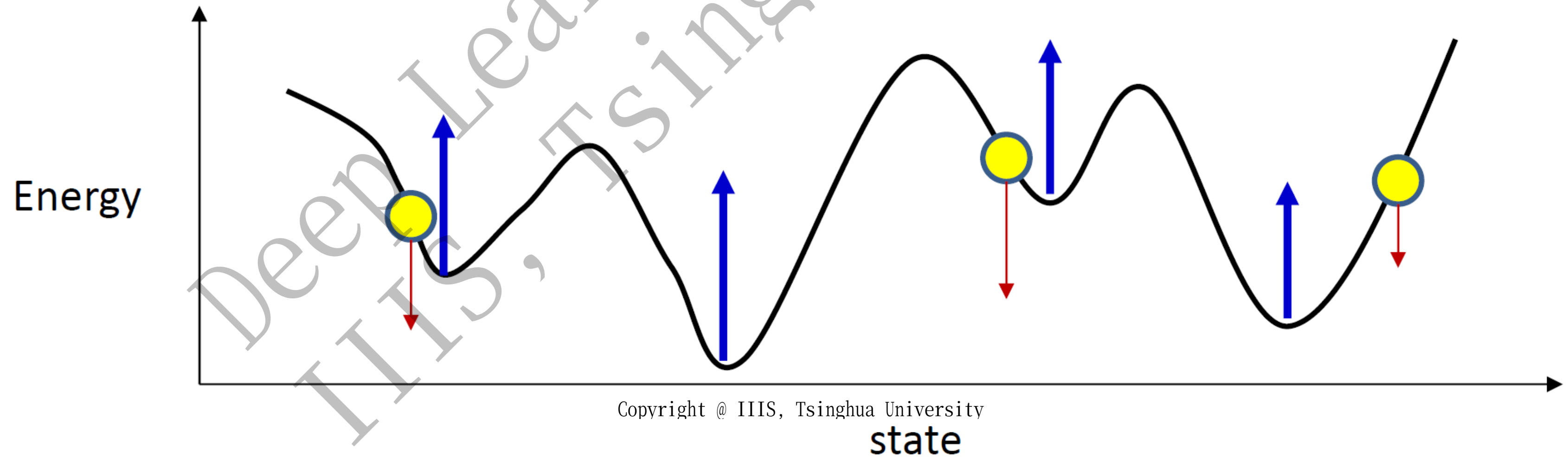


Hopfield Network: Optimization

- Update rule for W

$$W \leftarrow W - \eta \left(\sum_{y \in P} y y^T - \sum_{y' \notin P \ \& \ \boldsymbol{y' \in Valley}} y' y'^T \right)$$

- Let's just focus on the valleys!

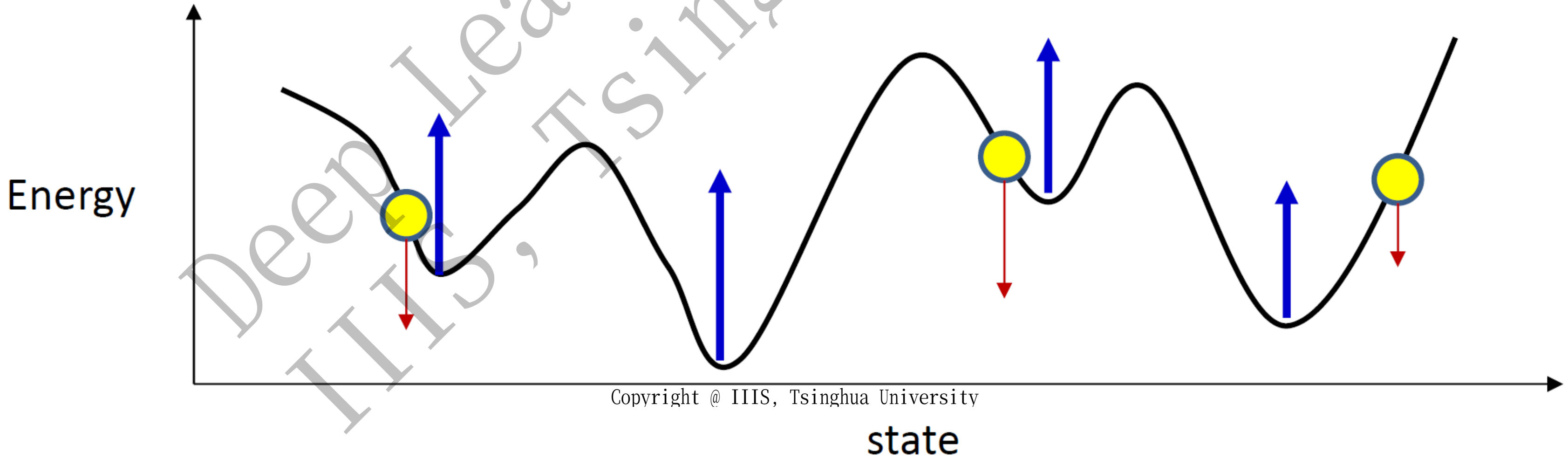


Hopfield Network: Optimization

- Update rule for W

$$W \leftarrow W - \eta\left(\sum_{y \in P} yy^T - \sum_{y' \notin P \ \& \ \boldsymbol{y' \in Valley}} y'y'^T\right)$$

- Let's just focus on the valleys!
- **But how can we find the valleys?**

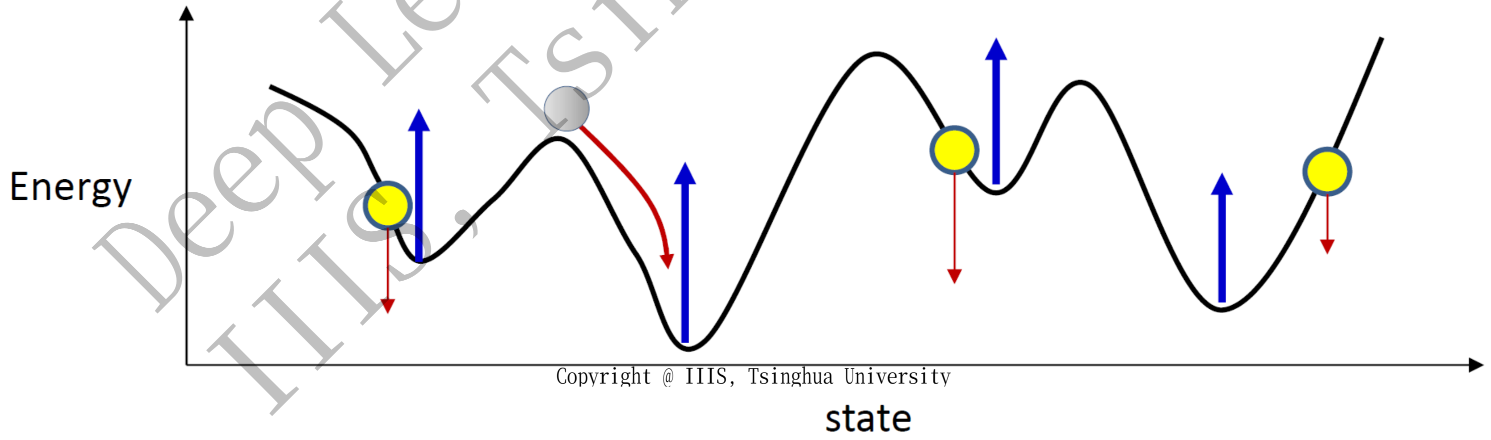


Hopfield Network: Optimization

- Update rule for W

$$W \leftarrow W - \eta \left(\sum_{y \in P} y y^T - \sum_{y' \notin P \,\&\, \boldsymbol{y' \in Valley}} y' y'^T \right)$$

  - Let's just focus on the valleys!
  - But how can we find the valleys?
  - **Evolution of Hopfield Network will converge to a valley**



Hopfield Network: Optimization

- Update rule for W

$$W \leftarrow W - \eta \left(\sum_{y \in P} yy^T - \sum_{y' \notin P \,\&\, \boldsymbol{y'} \in \boldsymbol{Valley}} y'y'^T \right)$$

  - Compute outer-products of desired patterns y
  - Randomly initialize y' for multiple times
    - Run evolution for random y' until convergence
    - Calculate outer-product of y'
  - Compute gradient and update W

Hopfield Network: Optimization

- Update rule for W

$$W \leftarrow W - \eta \left(\sum_{y \in P} yy^T - \sum_{y' \notin P \ \& \ \boldsymbol{y'} \in \boldsymbol{Valley}} y'y'^T \right)$$

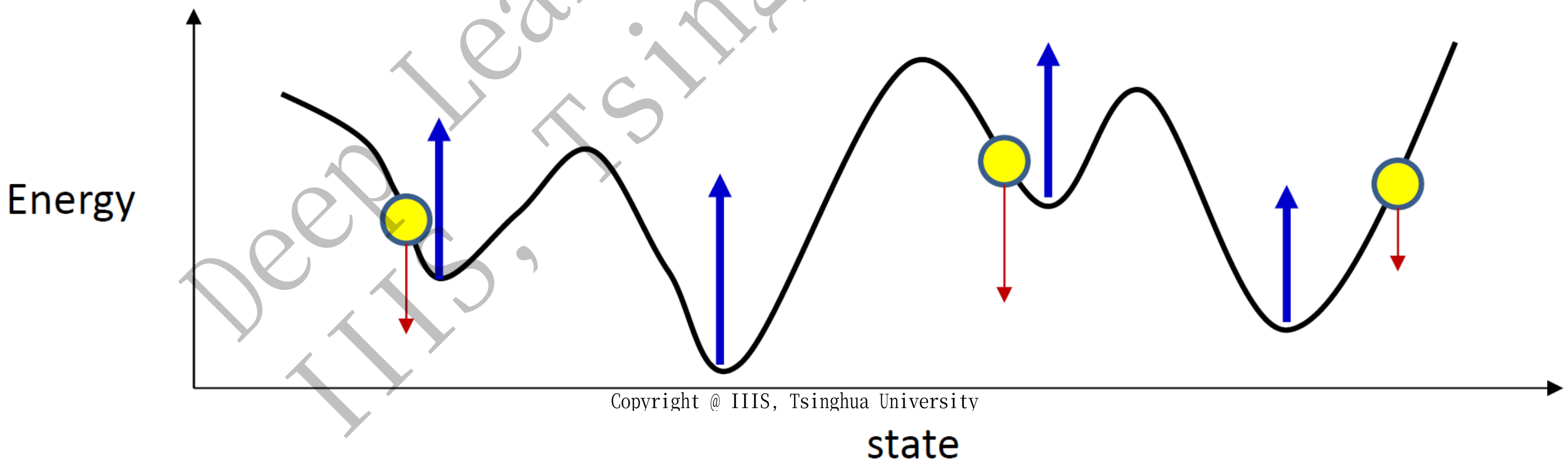
  - Compute outer-products of desired patterns y
  - Randomly initialize y' for multiple times
    - Run evolution for random y' until convergence
    - Calculate outer-product of y'
  - Compute gradient and update W

  - **Valleys are NOT equivalently important…**

Hopfield Network: Optimization

- Which valleys are important?

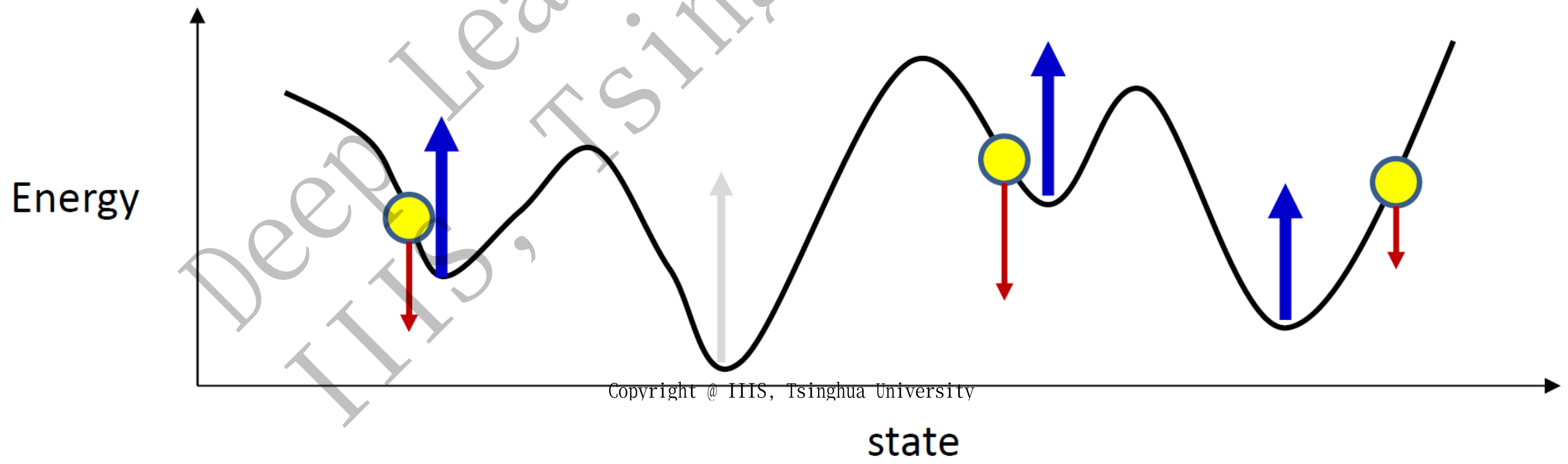
- Primary object: ensure desired pattens stable
  - We want to ensure desired patterns are in broad valleys



Hopfield Network: Optimization

- Which valleys are important?

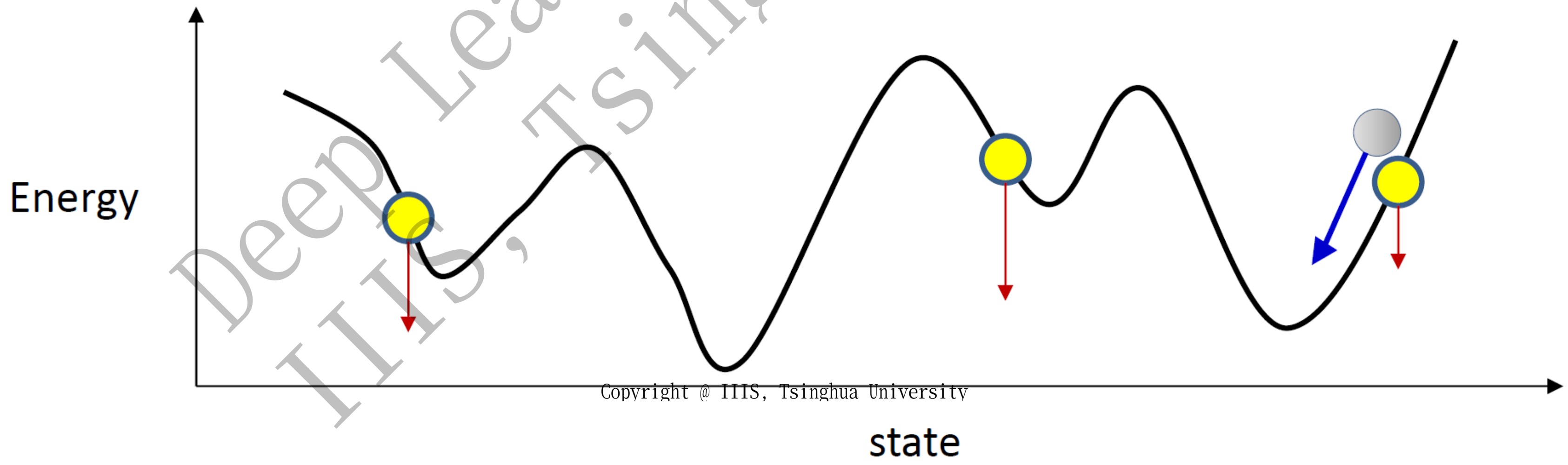
- Primary object: ensure desired pattens stable
  - We want to ensure desired patterns are in broad valleys
  - **Spurious valleys around desired patterns are more important to eliminate**



Hopfield Network: Optimization

- Which valleys are important?

