Neural Networks

Hopfield Nets, Auto Associators, Boltzmann machines

Fall 2024

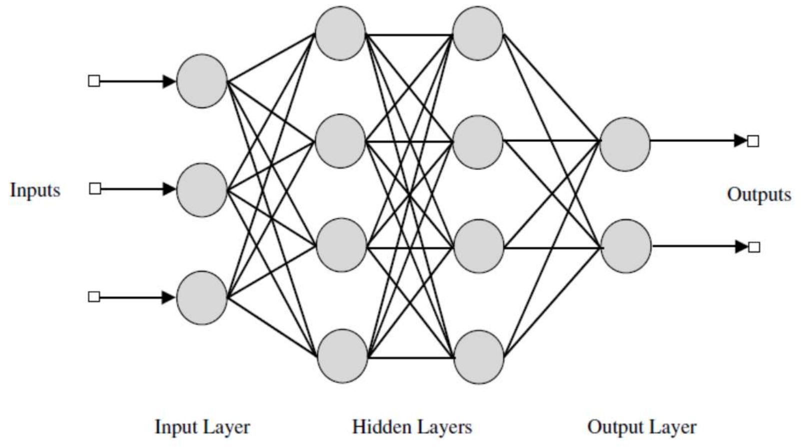
They trained artificial neural networks using physics

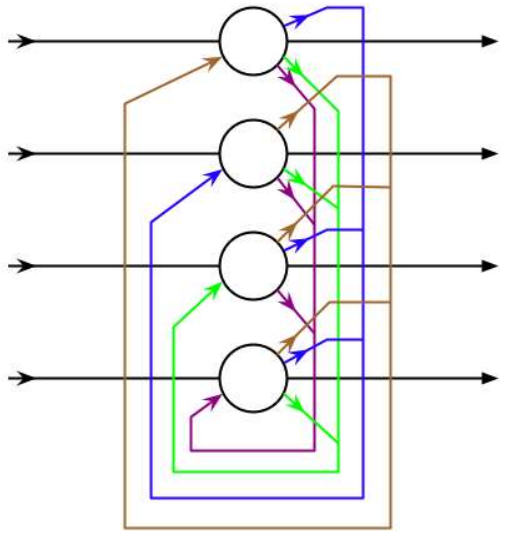
This year’s two Nobel Laureates in Physics have used tools from physics to develop methods that are the foundation of today’s powerful machine learning. John Hopfield created an associative memory that can store and reconstruct images and other types of patterns in data. Geoffrey Hinton invented a method that can autonomously find properties in data, and so perform tasks such as identifying specific elements in pictures.

Story so far

- **Neural networks for computation**
- All feedforward structures

- But what about..

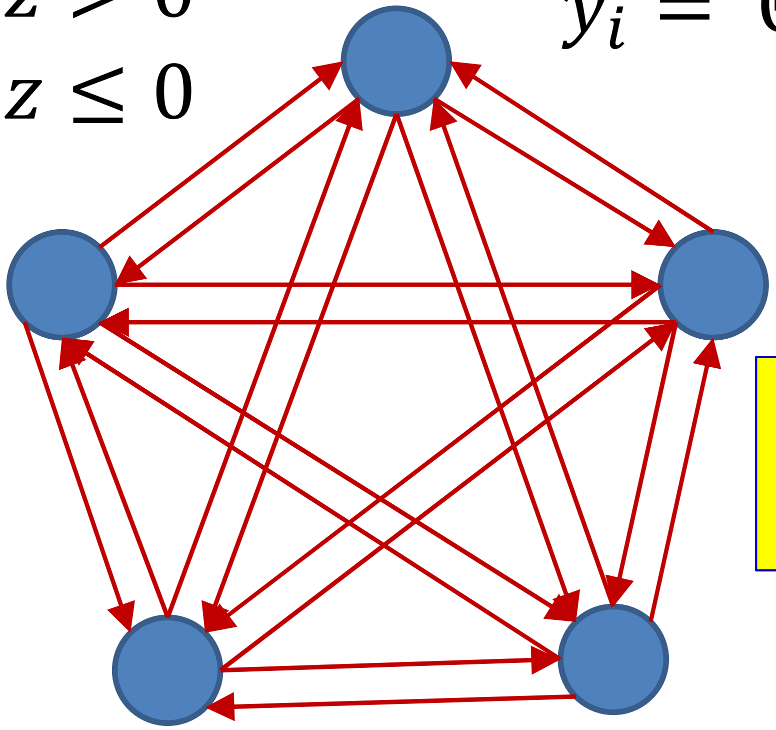




Consider this loopy network

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

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



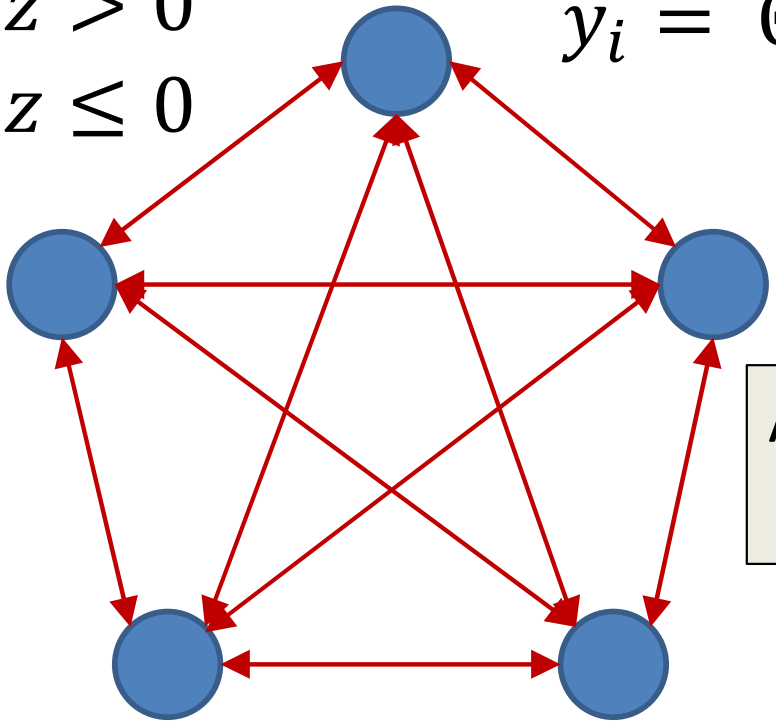
The output of a neuron affects the input to the neuron

- Each neuron is a perceptron with +1/-1 output
- Every neuron *receives* input from every other neuron
- Every neuron *outputs* signals to every other neuron

Consider this loopy network

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

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



A symmetric network:

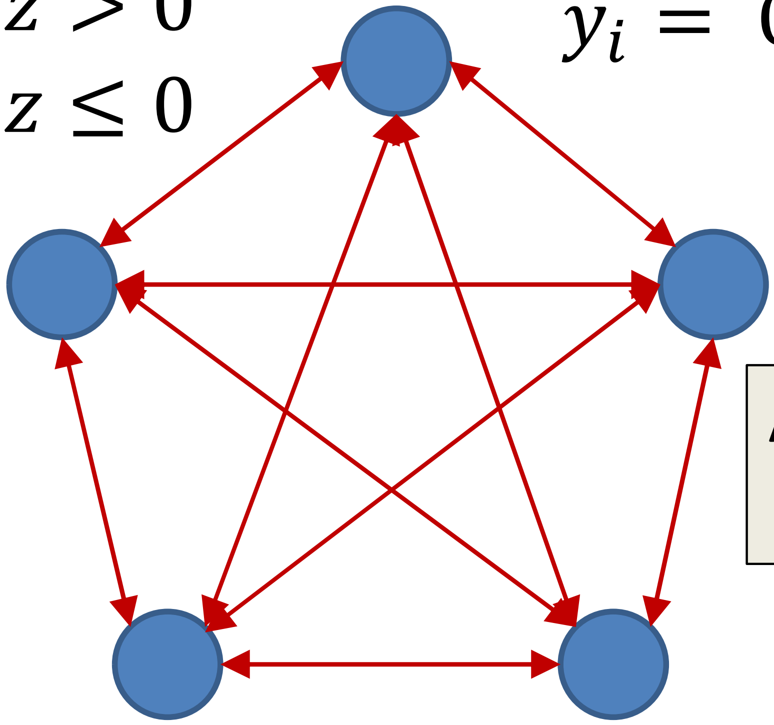
$$w_{ij} = w_{ji}$$

- Each neuron is a perceptron with +1/-1 output
- Every neuron *receives* input from every other neuron
- Every neuron *outputs* signals to every other neuron

Hopfield Net

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

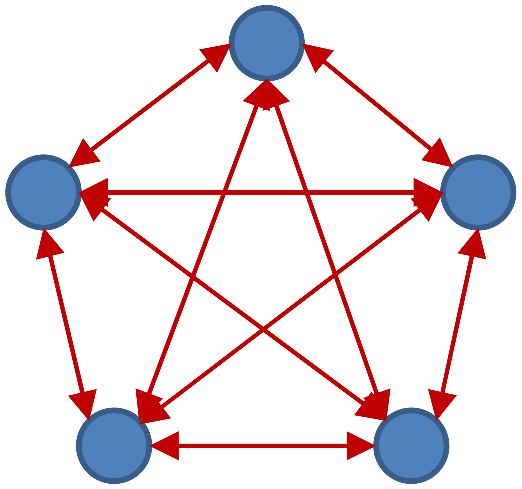
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$



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

- Each neuron is a perceptron with +1/-1 output
- Every neuron *receives* input from every other neuron
- Every neuron *outputs* signals to every other neuron

Loopy network

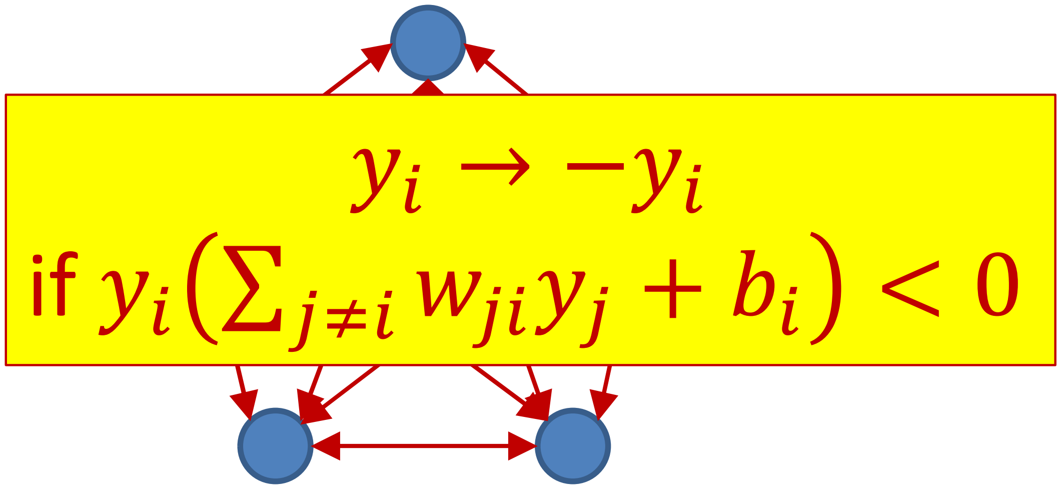


$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

- 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, it does not respond
- If the sign of the field opposes its own sign, it "flips" to match the sign of the field

Loopy network



$$y_i \rightarrow -y_i$$

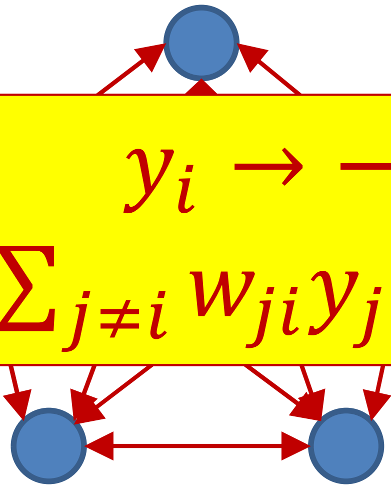
if $y_i\left(\sum_{j\neq i} w_{ji}y_j + b_i\right) < 0$

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

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

- 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, it does not respond
- If the sign of the field opposes its own sign, it “flips” to match the sign of the field

Loopy network



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

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

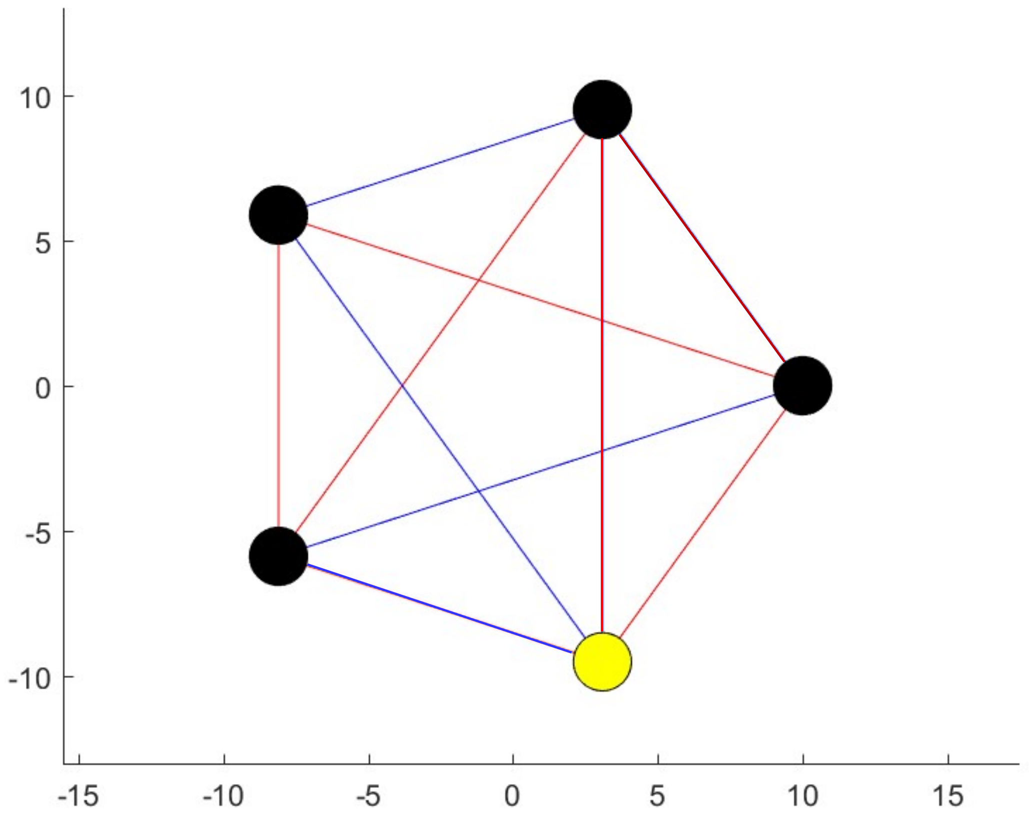
$$\Theta(z) = \begin{cases} +1 \; if \; z > 0 \\ -1 \; if \; z \leq 0 \end{cases}$$

A neuron "flips" if weighted sum of other neurons' outputs is of the opposite sign to its own current (output) value

But this may cause *other* neurons to flip!

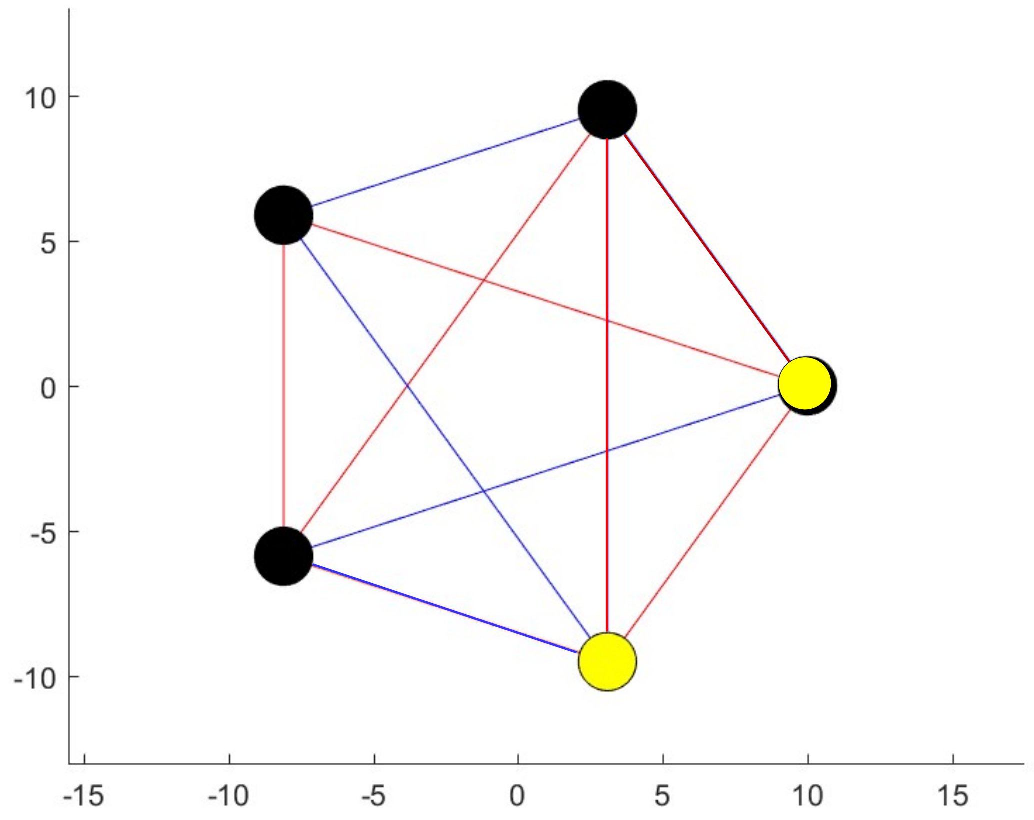
- es a "field" $\sum_{j\neq i} w_{ji}y_j + b_i$
- s own sign, it does not respond
- If the sign of the field opposes its own sign, it "flips" to match the sign of the field

Example



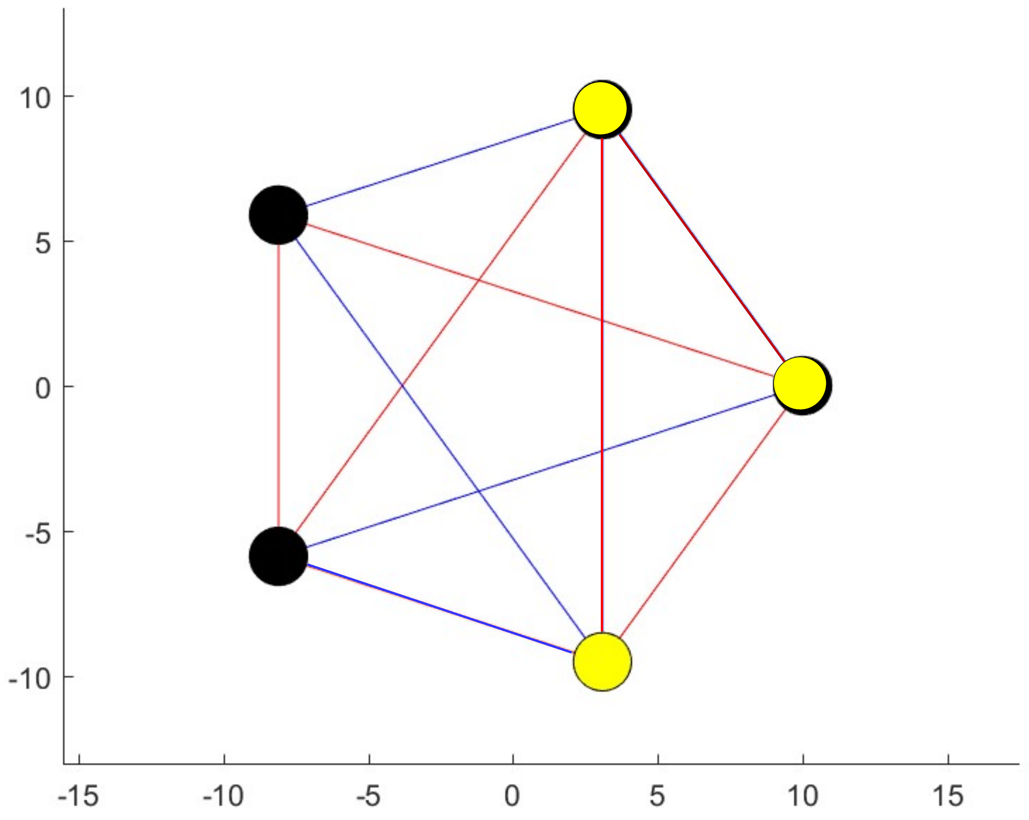
- Red edges are +1, blue edges are -1
- Yellow nodes are -1, black nodes are +1

