# Energy-based Models

## -- *Hopfield Network*

Hao Dong

Peking University

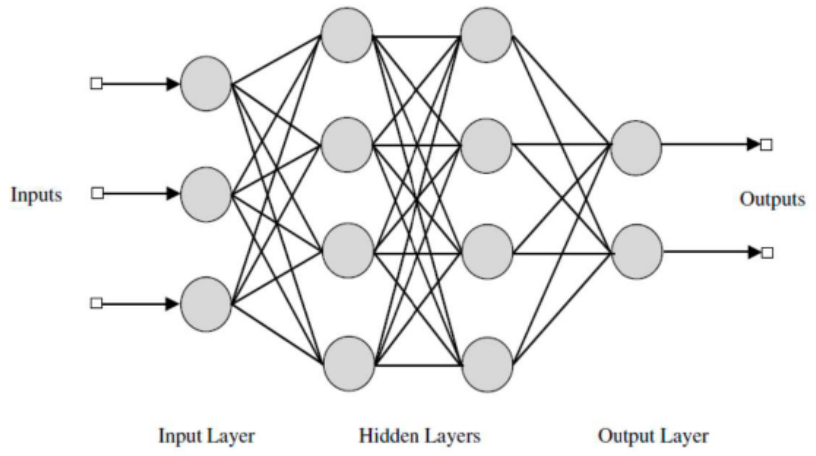
## Contents

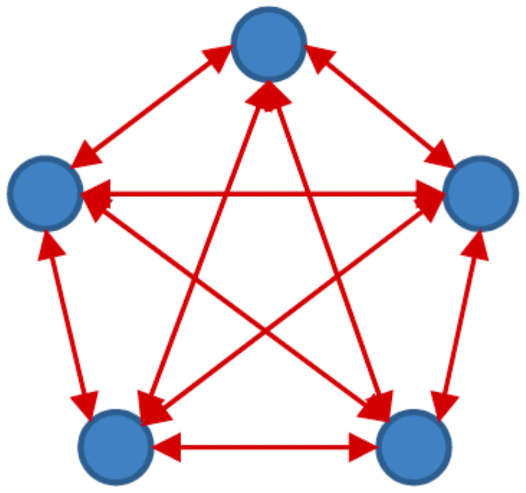
- **Discrete Hopfield Neural Networks**
  - Introduction
  - How to use
  - How to train
  - Thinking
- **Continuous Hopfield Neural Networks**

- **Before**

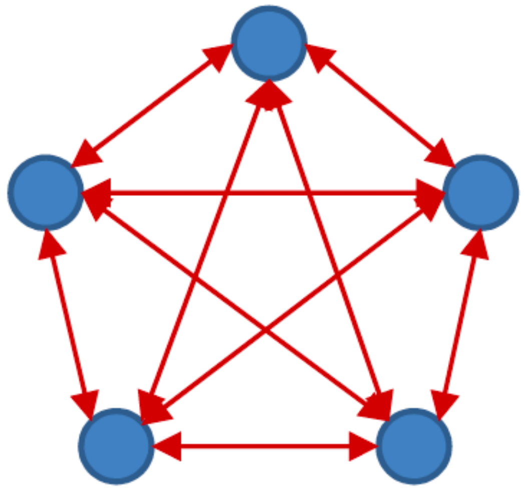
  All feed forward structures

- What about …





## Consider this network

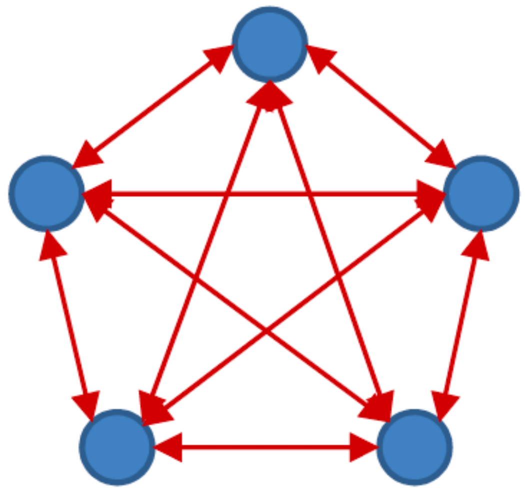


$$f(x) = \begin{cases} +1\ if\ x \geq 0 \\ -1\ if\ x < 0 \end{cases}$$

$$y_i = f(\sum_{j \neq i} w_{ji}\, y_j + b_i)$$

- The output of each neuron is +1/-1
- Every neuron *receives* input from every other neuron
- Every neuron *outputs* signals to every other neuron
- The weight is symmetric: $w_{ij} = w_{ji}$ assume $w_{ii} = 0$

## Hopfield Net



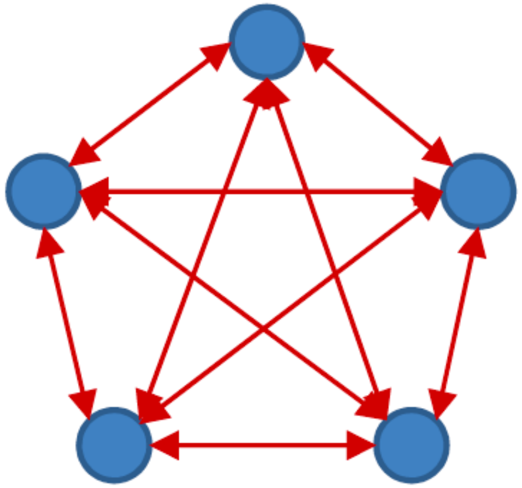
$$f(x) = \begin{cases} +1\ if\ x \geq 0 \\ -1\ if\ x < 0 \end{cases}$$

$$y_i = f\Big(\sum_{j \neq i} w_{ji}\, y_j + b_i\Big)$$

- At each time, each neuron receives a “field”: $\sum_{j \neq i} w_{ji}\, y_j + b_i$
  - If the sign of the field matches its own sign, nothing happens;
  - If the sign of the field opposes its own sign, it “flips” to match the sign of the field.

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

## Hopfield Net



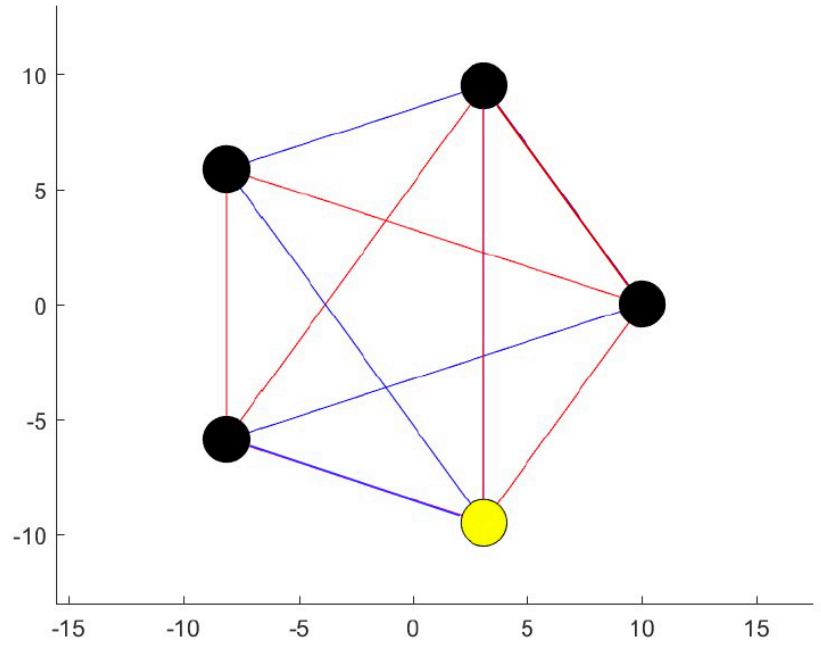
$$f(x) = \begin{cases} +1 \; if \; x \geq 0 \\ -1 \; if \; x < 0 \end{cases}$$

$$y_i = f(\sum_{j \neq i} w_{ji} \, y_j + b_i)$$

- If the sign of the field opposes its own sign, it "flips" to match the sign of the field.
- "Flips" of a neuron may cause other neurons to "flip"!