- Primary object: ensure desired pattens stable
  - We want to ensure desired patterns are in broad valleys
  - Spurious valleys around desired patterns are more important to eliminate
  - **Evolution from desired patterns**



Hopfield Network: Optimization

- Update rule for W

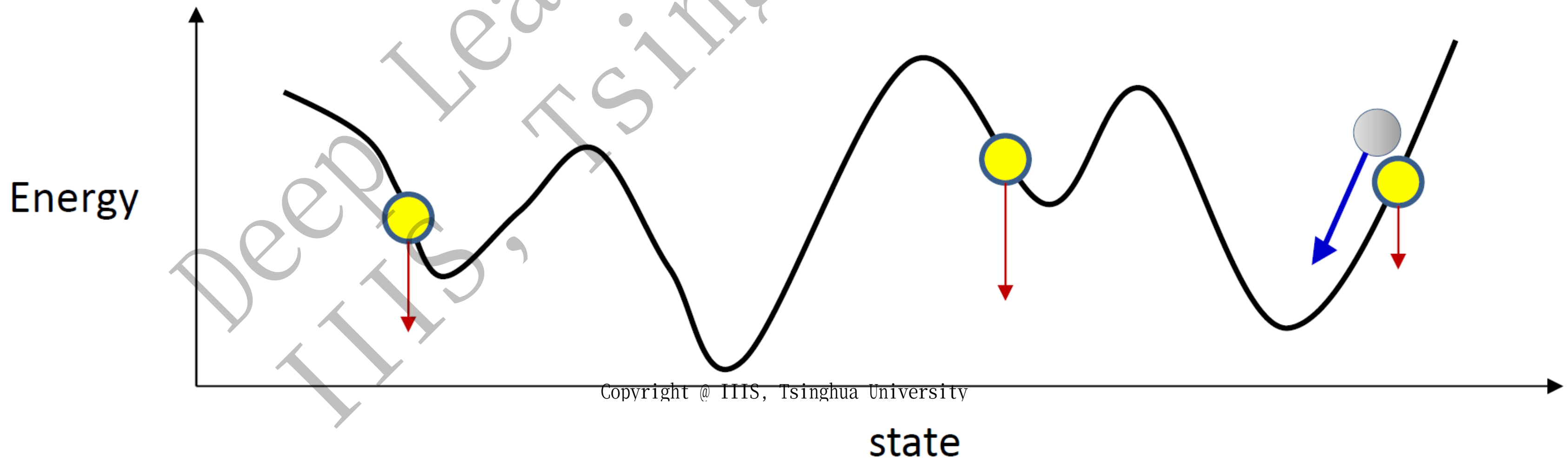
$$W \leftarrow W - \eta \left(\sum_{y \in P} y y^T - \sum_{y' \notin P \ \& \ \boldsymbol{y' \in Valley}} y' y'^T \right)$$

  - Compute outer-products of desired patterns y
  - Initialize y' by all the desired patterns
    - Run evolution for random y' until convergence
    - Calculate outer-product of y'
  - Compute gradient and update W

  - **Still issues?**

Hopfield Network: Optimization

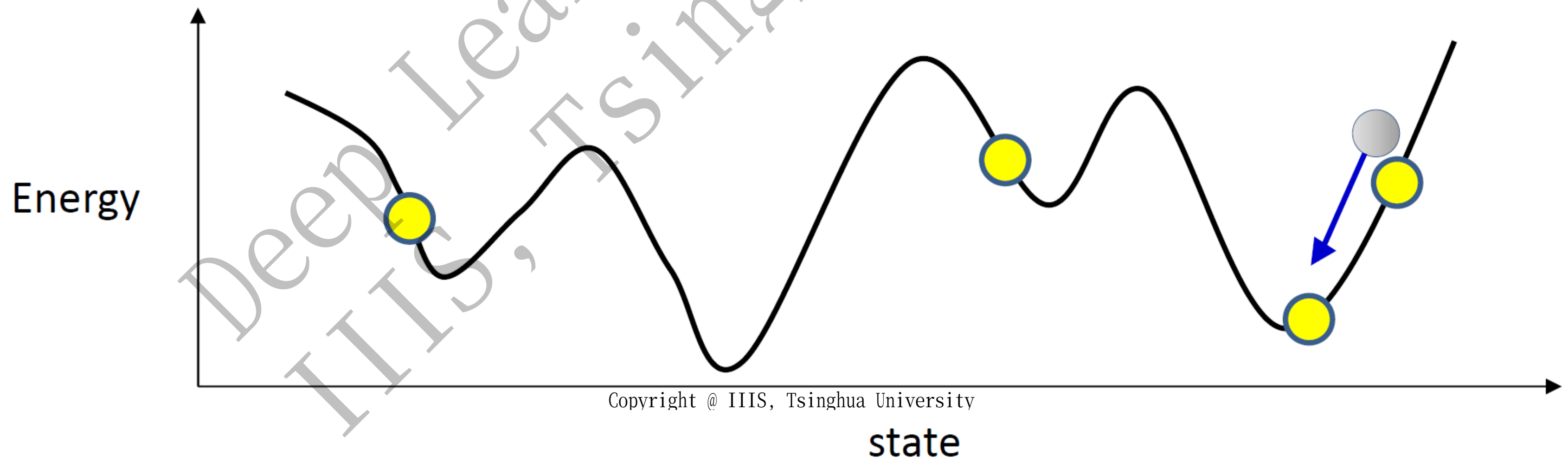
- Recap: we raise the valleys next to the desired patterns



Hopfield Network: Optimization

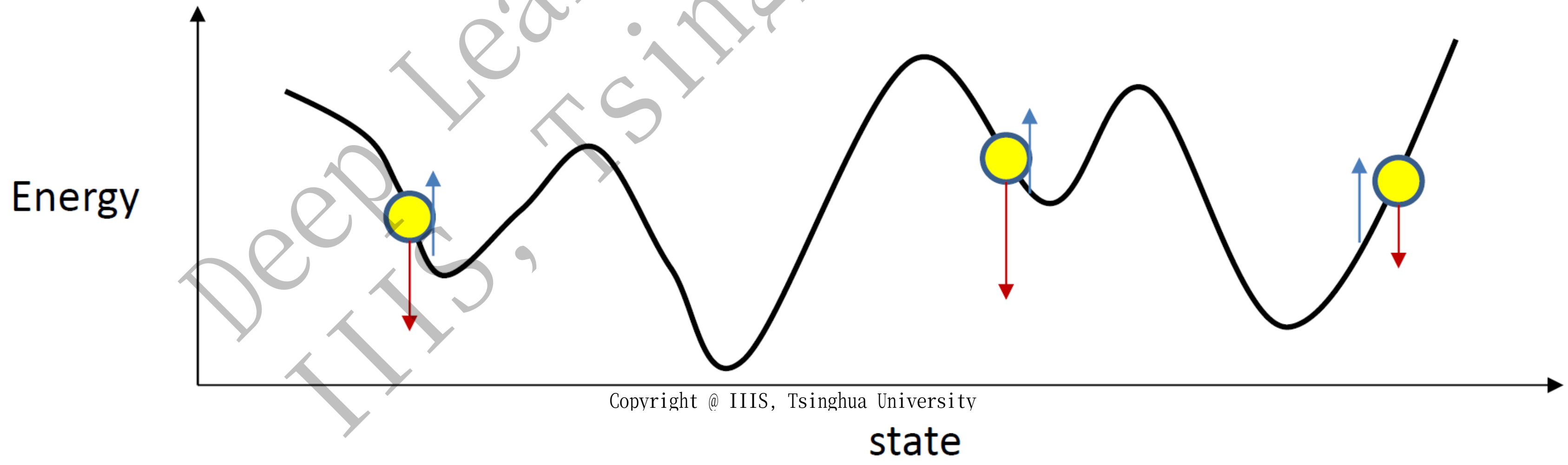
- Recap: we raise the valleys next to the desired patterns

- What if a pattern is close to the valley?
  - Naively forcing a valley to raise may hurt the learned representation
  - Particularly challenging when y are continuously valued (e.g., tanh activation)



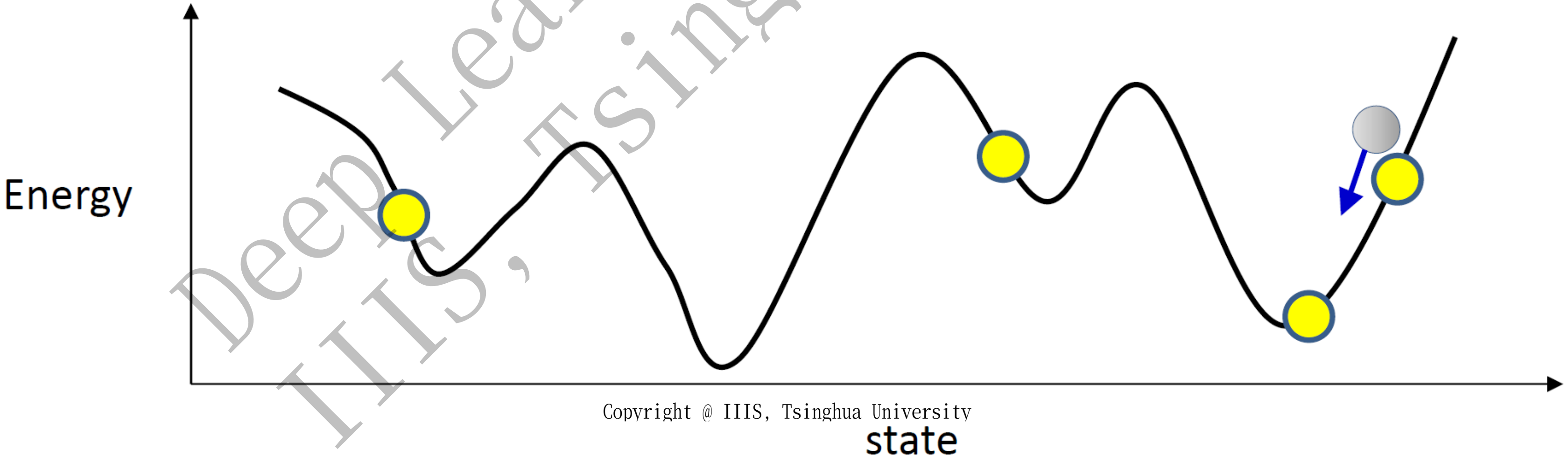
Hopfield Network: Optimization

- New idea: we only raise the “neighborhood” of desired patterns!
  - It is sufficient to make each desired pattern a valley
  - Note: we want to raise the neighborhood of the decent direction



Hopfield Network: Optimization

- New idea: we only raise the “neighborhood” of desired patterns!
  - It is sufficient to make each desired pattern a valley
  - Note: we want to raise the neighborhood of the decent direction
- Implementation
  - We initialize y' by the desired patterns
  - **Only perform evolution for a few steps!**



Hopfield Network: SGD Optimization

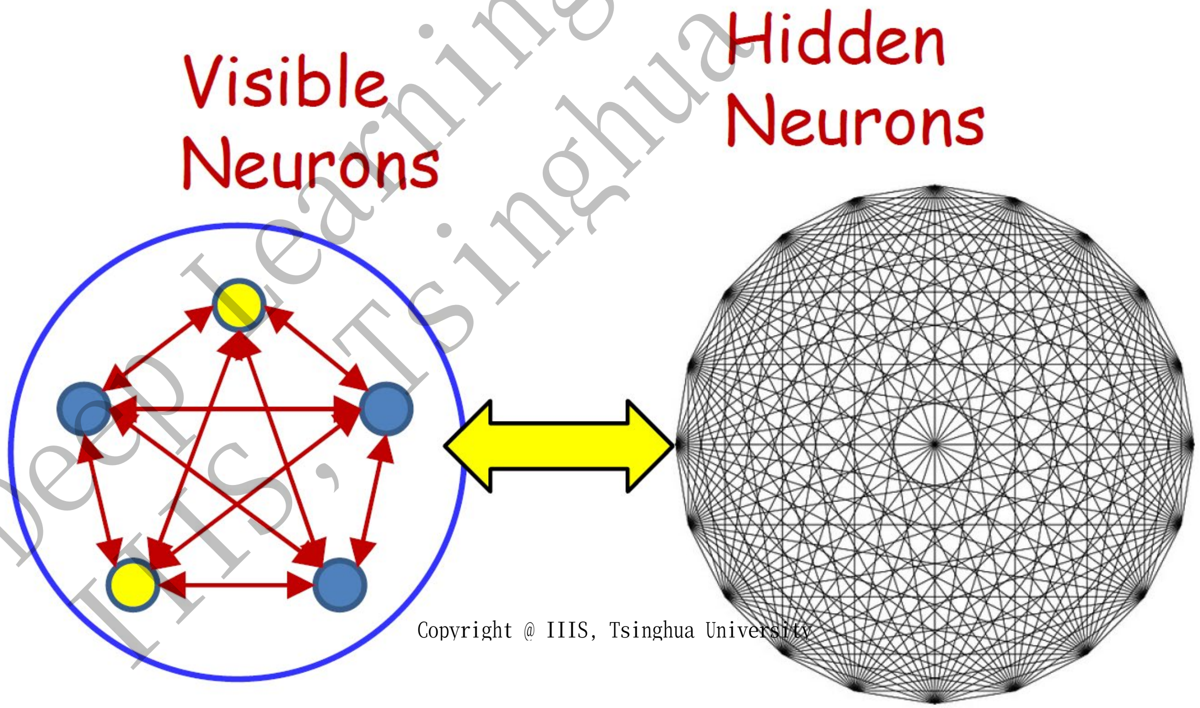
- SGD update rule for W

$$W \leftarrow W - \eta\left(\mathrm{E}_{y \in P}[yy^T] - \mathrm{E}_{y'}[y'y'^T]\right)$$

  - Compute outer-products of random desired pattern y
  - Initialize y' by a random desired pattern
    - Run evolution for random y' for a few timesteps (2~4)
    - Calculate outer-product of y'
  - Compute gradient and update W
- In theory, $O(N)$ patterns can be stored in the network (with undesired valleys)
  - How to store more patterns?