Example



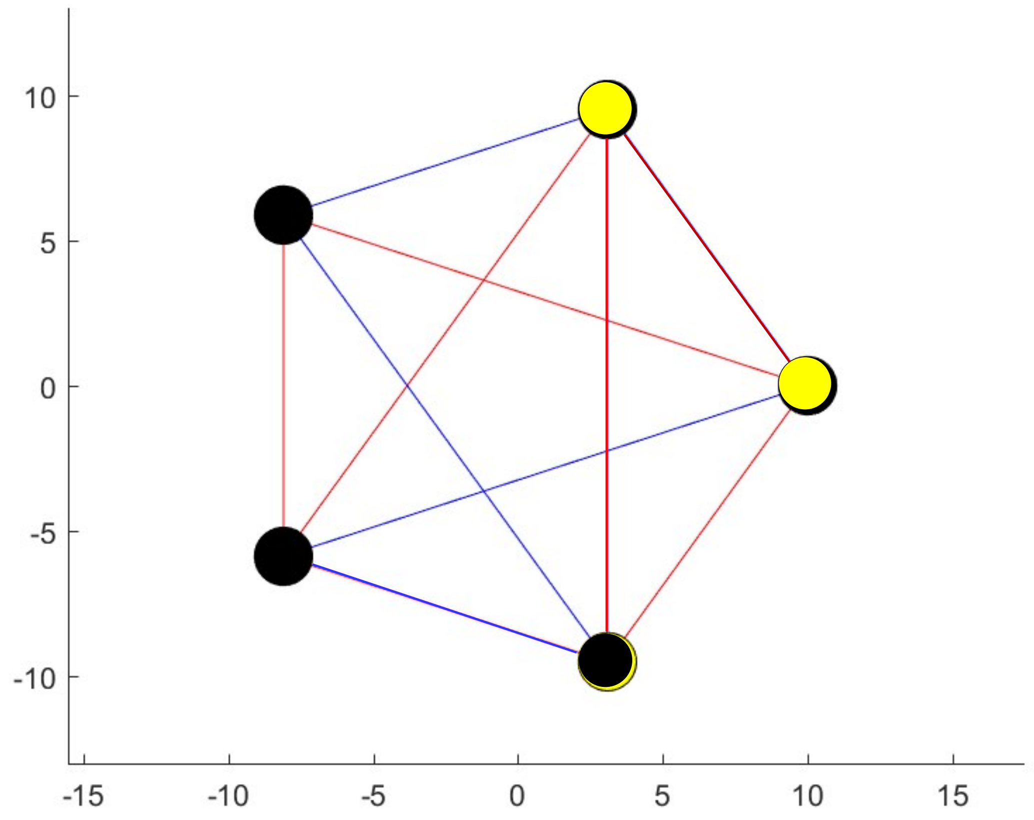
- Red edges are +1, blue edges are -1
- Yellow nodes are -1, black nodes are +1

Example



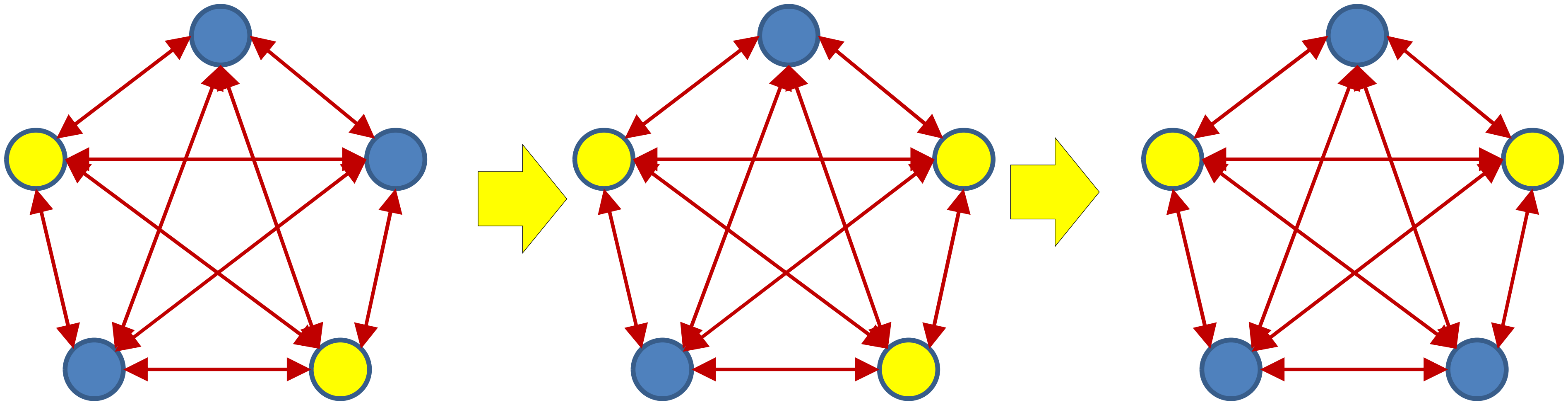
- Red edges are +1, blue edges are -1
- Yellow nodes are -1, black nodes are +1

Example



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

Loopy network

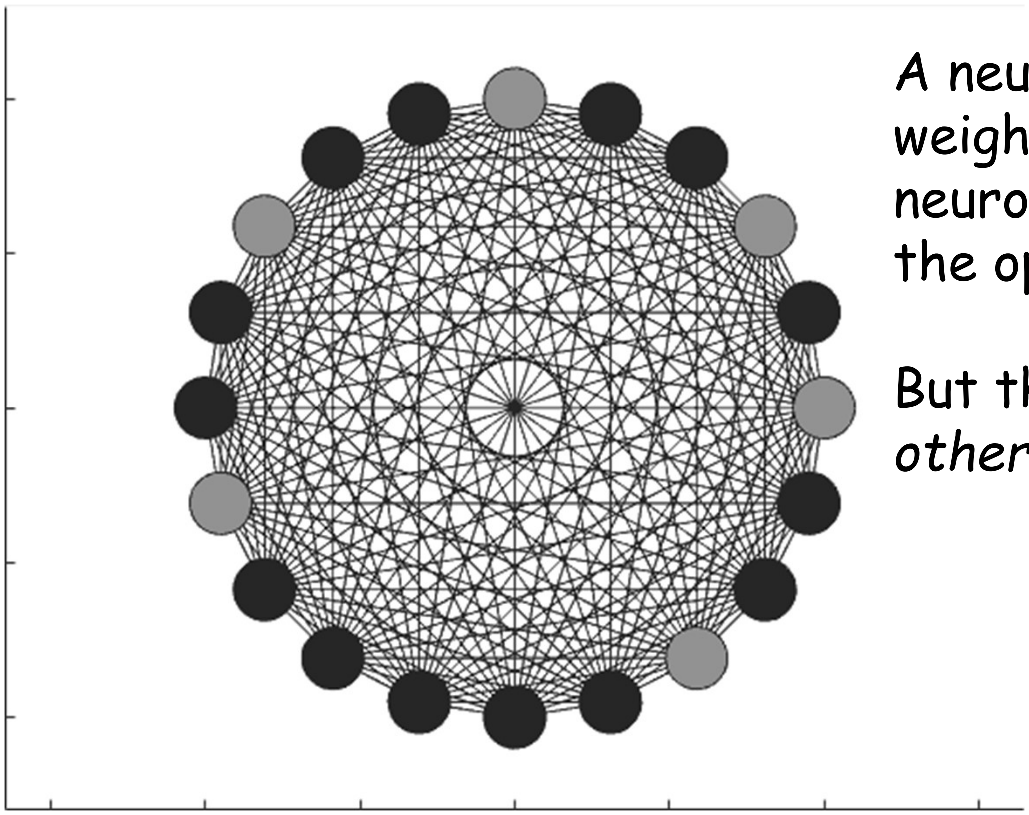


- If the sign of the field at any neuron opposes its own sign, it "flips" to match the field
  - Which will change the field at other nodes
    - Which may then flip
      - Which may cause other neurons including the first one to flip…
        - » And so on…

20 evolutions of a loopy net

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

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

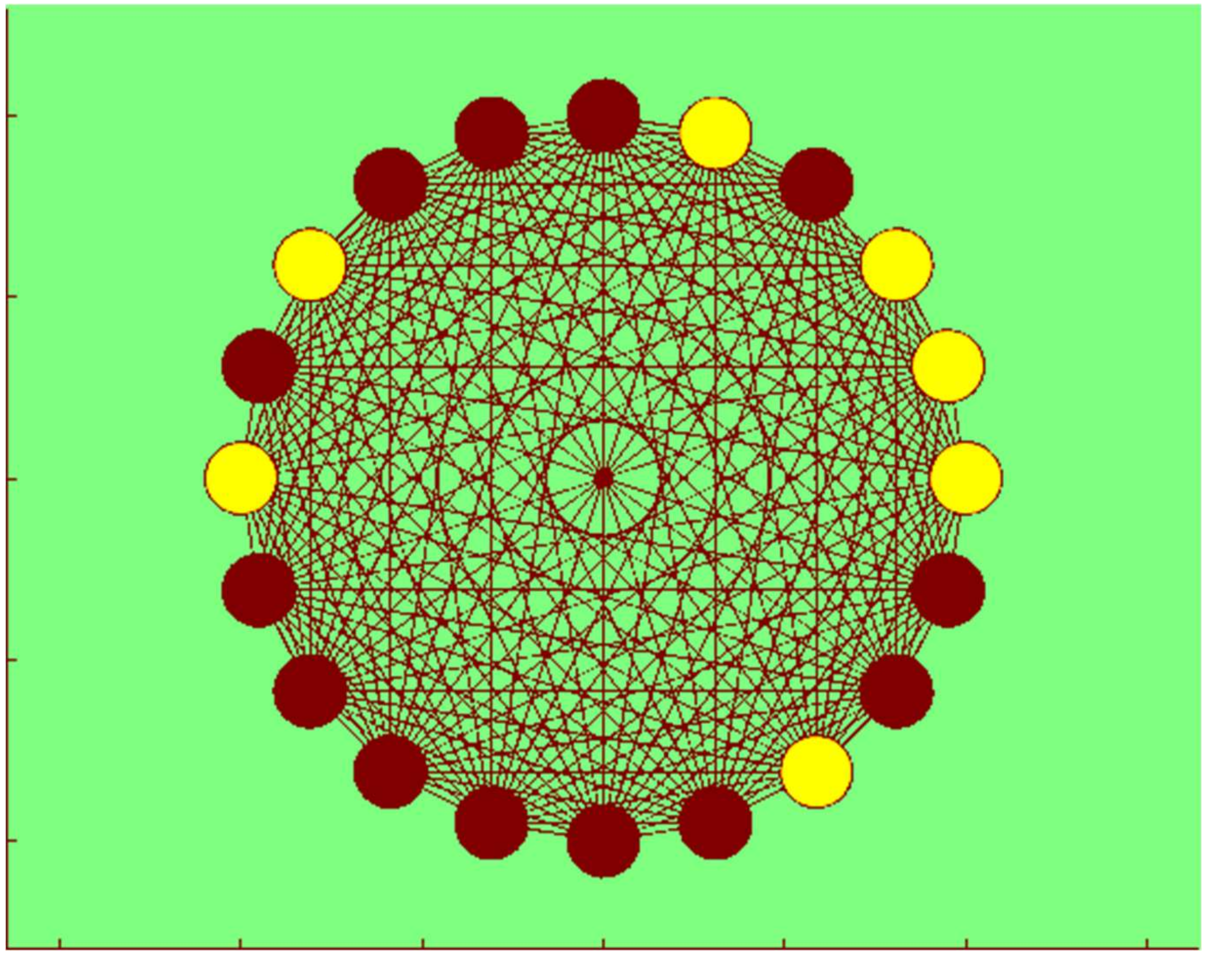


A neuron "flips" if weighted sum of other neuron's outputs is of the opposite sign

But this may cause *other* neurons to flip!

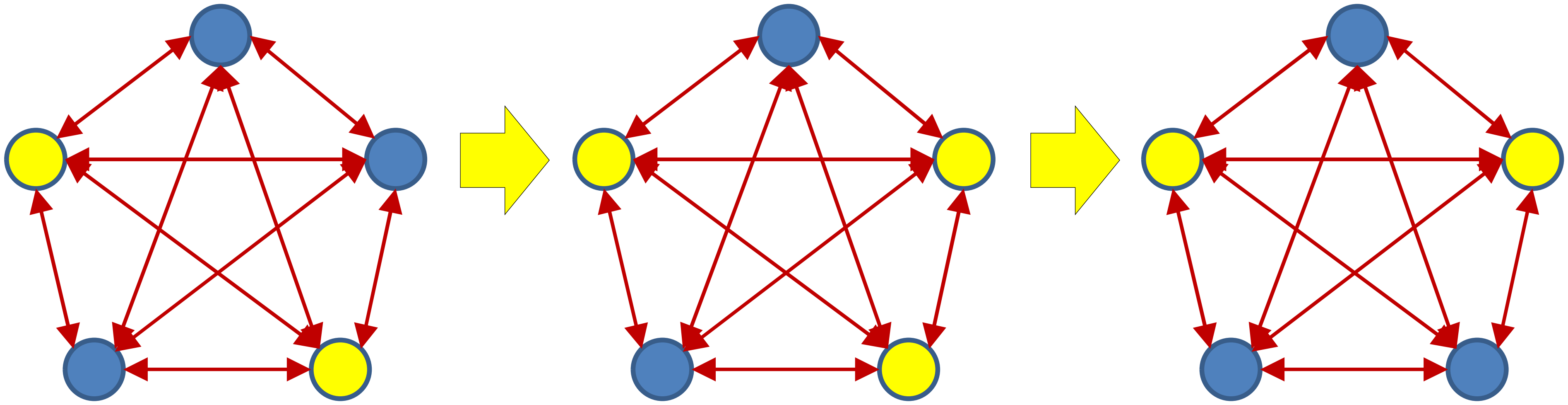
- All neurons which do not "align" with the local field "flip"

120 evolutions of a loopy net



- All neurons which do not "align" with the local field "flip"

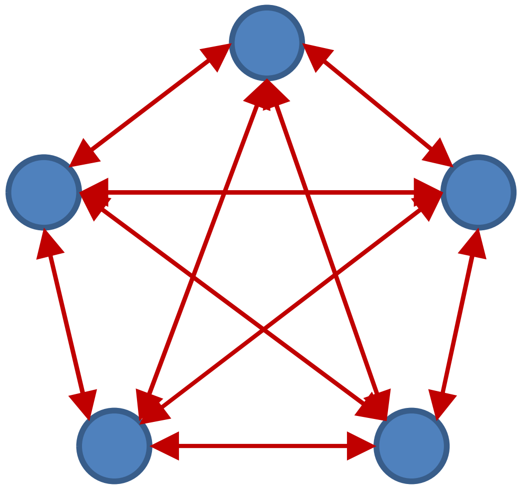
Loopy network



- If the sign of the field at any neuron 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 including the first one to flip…

- *Will this behavior continue for ever??*

Loopy network



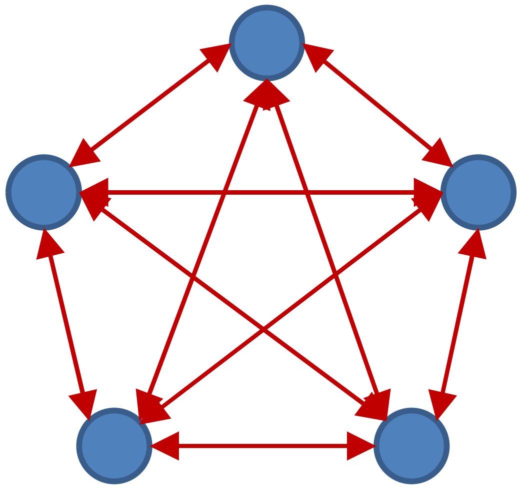
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

- Let y_i^- be the output of the *i*-th neuron just *before* it responds to the current field
- Let y_i^+ be the output of the *i*-th neuron just *after* it responds to the current field
- If $y_i^- = sign\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$, then $y_i^+ = y_i^-$
  - If the sign of the field matches its own sign, it does not flip

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

Loopy network



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

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

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

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

  – This term is always positive!

- *Every flip of a neuron is guaranteed to locally increase*

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

Globally

- Consider the following sum across *all* nodes

$$D(y_1, y_2, \dots, y_N) = \sum_i y_i \left(\sum_{j \neq i} w_{ji} y_j + b_i \right)$$

$$= \sum_{i, j \neq i} w_{ij} y_i y_j + \sum_i b_i y_i$$

  - Assume $w_{ii} = 0$

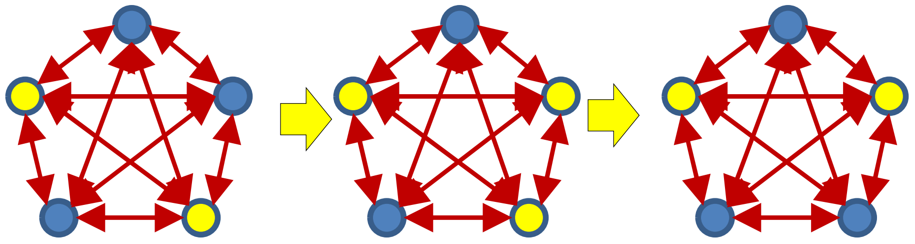
- For any unit k that "flips" because of the local field

$$\Delta D(y_k) = D(y_1, \dots, y_k^+, \dots, y_N) - D(y_1, \dots, y_k^-, \dots, y_N)$$

- This is strictly positive

$$\Delta D(y_k) = 2 y_k^+ \left(\sum_{j \neq k} w_{jk} y_j + b_k \right)$$

Hopfield Net



- Flipping a unit will result in an increase (non-decrease) of

$$D = \sum_{i,j\neq i} w_{ij} y_i y_j + \sum_i b_i y_i$$

- D is bounded

$$D_{max} = \sum_{i,j\neq i} |w_{ij}| + \sum_i |b_i|$$

- The minimum increment of D in a flip is

$$\Delta D_{min} = \min_{i, \{y_i,\, i=1..N\}} 2 \left| \sum_{j\neq i} w_{ji} y_j + b_i \right|$$

- Any sequence of flips must converge in a finite number of steps

The Energy of a Hopfield Net

- Define the *Energy* of the network as

$$E = -\frac{1}{2}\left(\sum_{i,j\neq i} w_{ij}y_iy_j - \sum_i b_iy_i\right)$$

  - Just 0.5 times the negative of D
    - The 0.5 is only needed for convention

- The evolution of a Hopfield network constantly decreases its energy

Story so far

- A Hopfield network is a loopy binary network with symmetric connections
- Every neuron in the network attempts to "align" itself with the sign of the weighted combination of outputs of other neurons
  - The local "field"
- Given an initial configuration, neurons in the net will begin to "flip" to align themselves in this manner
  - Causing the field at other neurons to change, potentially making them flip
- Each evolution of the network is guaranteed to decrease the "energy" of the network
  - The energy is lower bounded and the decrements are upper bounded, so the network is guaranteed to converge to a stable state in a finite number of steps