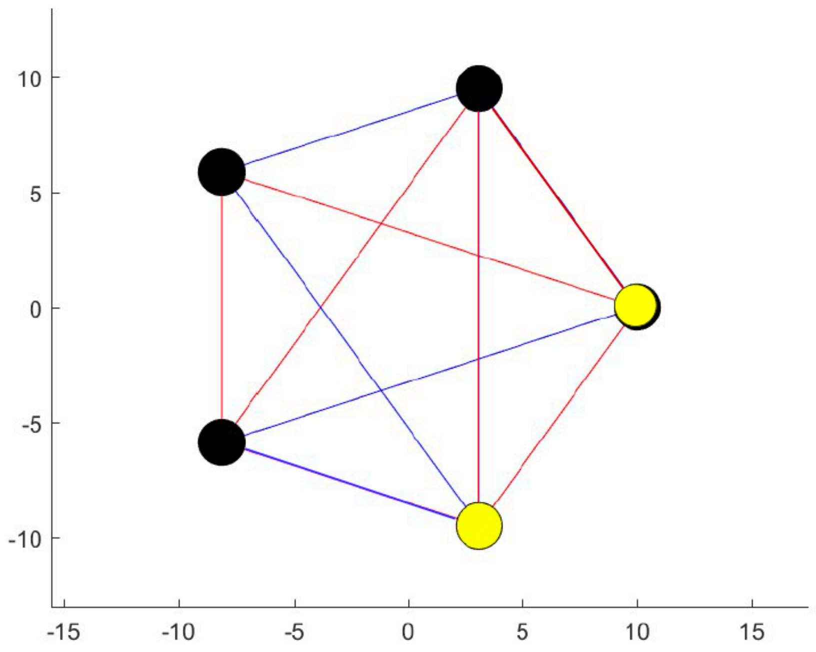
$$y_i \rightarrow -y_i, if \; y_i \left( \sum_{j \neq \mathrm{i}} w_{ji} y_j + b_i \right) < 0$$

## Example



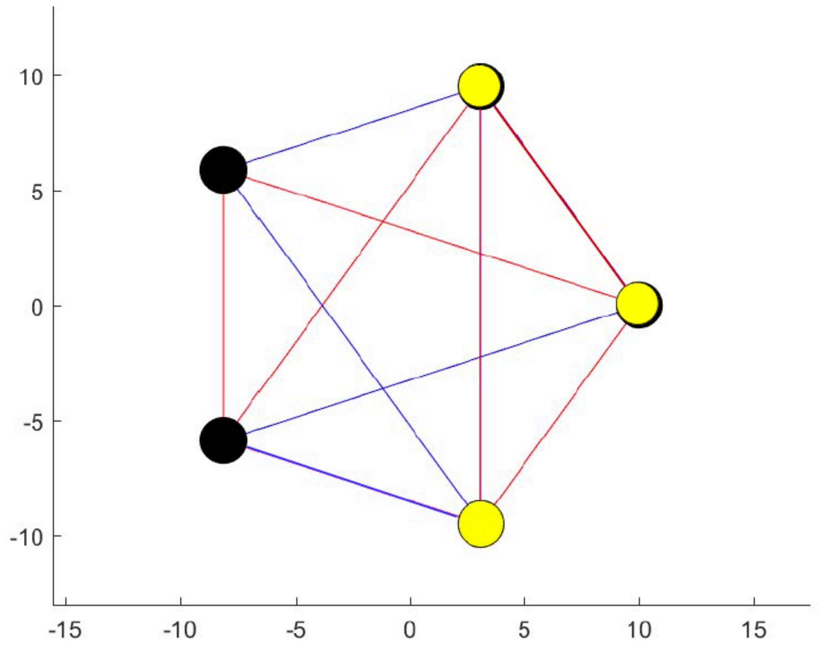
- **Red edges are +1, blue edges are -1**
- **Yellow nodes are -1, black nodes are +1**

## Example



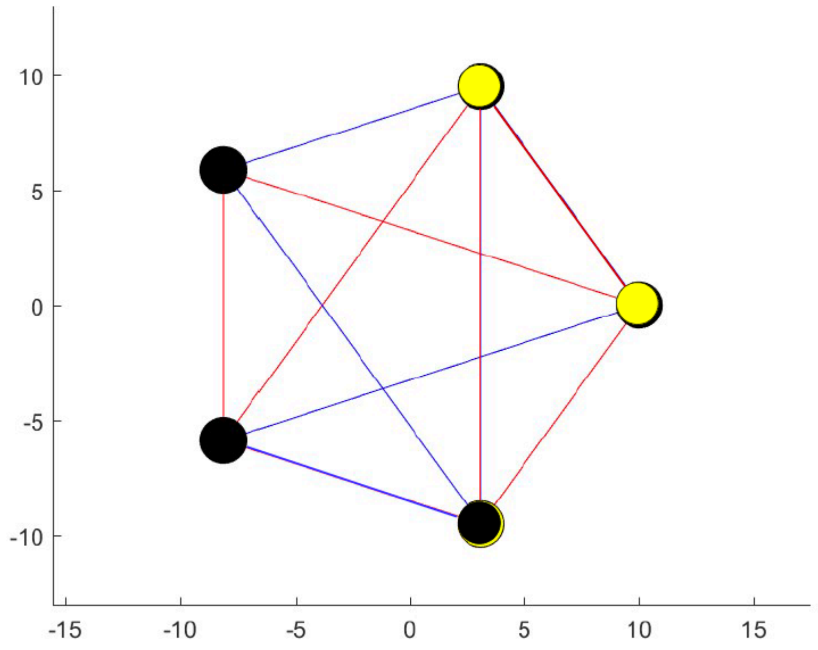
- **Red edges are +1, blue edges are -1**
- **Yellow nodes are -1, black nodes are +1**

## Example



- **Red edges are +1, blue edges are -1**
- **Yellow nodes are -1, black nodes are +1**

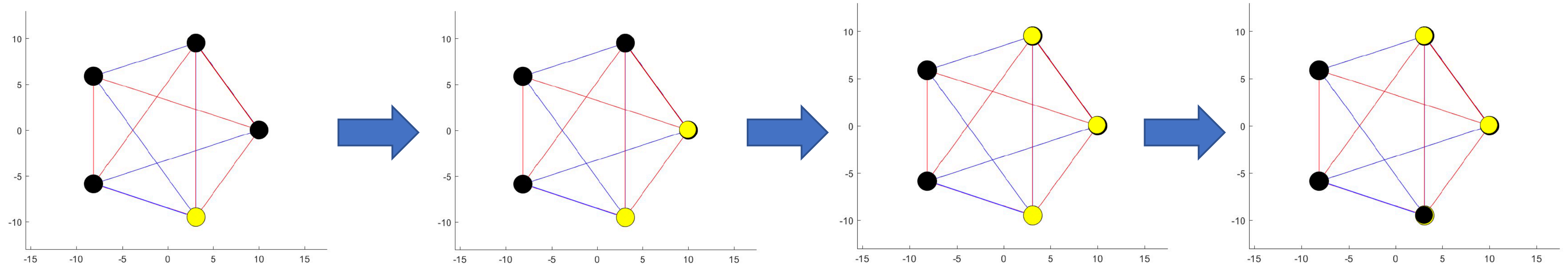
## Example



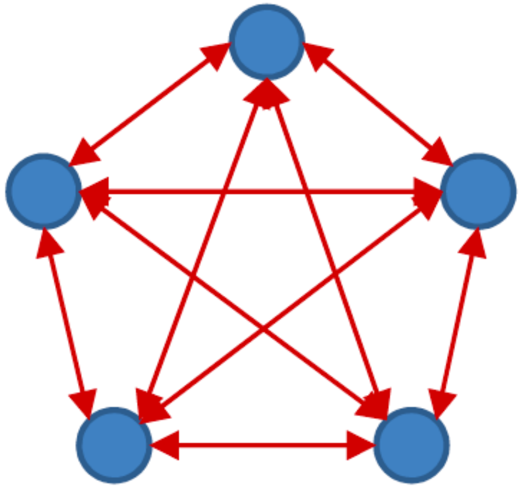
- **Red edges are +1, blue edges are -1**
- **Yellow nodes are -1, black nodes are +1**

## Hopfield Net

- If the sign of the field opposes its own sign, it “flips” to match the field
  - Which will change the field at other nodes
    - Which may then flip
      - Which may cause other neurons to flip
        - And so on…
- Will this continue forever?



## Hopfield Net



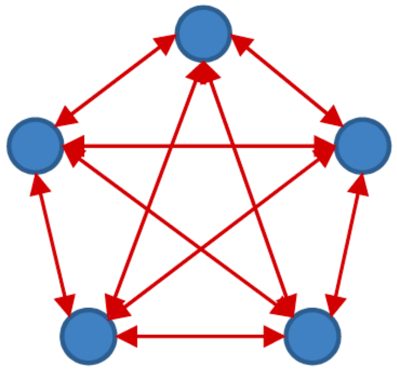
$$f(x) = \begin{cases} +1 \; if \; x \geq 0 \\ -1 \; if \; x < 0 \end{cases}$$

$$y_i = f(\sum_{j \neq i} w_{ji} \, y_j + b_i)$$

- Let $y_i^0$ be the output of the i-th neuron before it responds to the current field
- Let $y_i^1$ be the output of the i-th neuron before it responds to the current field

$$y_i \rightarrow -y_i, if \; y_i \left( \sum_{j \neq \mathrm{i}} w_{ji} y_j + b_i \right) < 0$$