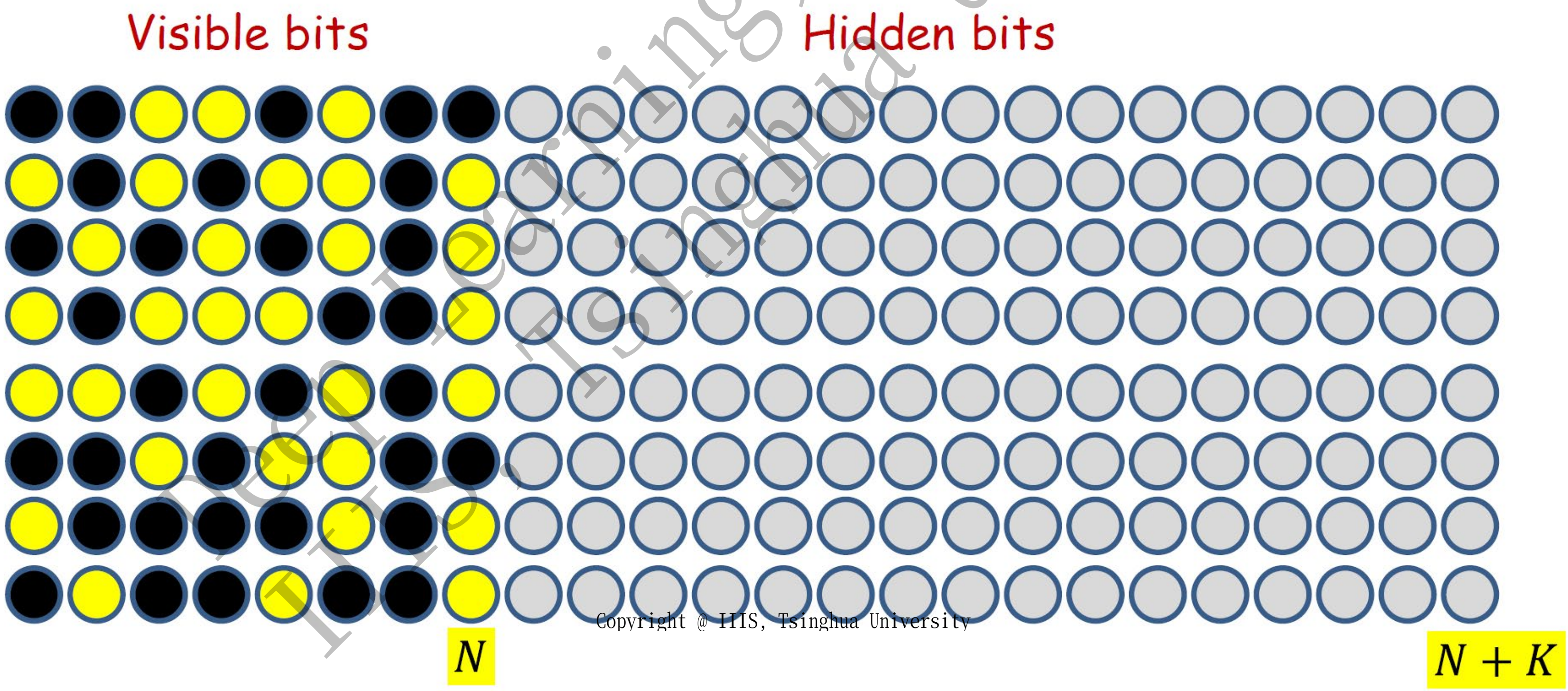
The Expanded Network

- Idea: introduce redundant neurons to increase network capacity
- Original N neurons for patterns: visible neurons
- Additional K neurons: hidden neurons



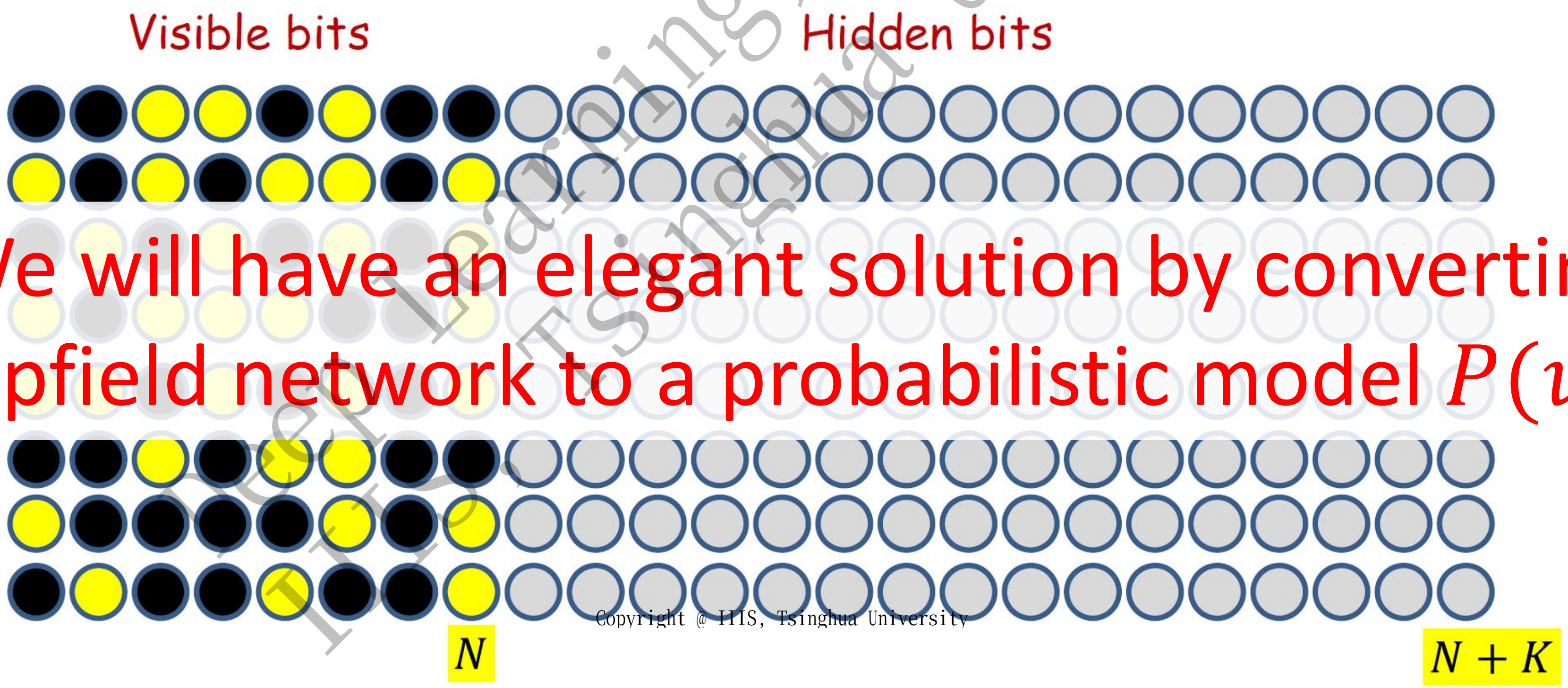
The Expanded Network

- Idea: introduce redundant neurons to increase network capacity
- Original N neurons for patterns: visible neurons
- Additional K neurons: hidden neurons



The Expanded Network

- N dimensional pattern → $N + K$ dimension
  - Q1: How can we store the patterns with K additional units? (random filling?)
  - Q2: How to retrieve the desired patterns? (perform evolution?)



We will have an elegant solution by converting a Hopfield network to a probabilistic model $P(v, h)$!

Today's Lecture: Energy-Based Models

- A particularly flexible and general form of ***generative model***

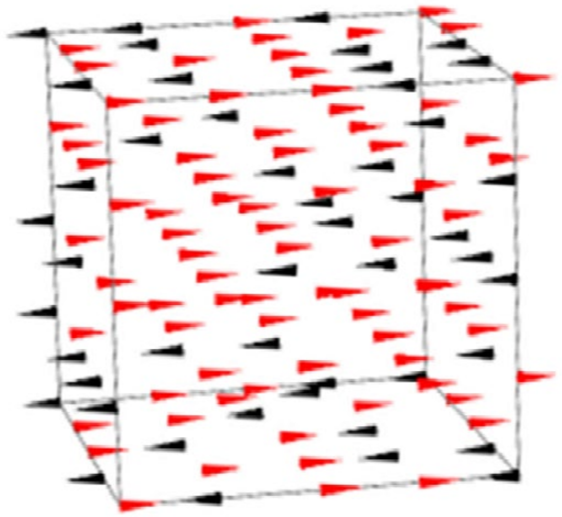
- Part 1: Hopfield Network
  - The simplest model that can memorize and generate patterns

- Part 2: Boltzmann Machine
  - The first deep generative model

- Part 3: General Energy-Based Models & Sampling Methods

The Helmholtz Free Energy

- Recap: A thermodynamic （热力学） system
  - A probabilistic system
  - Hopfield network is a simplified deterministic version
- A thermodynamic system at temperature T
  - $P_T(S)$ the probability of the system at state S
  - $E_T(S)$ the potential energy at state S
  - U_T the internal energy, the capability to do work
  - H_T the entropy, internal disorder of the system
  - k Boltzmann constant
  - Free energy $F_T = U_T - kTH_T$



The Helmholtz Free Energy

- Free energy

$$F_T = \sum_S P_T(S) E_T(S) + kT \sum_S P_T(S) \log P_T(S)$$

- Boltzmann distribution (also known as Gibbs distribution)

$$P_T(S) = \frac{1}{Z} \exp\left(-\frac{E_T(S)}{kT}\right)$$

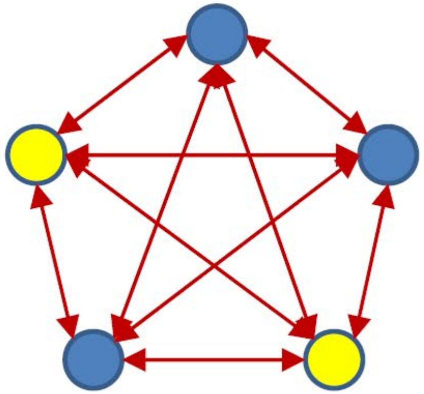
  - Minimum Free-Energy Principle: minimize F_T w.r.t. $P_T(S)$
  - The probability distribution of states at equilibrium
  - Z normalizing constant

Given an energy function $E_T(S)$, if we follow a proper physical evolution process, the system state will converge to the Boltzmann distribution

Stochastic Hopfield Network

- Let's model our Hopfield network as a thermodynamic system
  - $T = k = 1$ for simplicity
  - Energy

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

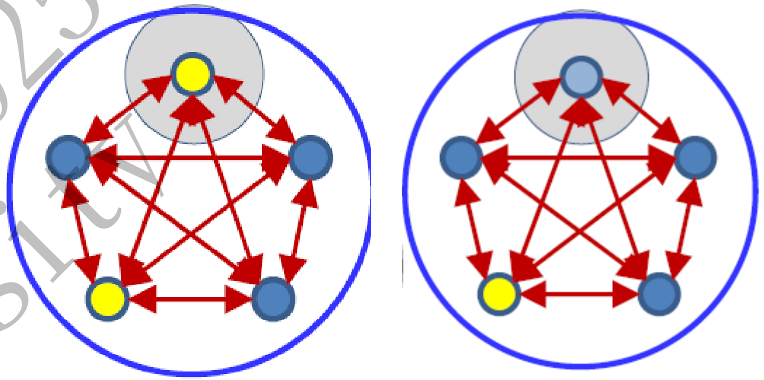


  - Boltzmann Probability

$$P(y) = \frac{1}{Z} \exp(-E(y)) = \frac{1}{Z} \exp\left(\sum_{i<j} w_{ij} y_i y_j + b_i y_i\right)$$

- Stochastic Hopfield Network
  - $P(y)$ models the stationary probability distribution of states y given $E(y)$
  - We generate patterns by sampling from $P(y)$

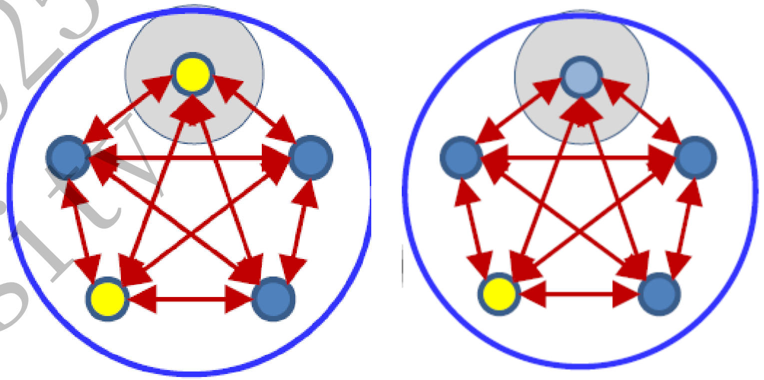
Stochastic Hopfield Network



- Let's consider the "flip" operation
  - Deterministic → probabilistic
  - Goal: change y_i to 1 with probability $P(y_i = 1|y_{j\neq i})$
- Assume y and y' only differ at position i and $y_i' = -1$
  - $\log P(y) = -E(y) + C$
  - $E(y) = -\sum_{i<j} w_{ij} y_i y_j - b_i y_i$
  - $\log P(y) - \log P(y') = E(y') - E(y) = -\sum_j w_{ij} y_j - 2b_i$

$$\log \frac{P(y)}{P(y')} = \log \frac{P(y_i = 1|y_{j\neq i})P(y_{j\neq i})}{P(y_i' = -1|y_{j\neq i}')P(y_{j\neq i}')} = \log \frac{P(y_i = 1|y_{j\neq i})}{1 - P(y_i = 1|y_{j\neq i})} = -\sum_j w_{ij} y_j - 2b_i$$

Stochastic Hopfield Network



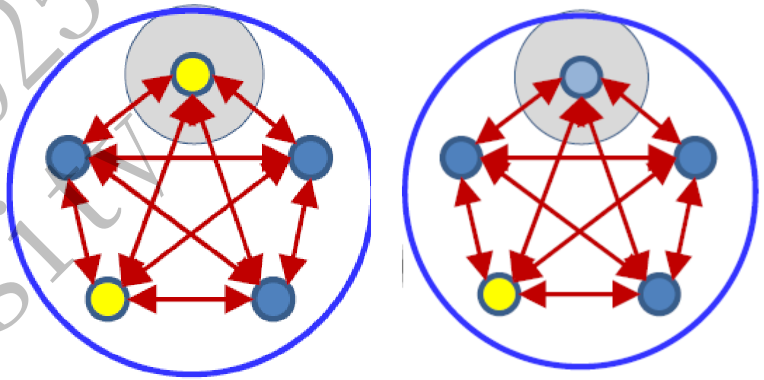
- Let's consider the "flip" operation
  - Deterministic → probabilistic
  - Goal: change y_i to 1 with probability $P(y_i = 1|y_{j\neq i})$
- Assume y and y' only differ at position i and $y_i' = -1$
  - $\log P(y) = -E(y) + C$
  - $E(y) = -\sum_{i<j} w_{ij} y_i y_j - b_i y_i$
  - $\log P(y) - \log P(y') = E(y') - E(y) = -\sum_j w_{ij} y_j - 2b_i$

$$\log \frac{P(y)}{P(y')} = \log \frac{P(y_i = 1|y_{j\neq i})P(y_{j\neq i})}{P(y_i' = -1|y_{j\neq i}')P(y_{j\neq i}')} = \log \frac{P(y_i = 1|y_{j\neq i})}{1 - P(y_i = 1|y_{j\neq i})} = -\sum_j w_{ij} y_j - 2b_i$$

- A sigmoid conditional: $P(y_i = 1|y_{j\neq i}) = \frac{1}{1+\exp\left(-\sum_j w_{ij} y_j - 2b_i\right)}$

This is also called Gibbs sampling (remember the name for now ☺)

Stochastic Hopfield Network

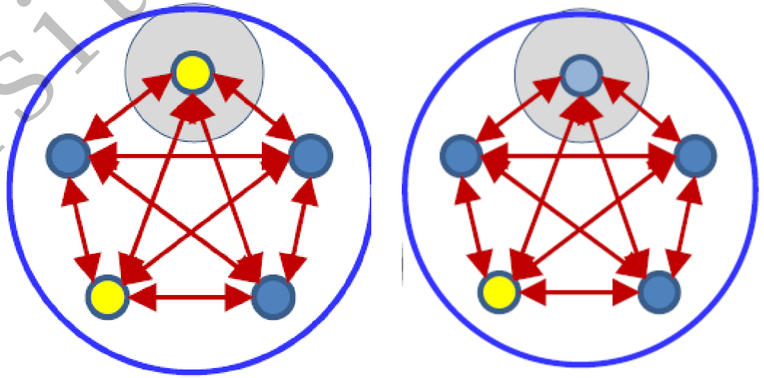


- The whole update rule
  - Field at y_i: $z_i = \sum_j w_{ij} y_j + 2b_i$
  - $P\left(y_i = 1 \middle| y_{j \neq i}\right) = \frac{1}{1+\exp(-z_i)} = \sigma(z_i)$

Field quantifies the delta energy of flip

- Evolving the network
  - Randomly initialize y
  - Cycle over y_i, fixed other variables fixed and sample y_i according to the conditional probability
  - After "convergence", we can get samples of y according to $P(y)$
  - *This sampling procedure is called Gibbs sampling*
  - **How can we retrieve a stored pattern???**
    - **This is a stochastic process!**