Poll 1

Hopfield networks are loopy networks whose output activations "evolve" over time

- True
- False

Hopfield networks will evolve continuously, forever

- True
- False

Hopfield networks can also be viewed as infinitely deep shared parameter MLPs

- True
- False

Poll 1

Hopfield networks are loopy networks whose output activations “evolve” over time

- **True**
- False

Hopfield networks will evolve continuously, forever

- True
- **False**

Hopfield networks can also be viewed as infinitely deep shared parameter MLPs

- **True**
- False

The Energy of a Hopfield Net

- Define the *Energy* of the network as

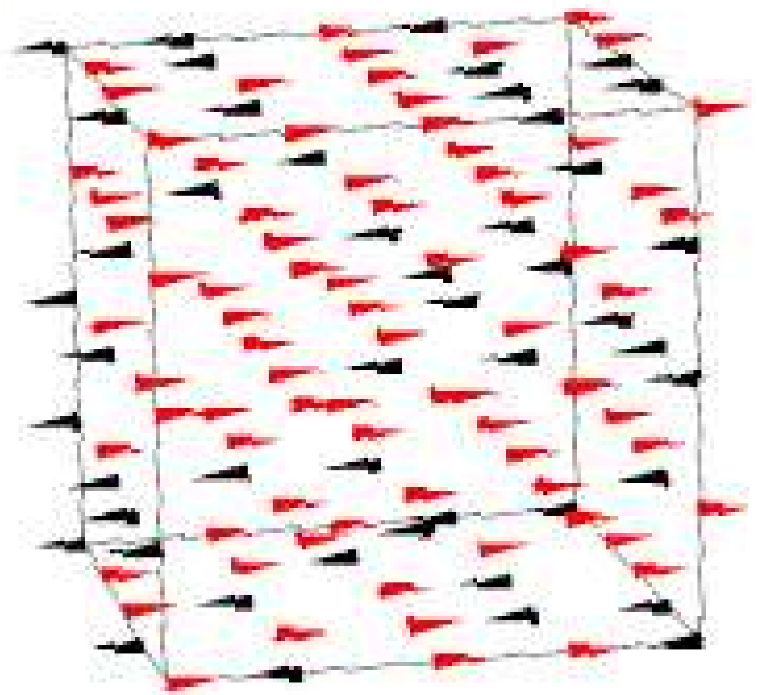
$$E = -\frac{1}{2}\left(\sum_{i,j\neq i} w_{ij}y_iy_j - \sum_i b_iy_i\right)$$

  – Just 0.5 times the negative of D

- The evolution of a Hopfield network constantly decreases its energy

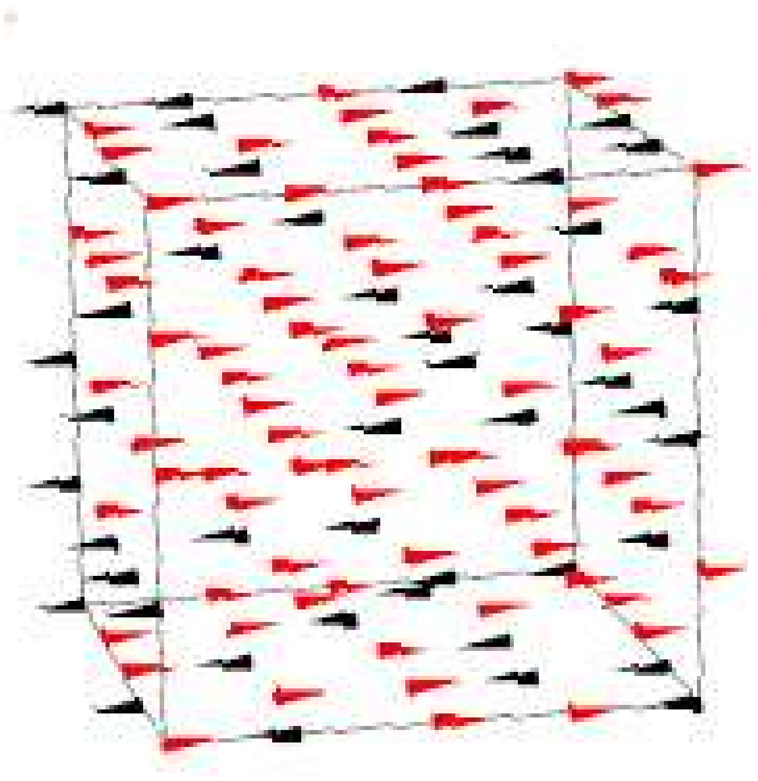
- Where did this “energy” concept suddenly sprout from?

Analogy: Spin Glass



- Magnetic diploes in a disordered magnetic material
- Each dipole tries to *align* itself to the local field
  - In doing so it may flip
- This will change fields at *other* dipoles
  - Which may flip
- Which changes the field at the current dipole…

Analogy: Spin Glasses



Total field at current dipole:

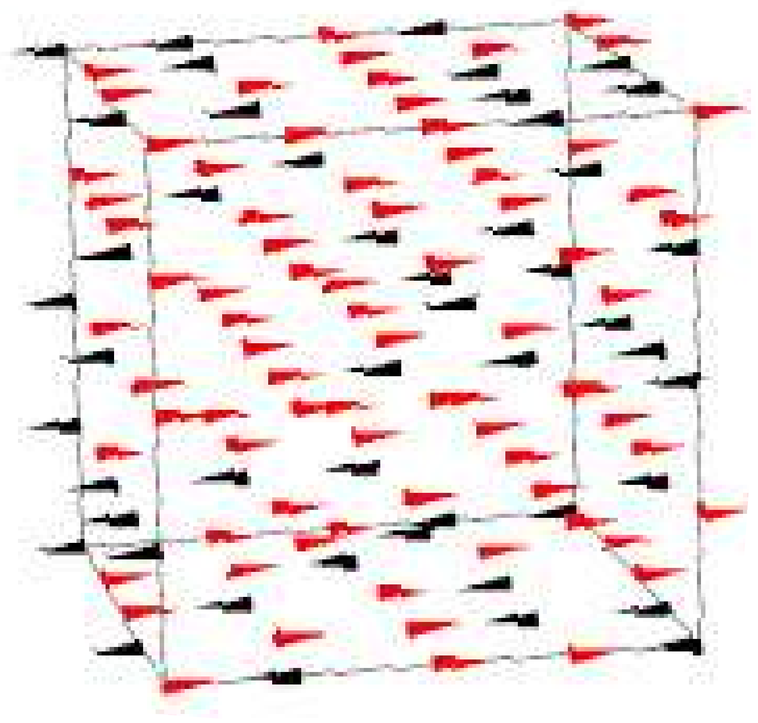
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

intrinsic

external

- p_i is vector position of i-th dipole
- The field at any dipole is the sum of the field contributions of all other dipoles
- The contribution of a dipole to the field at any point depends on interaction J
  - Derived from the "Ising" model for magnetic materials (Ising and Lenz, 1924)

Analogy: Spin Glasses



Total field at current dipole:

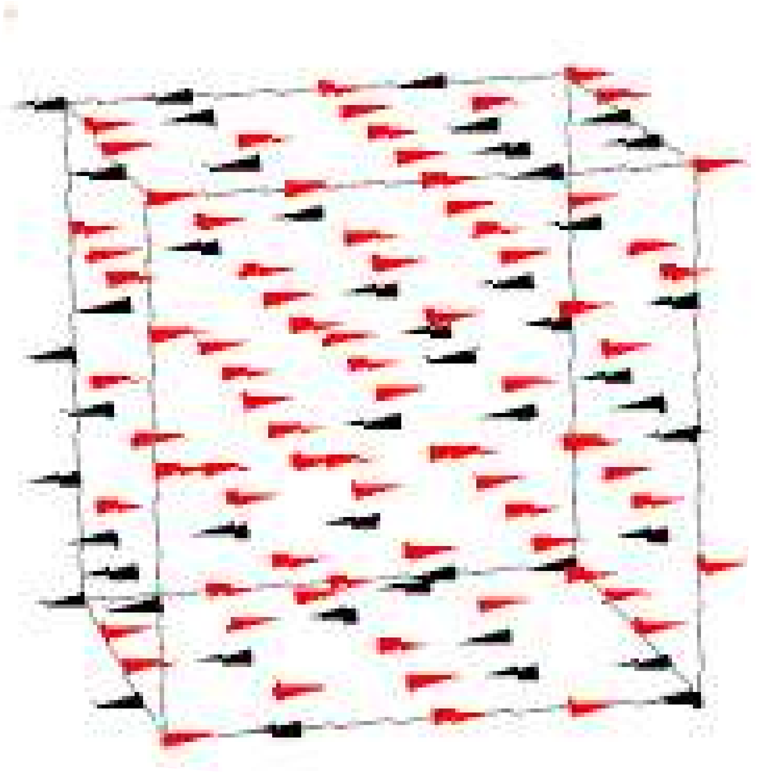
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

Response of current dipole

$$x_i = \begin{cases} x_i \ if \ sign\big(x_i \ f(p_i)\big) = 1 \\ -x_i \ otherwise \end{cases}$$

- A Dipole flips if it is misaligned with the field in its location

Analogy: Spin Glasses



Total field at current dipole:

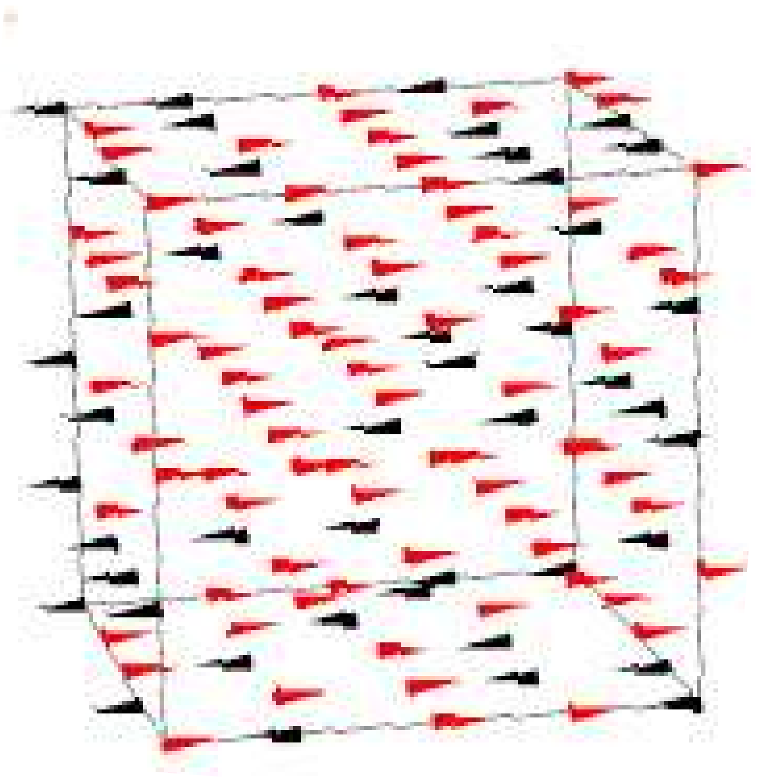
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

Response of current dipole

$$x_i = \begin{cases} x_i \; if \; sign\left(x_i \, f(p_i)\right) = 1 \\ -x_i \; otherwise \end{cases}$$

- Dipoles will keep flipping
  - A flipped dipole changes the field at other dipoles
    - Some of which will flip
  - Which will change the field at the current dipole
    - Which may flip
  - Etc..

Analogy: Spin Glasses



Total field at current dipole:

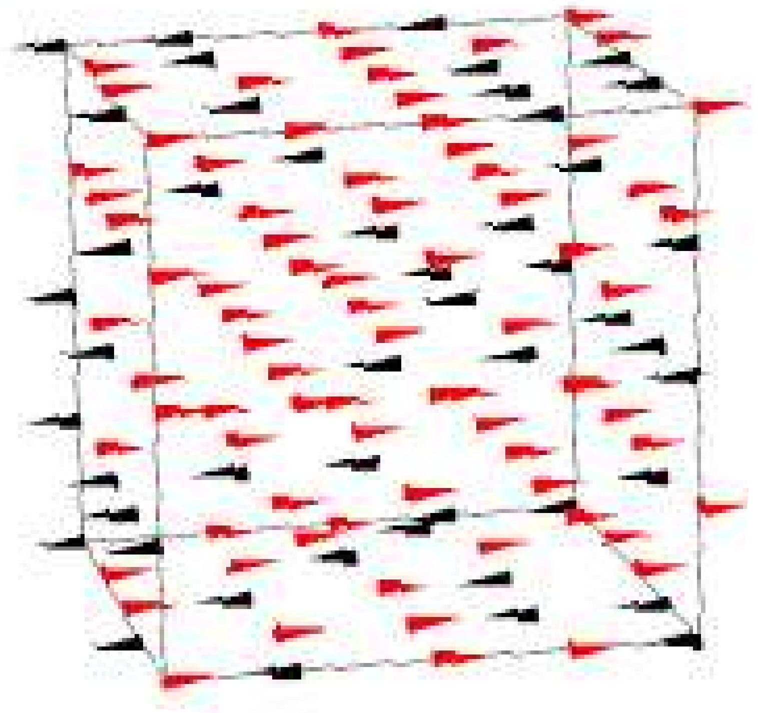
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

Response of current dipole

$$x_i = \begin{cases} x_i \; if \; sign(x_i \, f(p_i)) = 1 \\ -x_i \; otherwise \end{cases}$$

- When will it stop???

Analogy: Spin Glasses



Total field at current dipole:

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

Response of current dipole

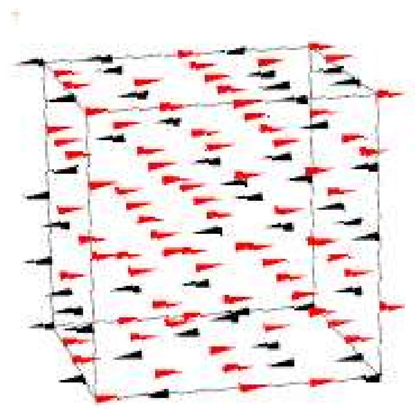
$$x_i = \begin{cases} x_i \; if \; sign\big(x_i \, f(p_i)\big) = 1 \\ -x_i \; otherwise \end{cases}$$

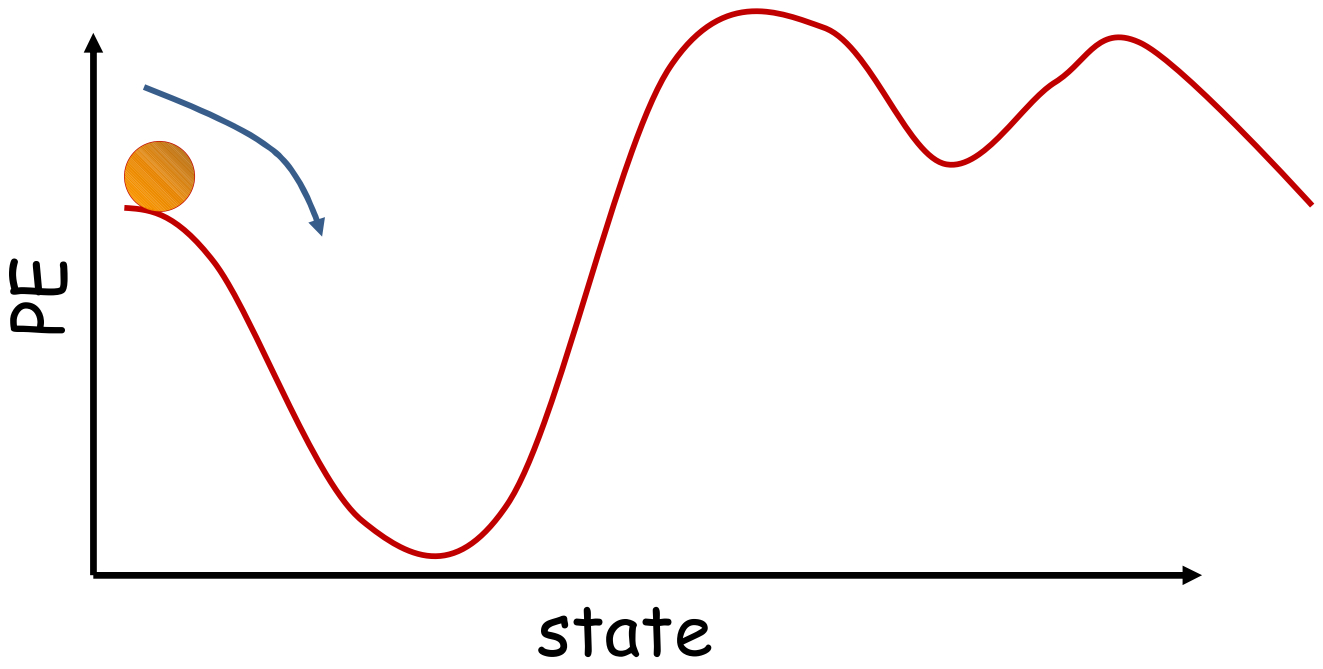
- The "Hamiltonian" (total energy) of the system

$$E = -\frac{1}{2}\sum_i x_i f(p_i) = -\sum_i \sum_{j>i} J_{ji} x_i x_j - \sum_i b_i x_i$$

- The system *evolves* to minimize the energy
  - Dipoles stop flipping if any flips result in increase of energy

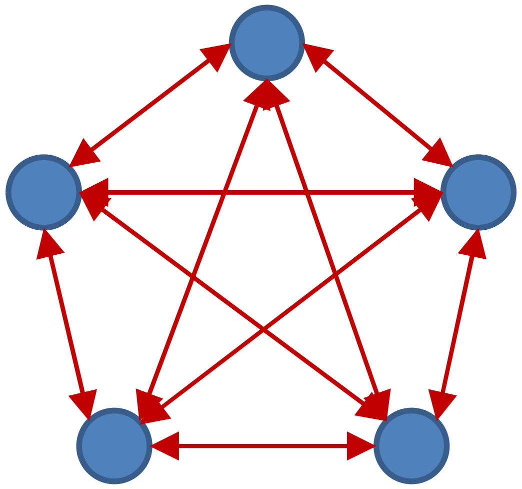
Spin Glasses





- The system stops at one of its *stable* configurations
  - Where energy is a local minimum
- Any small jitter from this stable configuration *returns it* to the stable configuration
  - I.e. the system *remembers* its stable state and returns to it

Hopfield Network



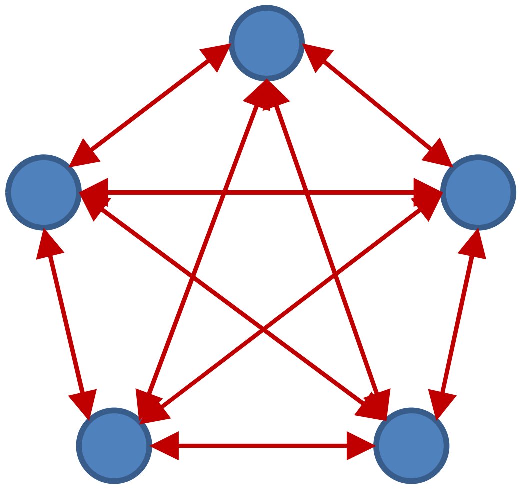
$$y_i = \Theta\left(\sum_{j\neq i} w_{ji} y_j + b_i\right)$$

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

$$E = -\frac{1}{2}\left(\sum_{i,j\neq i} w_{ij} y_i y_j + \sum_i b_i y_i\right)$$

- This is analogous to the potential energy of a spin glass
  - The system will evolve until the energy hits a local minimum

Hopfield Network



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

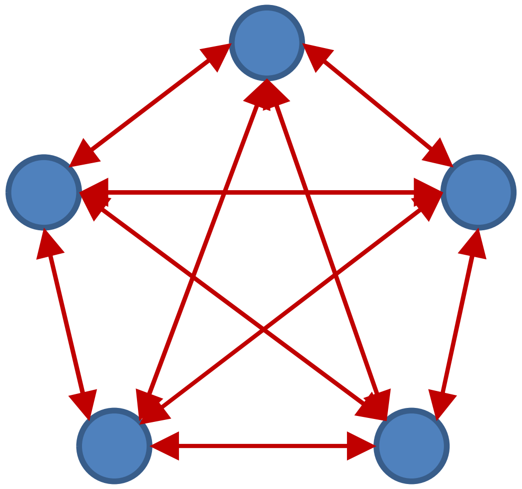
$$\Theta(z) = \begin{cases} +1 \; if \; z > 0 \\ -1 \; if \; z \leq 0 \end{cases}$$

The bias is equivalent to having a single extra unit pegged at 1

We will not always explicitly show the bias

Often, in fact, a bias is not used, although in our case we are just being lazy in not showing it explicitly

Hopfield Network



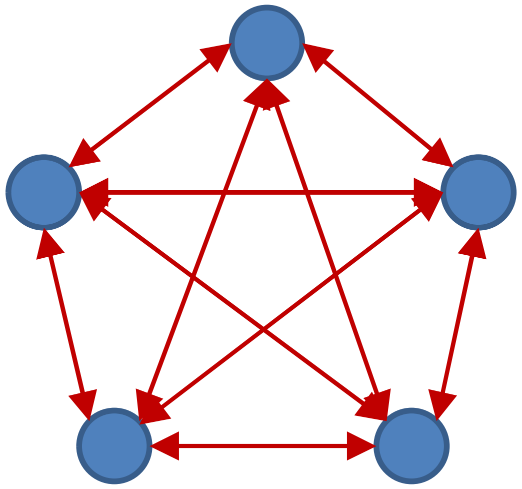
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j\right)$$