## Hopfield Net



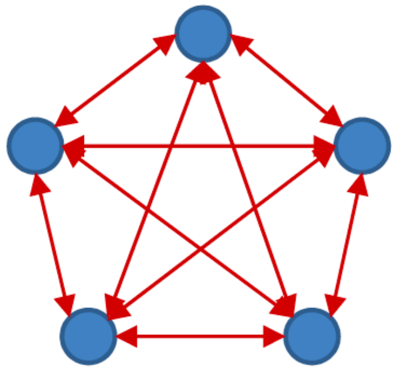
$$f(x) = \begin{cases} +1 \; if \; x \geq 0 \\ -1 \; if \; x < 0 \end{cases}$$

$$y_i = f(\sum_{j \neq i} w_{ji} \, y_j + b_i)$$

- If $y_i^0 = f(\sum_{j \neq i} w_{ji} \, y_j + b_i)$, then $y_i^1 = y_i^0$
  - No "flip" happens

$$y_i^1 \left( \sum_{j \neq i} w_{ji} \, y_j + b_i \right) - y_i^0 \left( \sum_{j \neq i} w_{ji} \, y_j + b_i \right) = 0$$

## Hopfield Net



$$f(x) = \begin{cases} +1 \; if \; x \geq 0 \\ -1 \; if \; x < 0 \end{cases}$$

$$y_i = f(\sum_{j \neq i} w_{ji} \, y_j + b_i)$$

- If $y_i^0 \neq f(\sum_{j \neq i} w_{ji} \, y_j + b_i)$, then $y_i^1 = -y_i^0$
  - ”Flip” happens

$$y_i^1\left(\sum_{j\neq i} w_{ji} \, y_j + b_i\right) - y_i^0\left(\sum_{j\neq i} w_{ji} \, y_j + b_i\right) = 2y_i^1\left(\sum_{j\neq i} w_{ji} \, y_j + b_i\right) > 0$$

- Every ”flip” is guaranteed to locally increase

## Globally

- Consider the following sum across all nodes:
  - $E(y_1, y_2, \dots, y_N) = -\sum_i y_i(\sum_{j\neq i} w_{ji} y_j + b_i)$
    $= -\sum_{i,j\neq i} w_{ij}\, y_i y_j - \sum_i b_i y_i$
  - Assume $w_{ii} = 0$

- For a neuron k that ”flips”:
  - $\Delta E(y_k) = E\left(y_1, \dots, y_k^1, \dots, y_N\right) - E\left(y_1, \dots, y_k^0, \dots, y_N\right)$
    $= -\left(y_k^1 - y_k^0\right)\left(\sum_{j\neq k} w_{jk}\, y_j + b_k\right)$
  - Always <0!
  - Every ”flip” results in a decrease in E

## Globally

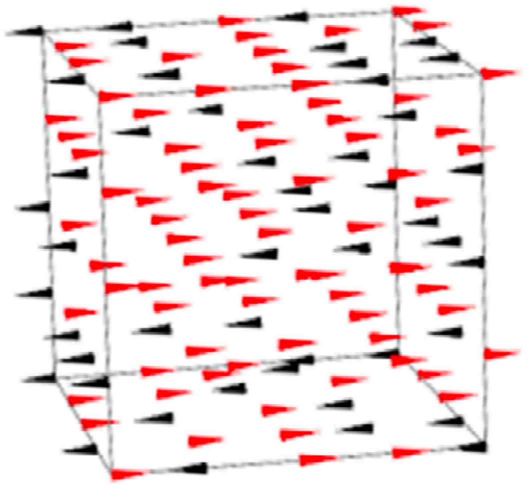
- Consider the following sum across all nodes:
  - $E(y_1, y_2, \dots, y_N) = -\sum_{i,j\neq i} w_{ij}\, y_i y_j - \sum_i b_i y_i$
- E is bounded:
  - $E_{min} = -\sum_{i,j\neq i} |w_{ij}| - \sum_i |b_i|$
- The minimum variation of E in a "flip" is:
  - $|\Delta E|_{min} = \min_{i,\{y_i, i=1\dots N\}} 2 |\sum_{j\neq i} w_{ji} y_j + b_i|$
- So any sequence of flips must converge in a finite number of steps

## The Energy of a Hopfield Net

- The E is the energy of the network
  - $E(y_1, y_2, \ldots, y_N) = -\sum_{i,j\neq i} w_{ij}\, y_i y_j - \sum_i b_i y_i$
- The evolution of a Hopfield network decreases its energy
- Analogy: Spin Glass

## Spin Glass

- Each dipole in a disordered magnetic material tries to align itself to the local field
  - --Filp
- $p_i$ is vector position of i-th dipole
  - -- output of each neuron $y_i$
- The contribution of a dipole to the field depends on interaction J
  - -- Weight $w_{ij}$
  - Derived from the “Ising” model for magnetic materials (Ising and Lenz, 1924)

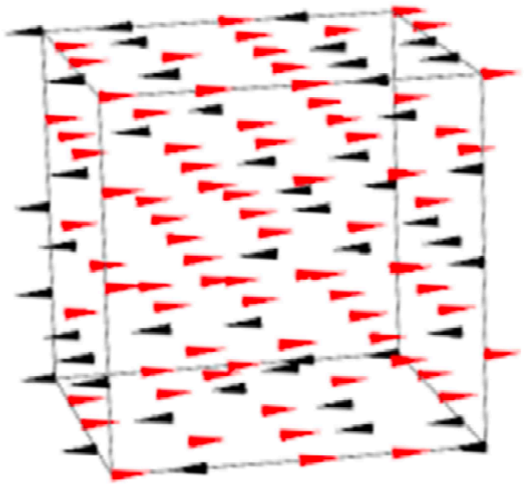


Total field at current dipole:

$$f(p_i) = \underbrace{\sum_{j \neq i} J_{ji} x_j}_{\text{intrinsic}} + \underbrace{b_i}_{\text{external}}$$

## Spin Glass

- Response of current dipole
  - $x_i = \begin{cases} x_i \; if \; sign(x_i \, f(p_i)) = 1 \\ -x_i \; otherwise \end{cases}$
- Total energy (Hamiltonian) of the system
  - $E = -\frac{1}{2}\sum_i x_i f(p_i)$
    $= -\sum_i \sum_{j>i} J_{ji} x_i x_j - \sum_i b_i x_i$
- Evolve to minimize the energy
  - "Flips" will stop



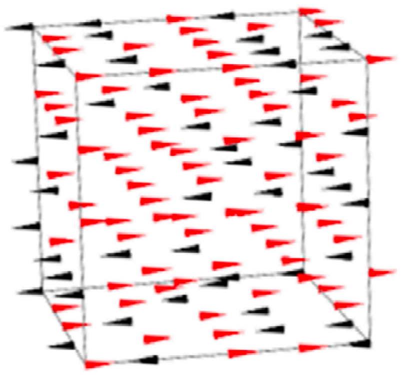
Total field at current dipole:

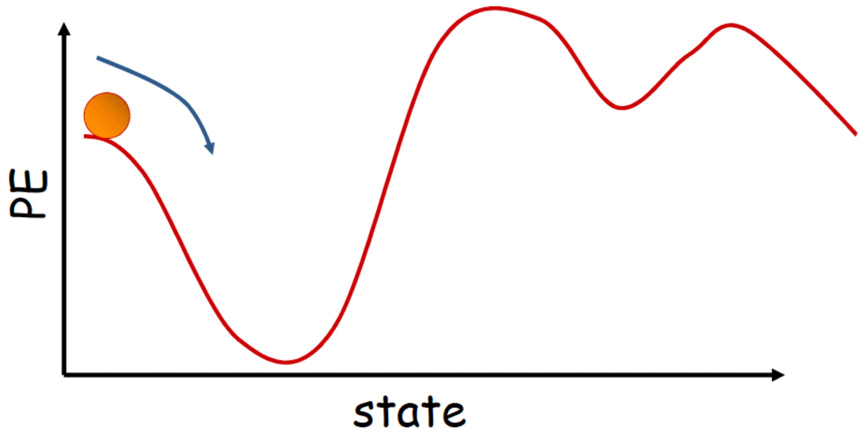
$$f(p_i) = \underset{\text{intrinsic}}{\underbrace{\sum_{j \neq i} J_{ji} x_j}} + \underset{\text{external}}{\underbrace{b_i}}$$

## Spin Glass

- The system stops at one of its stable point
    - local minimum of the energy

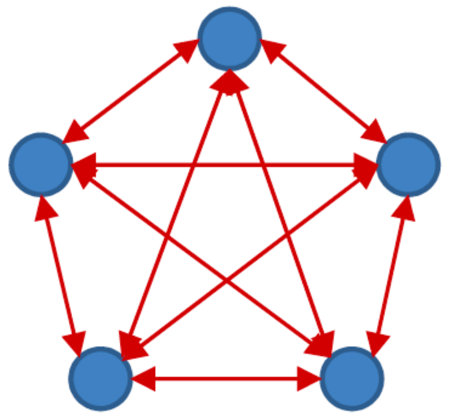
- Every point will return to the stable point after evolving
    - The system remembers its stable state