Stochastic Hopfield Network

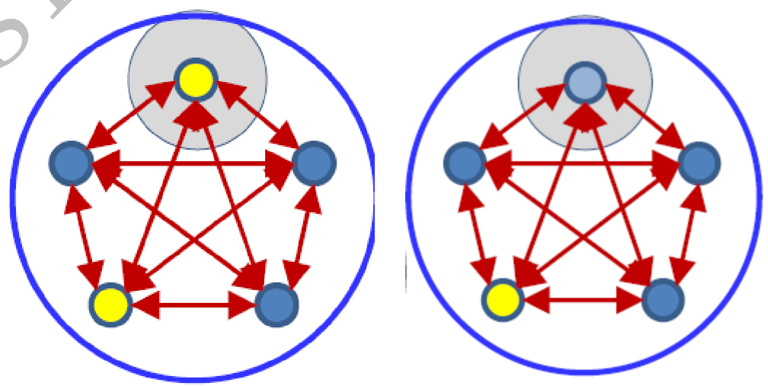


- Network evolution
  - initialize y_0
  - For $1 \le i \le N, y_i(t+1) \sim Bernoulli(\sigma(z_i(t)))$
  - Until convergence
- Retrieve a stored (low energy / high probability) pattern y
  - Given sequence of samples $y_0, \ldots, y_L$
  - Simply take the average of final M samples

$$y_i = I\left[\frac{1}{M}\sum_{t=L-M+1}^{L} y_i(t) > 0\right]$$

    - If you want a probability instead of a single vector, you can use the frequency derived from $\{y_{L-M+1}, \ldots, y_L\}$ to approximate the stationary distribution
- **In many applications, we simply take $M = 1$ (output y_L)**

Stochastic Hopfield Network: Annealing

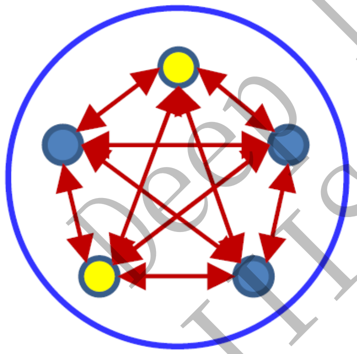


- Find the state with lowest energy?
- Network evolution with temperature annealing
  - initialize $y_0, T \leftarrow T_{\max}$
  - Repeat
    - Repeat a few cycles
      - For $1 \le i \le N$, $y_i(T) \sim Bernoulli\left(\sigma\left(\frac{1}{T} z_i(T)\right)\right)$
    - $y_i(\alpha T) \leftarrow y_i(T); T \leftarrow \alpha T$
  - Until convergence
- Final state as the retrieved pattern
  - With temperature annealing, the system will converge to the most likely state
  - Possibly local minimum in practice

Boltzmann Machine

- A generative Model (simplified)
  - $E(y) = -\frac{1}{2} y^T W y$
  - $P(y) = \frac{1}{Z} \exp\left(-\frac{E(y)}{T}\right)$
  - Parameter W
- It has a probability for producing any binary pattern y
  - We assume $y_i = 0$ or 1 (or ± 1)

How to learn W for desired patterns?



$$z_i = \frac{1}{T} \sum_j w_{j,i} y_j$$

$$P(y_i = 1 | y_{j \neq i}) = \frac{1}{1 + e^{-z_i}}$$

Boltzmann Machine: Training

- Goal
  - Remember a set of desired patterns $P = \{y^p\}$
  - Now we have a probability distribution $P(y)$ with parameter W
- Objective: maximum likelihood learning (assume $T = 1$)
  - Probability of a particular pattern

$$P(y) = \frac{\exp\left(\frac{1}{2} y^T W y\right)}{\sum_{y'} \exp\left(\frac{1}{2} y'^T W y'\right)}$$

  - Maximize log-likelihood

$$L(W) = \frac{1}{N_P} \sum_{y \in P} \frac{1}{2} y^T W y - \log \sum_{y'} \exp\left(\frac{1}{2} y'^T W y'\right)$$

Boltzmann Machine: Training

- Maximize log-likelihood

$$L(W) = \frac{1}{N_P} \sum_{y \in P} \frac{1}{2} y^T W y - \log \sum_{y'} \exp\left(\frac{1}{2} y'^T W y'\right)$$

- Gradient Ascent $\nabla_{w_{ij}} L$

Boltzmann Machine: Training

- Maximize log-likelihood

$$L(W) = \frac{1}{N_P} \boxed{\sum_{y \in P} \frac{1}{2} y^T W y} - \log \sum_{y'} \exp\left(\frac{1}{2} y'^T W y'\right)$$

- Gradient Ascent $\nabla_{w_{ij}} L$
  - $\nabla_{w_{ij}} L = \frac{1}{N_P} \sum_{y \in P} y_i y_j$

Boltzmann Machine: Training

- Maximize log-likelihood

$$L(W) = \frac{1}{N_P} \sum_{y \in P} \frac{1}{2} y^T W y - \boxed{\log \sum_{y'} \exp\left(\frac{1}{2} y'^T W y'\right)}$$

- Gradient Ascent $\nabla_{w_{ij}} L$
  - $\nabla_{w_{ij}} L = \frac{1}{N_P} \sum_{y \in P} y_i y_j - \sum_{y'} \frac{\exp\left(\frac{1}{2} y'^T W y'\right)}{Z} \cdot y_i' y_j'$

Exponentially many terms!

Boltzmann Machine: Training

- Maximize log-likelihood

$$L(W) = \frac{1}{N_P}\sum_{y\in P}\frac{1}{2}y^T W y - \boxed{\log\sum_{y'}\exp\left(\frac{1}{2}y'^T W y'\right)}$$

- Gradient Ascent $\nabla_{w_{ij}} L$
  - $\nabla_{w_{ij}} L = \frac{1}{N_P}\sum_{y\in P} y_i y_j - \sum_{y'}\frac{\exp\left(\frac{1}{2}y'^T W y'\right)}{Z}\cdot y_i' y_j'$
  - $\nabla_{w_{ij}} L = \frac{1}{N_P}\sum_{y\in P} y_i y_j - \mathrm{E}_{y'}\left[y_i' y_j'\right]$ **Monte-Carlo Approximation**
  - Draw a set of samples S for y' according to the probability,
  - $\nabla_{w_{ij}} L = \frac{1}{N_P}\sum_{y\in P} y_i y_j - \frac{1}{|S|}\sum_{y'\in S} y_i' y_j'$

Boltzmann Machine: Training

- Maximize log-likelihood with M Monte-Carlo samples

$$\nabla_{w_{ij}} L(W) = \frac{1}{N_P} \sum_{y \in P} y_i y_j - \frac{1}{M} \sum_{y' \in S} y'_i y'_j$$

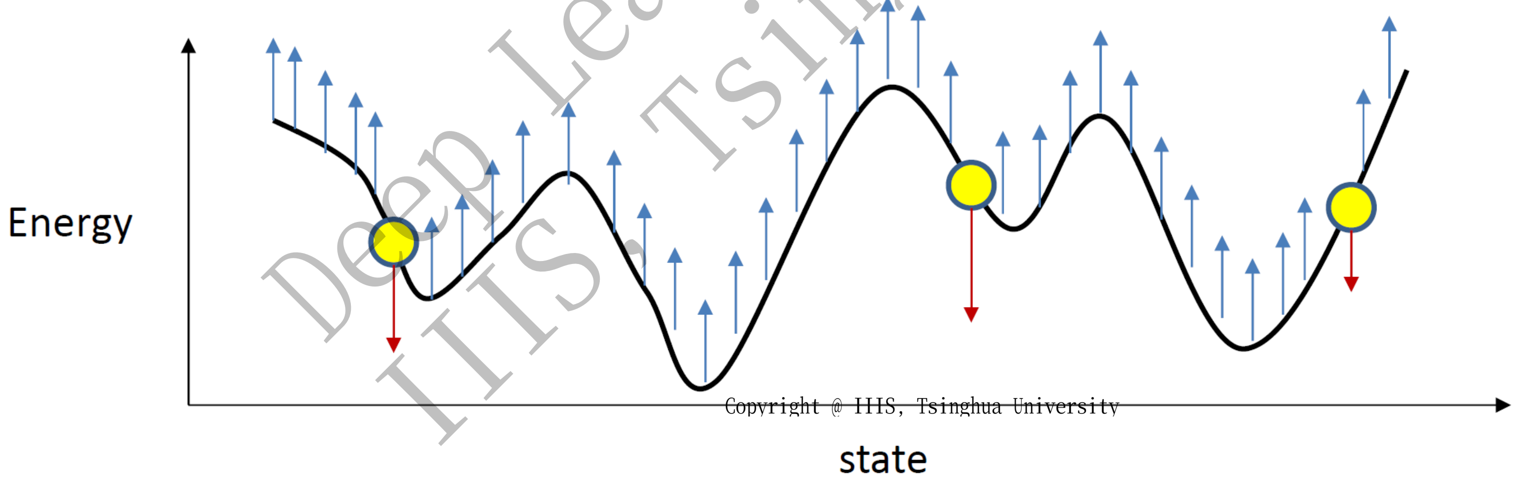
- How to draw samples from $P(y)$?
  - Running the stochastic network (Gibbs sampling)
    - Randomly initialize $y(0)$
    - Cycle over $y_i(t)$, sampling according to $P(y_i(t) | y_{j \neq i}(t))$
    - After convergence, we get a sequence of samples $\{y(0), \dots, y(L)\}$
    - Get the final M states as samples $S = \{y(L - M + 1), \dots, y(L)\}$

Boltzmann Machine: Training

- Overall Training
  - Initialize W
  - Maximize log-likelihood with M Monte-Carlo samples

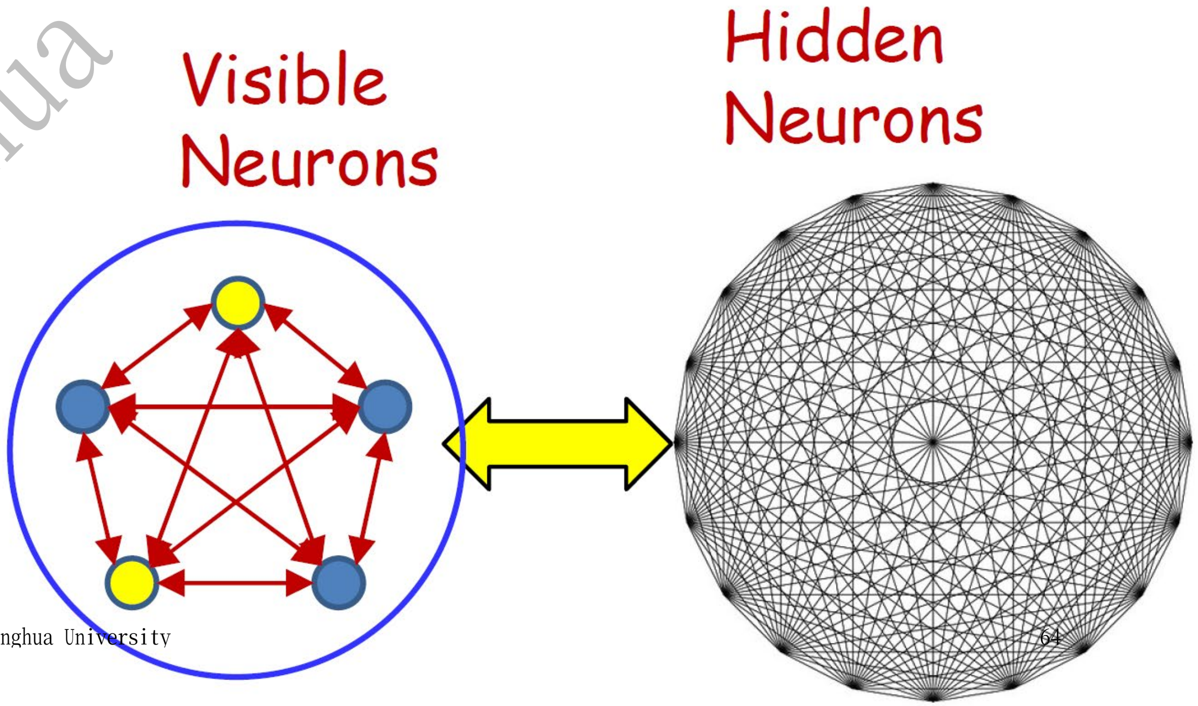
$$\nabla_{w_{ij}} L(W) = \frac{1}{N_P}\sum_{y\in P} y_i y_j - \frac{1}{M}\sum_{y'\in S} y_i' y_j'$$

  - $w_{ij} \leftarrow w_{ij} + \eta \nabla_{w_{ij}} L(W)$ (*we are maximizing likelihood*)



Boltzmann Machine with Hidden Neurons

- Let's get back to hidden neurons!
  - v visible neurons (visible patterns), h hidden neurons (latent variables)
  - $y = (v, h)$
- A joint probability distribution
  - $P(y) = P(v, h)$
  - $P(v) = \sum_h P(v, h)$
    - We only care about patterns
    - **The marginal distribution!**
- New objective
  - Maximize the marginal probability



Boltzmann Machine with Hidden Neurons

- Maximum log-likelihood learning

$$P(v) = \sum_h P(v,h) = \sum_h \frac{\exp(y^T W y)}{\sum_{y'} \exp(y'^T W y')}$$

$$L(W) = \frac{1}{|P|} \sum_{v \in P} \log\left(\sum_h \exp(y^T W y)\right) - \log\left(\sum_{y'} \exp(y'^T W y')\right)$$

- Gradient $\nabla L(W)$?

Boltzmann Machine with Hidden Neurons

- Maximum log-likelihood learning

$$P(v) = \sum_h P(v,h) = \sum_h \frac{\exp(y^T W y)}{\sum_{y'} \exp(y'^T W y')}$$

$$L(W) = \frac{1}{|P|} \sum_{v \in P} \log\left(\sum_h \exp(y^T W y)\right) - \log\left(\sum_{y'} \exp(y'^T W y')\right)$$

- Gradient $\nabla L(W)$?

Monte-Carlo Estimate!

Boltzmann Machine with Hidden Neurons

- Maximum log-likelihood learning

$$P(v) = \sum_h P(v,h) = \sum_h \frac{\exp(y^T W y)}{\sum_{y'} \exp(y'^T W y')}$$

$$L(W) = \frac{1}{|P|} \sum_{v \in P} \log\left(\sum_h \exp(y^T W y)\right) - \log\left(\sum_{y'} \exp(y'^T W y')\right)$$

- Gradient $\nabla L(W)$?
  - The first term is also in the form of log-sum
  - Monte Carlo estimates for each $v \in P$!