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

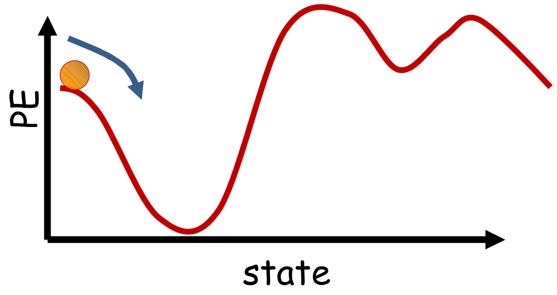
$$E = -\frac{1}{2} \sum_{i,j<i} w_{ij} y_i y_j$$

- This is analogous to the potential energy of a spin glass
  - The system will evolve until the energy hits a local minimum
    - Above equation is a factor of 0.5 off from earlier definition for conformity with thermodynamic system

Evolution

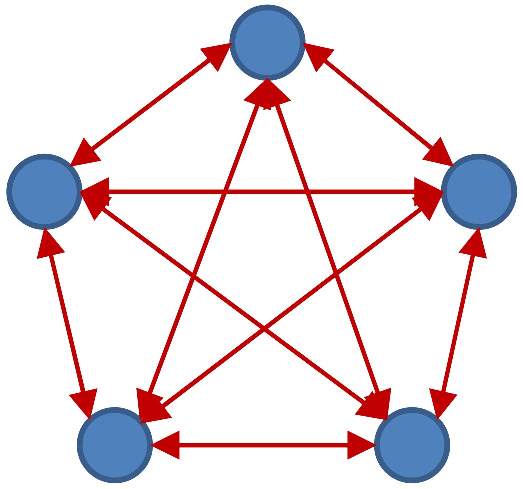


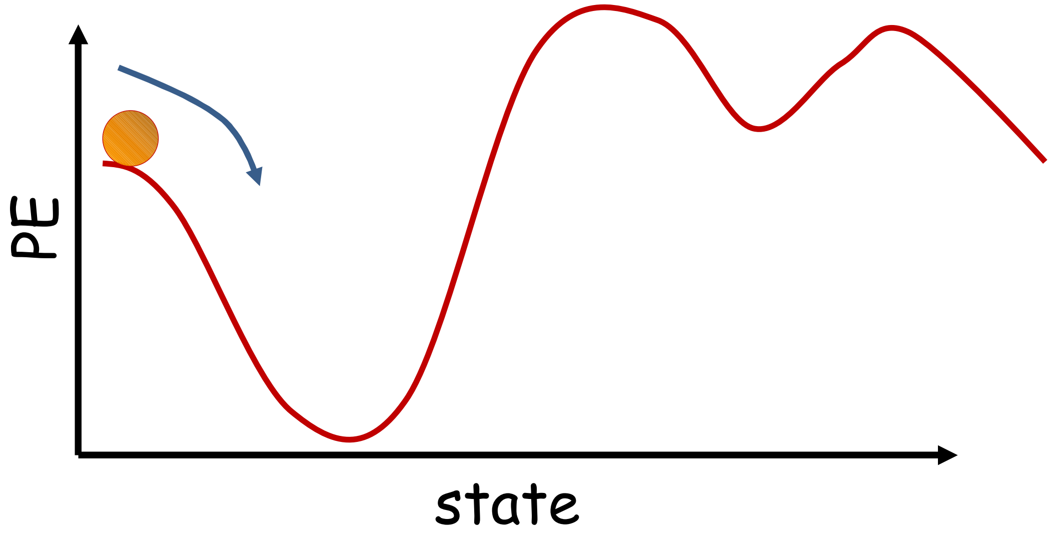
$$E = -\frac{1}{2}\sum_{i,j<i} w_{ij}y_iy_j$$



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

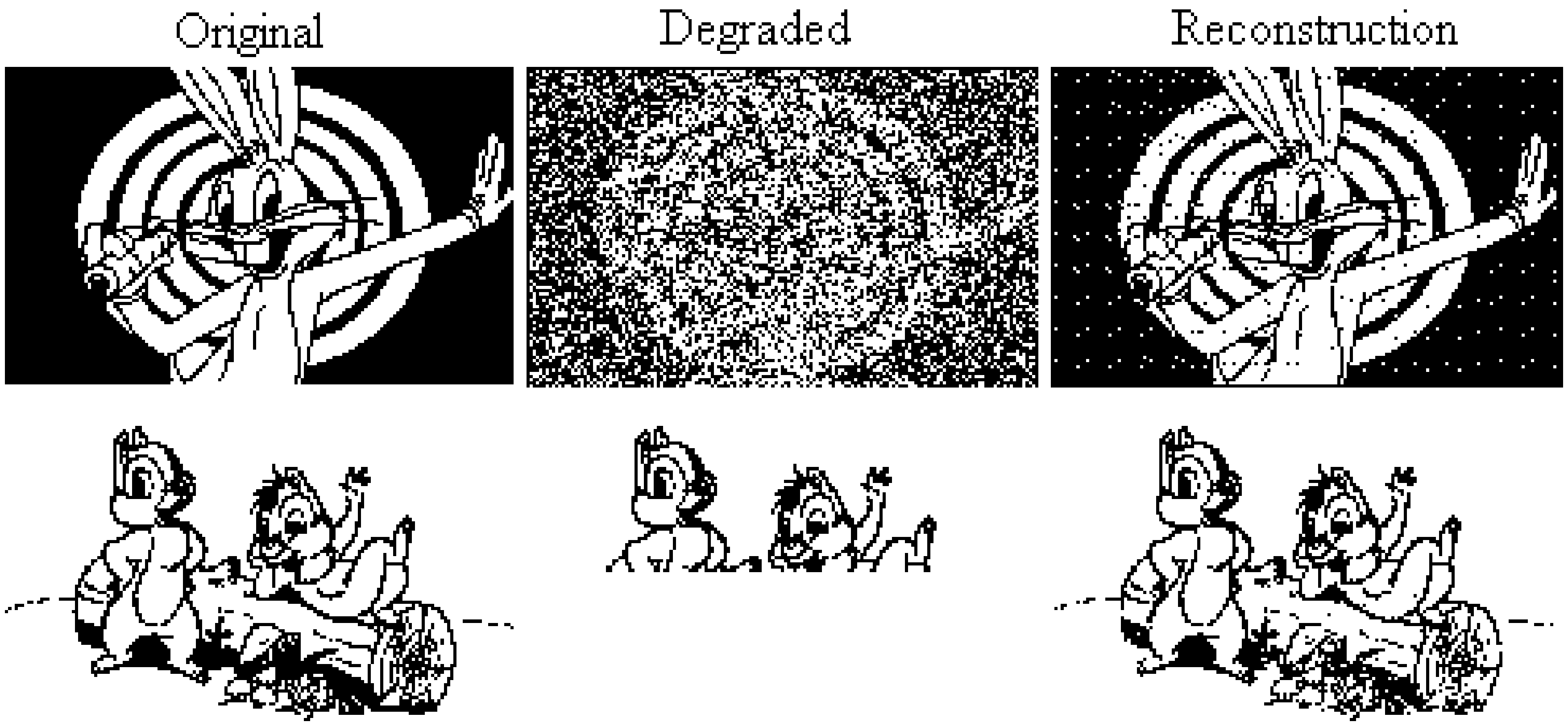
Content-addressable memory





- Each of the minima is a "stored" pattern
  - If the network is initialized close to a stored pattern, it will inevitably evolve to the pattern
- **This is a *content addressable memory***
  - Recall memory content from partial or corrupt values
- Also called ***associative memory***

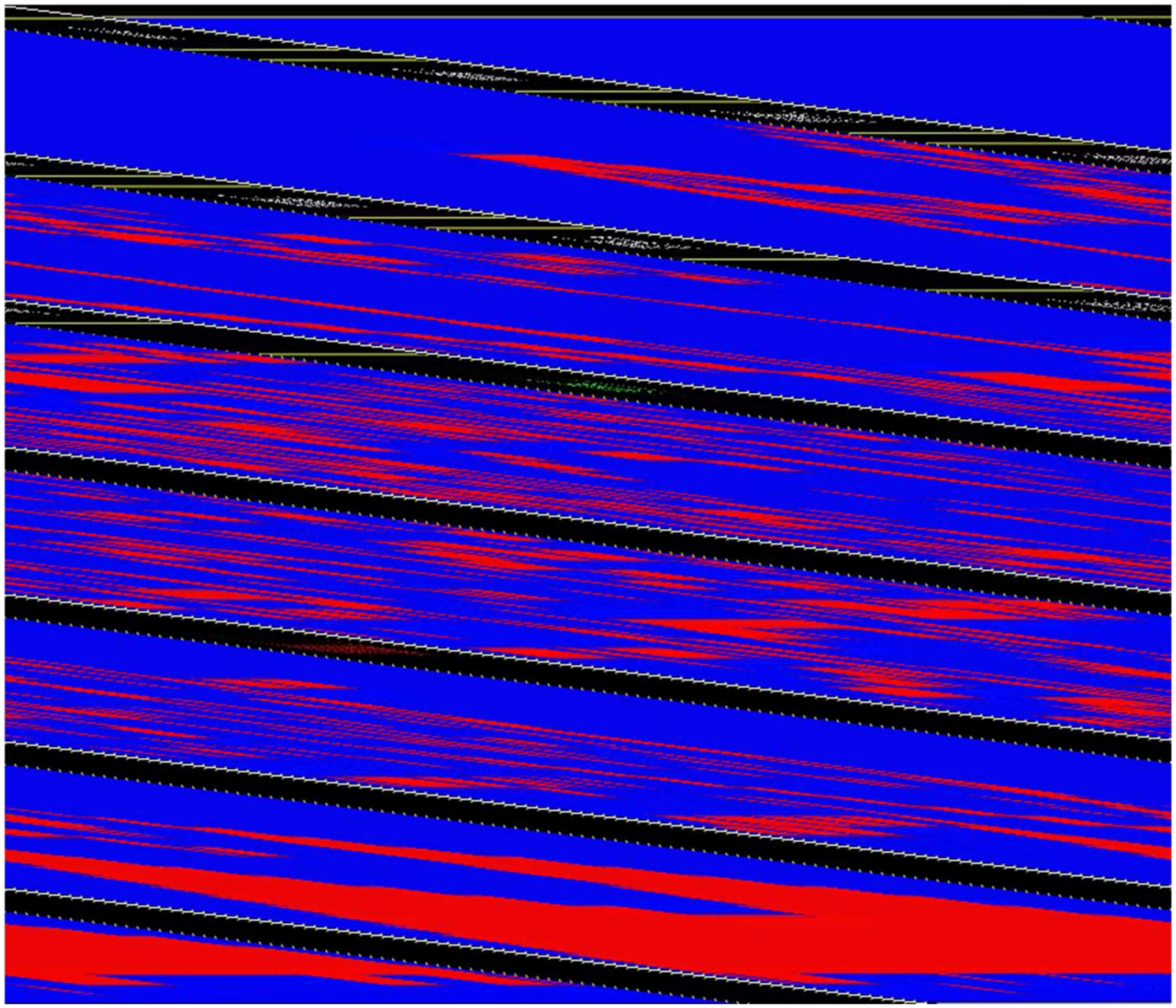
Examples: Content addressable memory



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

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

Hopfield net examples



Computational algorithm

1. Initialize network with initial pattern

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

2. Iterate until convergence

$$y_i(t+1) = \Theta\left(\sum_{j \neq i} w_{ji} y_j\right), \qquad 0 \le i \le N-1$$

- Very simple
- Updates can be done sequentially, or all at once
- Convergence

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

does not change significantly any more

Computational algorithm

1. Initialize network with initial pattern

$$\mathbf{y} = \mathbf{x}, \qquad 0 \leq i \leq N-1$$

2. Iterate until convergence

$$\mathbf{y} = \Theta(\mathbf{W}\mathbf{y})$$

Writing $\mathbf{y} = [y_1, y_2, y_3, \cdots, y_N]^\top$
and arranging the weights as a matrix $\mathbf{W}$

- Very simple
- Updates can be done sequentially, or all at once
- Convergence

$$E = -0.5\mathbf{y}^\top \mathbf{W}\mathbf{y}$$

does not change significantly any more

Story so far

- A Hopfield network is a loopy binary network with symmetric connections
  - Neurons try to align themselves to the local field caused by other neurons

- Given an initial configuration, the patterns of neurons in the net will evolve until the “energy” of the network achieves a local minimum
  - The evolution will be monotonic in total energy
  - The dynamics of a Hopfield network mimic those of a spin glass
  - The network is symmetric: if a pattern Y is a local minimum, so is $-Y$

- The network acts as a *content-addressable* memory
  - If you initialize the network with a somewhat damaged version of a local-minimum pattern, it will evolve into that pattern
  - Effectively “recalling” the correct pattern, from a damaged/incomplete version

Poll 2

Mark all that are correct about Hopfield nets

- The network activations evolve until the energy of the net arrives at a local minimum
- Hopfield networks are a form of content addressable memory
- It is possible to analytically determine the stored memories by inspecting the weights matrix

Poll 2

Mark all that are correct about Hopfield nets

- **The network activations evolve until the energy of the net arrives at a local minimum**
- **Hopfield networks are a form of content addressable memory**
- It is possible to analytically determine the stored memories by inspecting the weights matrix

Issues

- How do we make the network store *a specific* pattern or set of patterns?
- How many patterns can we store?
- How to “retrieve” patterns better..

Issues

- How do we make the network store *a specific* pattern or set of patterns?

- How many patterns can we store?

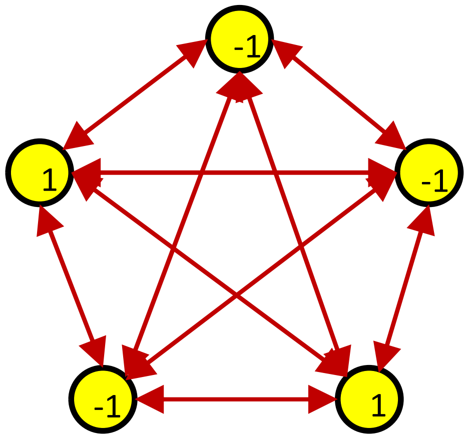
- How to “retrieve” patterns better..

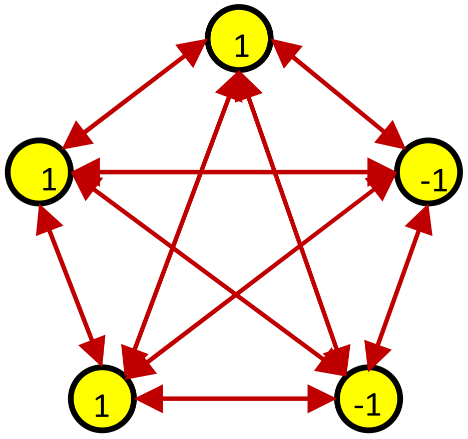
How do we remember a *specific* pattern?

- How do we teach a network to "remember" this image

- For an image with N pixels we need a network with N neurons
- Every neuron connects to every other neuron
- Weights are symmetric (not mandatory)
- $\frac{N(N-1)}{2}$ weights in all

Storing a pattern

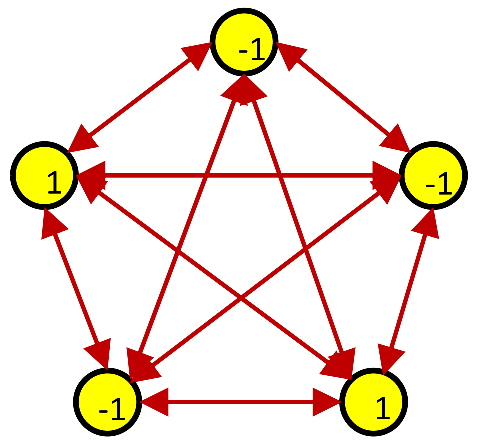


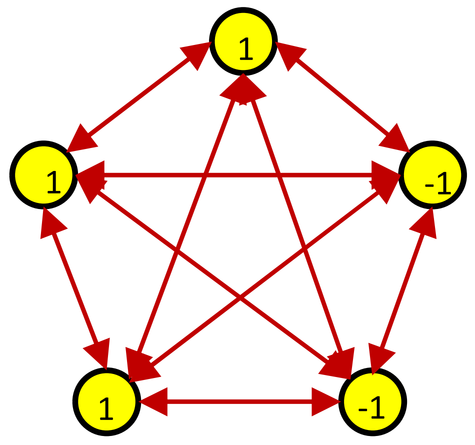


$$E = -\sum_i \sum_{j<i} w_{ji} y_j y_i$$

- Design $\{w_{ij}\}$ such that the energy is a local minimum at the desired $P = \{y_i\}$
  - Recall: the evolution is $Y \leftarrow sign(WY)$
  - For static patterns, $sign(WY) = Y$
  - For *stable* patterns $sign(W(Y+\epsilon)) = Y$ for small ϵ

Storing a pattern

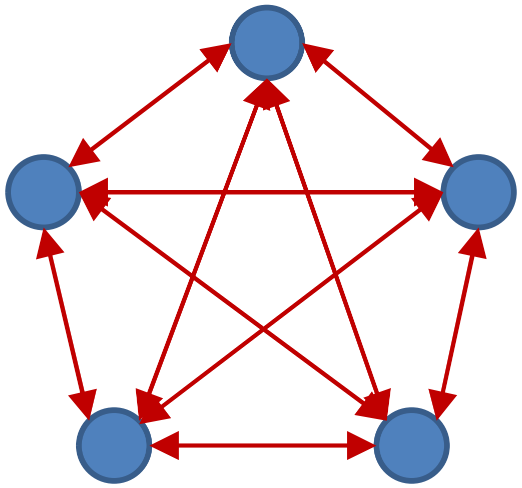




$$E = -\sum_{i}\sum_{j<i} w_{ji} y_j y_i$$

- Math: the 'stable' patterns must be close to the Eigen vectors of W
  - For a network with N neurons, we can store at most N patterns reliably
  - For the rest, $sign(WY)$ may end up at a different pattern

Consider the energy function

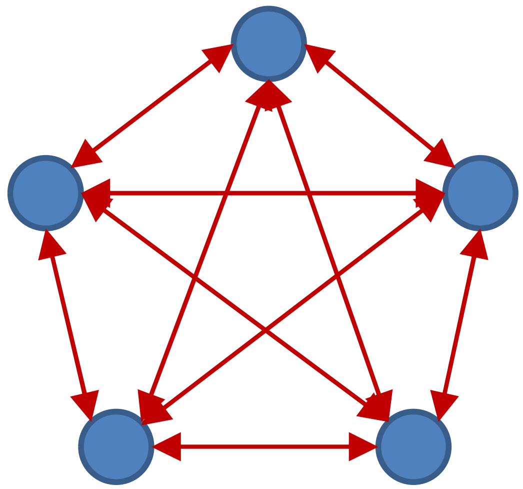


$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y}$$

- This must be *maximally* low for target patterns
- Must be *maximally* high for *all other patterns*
  - So that they are unstable and evolve into one of the target patterns

Estimating the Network

$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y}$$



- Estimate $\mathbf{W}$ (and $\mathbf{b}$) such that
  - E is minimized for $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_P$
  - E is maximized for all other $\mathbf{y}$
- Caveat: Unrealistic to expect to store more than N patterns, but can we make those N patterns *memorable*

Optimizing W (and b)

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

$$\widehat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{\mathbf{y}\in\mathbf{Y}_P} E(\mathbf{y})$$

The bias can be captured by another fixed-value component

- Minimize total energy of target patterns
  - Problem with this?