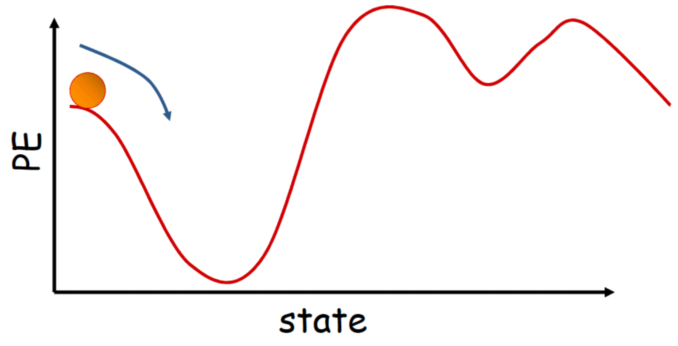
## Contents

## Hopfield Network



$$f(x) = \begin{cases} +1 \; if \; x \geq 0 \\ -1 \; if \; x < 0 \end{cases}$$

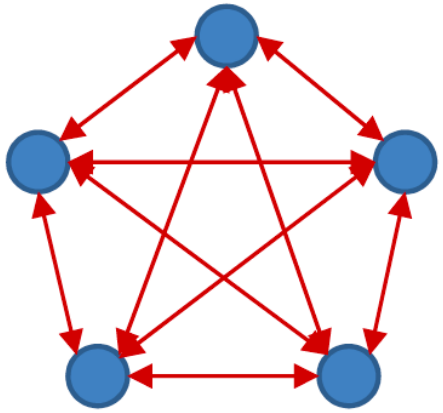
$$y_i = f(\sum_{j \neq i} w_{ji} \, y_j + b_i)$$

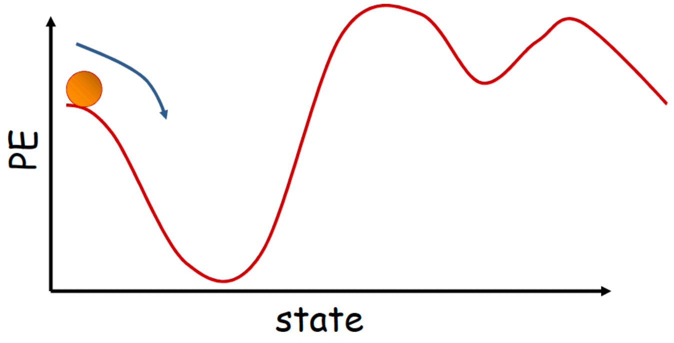


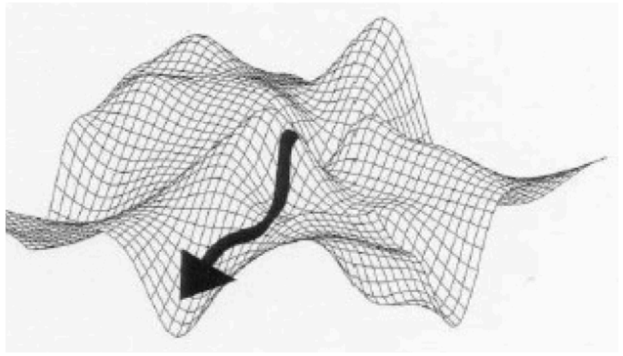
- The bias is typically not utilized
  - It’s similar to having a single extra neuron that is pegged to 1.0

- The network will evolve until it arrives at a local minimum in the energy contour

## Content-addressable memory







- Each minima is a “stored” pattern
  - How to store?
- Recall memory content from partial or corrupt values
- Also called associative memory
- The path is not unique

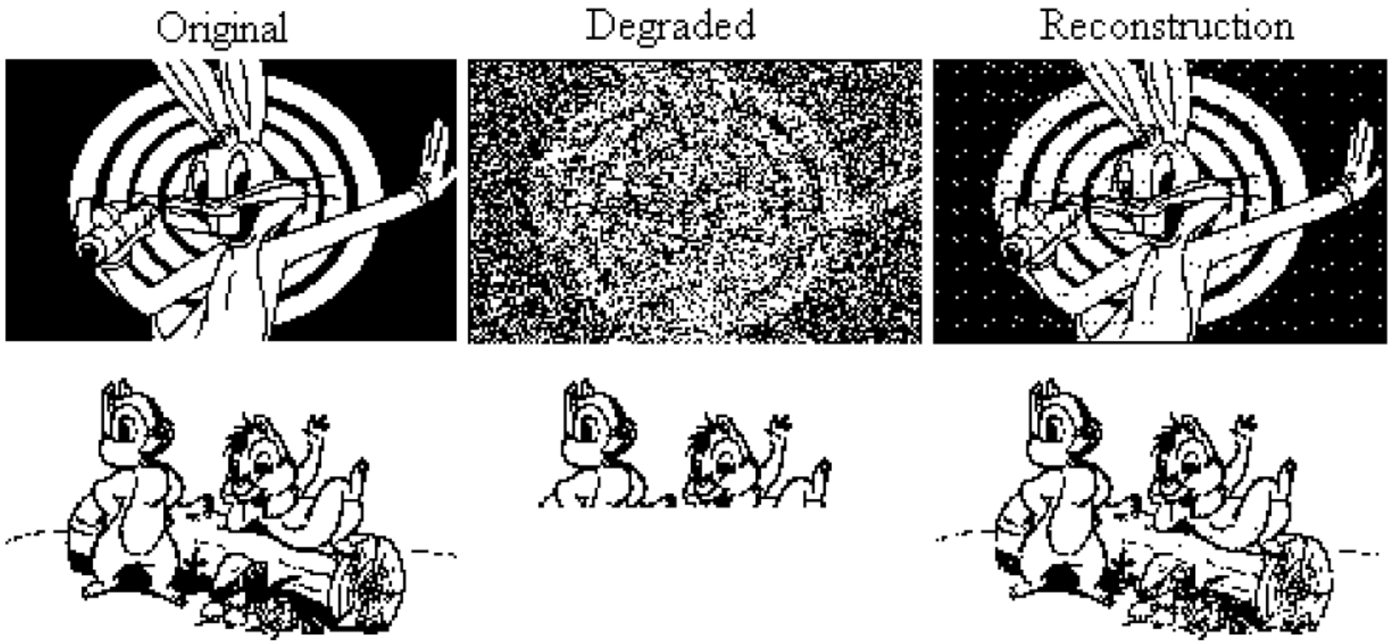
## Real-world Examples

- Take advantage of content-addressable memory

Input

Process of Evolution

## Real-world Examples



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

http://staff.itee.uq.edu.au/janetw/cmc/chapters/Hopfield

## Computation

1. Initialize network with initial pattern

$$y_i = x_i, \qquad 0 \le i \le N-1$$

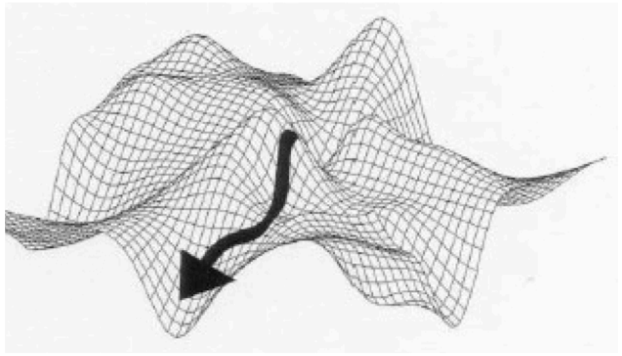
2. Iterate until convergence

$$y_i = f\left(\sum_{j \ne i} w_{ji}\, y_j + b_i\right), 0 \le i \le N-1$$

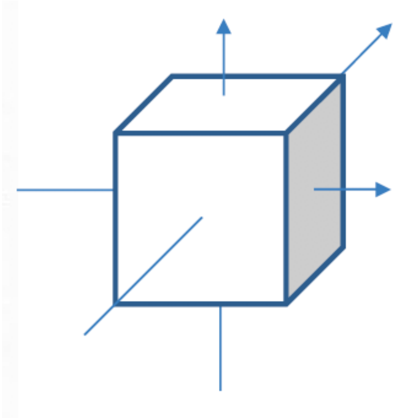
- Updates can be done sequentially, or all at once
    - Usually update all nodes once per epoch
    - In one epoch, the nodes are updated randomly
- The system will converge to the local minimum
    - Not deterministic

## Evolution

- The energy is a quadratic function.
  - $E = -\sum_{i,j\neq i} w_{ij}\, y_i y_j - \sum_i b_i y_i$
  - $E = -\frac{1}{2} y^T W y - b^T y$
- But why not global minimum?