Boltzmann Machine with Hidden Neurons

- Maximum log-likelihood learning

$$\nabla_{w_{ij}} L(W) = \frac{1}{|P|} \sum_{v \in P} E_h[y_i y_j] - E_{y'}[y'_i y'_j]$$

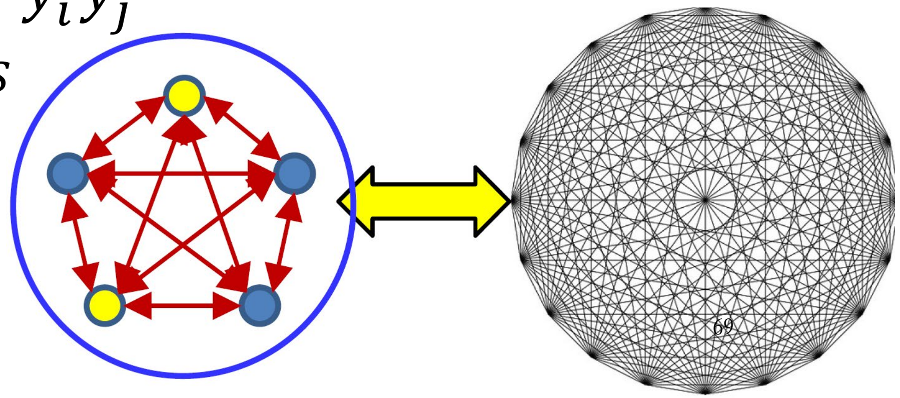
- Second term
  - Freely generate samples w.r.t. $p(y)$
  - Random initialization, cyclic Gibbs sampling
- First term
  - Generate samples w.r.t. $p(y)$ conditioned on a fixed v
  - Randomly initialize h, run Gibbs sampling over h

Boltzmann Machine with Hidden Neurons

- Overall Training
  - Initialize W
  - For $v \in P$, fixed the visible neurons, run Gibbs sampling to get K samples
    - Collect all conditioned samples as S_c
  - Randomly initialize all neurons, run Gibbs sampling to get M samples
    - Collect free samples as S
  - Maximize log-likelihood with $N_p K + M$ Monte-Carlo samples

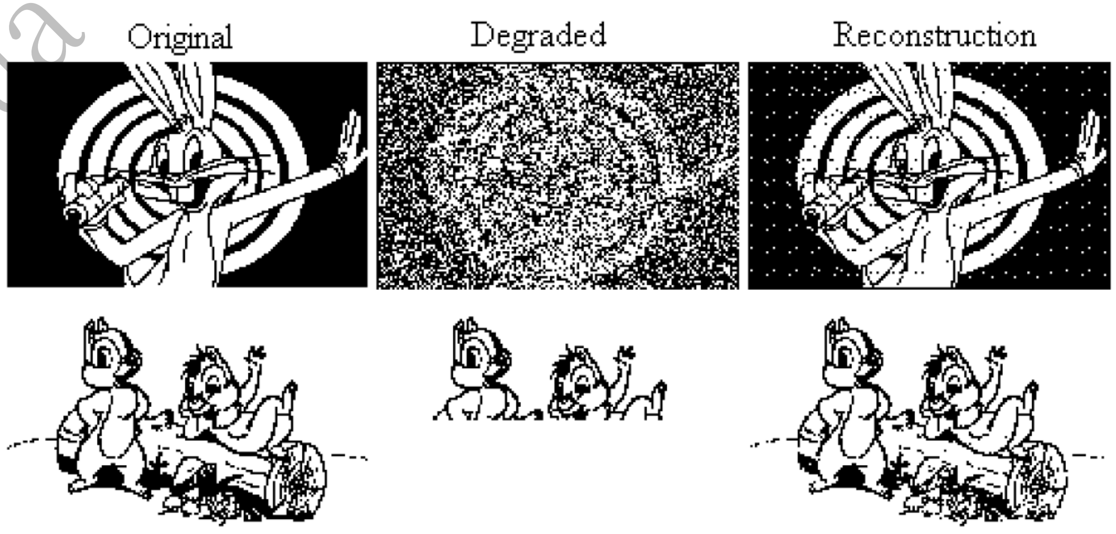
$$\nabla_{w_{ij}} L(W) = \frac{1}{N_P K} \sum_{y \in S_c} y_i y_j - \frac{1}{M} \sum_{y' \in S} y'_i y'_j$$

  - $w_{ij} \leftarrow w_{ij} + \eta \nabla_{w_{ij}} L(W)$



Boltzmann Machine

- Summary
  - A stochastic version of Hopfield Network
    - Nice mathematical properties
    - Large capacity for storing patterns (with hidden neurons)
  - Pattern generation
    - Gibbs sampling
  - Pattern completion
    - Conditioned Gibbs sampling
  - ***Classification??***
    - $y = (v, h, c)$, *c is label*
    - *c as a one-hot vector (0-1 variables)*
    - Posterior $P(c|v)$
    - Even conditional generation: $P(v|c)$!



Hopfield network reconstructing degraded images from noisy (top) or partial (bottom) cues.

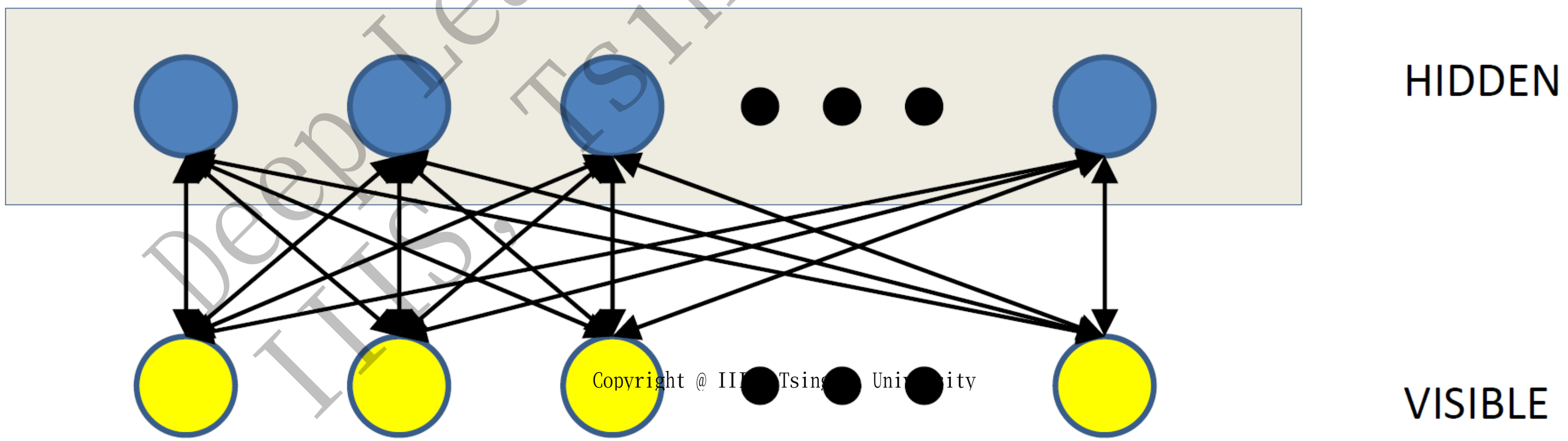
Boltzmann Machine

- The issue
  - Training is hard!
  - Gibbs sampling may take a very long time to converge
    - also called ***mixing-time***
  - Not really applicable for large problems

- Can we design a better structure for faster Gibbs sampling mixing?

Restricted Boltzmann Machine

- A particularly structured Boltzmann Machine
  - A partitioned structure
  - Hidden neurons are only connected to visible neurons
  - No intra-layer connections
  - *Invented under the name Harmonium by Paul Smolensky in 1986*
  - *Became promise after Hinton invented fast learning algorithms in mid-2000*



Restricted Boltzmann Machine

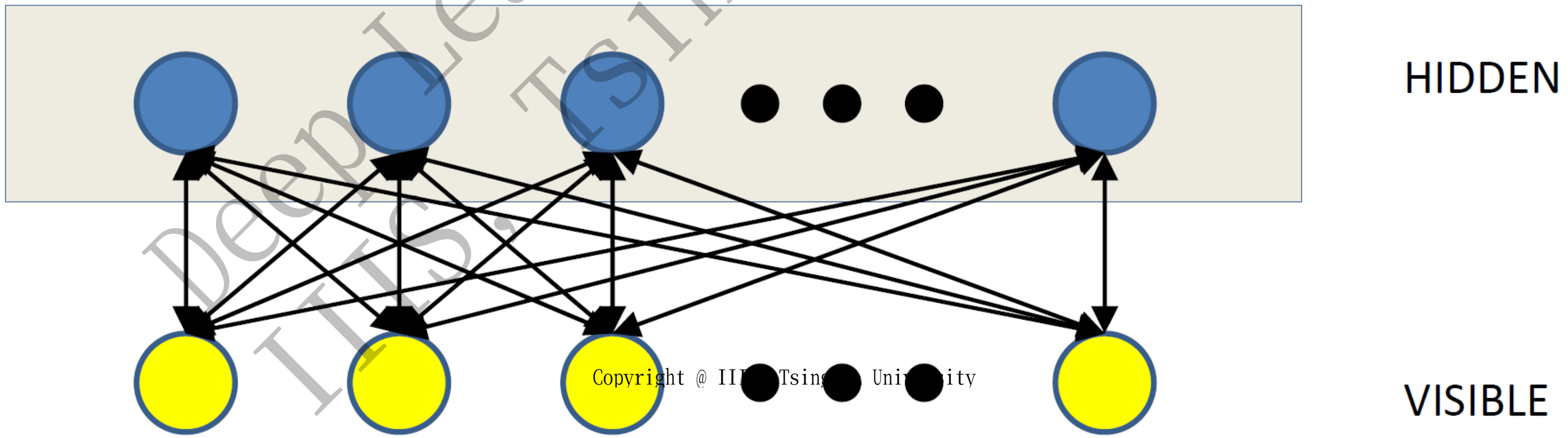
- Computation Rules: same as Boltzmann machine
  - Hidden neurons h_i

$$z_i = \sum_j w_{ij} v_j , \qquad P(h_i = 1 | v_j) = \frac{1}{1 + \exp(-z_i)}$$

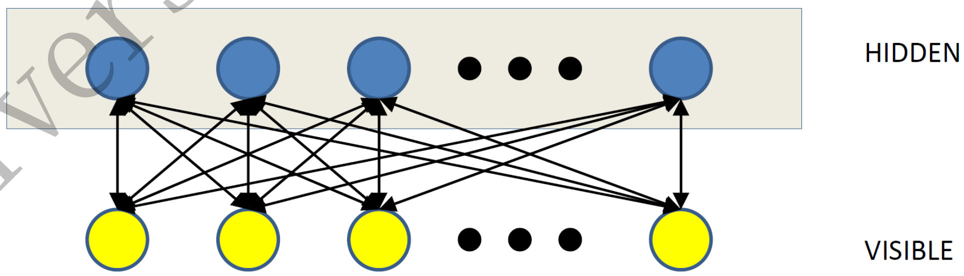
  - Visible neurons v_j

$$z_j = \sum_i w_{ij} h_i , \qquad P(v_j = 1 | h_i) = \frac{1}{1 + \exp(-z_j)}$$

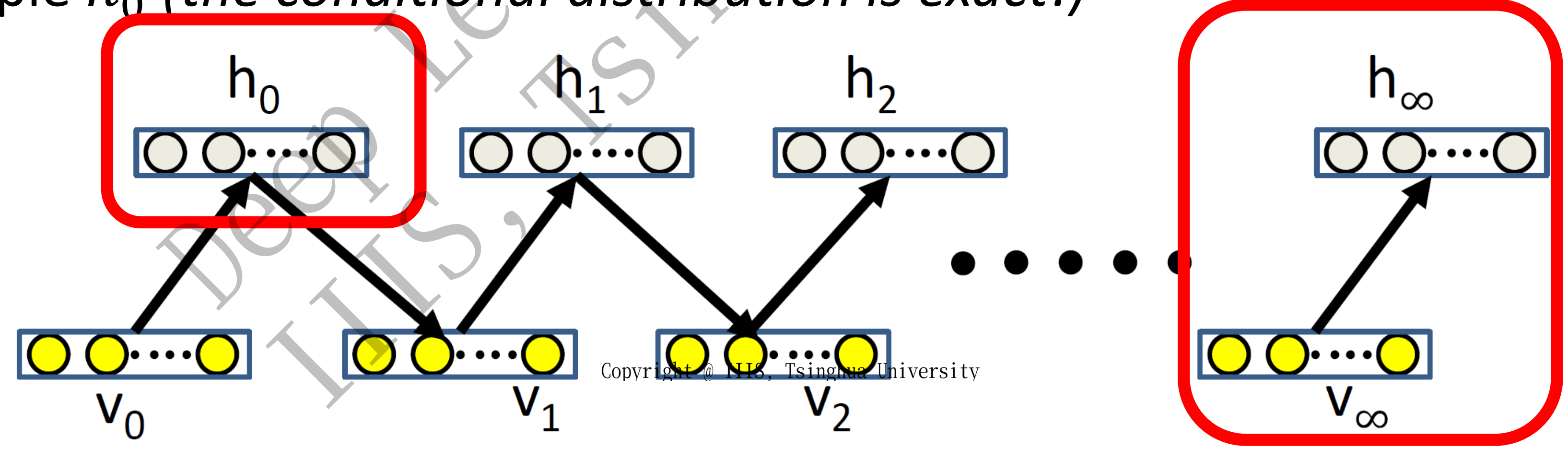
Iterative Sampling!



Restricted Boltzmann Machine



- Sampling
  - Randomly initialize visible neurons v_0
  - Iterative between hidden and visible neurons
  - Get final sample (v_∞, h_∞)
- Conditioned sampling?
  - Initialize v_0 as the desired pattern
  - Sample h_0 *(the conditional distribution is exact!)*

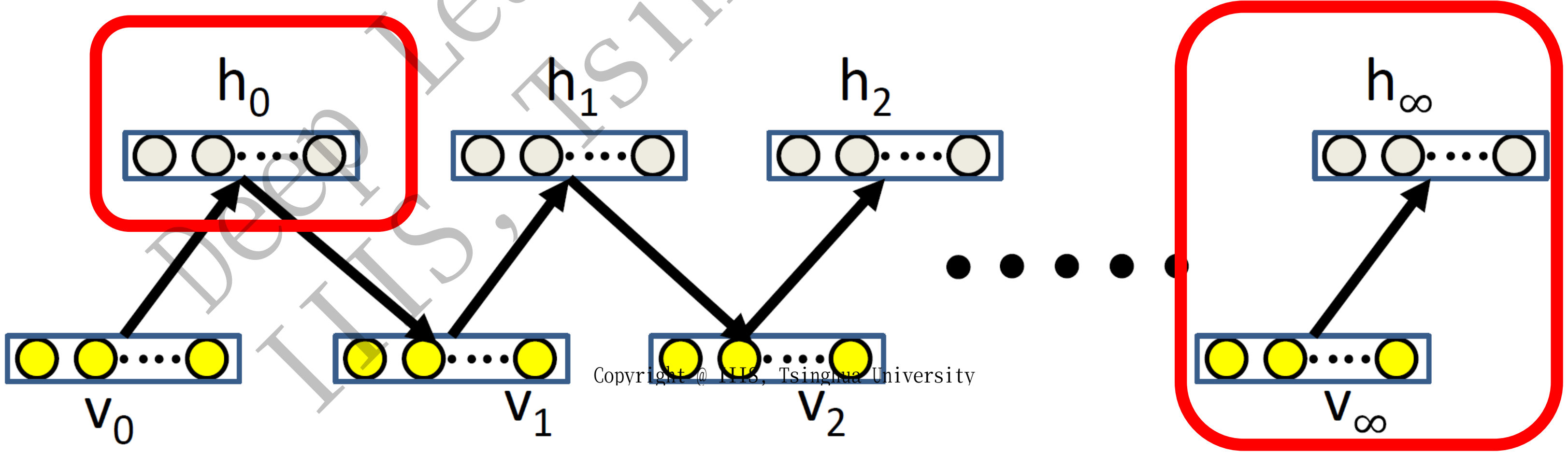


Restricted Boltzmann Machine

- Maximum Likelihood Estimate

$$\nabla_{w_{ij}} L(W) = \frac{1}{N_P K} \sum_{v \in P} v_{0_i} h_{0_j} - \frac{1}{M} \sum v_{\infty_i} h_{\infty_j}$$

  - No need to lift up the entire energy landscape … (recap)

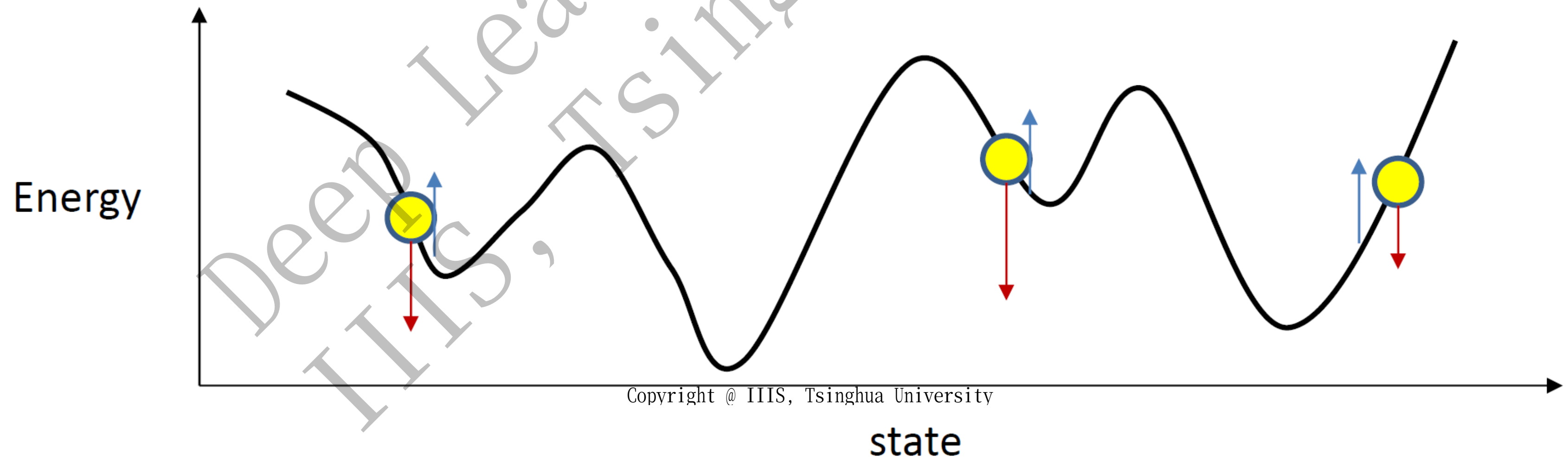


Restricted Boltzmann Machine

- Maximum Likelihood Estimate

$$\nabla_{w_{ij}} L(W) = \frac{1}{N_P K} \sum_{v \in P} v_{0_i} h_{0_j} - \frac{1}{M} \sum v_{\infty_i} h_{\infty_j}$$