Optimizing W

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

$$\widehat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{\mathbf{y}\in\mathbf{Y}_P} E(\mathbf{y}) - \sum_{\mathbf{y}\notin\mathbf{Y}_P} E(\mathbf{y})$$

- Minimize total energy of target patterns
- Maximize the total energy of all *non-target* patterns

Optimizing W

$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} \qquad \widehat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{\mathbf{y}\in\mathbf{Y}_P} E(\mathbf{y}) - \sum_{\mathbf{y}\notin\mathbf{Y}_P} E(\mathbf{y})$$

- Simple gradient descent:

$$\mathbf{W} = \mathbf{W} + \eta\left(\sum_{\mathbf{y}\in\mathbf{Y}_P} \mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y}\notin\mathbf{Y}_P} \mathbf{y}\mathbf{y}^T\right)$$

Optimizing W

$$\mathbf{W} = \mathbf{W} + \eta\left(\sum_{\mathbf{y}\in\mathbf{Y}_P}\mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y}\notin\mathbf{Y}_P}\mathbf{y}\mathbf{y}^T\right)$$

- Can "emphasize" the importance of a pattern by repeating
  - More repetitions → greater emphasis

Optimizing W

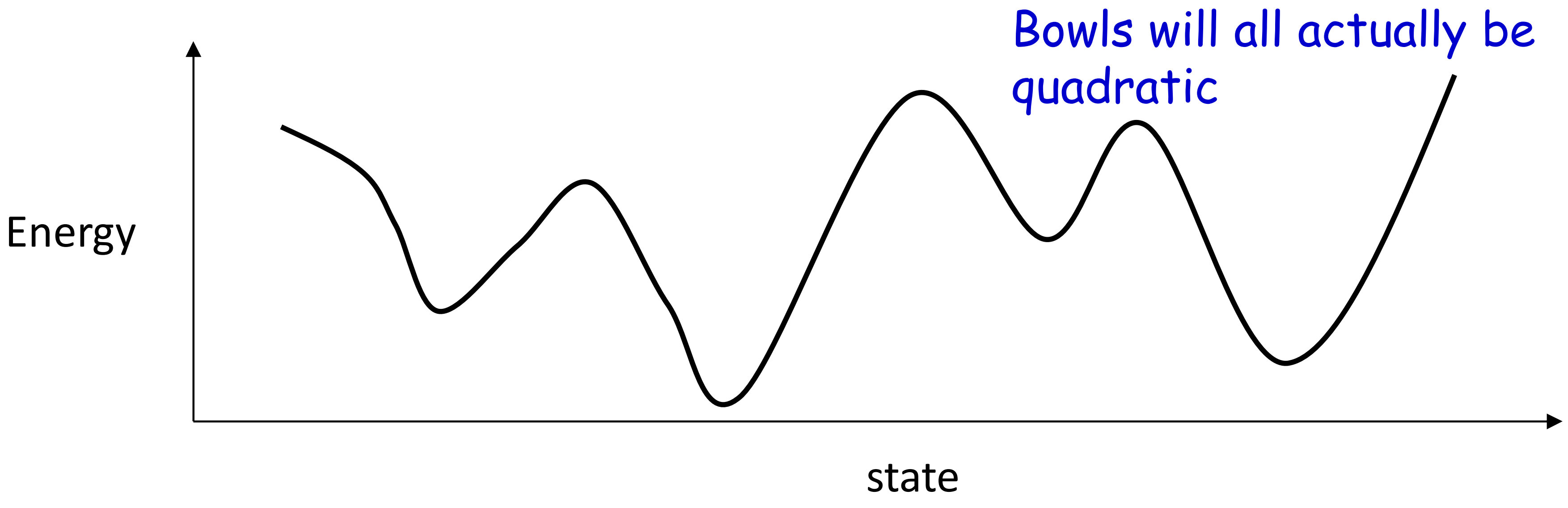
$$\mathbf{W} = \mathbf{W} + \eta\left(\sum_{\mathbf{y}\in\mathbf{Y}_P}\mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y}\notin\mathbf{Y}_P}\mathbf{y}\mathbf{y}^T\right)$$

- Can “emphasize” the importance of a pattern by repeating
  - More repetitions → greater emphasis
- How many of these?
  - Do we need to include *all* of them?
  - Are all equally important?

The training again..

$$\mathbf{W} = \mathbf{W} + \eta\left(\sum_{\mathbf{y}\in\mathbf{Y}_P}\mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y}\notin\mathbf{Y}_P}\mathbf{y}\mathbf{y}^T\right)$$

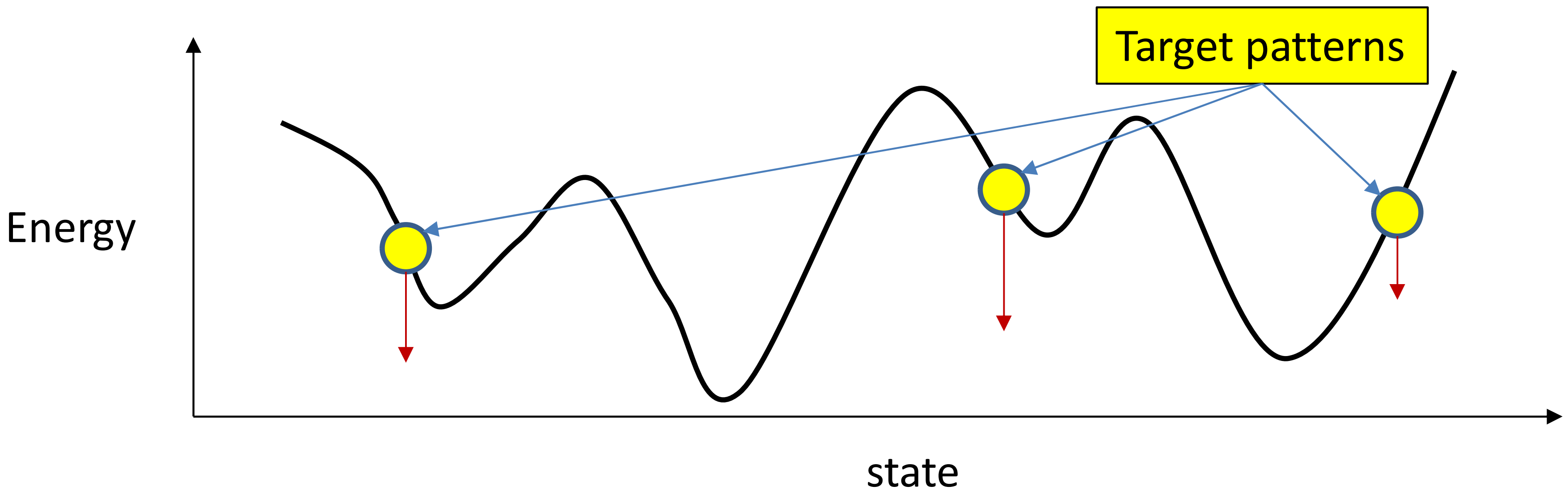
- Note the energy contour of a Hopfield network for any weight $\mathbf{W}$



The training again

$$\mathbf{W} = \mathbf{W} + \eta\left(\sum_{\mathbf{y}\in\mathbf{Y}_P} \mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y}\notin\mathbf{Y}_P} \mathbf{y}\mathbf{y}^T\right)$$

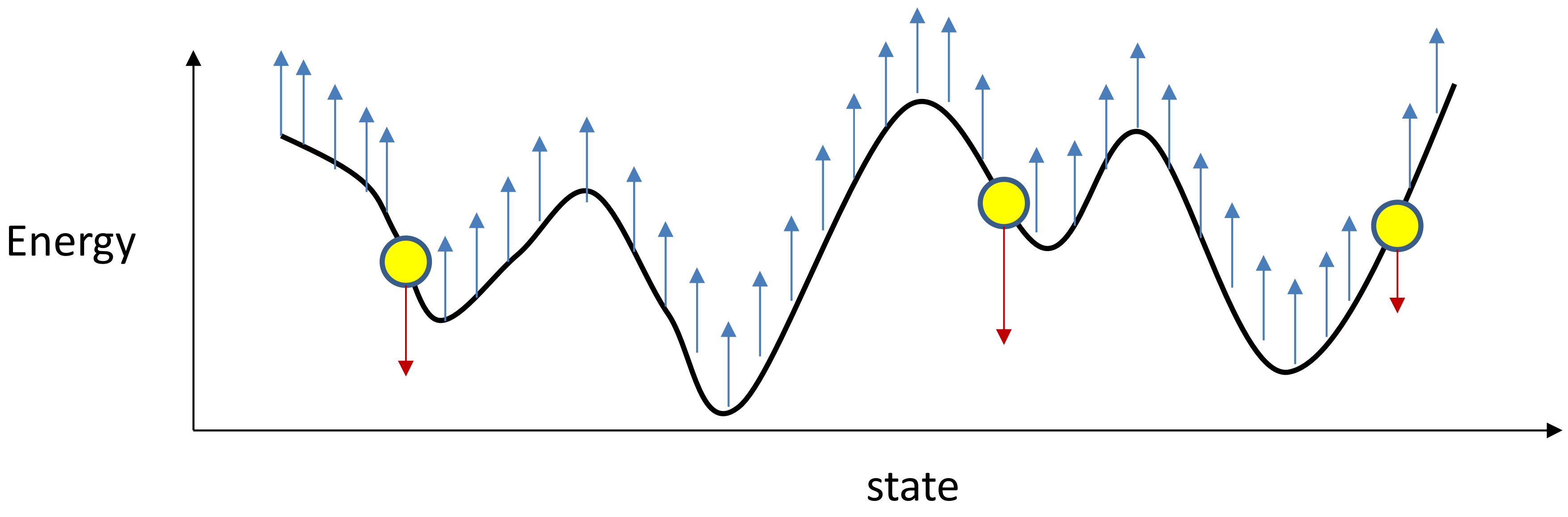
- The first term tries to *minimize* the energy at target patterns
  - Make them local minima
  - Emphasize more "important" memories by repeating them more frequently



The negative class

$$\mathbf{W} = \mathbf{W} + \eta\left(\sum_{\mathbf{y}\in\mathbf{Y}_P}\mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y}\notin\mathbf{Y}_P}\mathbf{y}\mathbf{y}^T\right)$$

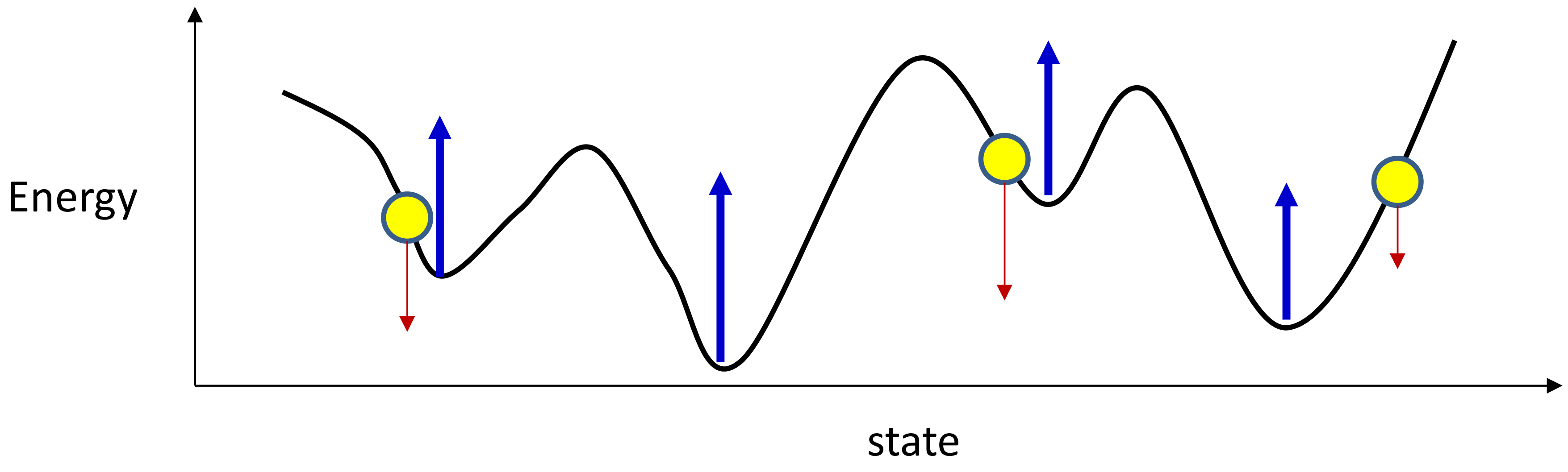
- The second term tries to "raise" all non-target patterns
  - Do we need to raise *everything*?



Option 1: Focus on the valleys

$$\mathbf{W} = \mathbf{W} + \eta\left(\sum_{\mathbf{y}\in\mathbf{Y}_P}\mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y}\notin\mathbf{Y}_P \& \mathbf{y}=valley}\mathbf{y}\mathbf{y}^T\right)$$

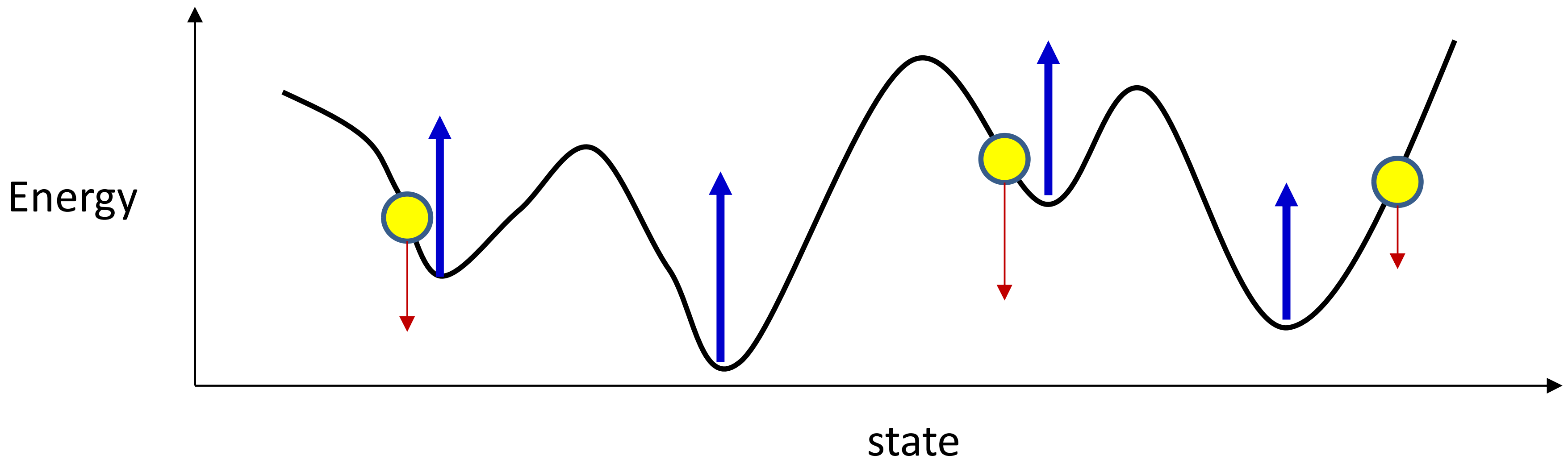
- Focus on raising the valleys
  - If you raise *every* valley, eventually they'll all move up above the target patterns, and many will even vanish



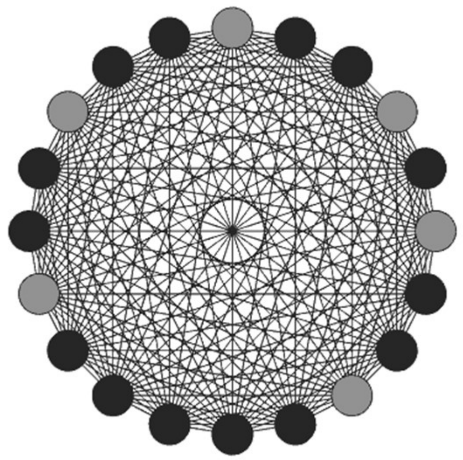
Identifying the valleys..

$$\mathbf{W} = \mathbf{W} + \eta\left(\sum_{\mathbf{y}\in\mathbf{Y}_P} \mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y}\notin\mathbf{Y}_P \& \mathbf{y}=valley} \mathbf{y}\mathbf{y}^T\right)$$

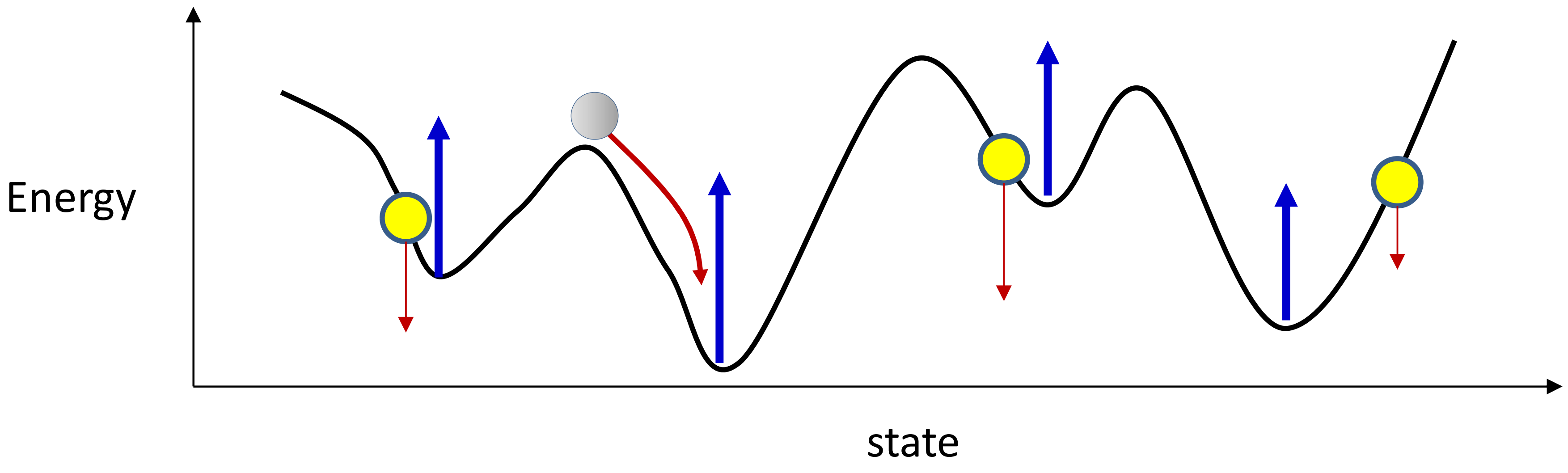
- Problem: How do you identify the valleys for the current $\mathbf{W}$?



Identifying the valleys..