- For DHN, the energy contour is only defined on a lattice
  - Corners of a unit cube on $[-1, 1]^N$

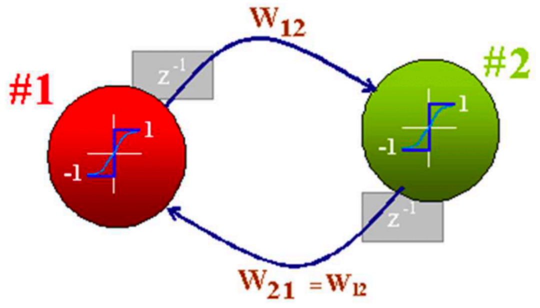


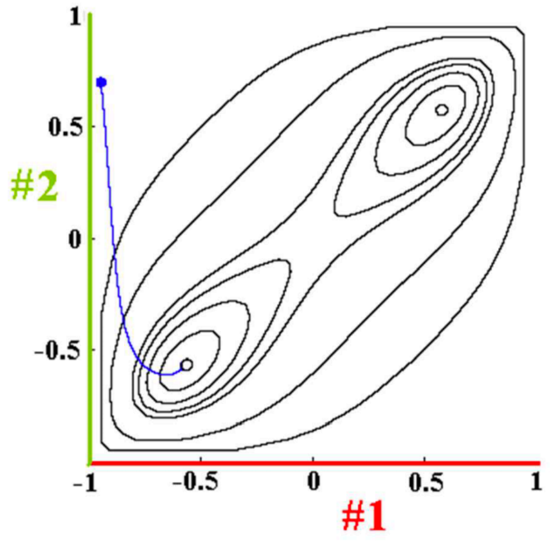
## Evolution

- If we use tanh for activation
  - Still not global minimum, why?
  - Local minimum still exists

- An example for a 2-neuron net
  - Without bias, the local minimum is symmetric, why?

$$-\frac{1}{2}y^T W\, y = -\frac{1}{2}(-y)^T W(-y)$$





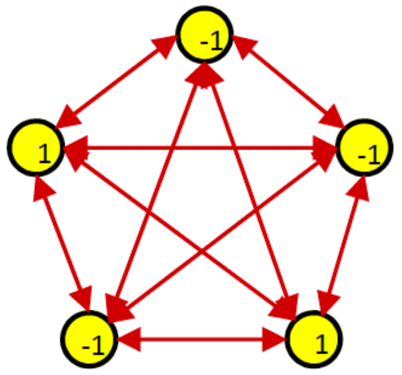
## Contents

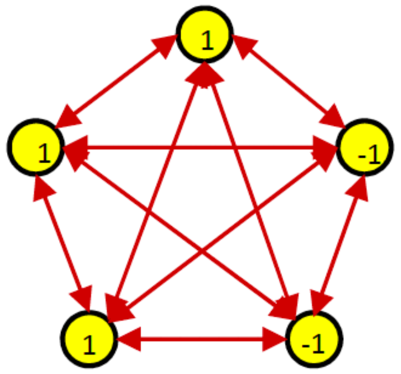
## Issues to be solved

- How to store a specific pattern?
- How many patterns can we store?
- How to “retrieve” patterns better?

## How to store a specific pattern?

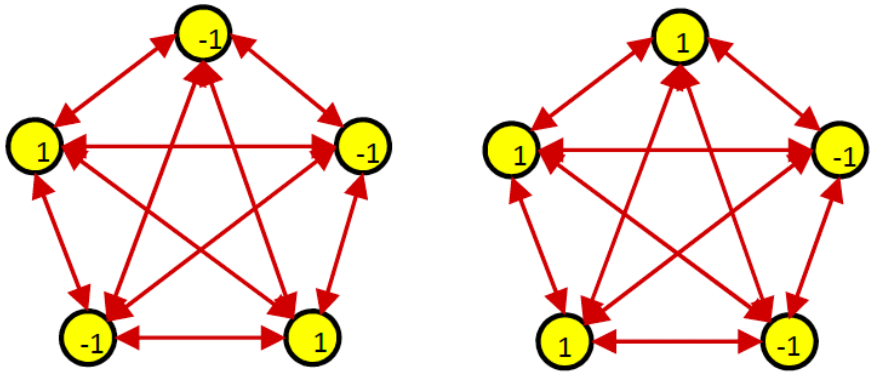
- For an image with N pixels, we need:
  - N neurons
  - $\frac{N(N-1)}{2}$ weights (symmetric)
- Consider the setting without bias
  - $E = -\sum_{i,j\neq i} w_{ij}\, y_i y_j$
- Goal: Design W so that the energy is local minimum at pattern $P = \{y_i\}$





## Method1: Hebbian Learning

- We want:
  - $f\left(\sum_{j\neq i} w_{ji} y_j\right) = y_i \ \forall i$
- Hebbian Learning:
  - $w_{ji} = y_j y_i$

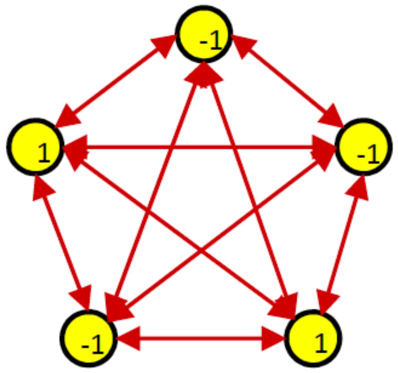


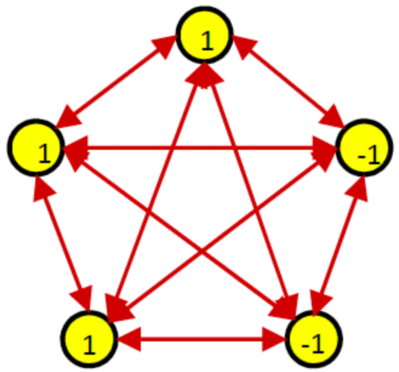
- $f\left(\sum_{j\neq i} w_{ji} y_j\right) = f\left(\sum_{j\neq i} y_j y_i y_j\right) = f\left(\sum_{j\neq i} y_j^2 y_i\right) = f(y_i) = y_i$
- The pattern is stationary
- $\mathrm{E} = -\sum_{i,j\neq i} w_{ij}\, y_i y_j = -\frac{1}{2} N(N-1)$

## Method1: Hebbian Learning

- Note:
  - If we store P, we will also store –P

- For K patterns:
  - $y_k = [y_1^k, y_2^k, \dots, y_N^k], k = 1, \dots, K$
  - $w_{ij} = \frac{1}{N} \sum_k y_i^k y_j^k$
- Each pattern is stable





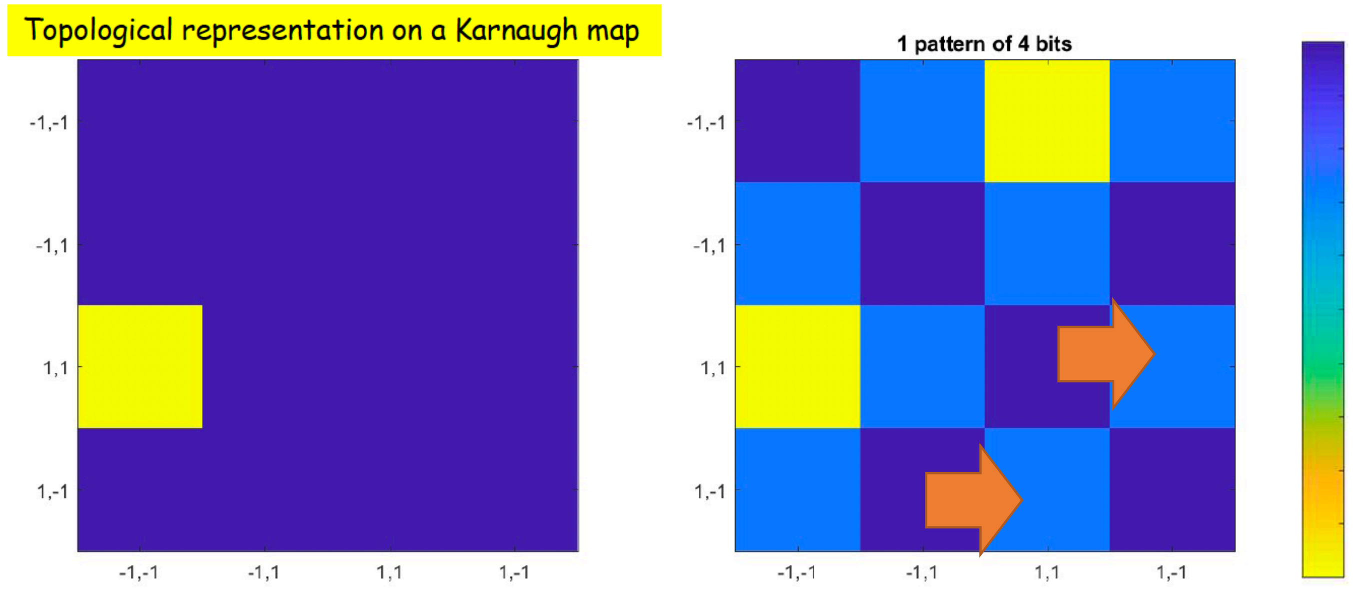
## Method1: Hebbian Learning - How many patterns can we store?