  - We can starting sampling with a given v_0
    - Raising the neighborhood of the desired patterns will be sufficient

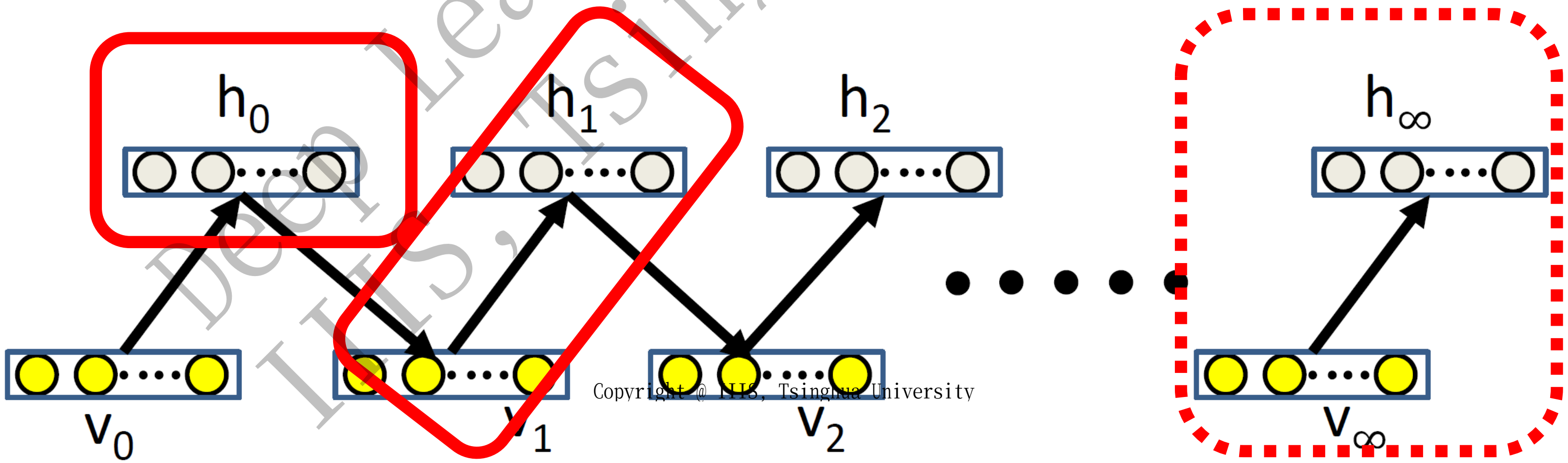


Restricted Boltzmann Machine

- Maximum Likelihood Estimate

$$\nabla_{w_{ij}} L(W) = \frac{1}{N_P K} \sum_{v \in P} v_{0_i} h_{0_j} - \frac{1}{M} \sum v_{\infty_i} h_{\infty_j}$$

  - Directly run Gibbs sampling from v_0 for 3 steps will be sufficient!



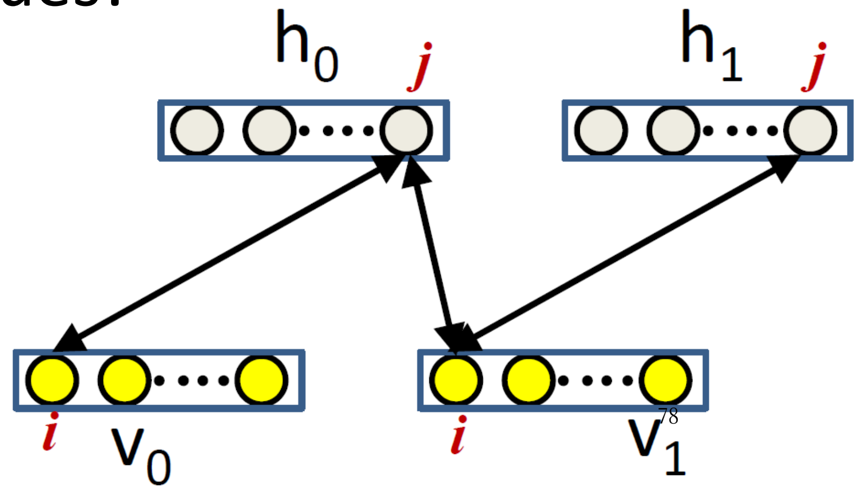
Restricted Boltzmann Machine

- Maximum Likelihood Estimate

$$\nabla_{w_{ij}} L(W) = \frac{1}{N_P} \sum_{v \in P} v_{0_i} h_{0_j} - v_{1_i} h_{1_j}$$

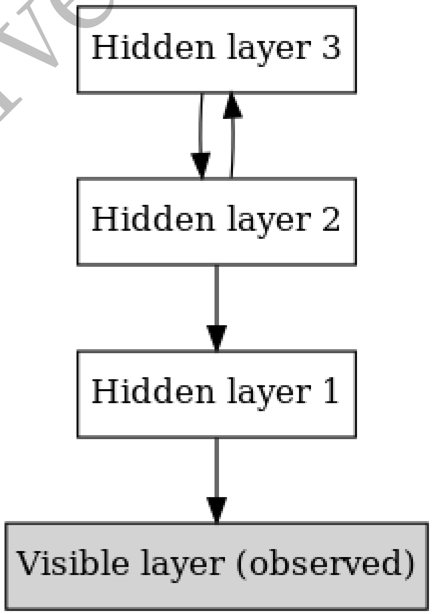
  - Only 3 Gibbs sampling steps are needed!

- We can also extend (R)BMs to to continuous values!
  - If we can explicitly sample from $P(y_i | y_{j \neq i})$
  - Exponential family! (FYI ☺)
    - “Exponential Family Harmoniums with an Application to Information Retrieval”, Welling et al., 2004

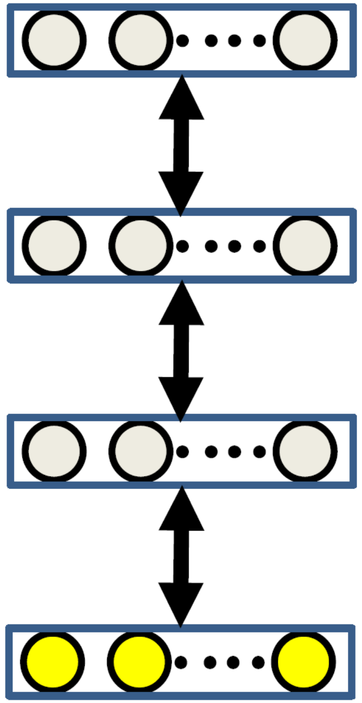


Deep Boltzmann Machine

- Can we have a ***deep*** version of RBM?
    - Deep Belief Net (2006)
    - Deep Boltzmann Machine (2009)
- Sampling?
    - Forward pass: bottom-up
    - Backward pass: top-down
    - Practical Trick: Layer-by-layer pretraining
- “Deep Boltzmann Machine”, AISTATS 2009
    - The very first deep generative model
    - Ruslan Salakhutdinov & Geoffrey Hinton



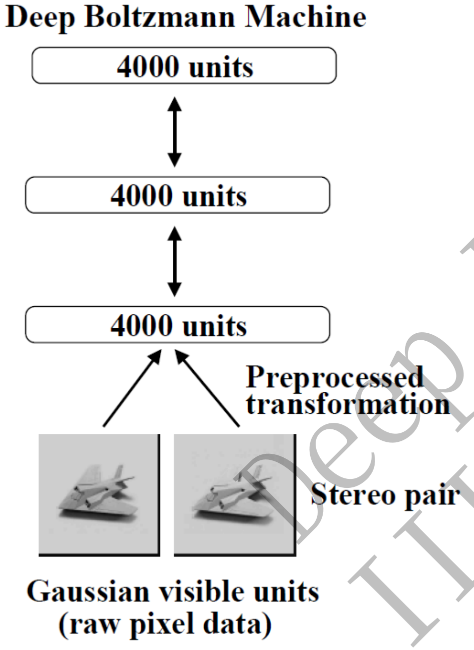
deep belief net

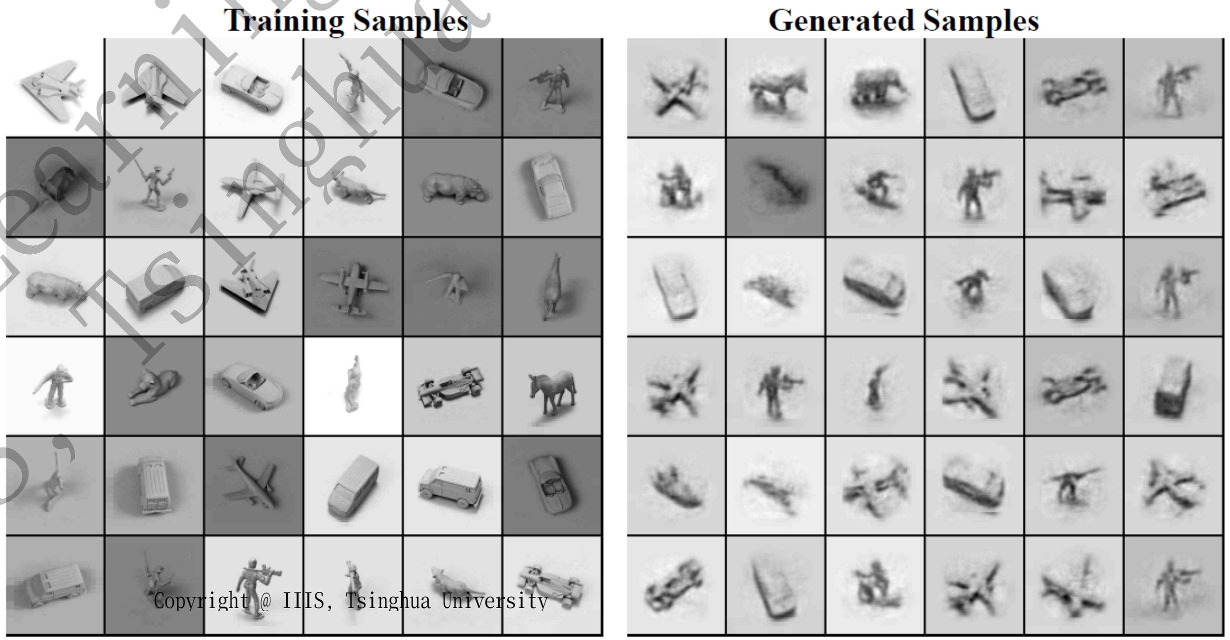


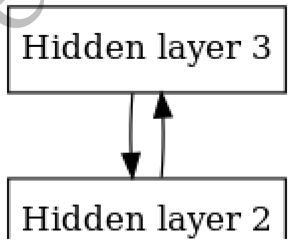
Deep Boltzmann Machine

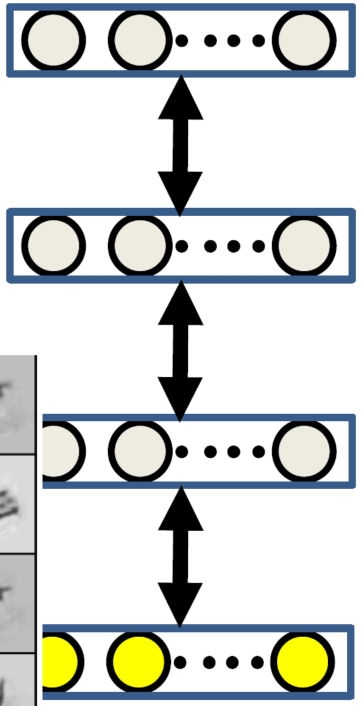
Deep Boltzmann Machine

- Can we have a ***deep*** version of RBM?
  - Deep Belief Net (2006)
  - Deep Boltzmann Machine (2009)
- Sampling?
  - F
  - B
  - P
- “Dee
  - T
  - R









Boltzmann Machine

Nobel Prize in Physics 2024

© Nobel Prize Outreach. Photo: Nanaka Adachi

John J. Hopfield

Prize share: 1/2

© Nobel Prize Outreach. Photo: Clément Morin

Geoffrey Hinton

Prize share: 1/2

Today's Lecture: Energy-Based Models

- A particularly flexible and general form of ***generative model***

- Part 1: Hopfield Network
  - The simplest model that can memorize and generate patterns

- Part 2: Boltzmann Machine
  - The first deep generative model

- Part 3: General Energy-Based Models & Sampling Methods

Energy-Based Model

- Goal of generative model
  - A probability distribution of "patterns" $P(x)$
- Requirement
  - $P(x) \geq 0$ (non-negative)
  - $\int_x P(x)dx = 1$ (sum to 1)
- Energy-Based Model
  - Energy function: $E(x;\theta)$ parameterized by θ
  - $P(x) = \frac{1}{Z}\exp(-E(x;\theta))$
  - $Z = \int_x \exp\left(-E(x;\theta)\right) dx$ *partition function*

Why use exp() function?
e.g. $|x|$ or $|x|^2$

Energy-Based Model

- A particular class of density function

$$P(x) = \frac{1}{Z}\exp(-E(x;\theta))$$

- Pros
  - Common in statistical physics
  - Compatible with log-probability measure to capture large variations
  - Exponential family (e.g., Gaussian)
  - Extremely flexible, i.e., use any $E(x)$ you like (e.g., any $f(x)\colon \mathbb{R}^d \to \mathbb{R}$, even CNNs)
- Cons
  - Non-trivial to sample and train due to the partition function Z

Energy-Based Model: Training

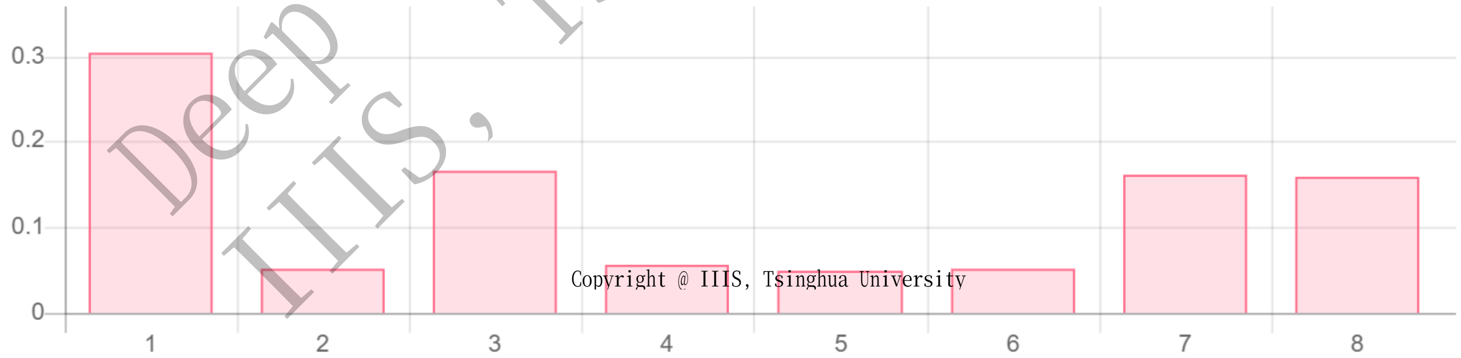
- A particular class of density function

$$P(x) = \frac{1}{Z}\exp(-E(x;\theta))$$

- Maximum Likelihood Training
  - $L(\theta) = \log P(x) = -E(x;\theta) - \log Z(\theta)$
  - Monte-Carlo estimates for partition function $Z(\theta)$
- Contrastive Divergence Algorithm
  - $\nabla_\theta L(\theta) \approx \nabla_\theta\left(-E(x_{train};\theta) + E(x_{sample};\theta)\right)$
  - Generating samples is the foundation for both training and generation!
- How to sample from an general energy-based model?
  - Or in general: sample from an arbitrary distribution $p(x)$