- Initialize the network randomly and let it evolve
  - It will settle in a valley



Training the Hopfield network

$$\mathbf{W} = \mathbf{W} + \eta \left(\sum_{\mathbf{y} \in \mathbf{Y}_P} \mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y} \notin \mathbf{Y}_P \& \mathbf{y} = valley} \mathbf{y}\mathbf{y}^T \right)$$

- Initialize $\mathbf{W}$
- Compute the total outer product of all target patterns
  - More important patterns presented more frequently
- Randomly initialize the network several times and let it evolve
  - And settle at a valley
- Compute the total outer product of valley patterns
- Update weights

Training the Hopfield network: SGD version

$$\mathbf{W} = \mathbf{W} + \eta \left(\sum_{\mathbf{y} \in \mathbf{Y}_P} \mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y} \notin \mathbf{Y}_P \& \mathbf{y} = valley} \mathbf{y}\mathbf{y}^T \right)$$

- Initialize $\mathbf{W}$
- Do until convergence, satisfaction, or death from boredom:
  - Sample a target pattern $\mathbf{y}_p$
    - Sampling frequency of pattern must reflect importance of pattern
  - Randomly initialize the network and let it evolve
    - And settle at a valley $\mathbf{y}_v$
  - Update weights
    - $\mathbf{W} = \mathbf{W} + \eta\left(\mathbf{y}_p\mathbf{y}_p^T - \mathbf{y}_v\mathbf{y}_v^T\right)$

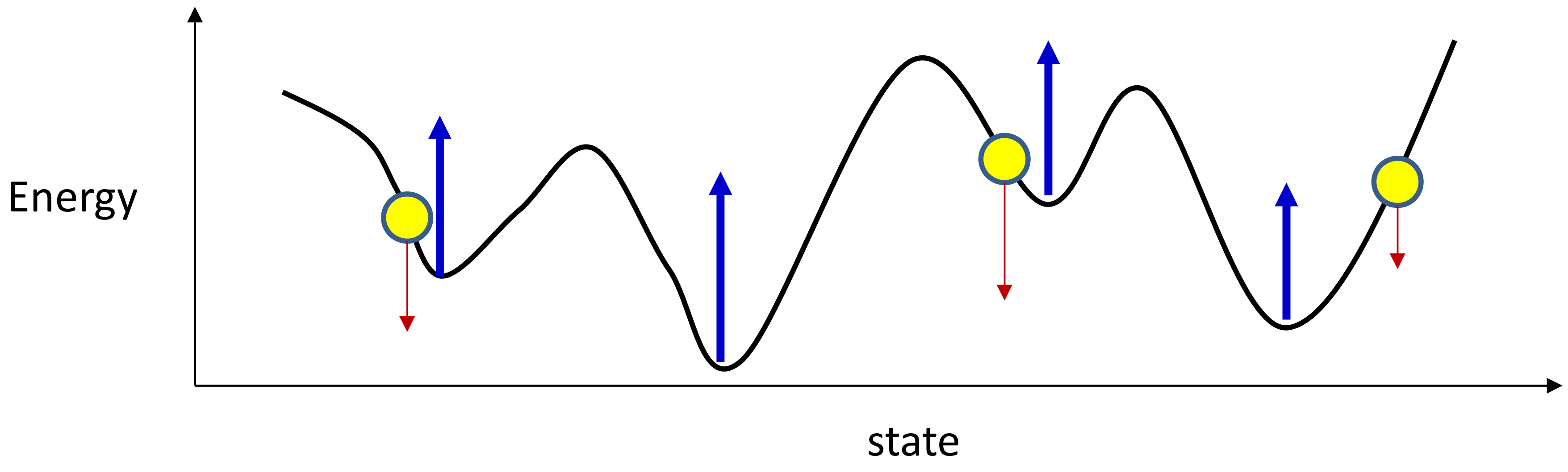
Training the Hopfield network

$$\mathbf{W} = \mathbf{W} + \eta\left(\sum_{\mathbf{y}\in\mathbf{Y}_P} \mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y}\notin\mathbf{Y}_P \& \mathbf{y}=valley} \mathbf{y}\mathbf{y}^T\right)$$

- Initialize $\mathbf{W}$
- Do until convergence, satisfaction, or death from boredom:
  - Sample a target pattern $\mathbf{y}_p$
    - Sampling frequency of pattern must reflect importance of pattern
  - Randomly initialize the network and let it evolve
    - And settle at a valley $\mathbf{y}_v$
  - Update weights
    - $\mathbf{W} = \mathbf{W} + \eta\left(\mathbf{y}_p\mathbf{y}_p^T - \mathbf{y}_v\mathbf{y}_v^T\right)$

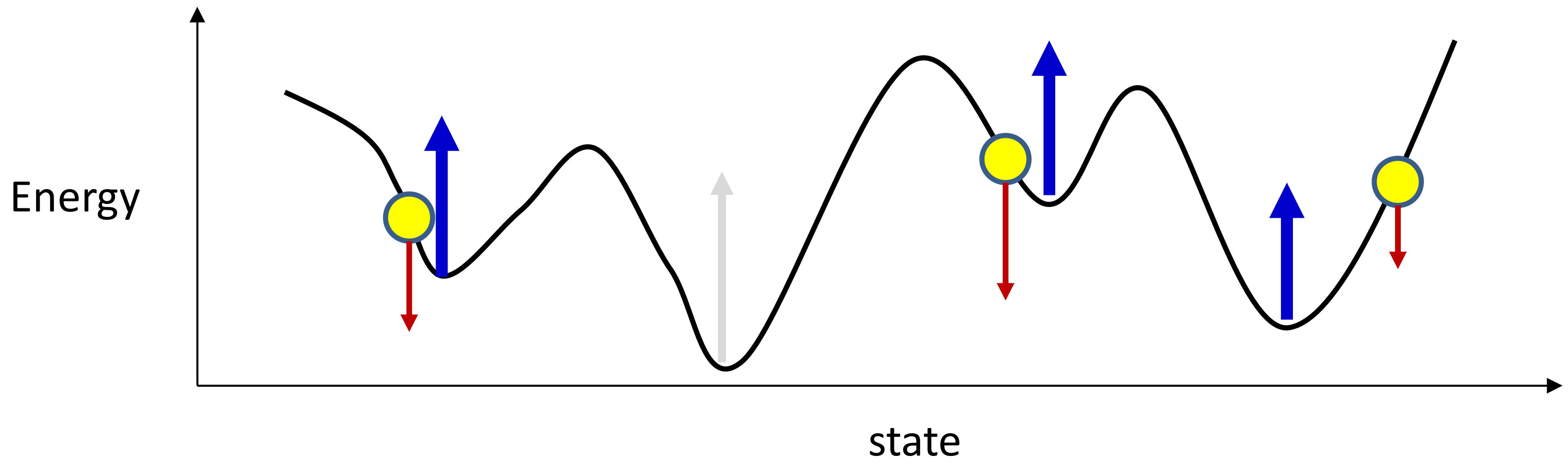
Which valleys?

- Should we *randomly* sample valleys?
  - Are all valleys equally important?

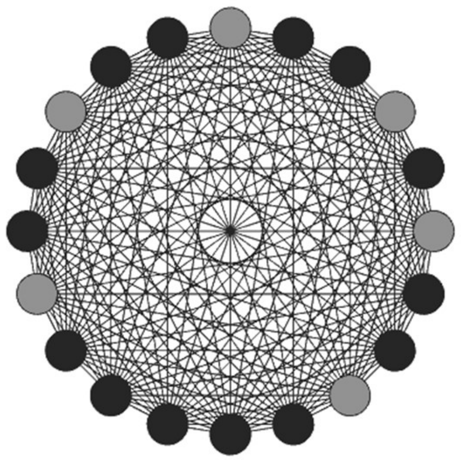


Which valleys?

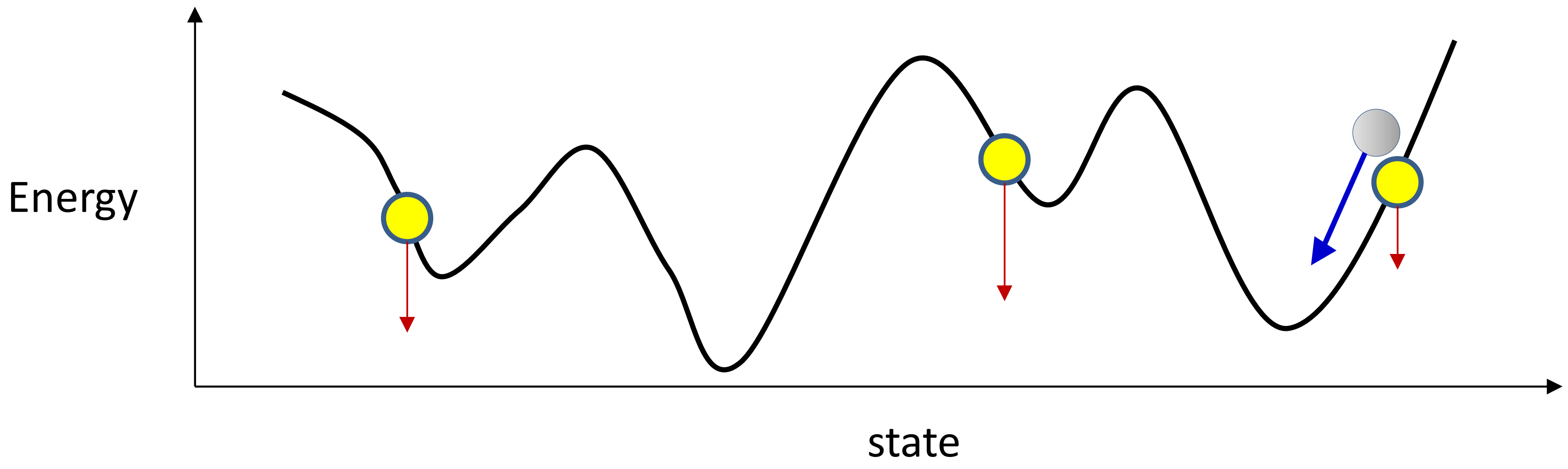
- Should we *randomly* sample valleys?
  – Are all valleys equally important?
- Major requirement: memories must be stable
  – They *must* be broad valleys
- Spurious valleys in the neighborhood of memories are more important to eliminate



Identifying the valleys..



- Initialize the network at valid memories and let it evolve
  - It will settle in a valley. If this is not the target pattern, raise it



Training the Hopfield network

$$\mathbf{W} = \mathbf{W} + \eta\left(\sum_{\mathbf{y}\in\mathbf{Y}_P} \mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y}\notin\mathbf{Y}_P \& \mathbf{y}=valley} \mathbf{y}\mathbf{y}^T\right)$$

- Initialize $\mathbf{W}$
- Compute the total outer product of all target patterns
  - More important patterns presented more frequently
- Initialize the network with each target pattern and let it evolve
  - And settle at a valley
- Compute the total outer product of valley patterns
- Update weights

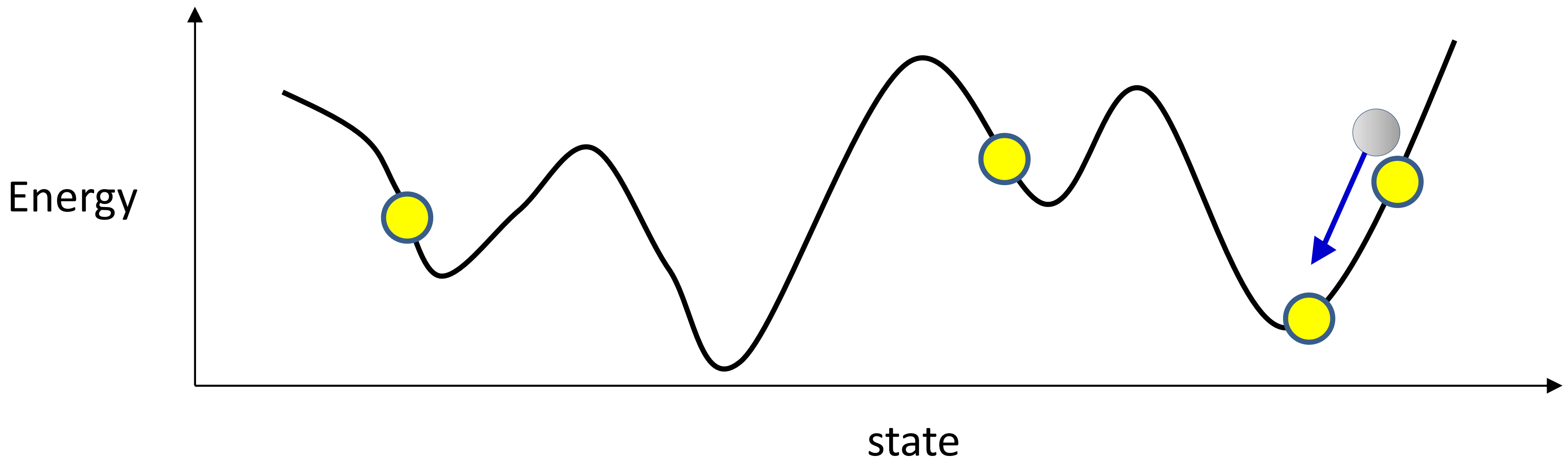
Training the Hopfield network: SGD version

$$\mathbf{W} = \mathbf{W} + \eta \sum_{\mathbf{y} \in \mathbf{Y}_P} \left(\mathbf{y}\mathbf{y}^T - \mathbf{y}_\mathbf{v}\mathbf{y}_v^T\right)$$

- Initialize $\mathbf{W}$
- Do until convergence, satisfaction, or death from boredom:
  - Sample a target pattern $\mathbf{y}_p$
    - Sampling frequency of pattern must reflect importance of pattern
  - Initialize the network at $\mathbf{y}_p$ and let it evolve
    - And settle at a valley $\mathbf{y}_v$
  - Update weights
    - $\mathbf{W} = \mathbf{W} + \eta\left(\mathbf{y}_p\mathbf{y}_p^T - \mathbf{y}_v\mathbf{y}_v^T\right)$

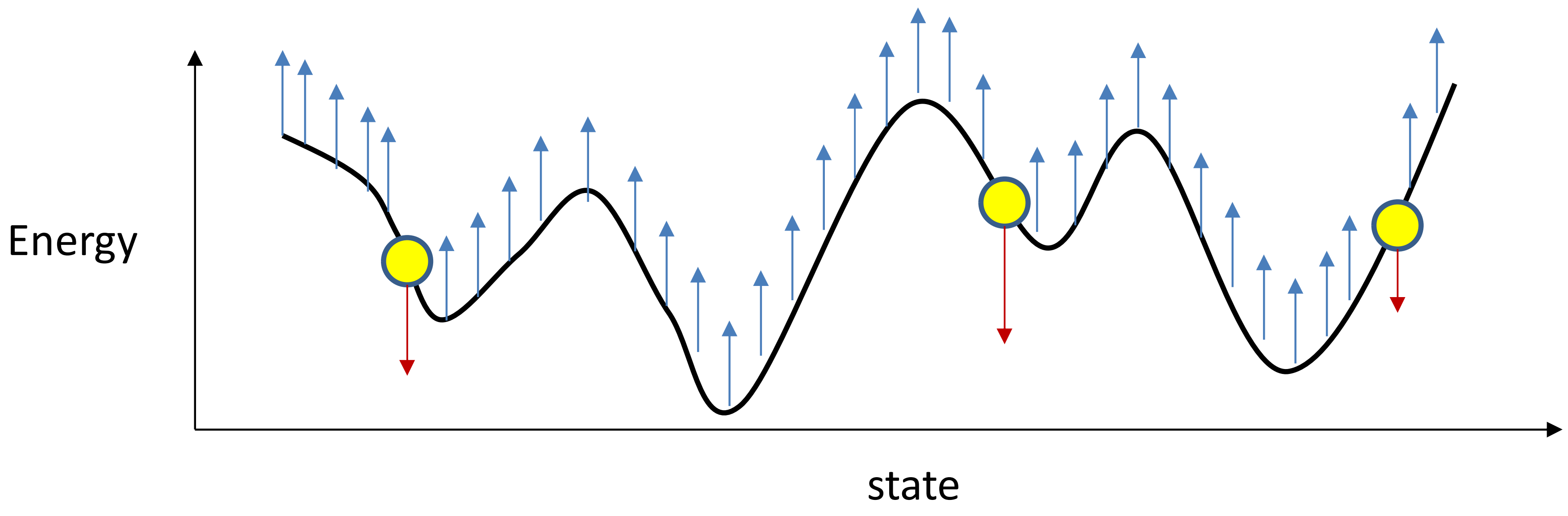
A possible problem

- What if there's another target pattern downvalley
  - Raising it will destroy a better-represented or stored pattern!



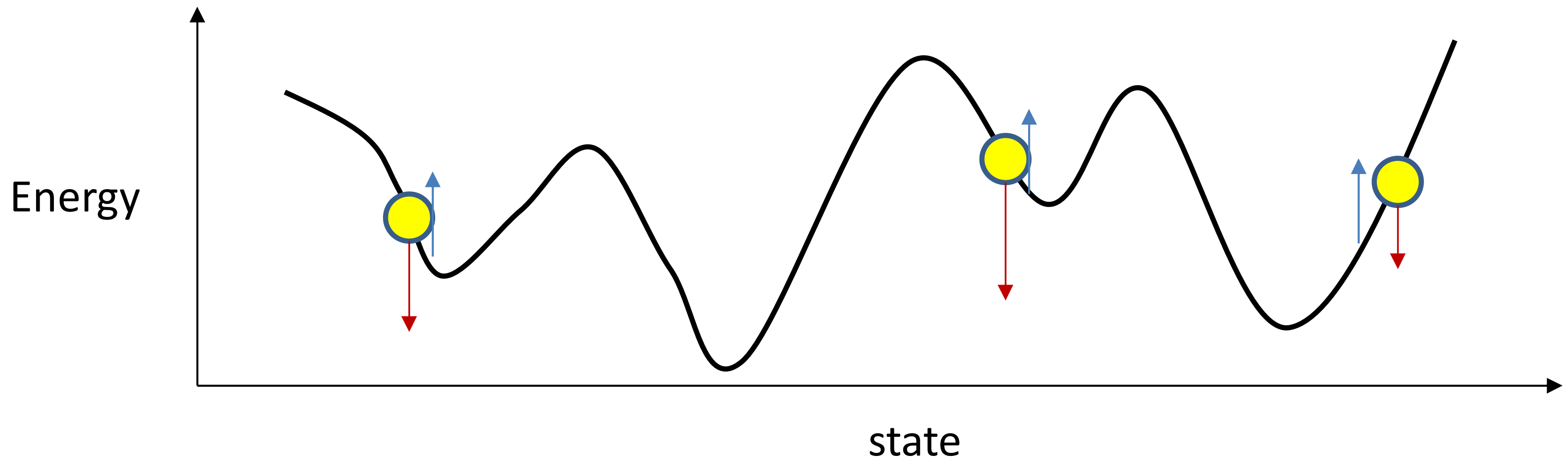
A related issue

- Really no need to raise the entire surface, or even every valley



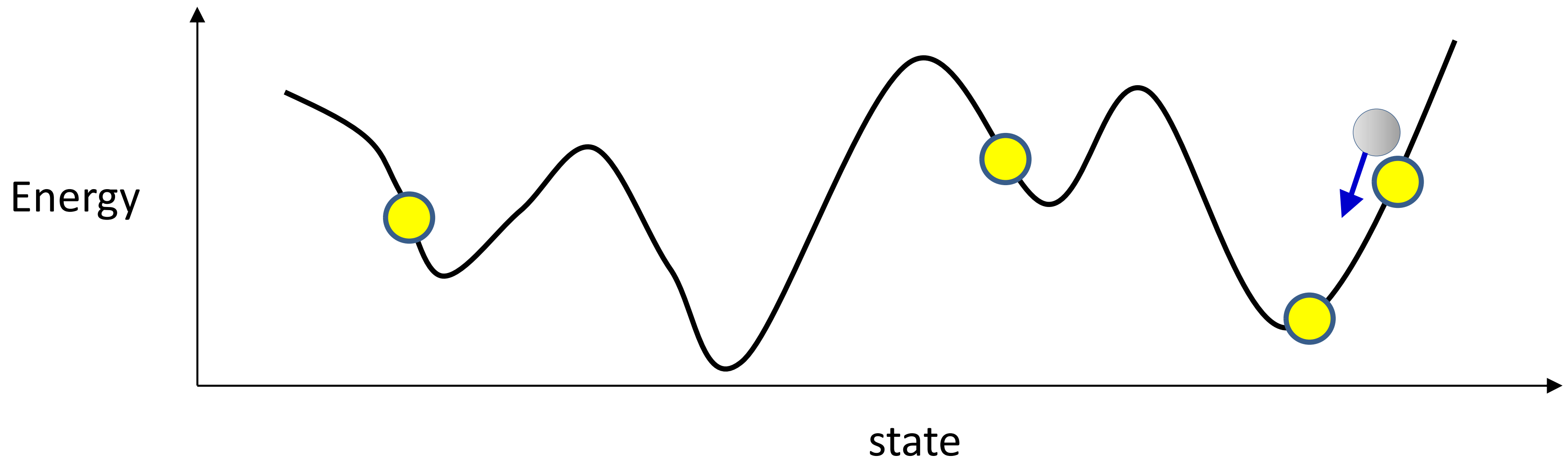
A related issue

- Really no need to raise the entire surface, or even every valley
- Raise the *neighborhood* of each target memory
  - Sufficient to make the memory a valley
  - The broader the neighborhood considered, the broader the valley



Raising the neighborhood

- Starting from a target pattern, let the network evolve only a few steps
  - Try to raise the resultant location
- Will raise the neighborhood of targets
- Will avoid problem of down-valley targets



Training the Hopfield network: SGD version

$$\mathbf{W} = \mathbf{W} + \eta \sum_{\mathbf{y}\in\mathbf{Y}_P} \left(\mathbf{y}\mathbf{y}^T - \mathbf{y}_d\mathbf{y}_d^T\right)$$

- Initialize $\mathbf{W}$
- Do until convergence, satisfaction, or death from boredom:
  - Sample a target pattern $\mathbf{y}_p$
    - Sampling frequency of pattern must reflect importance of pattern
  - Initialize the network at $\mathbf{y}_p$ and let it evolve ***a few steps (2-4)***
    - And arrive at a down-valley position $\mathbf{y}_d$
  - Update weights
    - $\mathbf{W} = \mathbf{W} + \eta\left(\mathbf{y}_p\mathbf{y}_p^T - \mathbf{y}_d\mathbf{y}_d^T\right)$