- A network of N neurons trained by Hebbian learning can store ~0.14N patterns with low probability of error (<0.4%)
  - Assume P(bit=1)=0.5
  - Patterns are orthogonal – maximally distant
    - The maximum Hamming distance between two N-bit patterns is N/2 (because symmetry)
    - Two patterns differ in N/2 bits are orthogonal

- The proof can be found in 11-785 CMU Lec 17

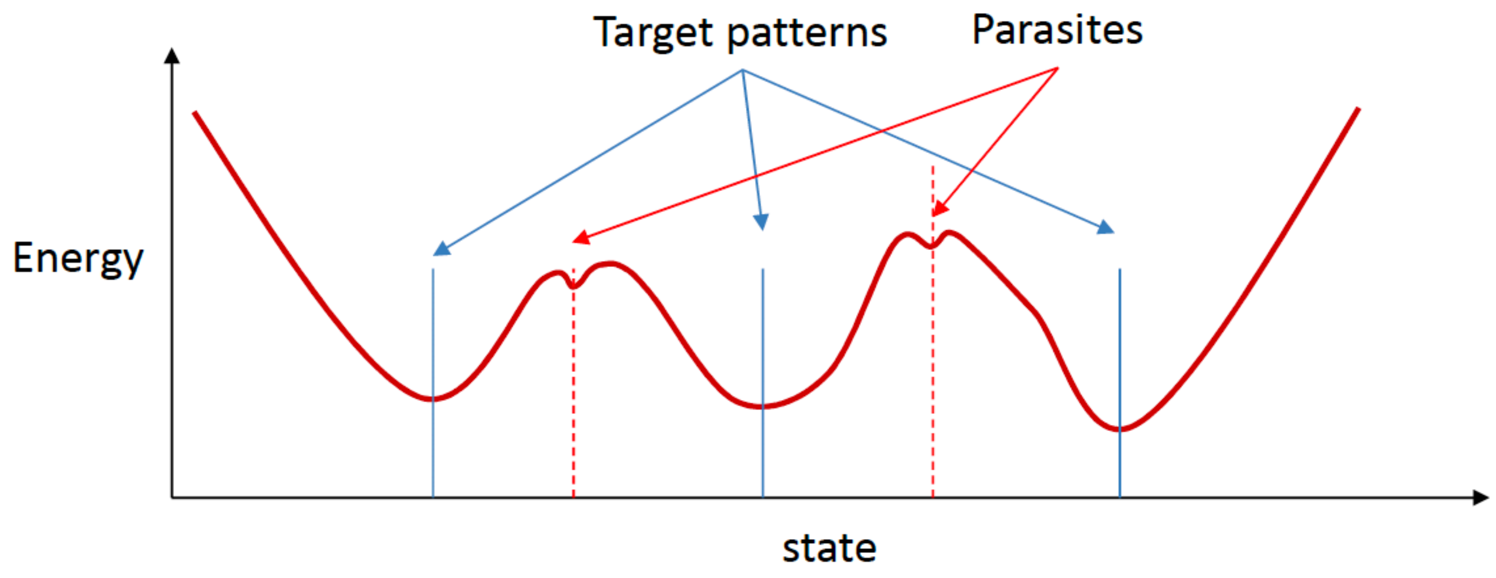
## Method1: Hebbian Learning - Example: 4-bit pattern

- Left: stored pattern. Right: energy map
- Local minima exists



## Method1: Hebbian Learning - Parasitic Patterns

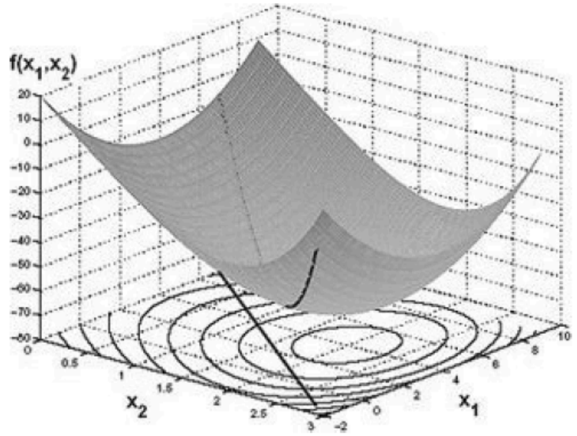
- Parasitic patterns are not expected

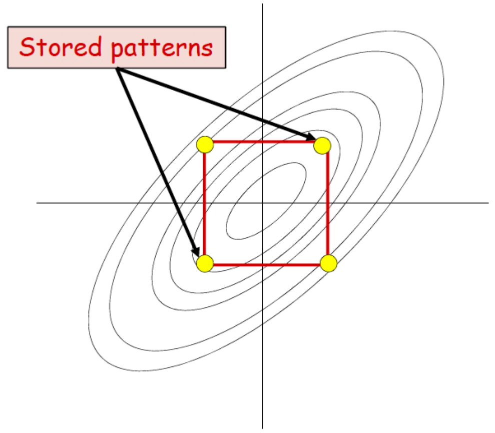


## Method2: Geometric approach

- Consider $W = yy^T$ i.e., $w_{ji} = y_j y_i$
  - W is a positive semidefinite matrix
- $E = -\frac{1}{2} y^T W y - b^T y$ is convex quadratic

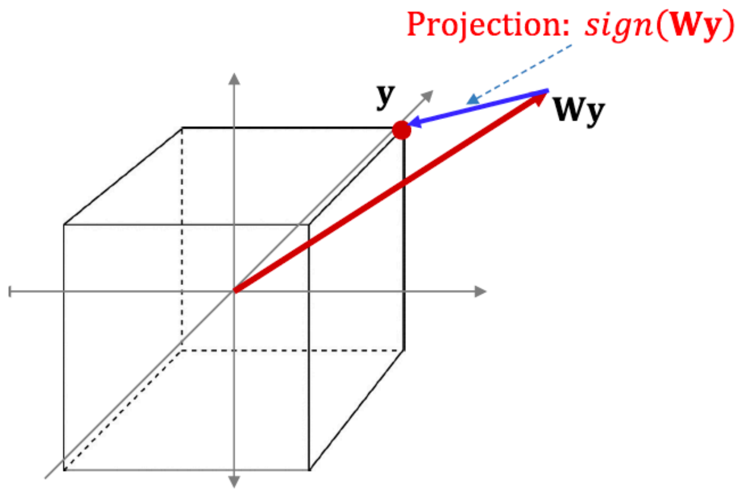
- But remember y is the corner of the unit hypercube





## Method2: Geometric approach

- Evolution of the network:
  - Rotate y and project it onto the nearest corner.



## Method2: Geometric approach

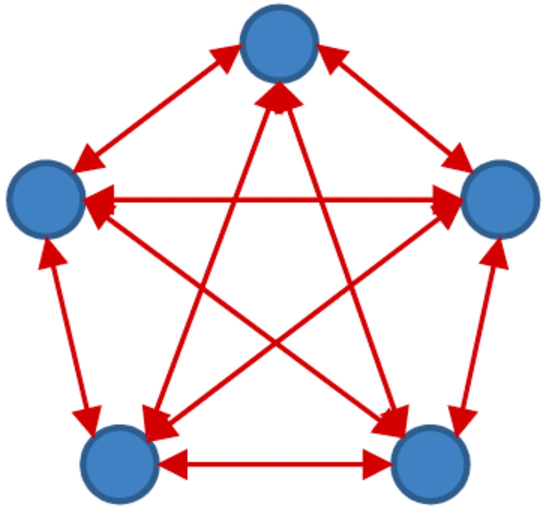
- Goal: Design W such that $f(Wy) = y$
- Simple solution: y is the Eigenvector of W
  - Note the eigenvalue of W are non-negative
  - The eigenvector of any symmetric matrix are orthogonal
- Storing K orthogonal patterns $Y = [y_1, y_2, \dots, y_K]$
  - $W = Y\Lambda Y^T$
  - $\Lambda$ is a positive diagonal matrix *diag*$(\lambda_1, \lambda_2, \dots \lambda_K)$
  - Hebbian rule: $\lambda = 1$.
    - All patterns are equally important

## Method3: Optimization

- $E = -\frac{1}{2} y^T W y - b^T y$
- This must be maximally low for target patterns
- Also must be maximally high for all other patterns