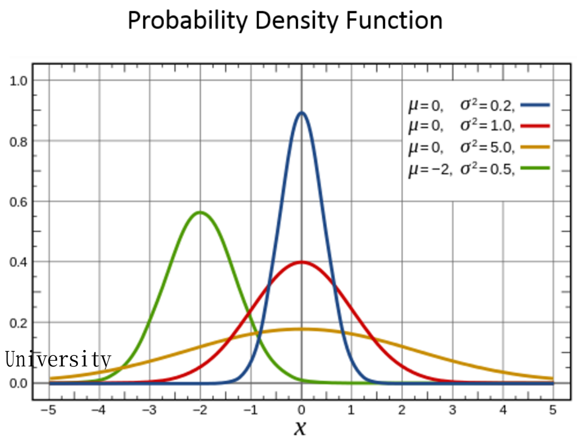
Sampling Methods

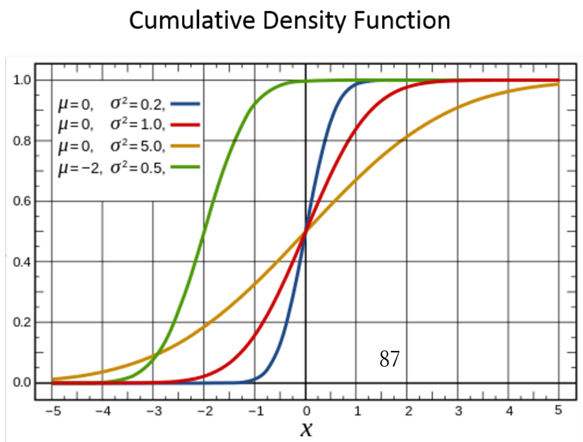
- Goal: sampling from $P(x)$
  - Assume we have a valid probability measure
  - $P(x)$ can be arbitrarily complex (e.g., high-dimensional, continuous, etc)
- Let’s start from an easy example
  - Categorical distribution?
  - Solution: uniform sampling, find the category with cumulative density
    - *The mapping from CDF to value is called Inverse distribution function (quantile function)*



Sampling Methods

- Goal: sampling from $P(x)$
  - Assume we have a valid probability measure
  - $P(x)$ can be arbitrarily complex (e.g., high-dimensional, continuous, etc)
- Let’s start from an easy example
  - Categorical distribution
  - Gaussian distribution?
    - No closed-form CDF!
    - Central-limit theorem
      - Sample $X_i \sim Beroulli(0.5)$
      - $E[X_i] = 0.5; Var[X_i] = 0.5^2$
      - $S_N = \frac{1}{N}\sum_{i=1}^{N} X_i$
      - As $N \to \infty, \sqrt{N}(S_N - 0.5) \sim N(0{,}0.5^2)$





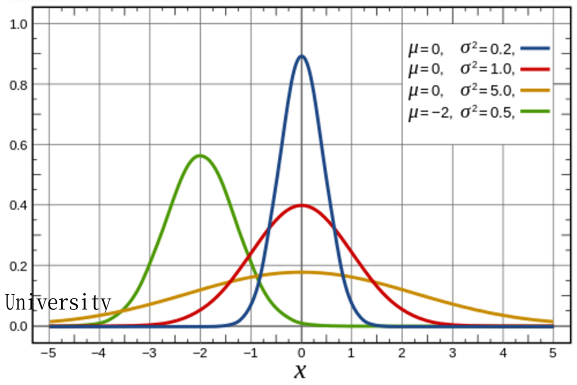
Sampling Methods

- Goal: sampling from $P(x)$
  - Assume we have a valid probability measure
  - $P(x)$ can be arbitrarily complex (e.g., high-dimensional, continuous, etc)
- Let's start from an easy example
  - Categorical distribution
  - Gaussian distribution?
    - No closed-form CDF!
    - Central-limit theorem
    - Box–Muller transform
      - Most practical method (FYI)
      - Uniform → Normal
      - Polar form transformation

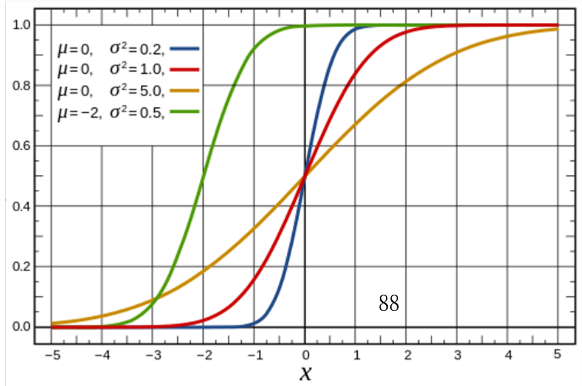
```python
def box_muller():
    # Avoid getting u == 0.0
    u1, u2 = 0.0, 0.0
    while u1 < epsilon or u2 < epsilon:
        u1 = random.random()
        u2 = random.random()

    n1 = math.sqrt(-2 * math.log(u1)) * math.cos(2 * math.pi * u2)
    n2 = math.sqrt(-2 * math.log(u1)) * math.sin(2 * math.pi * u2)
    return n1, n2
```

Probability Density Function



Cumulative Density Function



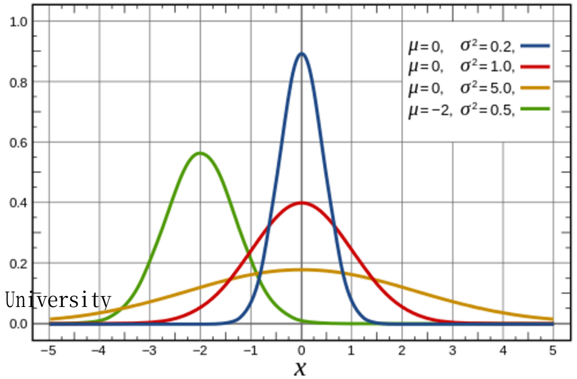
Sampling Methods

- Goal: sampling from $P(x)$
  - Assume we have a valid probability measure
  - $P(x)$ can be arbitrarily complex (e.g., high-dimensional, continuous, etc)
- Let's start from an easy example
  - Categorical distribution
  - Gaussian distribution?
    - No closed-form CDF!
    - Central-limit theorem
    - Box–Muller transform
    - General case $x \sim N(\mu, \sigma^2)$
    - High-dimensional case $x \sim N(\mu, \Sigma)$
      - $z \sim N(0, I)$
      - $x = \Sigma z + \mu$

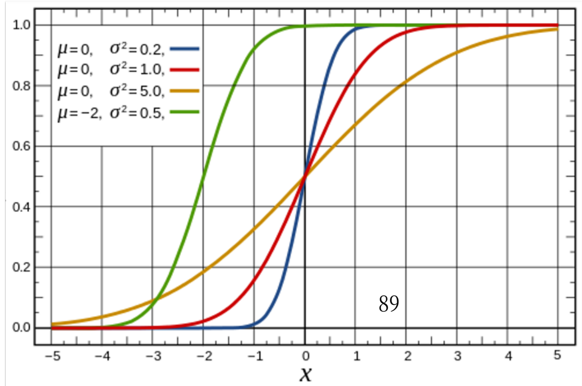
```python
def box_muller():
    # Avoid getting u == 0.0
    u1, u2 = 0.0, 0.0
    while u1 < epsilon or u2 < epsilon:
        u1 = random.random()
        u2 = random.random()

    n1 = math.sqrt(-2 * math.log(u1)) * math.cos(2 * math.pi * u2)
    n2 = math.sqrt(-2 * math.log(u1)) * math.sin(2 * math.pi * u2)
    return n1, n2
```

Probability Density Function

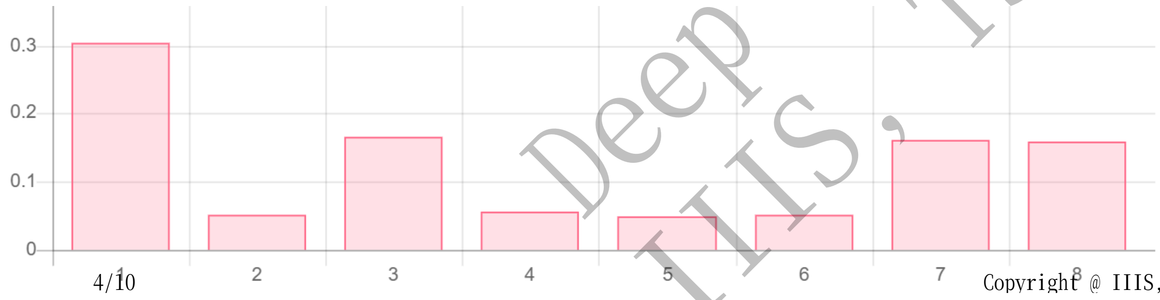


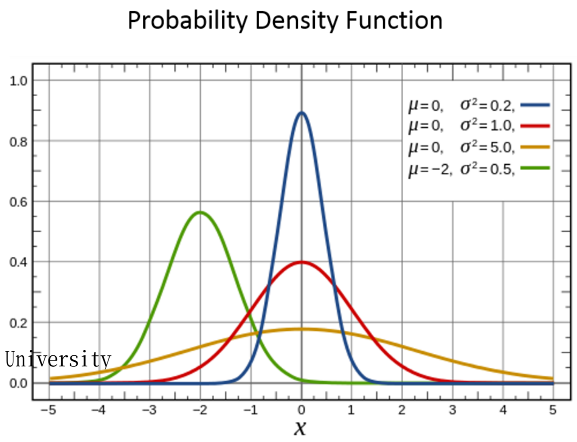
Cumulative Density Function

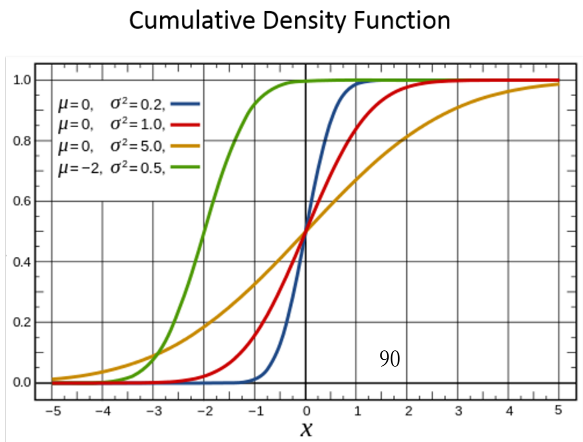


Sampling Methods

- Goal: sampling from $P(x)$
  - Assume we have a valid probability measure
  - $P(x)$ can be arbitrarily complex (e.g., high-dimensional, continuous, etc)
- Let’s start from an easy example
  - Categorical distribution
  - Gaussian distribution
    - Idea: (1) use “easy” distributions to draw sample & (2) apply mathematical transform
  - More complex distribution $p(x)$?

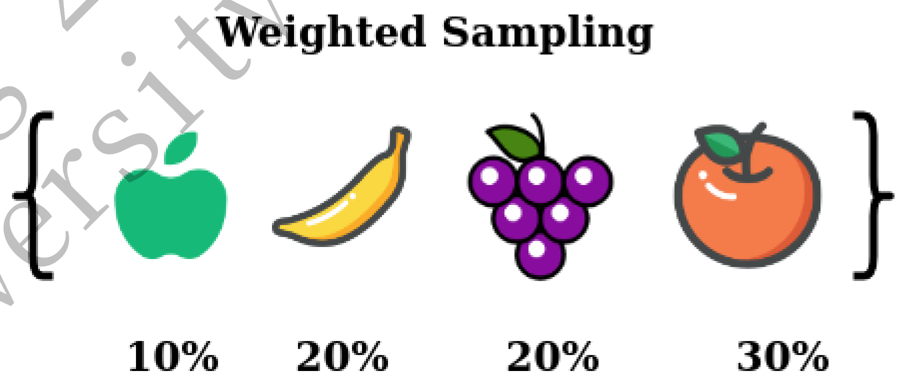


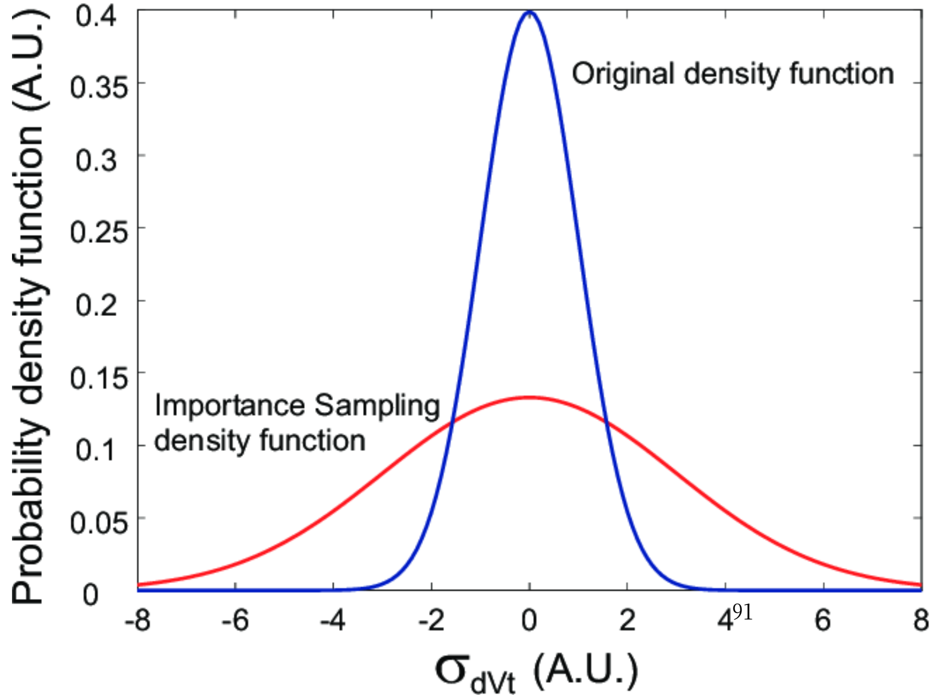




Sampling Methods

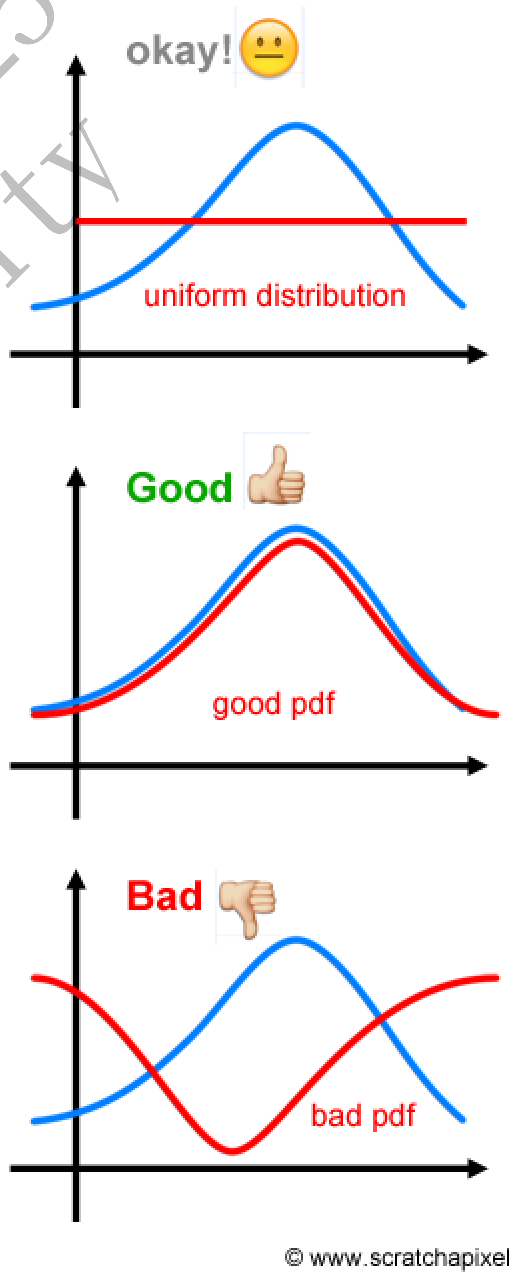
- Goal: sampling from $p(x)$
  - No CDF or nice mathematical property available
- Idea: weighted samples
  - sample from “easy” distribution $q(x)$ (e.g., uniform)
  - Use $p(x)/q(x)$ as the weight for the sample
- Importance Sampling
  - $q(x)$ proposal distribution
  - $\frac{p(x)}{q(x)}$ importance weight
  - $\mathrm{E}_{x\sim p}[f(x)] = \mathrm{E}_{x\sim q}\left[\frac{p(x)}{q(x)} f(x)\right]$