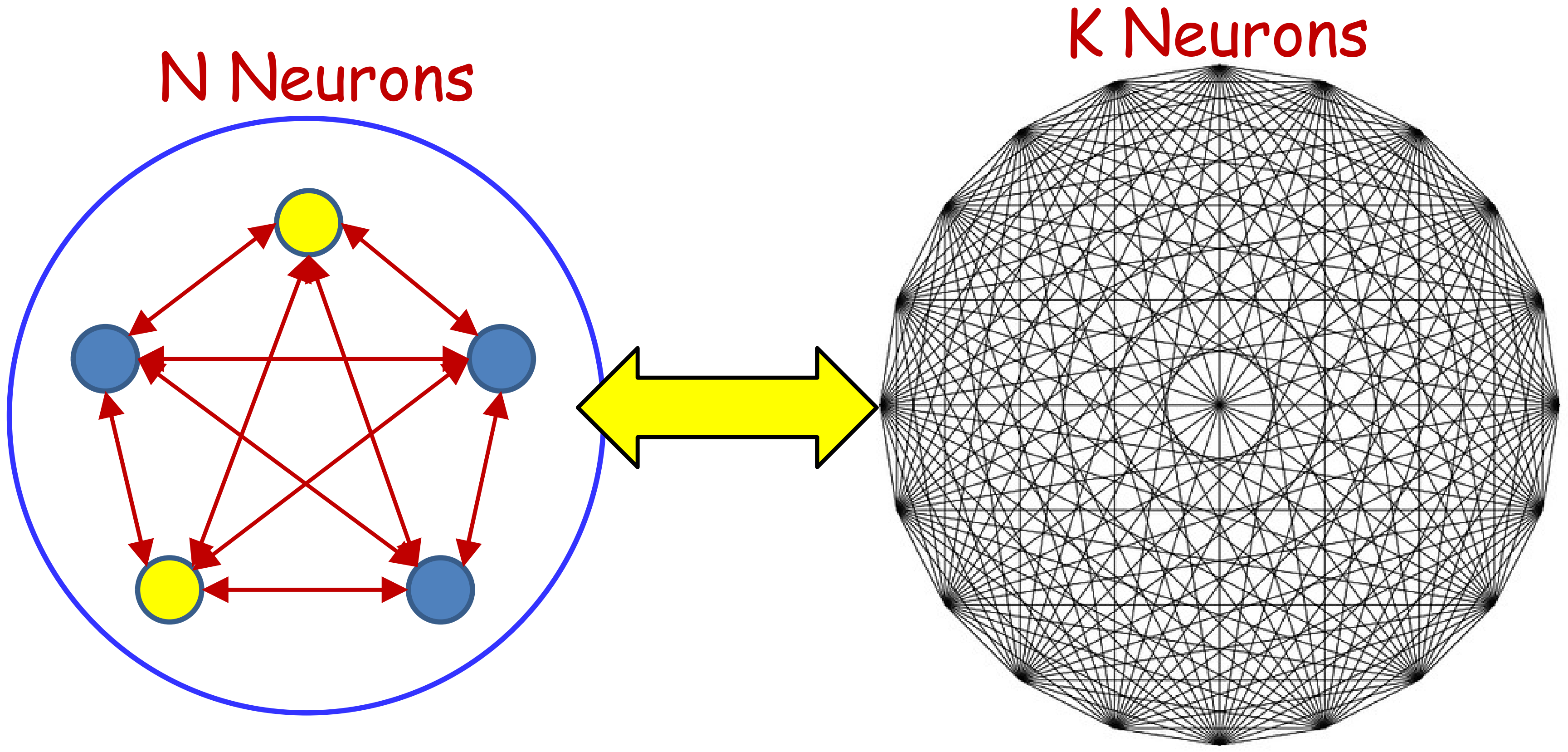
Story so far

- Hopfield nets with N neurons can store up to N random patterns
  - But comes with many parasitic memories
- Networks that store $O(N)$ memories can be trained through optimization
  - By minimizing the energy of the target patterns, while increasing the energy of the neighboring patterns

Storing more than N patterns

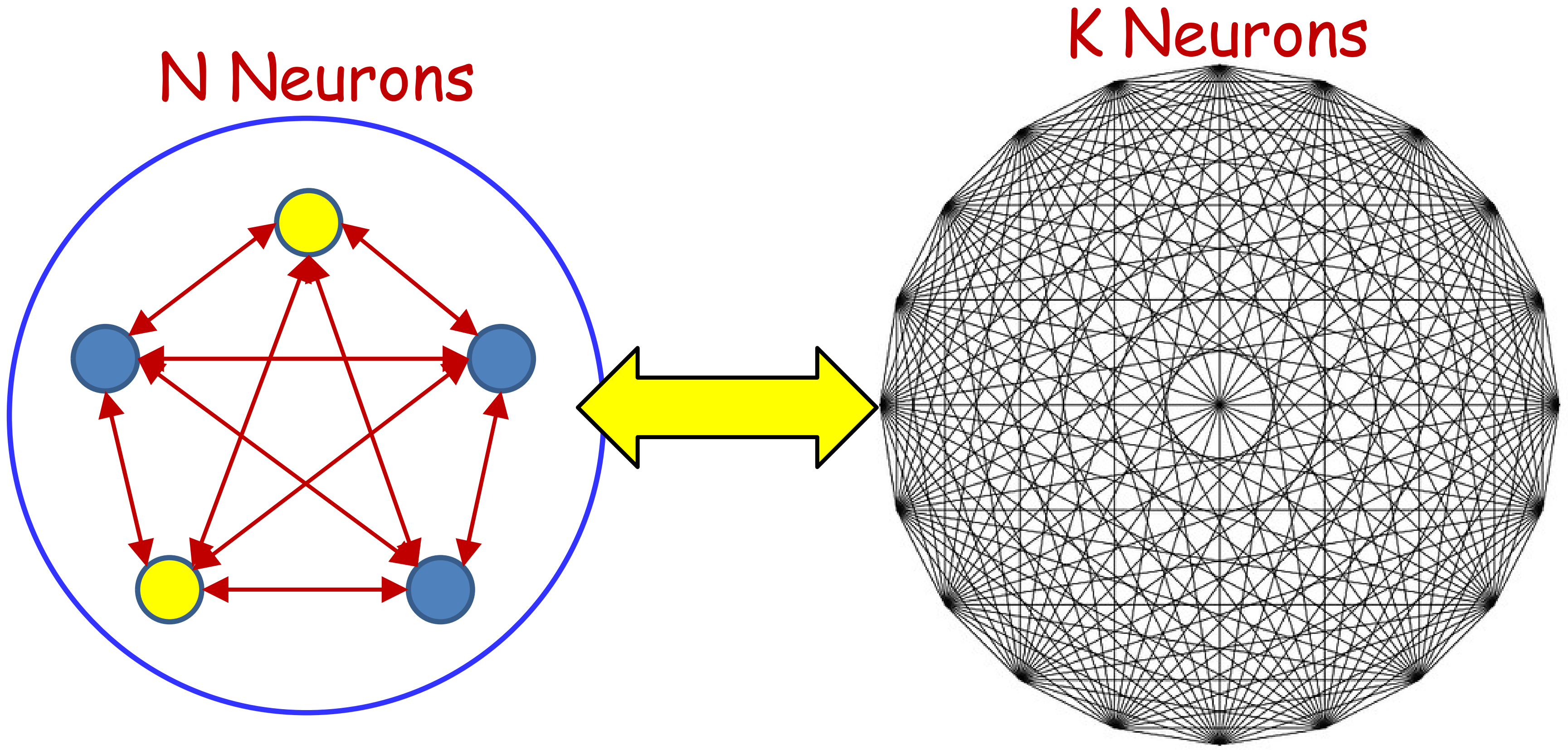
- The memory capacity of an N-bit network is at most N
  - Stable patterns (not necessarily even stationary)
    - Abu Mustafa and St. Jacques, 1985
    - Although "information capacity" is $\mathcal{O}(N^3)$
- How do we increase the capacity of the network
  - How to store more than N patterns

Expanding the network



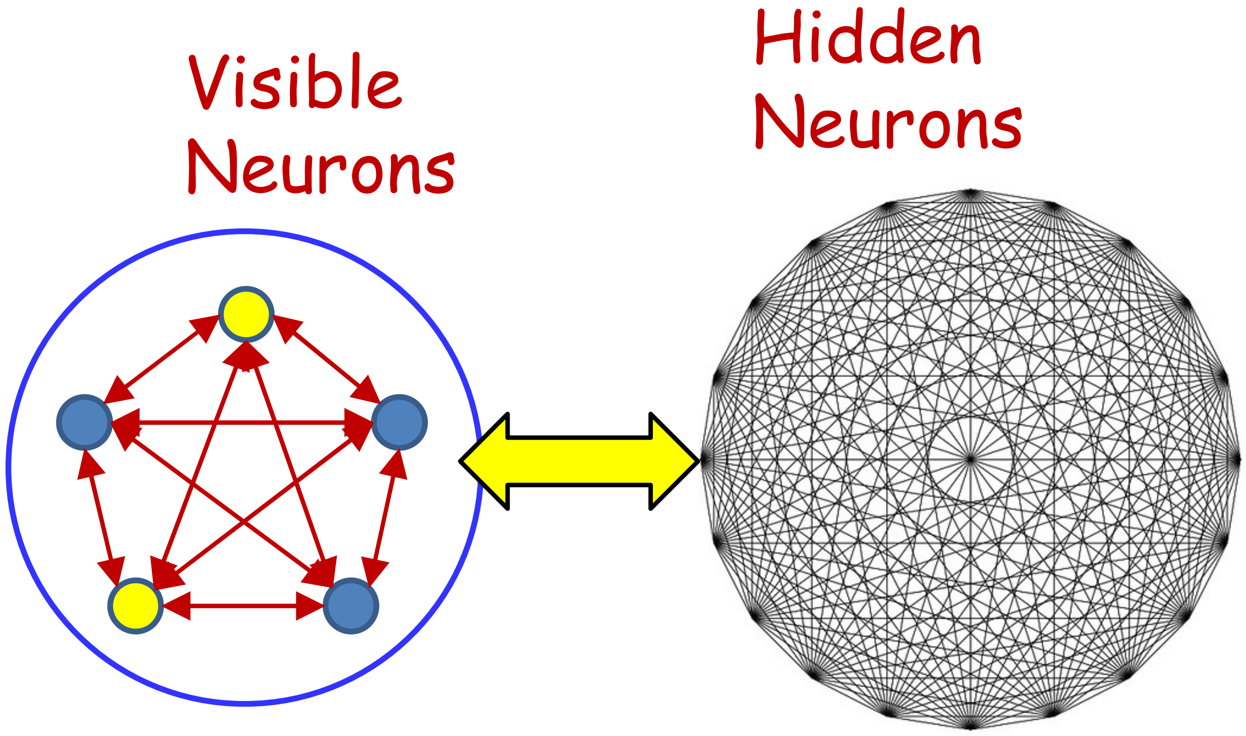
- Add a large number of neurons whose actual values you don't care about!

Expanded Network



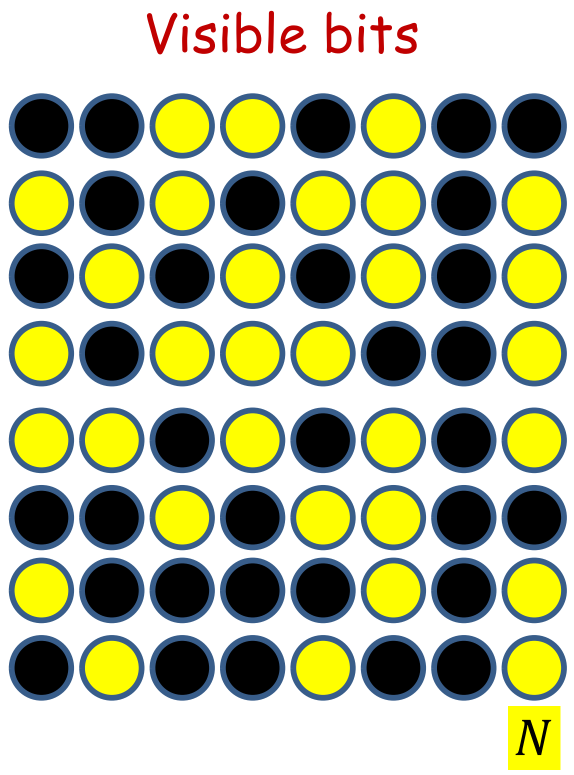
- New capacity: $\sim(N + K)$ patterns
  - Although we only care about the pattern of the first N neurons
  - We're interested in *N-bit* patterns

Terminology



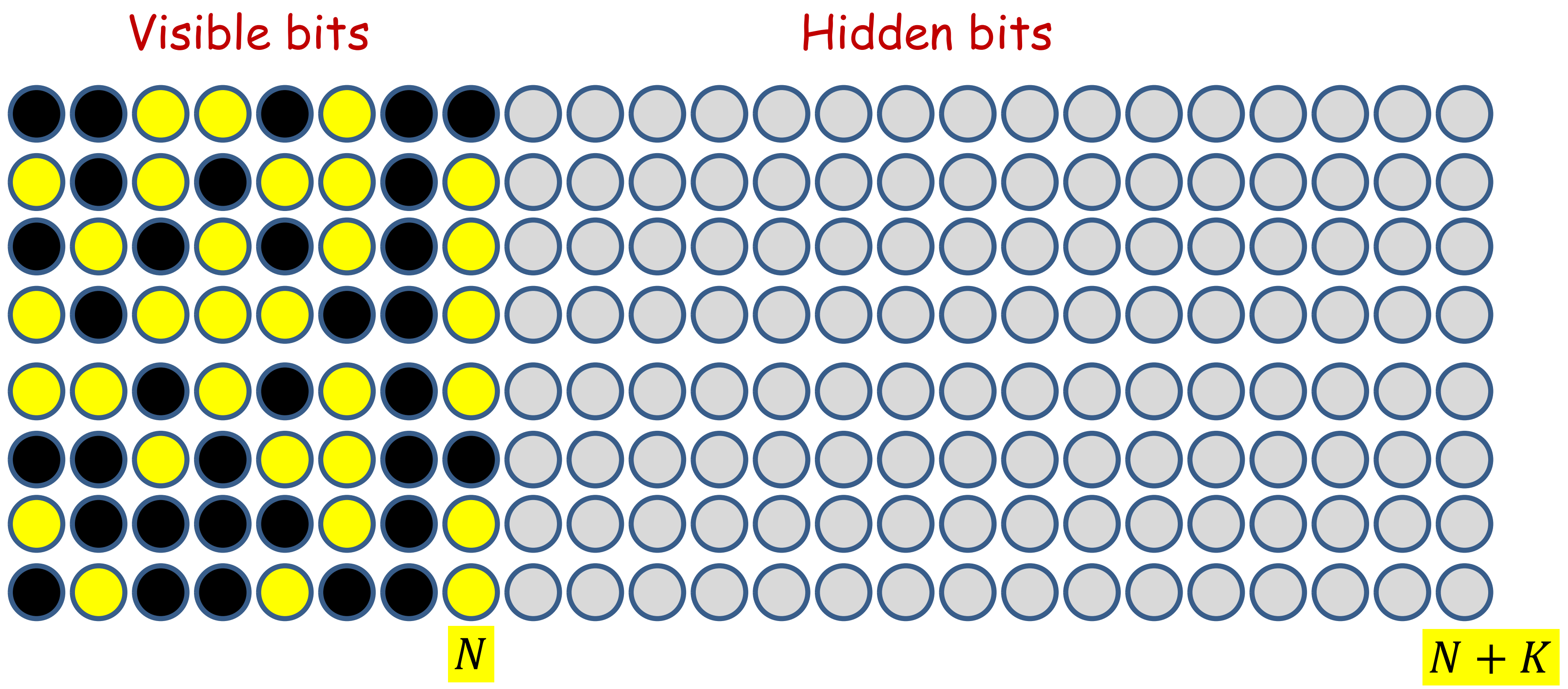
- Terminology:
  - The neurons that store the actual patterns of interest: *Visible neurons*
  - The neurons that only serve to increase the capacity but whose actual values are not important: *Hidden neurons*
  - These can be set to anything in order to store a visible pattern

Increasing the capacity: bits view



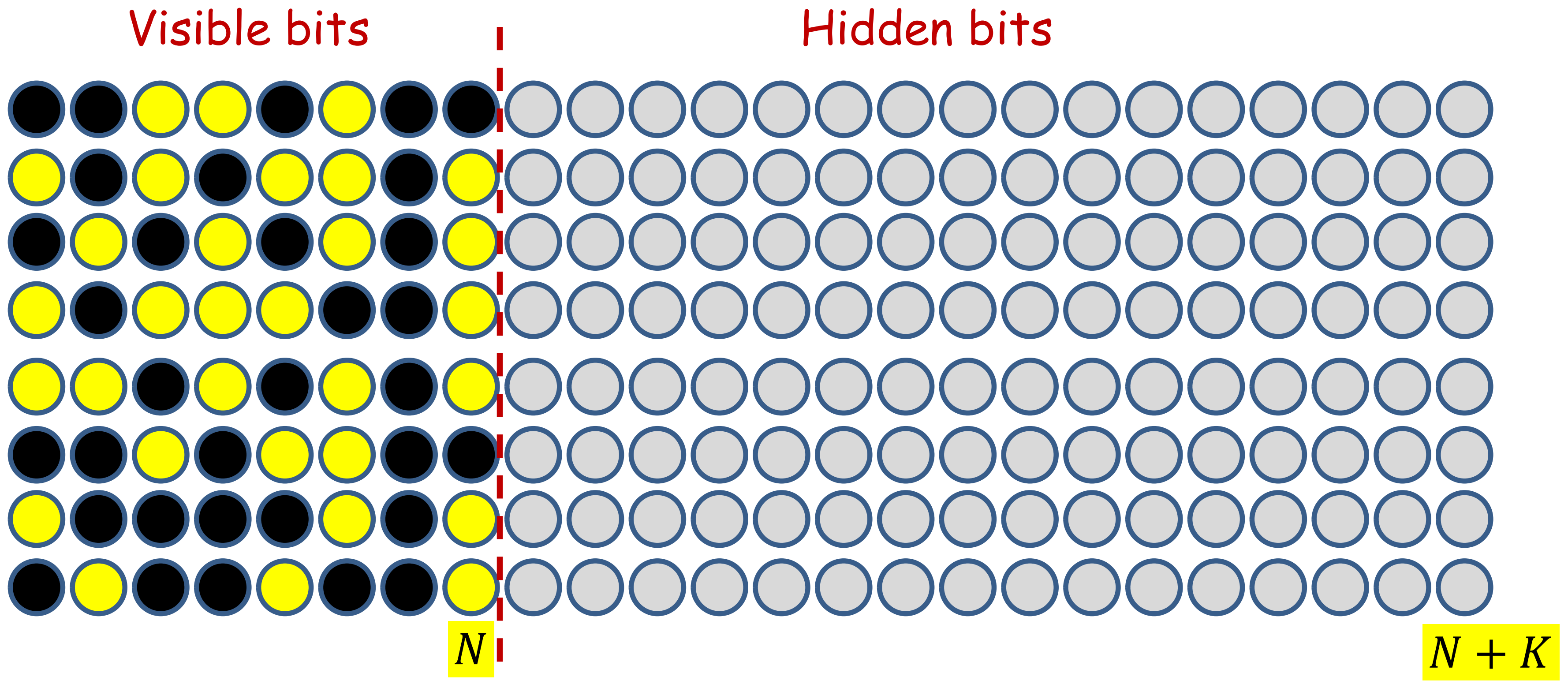
- The maximum number of patterns the net can store is bounded by the width *N* of the patterns..