- $W = argmin_W \sum_{y \in Y_p} E(y) - \sum_{y \notin Y_p} E(y)$

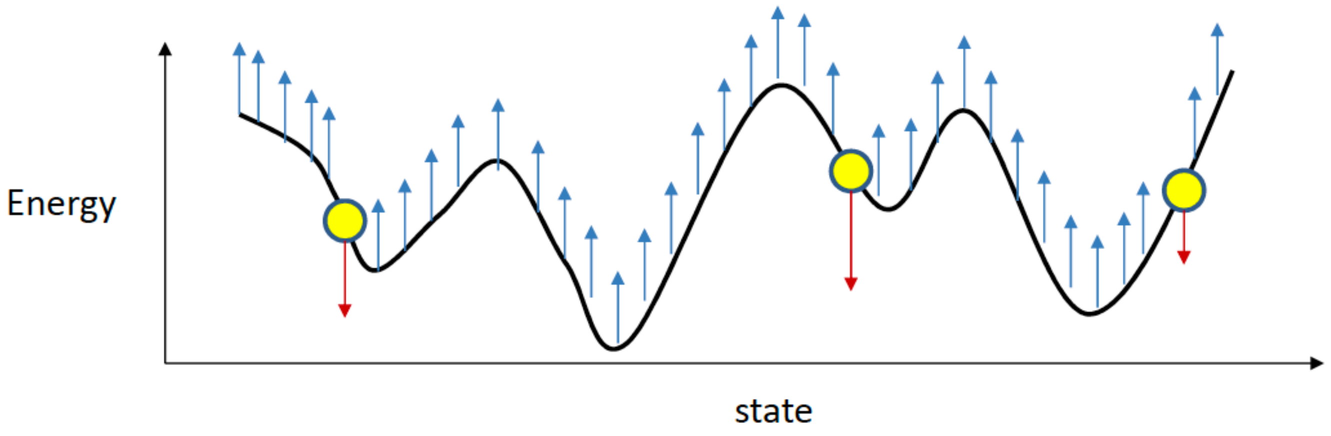
  $Y_\mathrm{p}$: set of target pattern



## Method3: Optimization

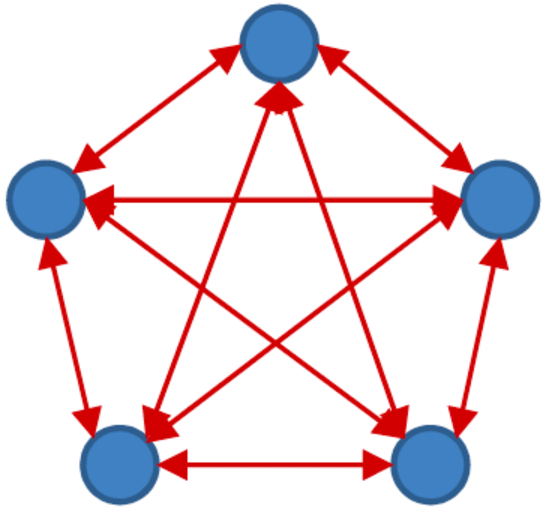
- $W = argmin_W \sum_{y \in Y_p} E(y) - \sum_{y \notin Y_p} E(y)$
- $\mathrm{Y_p}$: set of target pattern

- Intuitively:



## Method3: Optimization

- $W = argmin_W \sum_{y \in Y_p} E(y) - \sum_{y \notin Y_p} E(y)$
- So gradient descent:
  - $W = W + \alpha(\sum_{y \in Y_p} yy^T - \sum_{y \notin Y_P} yy^T)$
- Repeating a pattern can emphasize the importance.
- What about $y \notin Yp$?

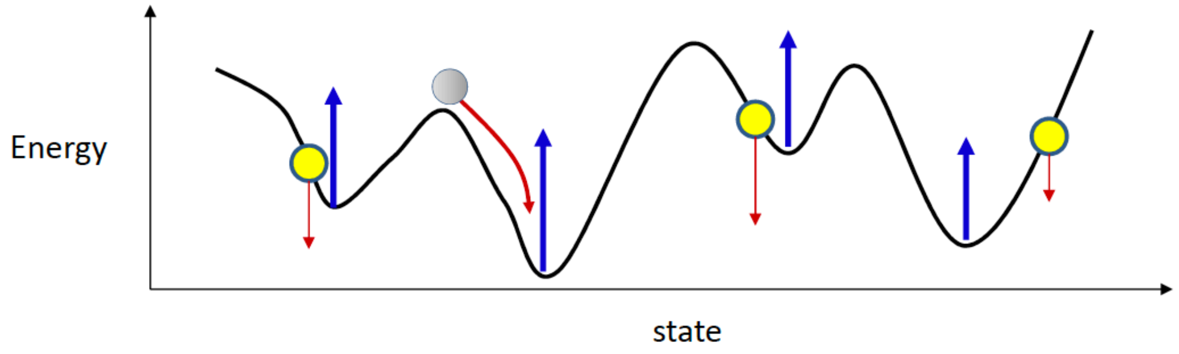


## Method3: Optimization

- $W = W + \alpha(\sum_{y \in Y_p} yy^T - \sum_{y \notin Y_P} yy^T)$

- We only need to focus on valleys.
- How to find valleys?

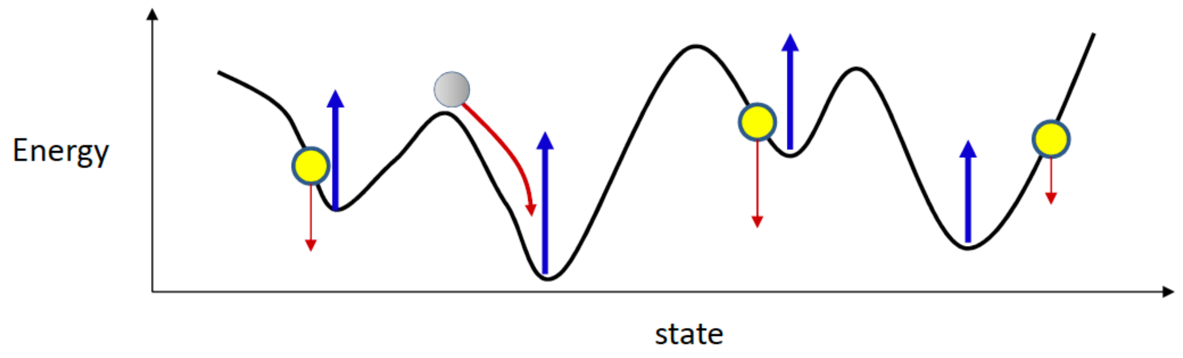
- Random sample and let it evolve



## Method3: Optimization

- $W = W + \alpha(\sum_{y \in Y_p} yy^T - \sum_{y \notin Y_P, y=valley} yy^T)$

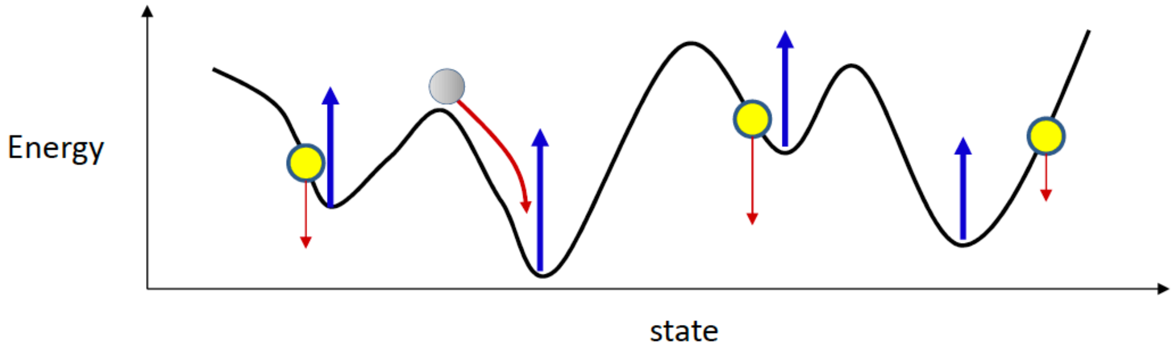
- Initialize W
- Repeat until convergence or limitation:
  - Sample target pattern
  - Randomly initialize the network and let it evolve
  - Update weights



## Method3: Optimization

- $W = W + \alpha(\sum_{y \in Y_p} yy^T - \sum_{y \notin Y_P, y=valley} yy^T)$

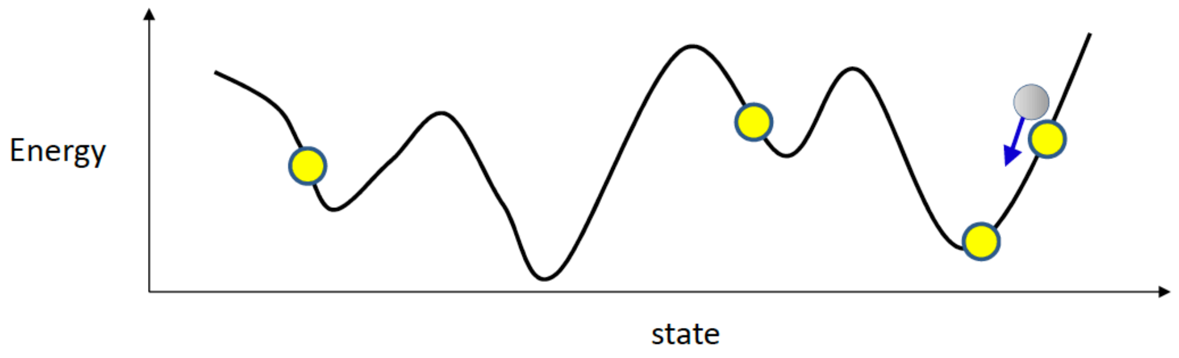
- Initialize W
- Repeat until convergence or limitation:
  - Sample target pattern
  - Initialize the network with target pattern and let it evolve
  - Update weights



## Method3: Optimization

- $W = W + \alpha(\sum_{y \in Y_p} yy^T - \sum_{y \notin Y_P, y=valley} yy^T)$

- Initialize W
- Repeat until convergence or limitation:
  - Sample target pattern
  - Initialize the network with target pattern and let it evolve a few steps
  - Update weights