Sampling Methods

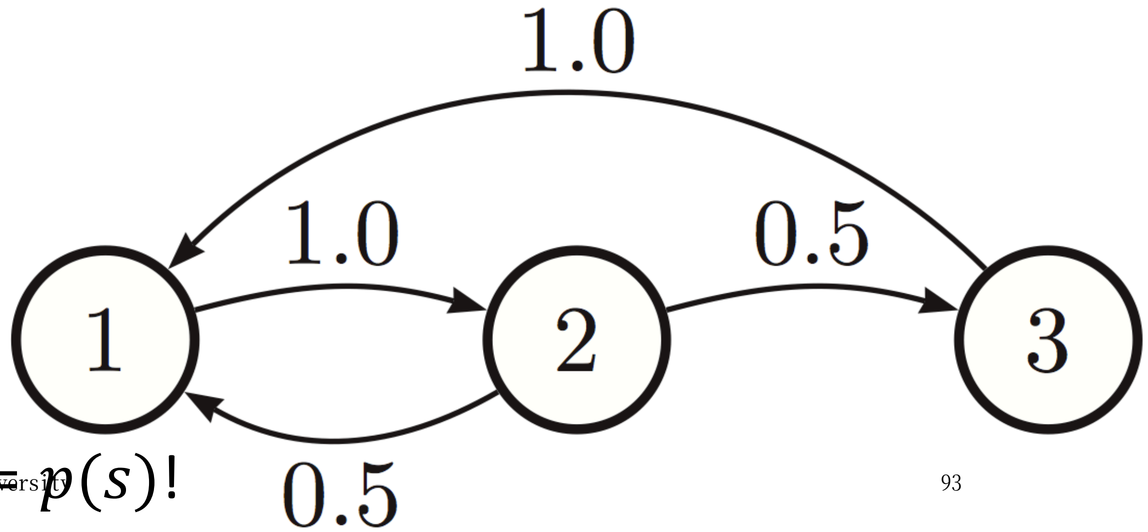
- Goal: sampling from $p(x)$
  - No CDF or nice mathematical property available
- Idea: weighted samples
  - sample from "easy" distribution $q(x)$ (e.g., uniform)
  - Use $p(x)/q(x)$ as the weight
- Importance Sampling
  - $q(x)$ proposal distribution
  - **How to choose $q(x)$???**
  - $q(x)$ needs to similar to $p(x)$
    - Your homework ☺



What if we don't have a universally good proposal?

Markov Chain Monte-Carlo

- Markov Chain
  - A state space S, a transition probability $P(s_j|s_i) = T_{ij}$
  - T is the transition matrix
  - We also use $T(s_i \to s_j)$ to denote T_{ij}
- A Markov Chain has a stationary distribution with a proper T
  - Current distribution over states π_t
  - Single step transition $\pi_{t+1} = T\pi_t$
  - Stationary distribution $\pi = T^{\infty}\pi_0$
- Sampling is easy!
- Goal: construct a Markov Chain
  - With a desired stationary distribution $\pi = p(s)$!



Markov Chain Monte-Carlo

- How to ensure π is a stationary distribution of a Markov Chain?
  - Detailed Balance (sufficient condition)

$$\pi(s)T(s \to s') = \pi(s')T(s' \to s)$$

Markov Chain Monte-Carlo

- How to ensure π is a stationary distribution of a Markov Chain?
  - Detailed Balance (sufficient condition)

$$\pi(s)T(s \to s') = \pi(s')T(s' \to s)$$

  - Design a Markov chain satisfying detailed balance for desired density $p(s)$!

Markov Chain Monte-Carlo

- How to ensure π is a stationary distribution of a Markov Chain?
  - Detailed Balance (sufficient condition)

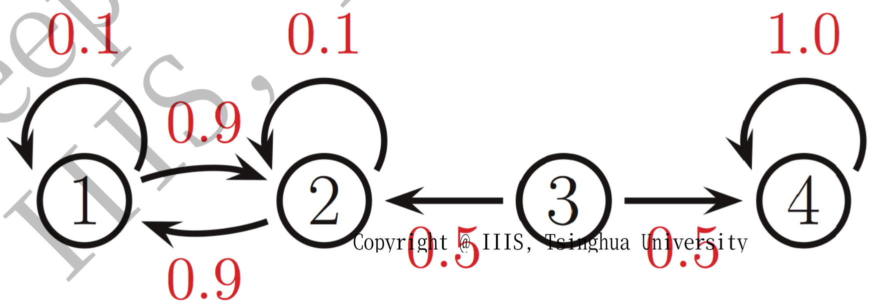
$$\pi(s)T(s \to s') = \pi(s')T(s' \to s)$$

  - Design a Markov chain satisfying detailed balance for desired density $p(s)$!
- How to ensure a unique stationary distribution exist?
  - The Markov chain is ergodic（遍历性）！

$$\min_{z} \min_{z':\pi(z')>0} \frac{T(z \to z')}{\pi(z')} = \delta > 0$$

Intuitively: you can visit any desired state with positive probability from any state

  - Examples:



$$T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Metropolis Hastings Algorithm

- Construct a valid Markov Chain $T(s' \to s)$ for distribution $p(s)$
  - Detailed balance: $p(s)T(s \to s') = p(s')T(s' \to s)$
  - Ergodicity
- Metropolis Hastings Algorithm
  - A proposal distribution $q(s'|s)$ to produce next state s' based on s
  - Draw $s' \sim q(s'|s)$
  - $\alpha = \min\left(1, \frac{p(s')q(s' \to s)}{p(s)q(s \to s')}\right)$ ($q(s \to s')$ to denotes $q(s'|s)$ for simplicity)
  - Transition to s' (accept) with probability α (acceptance ratio);
  - O.w., stays at s (reject)
- MH constructs a valid Markov chain with a proper proposal q!
  - Homework ☺

Metropolis Hastings Algorithm: Example

- Choice of $q(s \to s')$
  - Random proposal $q(s \to s') = s + \text{noise}$ (i.e., Gaussian/Uniform Noise)
- Acceptance ratio for $s \to s'$
  - $\alpha(s \to s') = \min\left(1, \frac{p(s')q(s' \to s)}{p(s)q(s \to s')}\right) = \min\left(1, \frac{p(s')}{p(s)}\right)$
- MH sampling for the energy-based model $p(s) = \frac{1}{Z}\exp(-E(s))$
  - Random initialize s^0
  - $s' \leftarrow q(s \to s')$
  - Transition to s' with probability $\alpha(s \to s')$;
  - O.w., stays at s
  - Repeat

Metropolis Hastings Algorithm: Example

- Choice of $q(s \to s')$
  - Random proposal $q(s \to s') = s + \text{noise}$ (i.e., Gaussian/Uniform Noise)
- Acceptance ratio for $s \to s'$
  - $\alpha(s \to s') = \min\left(1, \frac{p(s')q(s' \to s)}{p(s)q(s \to s')}\right) = \min\left(1, \frac{p(s')}{p(s)}\right)$
- MH sampling for the energy-based model $p(s) = \frac{1}{Z}\exp(-E(s))$
  - Random initialize s^0
  - $s' \leftarrow s + \text{noise}$
  - Transition to s' with probability $\alpha(s \to s')$;
  - O.w., stays at s
  - Repeat

Metropolis Hastings Algorithm: Example

- Choice of $q(s \to s')$
  - Random proposal $q(s \to s') = s + \text{noise}$ (i.e., Gaussian/Uniform Noise)
- Acceptance ratio for $s \to s'$
  - $\alpha(s \to s') = \min\left(1, \frac{p(s')q(s' \to s)}{p(s)q(s \to s')}\right) = \min\left(1, \frac{p(s')}{p(s)}\right)$
- MH sampling for the energy-based model $p(s) = \frac{1}{Z}\exp(-E(s))$
  - Random initialize s^0
  - $s' \leftarrow s + \text{noise}$
  - Transition to s' with probability $\min\left(1, \frac{p(s')}{p(s)}\right)$; No partition function involved!
  - O.w., stays at s
  - Repeat

Metropolis Hastings Algorithm: Example

- Choice of $q(s \to s')$
  - Random proposal $q(s \to s') = s + \text{noise}$ (i.e., Gaussian/Uniform Noise)
- Acceptance ratio for $s \to s'$
  - $\alpha(s \to s') = \min\left(1, \frac{p(s')q(s' \to s)}{p(s)q(s \to s')}\right) = \min\left(1, \frac{p(s')}{p(s)}\right)$
- MH sampling for the energy-based model $p(s) = \frac{1}{Z}\exp(-E(s))$
  - Random initialize s^0
  - For each iteration t
    - $s' \leftarrow s^t + \text{noise}$
    - If $E(s') < E(s^t)$; then accept $s^{t+1} \leftarrow s'$
    - Else accept $s^{t+1} \leftarrow s'$ with probability $\exp\left(E(s^t) - E(s')\right)$
  - Repeat

Metropolis Hastings Algorithm

- The simplest way to construct a valid Markov chain
  - Flexible, simple and general
  - **Quiz: proposal q in MH v.s. Importance Sampling**
    - A: $q(s'|s)$ v.s. $q(s)$; in MH, q generates local samples; in IS, q outputs "blind" guesses

- Issues
  - Curse of dimensionality: samples a completely new state
  - Acceptance ratio: what if acceptance rate is low?

Metropolis Hastings Algorithm

- The simplest way to construct a valid Markov chain
  - Flexible, simple and general
  - **Quiz: proposal q in MH v.s. Importance Sampling**
    - A: $q(s'|s)$ v.s. $q(s)$; in MH, q generates local samples; in IS, q outputs “blind” guesses

- Issues
  - Curse of dimensionality: samples a completely new state
  - Acceptance ratio: what if acceptance rate is low?

- Can we design a proposal distribution $q(s \rightarrow s')$ such that it always gets accepted?

Gibbs Sampling

- Gibbs sampling
  - $s = (s_0, s_1, \ldots, s_N)$, we construct a coordinate-wise $q(s_i \to s_i')$
  - $q(s_i \to s_i') = p(s_i' | s_{j \neq i})$ (conditional distribution)
- Dimensionality
  - Sample a single coordinate per step.
- Gibbs sampling always get accepted!
  - Acceptance ratio is always 1, $\alpha(s_i \to s_i') = 1$ Prove it in your homework ☺
- Assumption
  - An easy to sample conditional distribution
    - Conjugate Prior and Exponential Family (https://en.wikipedia.org/wiki/Conjugate_prior)
  - What if no closed-form posterior?
    - Learn a neural proposal to approximate the true posterior! ☺ (meta-learning MCMC proposals, Wang, Wu, et al NIPS2018)

Sampling Methods

- What we have learned so far …
  - Importance Sampling
    - Simplest solution by any proposal distribution
  - Metropolis-Hastings algorithm
    - Good local proposal → high acceptance ratio
  - Gibbs sampling
    - Posterior is easy-to-sample
    - The "default" method for machine learning among 2002~2012
- General Issues for MCMC methods
  - Slow convergence due to sampling (recap: SGD v.s. GD)
  - Can we use gradient information for MCMC?
    - Energy function is differentiable!

Stochastic Gradient MCMC

- MCMC with Langevin dynamics
  - "Bayesian learning via stochastic gradient langevin dynamics"
    - ICML 2011, Max Welling& Yee Whye The (ICML 2021 test-of-time award)
  - Given N data $X_1, \ldots, X_N$, define $p(\theta \to \theta')$ by

$$\theta' \leftarrow \theta + \frac{\epsilon_t}{2}\left(\nabla_\theta \log p(\theta) + \sum_i \nabla_\theta \log P(x_i|\theta)\right) + N(0, \epsilon_t I)$$

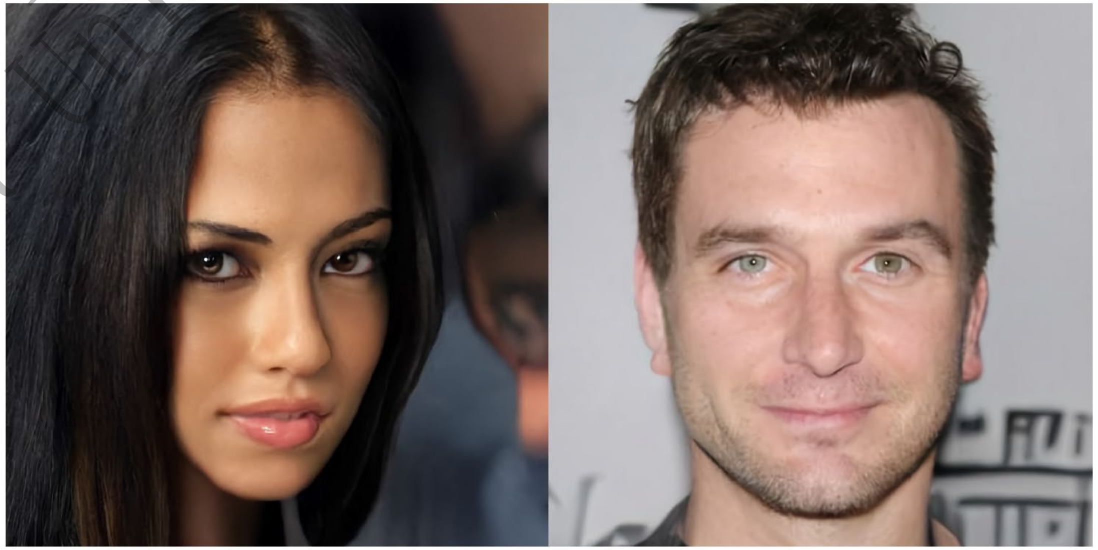
  - Condition for a valid Markov Chain
    - $\sum_t \epsilon_t = \infty$ and $\sum_t \epsilon_t^2 < \infty$
    - Interpretation
      - (stochastic) gradient descent first (∇_θ is large); MCMC around local minimum ($\nabla_\theta \approx 0$)
    - No need of MH acceptance rule
- Additional Reading:
  - Hamiltonian Monte Carlo (SGD with momentum): https://arxiv.org/pdf/1701.02434.pdf
  - https://arogozhnikov.github.io/2016/12/19/markov_chain_monte_carlo.html

Summary

- Hopfield Network
  - The first generative neural network
  - Undirected complete graph
- Boltzmann Machine
  - A probabilistic interpretation of Hopfield Network
  - The first deep generative model
- Energy-Based
  - Extremely flexible and powerful, designed to be multi-modal
  - Hard to sample and learn
  - Sampling is the core challenge!!

What's Next: Non-Sampling Methods

- Approximate Bayesian Inference
  - Variational Inference (next lecture ☺)
    - Learn an parameterized distribution to approximate the true posterior
- Design a model from which we can easily draw sample!
  - Lectures 6 & 7a
- Modern energy-based models
  - Scoring matching
  - Lecture 7b



Song et. al., 2021
OpenAI Blog: https://openai.com/blog/energy-based-models/

Thanks!