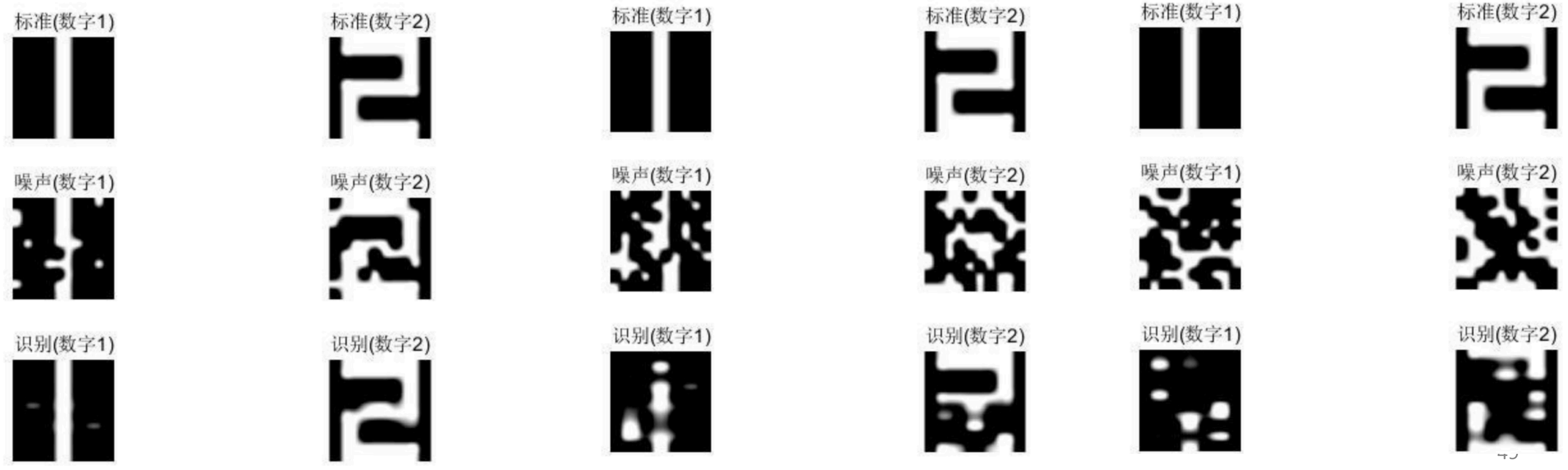


## Contents

## Thinking

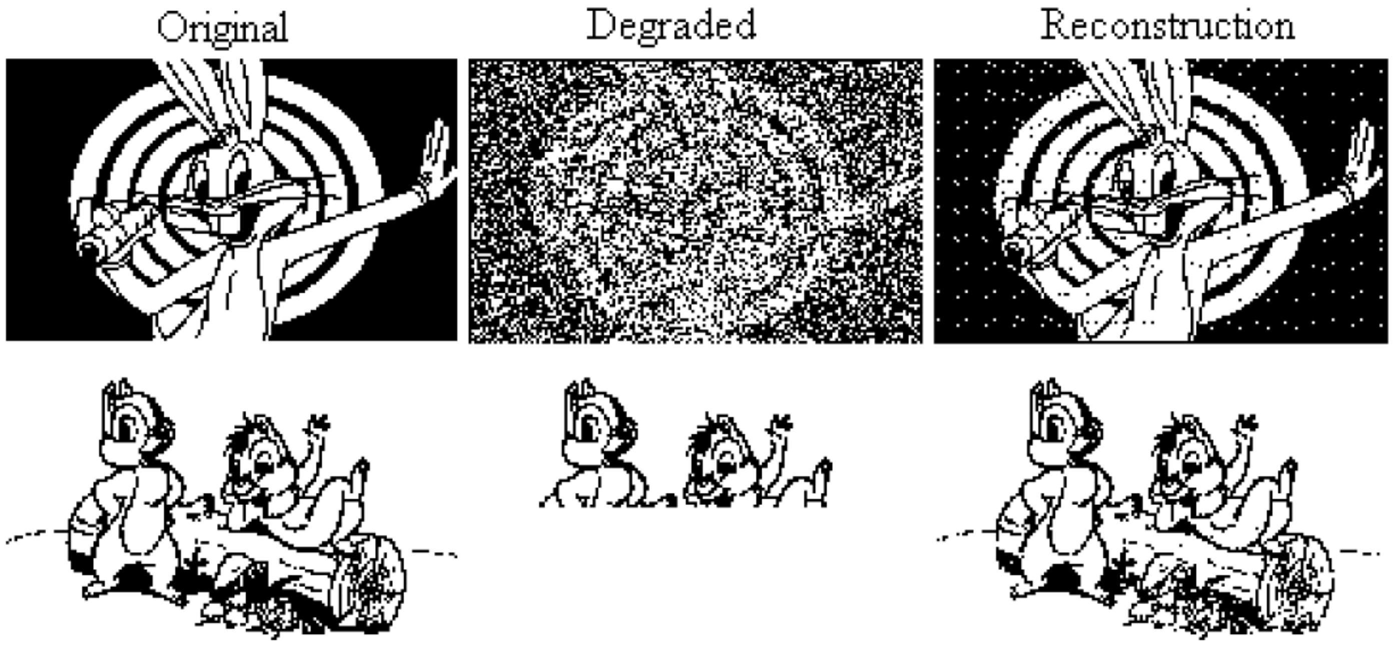
- The capacity of Hopfield Network
  - How many patterns can be stored?
  - Orthogonal <N; Non-orthogonal?

- Something bad happens:
  - When noise increase…



## Thinking

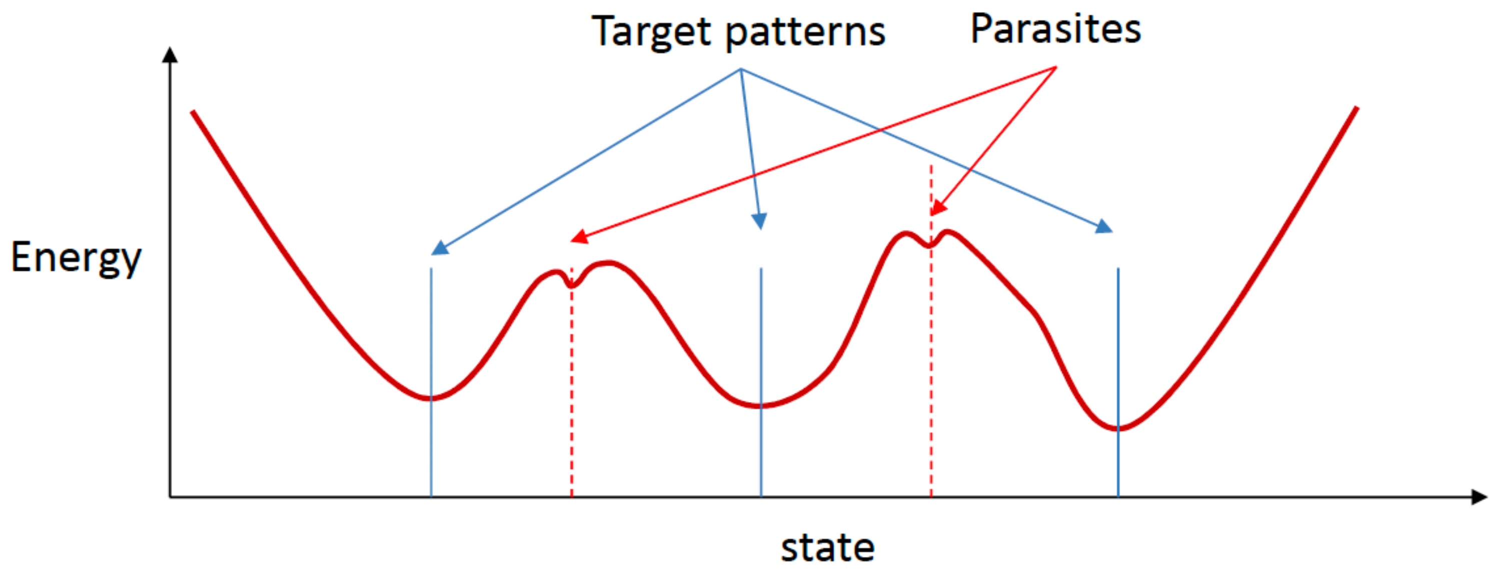
- Something bad happens:
  - The results are not perfect…



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

## Thinking

- Something bad happens:
  - The results are not perfect…
    - Because of the local minima



## Thinking – Stochastic Hopfield Net

- Something bad happens:
  - The results are not perfect…

- We can make Hopfield net stochastic
  - Each neuron responds probabilistically
  - If the difference if not large, the probability of flipping approaches 0.5
  - T is a “temperature” parameter

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

## Thinking – Stochastic Hopfield Nets

- What's the final state? (How do we recall a memory?)
  - The average of the final few iterations

$$\mathbf{y} = \left(\frac{1}{M}\sum_{t=L-M+1}^{L} \mathbf{y}_t\right) > 0?$$

## Contents

## Continuous Hopfield Neural Network

- Energy function :

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}V_iV_j - \sum_{i=1}^{n} V_iI_i + \sum_{i=1}^{n}\frac{1}{R}\int_{1}^{V_i} f^{-1}(v)dv$$

- The output of each neuron are real numbers in [-1,+1]

- Application: optimization (TSP)

- Issues:
  - Design the energy function for specific problems
  - The variable of the problem and the neuron of the CHNN

## Reference

- CMU 11-785 Lec17, 18

Thanks