Increasing the capacity: bits view



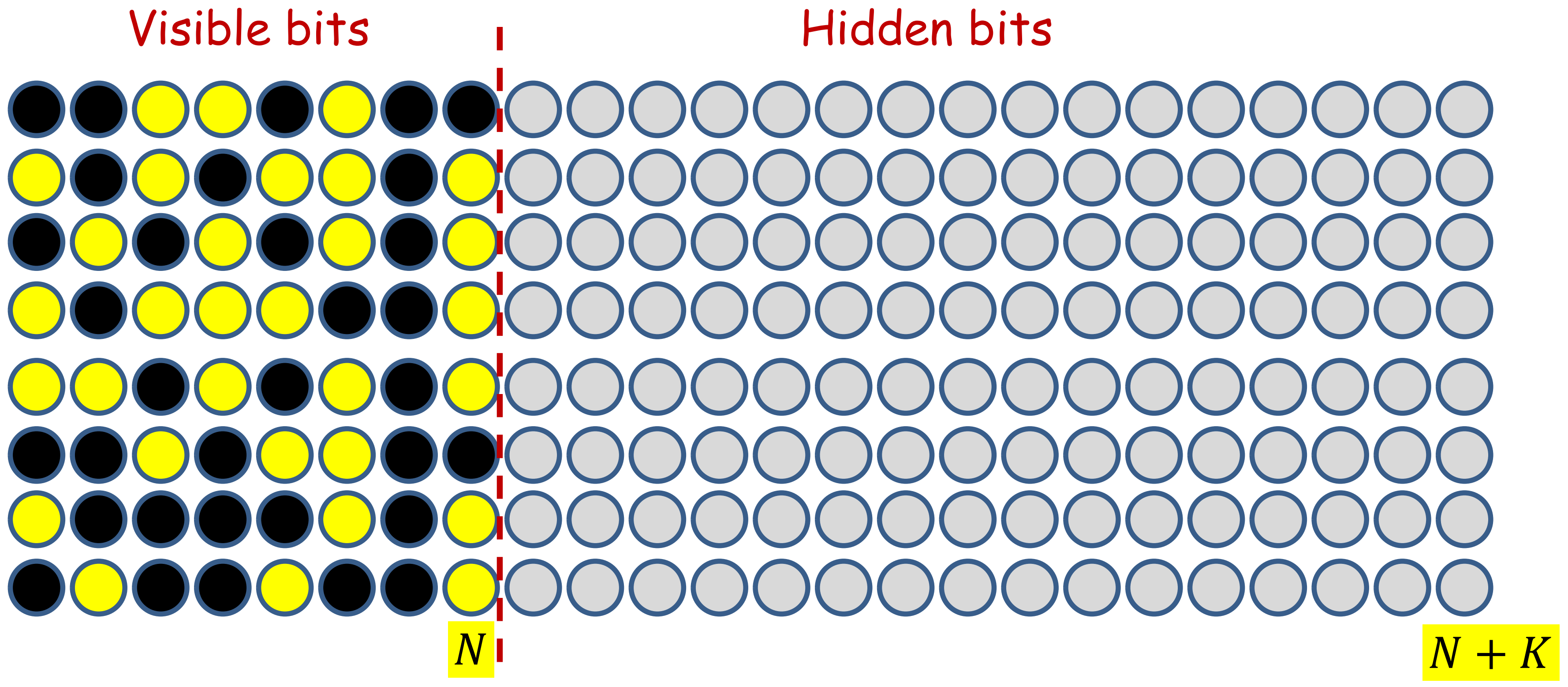
- The maximum number of patterns the net can store is bounded by the width *N* of the patterns..
- So, let's *pad* the patterns with *K* "don't care" bits
  - The new width of the patterns is N+K
  - Now we can store N+K patterns!

Issues: Storage



- What patterns do we fill in the don't care bits?
  - Simple option: Randomly
    - Flip a coin for each bit
  - Optimize
- How do we store the patterns?
  - Standard optimization method should work

Issues: Recall



- How do we retrieve a memory?
- Can do so using usual “evolution” mechanism
- But this is not taking advantage of a key feature of the extended patterns:
  - Making errors in the don’t care bits doesn’t matter

Taking advantage of don’t care bits

- Simple random setting of don’t care bits, and using the usual training and recall strategies for Hopfield nets should work

- However, it doesn’t sufficiently exploit the redundancy of the don’t care bits
  - Possible to set the don’t care bits such that the overall pattern (and hence the “visible” bits portion of the pattern) is more memorable
  - Also, may have multiple don’t-care patterns for a target pattern
    - Multiple valleys, in which the visible bits remain the same, but don’t care bits vary

- To exploit it properly, it helps to view the Hopfield net differently: as a probabilistic machine

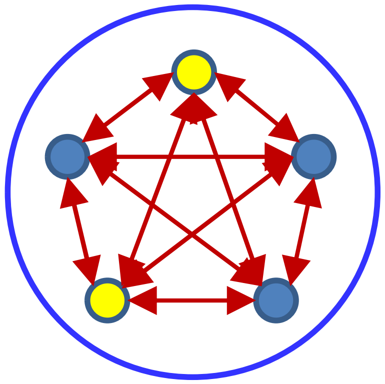
A probabilistic interpretation of Hopfield Nets

- For *binary* y the energy of a pattern is the analog of the negative log likelihood of a *Boltzmann distribution*
  - **Minimizing energy maximizes log likelihood**

$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} \qquad P(\mathbf{y}) = Cexp(-E(\mathbf{y}))$$

The Energy of the Network

Visible Neurons



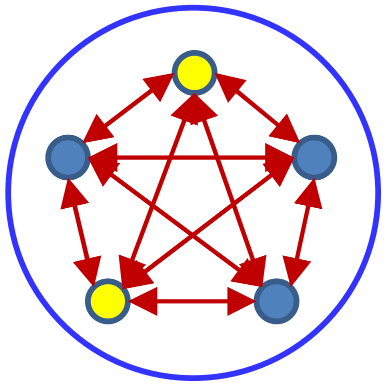
$$E(S) = -\sum_{i<j} w_{ij} s_i s_j - b_i s_i$$

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

- We can define the energy of the system as before
- *Neurons are stochastic*, with disorder or entropy
- The *equilibribum* probability distribution over states is the Boltzmann distribution at T=1
  - This is the probability of different states that the network will wander over *at equilibrium*

The Hopfield net is a distribution

Visible Neurons



$$E(S) = -\sum_{i<j} w_{ij}s_i s_j - b_i s_i$$

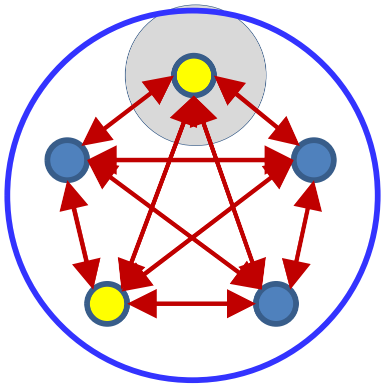
$$\boxed{P(S) = \frac{exp(-E(S))}{\sum_{S'} exp(-E(S'))}}$$

- The stochastic Hopfield network models a ***probability distribution*** over states
  - Where a state is a binary string
  - Specifically, it models a *Boltzmann distribution*
  - **The parameters of the model are the weights of the network**

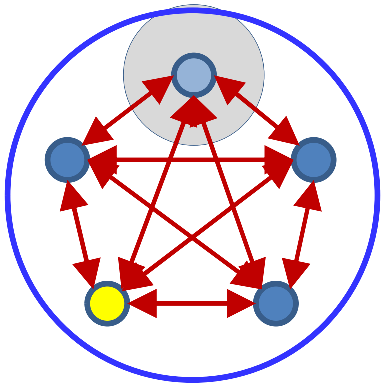
- The probability that (at equilibrium) the network will be in any state is $P(S)$
  - It is a *generative* model: generates states according to $P(S)$

The field at a single node

- Let S and S' be otherwise identical states that only differ in the i-th bit
  - S has i-th bit = $+1$ and S' has i-th bit = -1



$$P(S) = P\left(s_i = 1 \middle| s_{j\neq i}\right) P(s_{j\neq i})$$
$$P(S') = P\left(s_i = -1 \middle| s_{j\neq i}\right) P(s_{j\neq i})$$

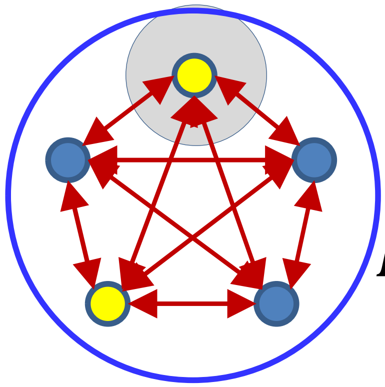


$$logP(S) - logP(S') = logP\left(s_i = 1 \middle| s_{j\neq i}\right) - logP\left(s_i = -1 \middle| s_{j\neq i}\right)$$

$$logP(S) - logP(S') = log\frac{P\left(s_i = 1 \middle| s_{j\neq i}\right)}{1 - P\left(s_i = 1 \middle| s_{j\neq i}\right)}$$

The field at a single node

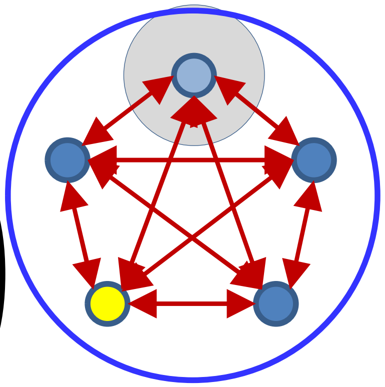
- Let S and S' be the states with the ith bit in the $+1$ and -1 states



$$\log P(S) = -E(S) + C$$

$$E(S) = -\frac{1}{2}\left(E_{not\ i} + \sum_{j \neq i} w_j s_j + b_i\right)$$

$$E(S') = -\frac{1}{2}\left(E_{not\ i} - \sum_{j \neq i} w_j s_j - b_i\right)$$



- $logP(S) - logP(S') = E(S') - E(S) = \sum_{j \neq i} w_j s_j + b_i$

The field at a single node

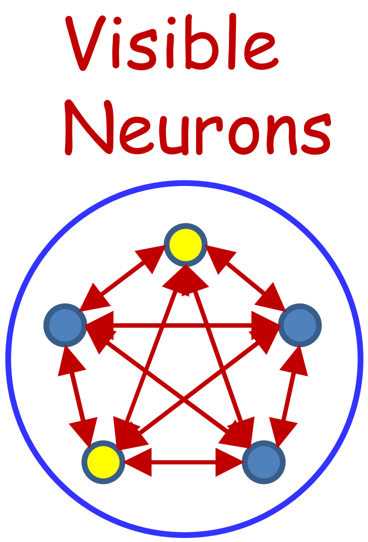
$$log\left(\frac{P\left(s_i = 1 \middle| s_{j\neq i}\right)}{1 - P\left(s_i = 1 \middle| s_{j\neq i}\right)}\right) = \sum_{j\neq i} w_j s_j + b_i$$

- Giving us

$$P\left(s_i = 1 \middle| s_{j\neq i}\right) = \frac{1}{1 + e^{-\left(\sum_{j\neq i} w_j s_j + b_i\right)}}$$

- The probability of any node taking value 1 given other node values is a logistic

The Hopfield net is a distribution

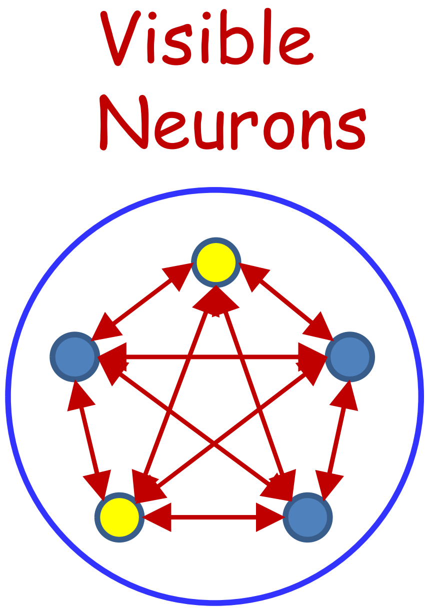


$$z_i = \sum_j w_{ji} s_j + b_i$$

$$\boxed{P(s_i = 1 | s_{j \neq i}) = \frac{1}{1 + e^{-z_i}}}$$

- The Hopfield net is a probability distribution over binary sequences
  - The Boltzmann distribution
- The *conditional* distribution of individual bits in the sequence is a logistic

Running the network



$$z_i = \sum_j w_{ji} s_j + b_i$$

$$\boxed{P(s_i = 1 | s_{j \neq i}) = \frac{1}{1 + e^{-z_i}}}$$

- Initialize the neurons
- Cycle through the neurons and randomly set the neuron to 1 or -1 according to the probability given above
  - Gibbs sampling: Fix N-1 variables and sample the remaining variable
  - As opposed to energy-based update (mean field approximation): run the test $z_i > 0$?
- After many many iterations (until “convergence”), *sample* the individual neurons

Evolution of a stochastic Hopfield net

1. Initialize network with initial pattern

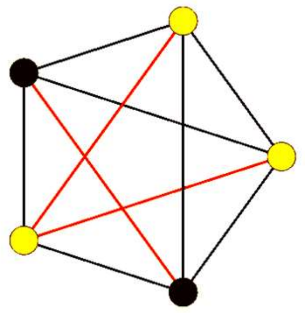
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

2. Iterate $0 \le i \le N-1$

$$P = \sigma\left(\sum_{j \ne i} w_{ji} y_j\right)$$

$$y_i(t+1) \sim Binomial(P)$$

Assuming T = 1



Evolution of a stochastic Hopfield net

1. Initialize network with initial pattern

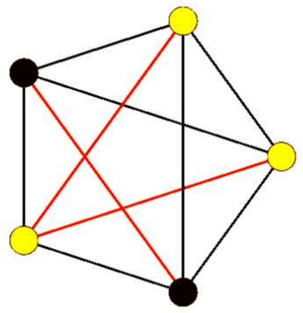
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

2. Iterate $0 \le i \le N-1$

$$P = \sigma\left(\sum_{j \neq i} w_{ji} y_j\right)$$

$$y_i(t+1) \sim Binomial(P)$$

Assuming T = 1



- When do we stop?
- What is the final state of the system
  - How do we “recall” a memory?

Evolution of a stochastic Hopfield net

1. Initialize network with initial pattern

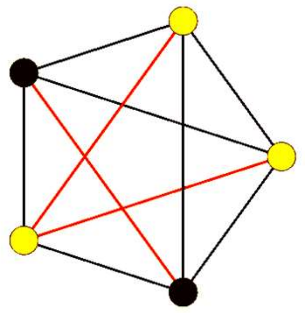
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

2. Iterate $0 \le i \le N-1$

$$P = \sigma\left(\sum_{j \ne i} w_{ji} y_j\right)$$

$$y_i(t+1) \sim Binomial(P)$$

Assuming T = 1



- When do we stop?
- What is the final state of the system
  - How do we "recall" a memory?

Evolution of a stochastic Hopfield net

1. Initialize network with initial pattern

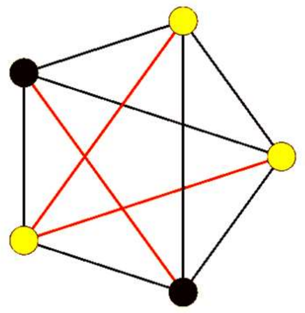
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

2. Iterate $0 \le i \le N-1$

$$P = \sigma\left(\sum_{j \neq i} w_{ji} y_j\right)$$

$$y_i(t+1) \sim Binomial(P)$$

Assuming T = 1



- Let the system evolve to "equilibrium"
- Let $\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_L$ be the sequence of values (L large)
- Final predicted configuration: from the average of the final few iterations

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

  - Estimates the probability that the bit is 1.0.
  - If it is greater than 0.5, sets it to 1.0

Evolution of the stochastic network

1. Initialize network with initial pattern

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

2. For $T = T_0 \; down \; to \; T_{min}$

Noisy pattern completion: Initialize the entire network and let the entire network evolve

Pattern completion: Fix the "seen" bits and only let the "unseen" bits evolve

- Let the system evolve to "equilibrium"
- Let $\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_L$ be the sequence of values (L large)
- Final predicted configuration: from the average of the final few iterations

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

Maximum Likelihood Training

$$\log(P(S)) = \left(\sum_{i<j} w_{ij} s_i s_j\right) - \log\left(\sum_{S'} exp\left(\sum_{i<j} w_{ij} s_i' s_j'\right)\right)$$

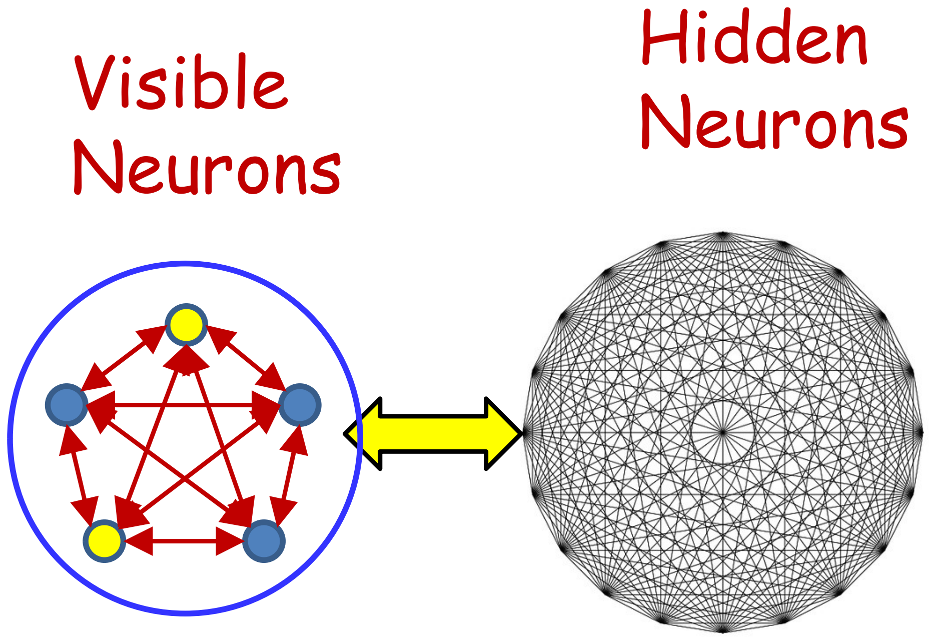
$$\mathcal{L} = \frac{1}{N} \sum_{S \in \mathbf{S}} \log(P(S))$$

Average log likelihood of training vectors (to be maximized)

$$= \frac{1}{N} \sum_{S} \left(\sum_{i<j} w_{ij} s_i s_j\right) - \log\left(\sum_{S'} exp\left(\sum_{i<j} w_{ij} s_i' s_j'\right)\right)$$

- Maximize the average log likelihood of all "training" vectors $\mathbf{S} = \{S_1, S_2, \dots, SN\}$
  - In the first summation, s_i and s_j are bits of S
  - In the second, s_i' and s_j' are bits of S'

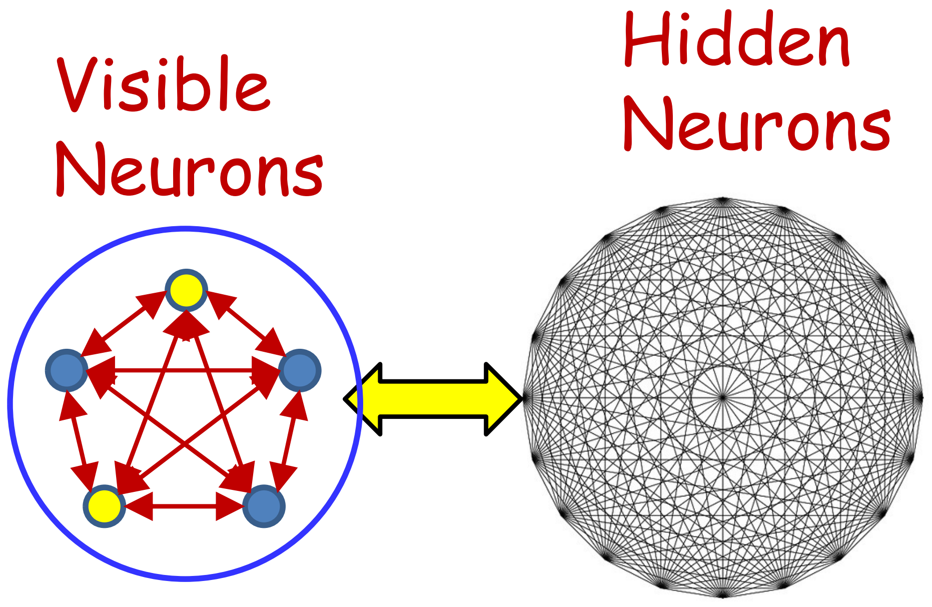
Step 1



- For each training pattern V_i
  - Fix the visible units to V_i
  - Let the hidden neurons evolve from a random initial point to generate H_i
  - Generate $S_i = [V_i, H_i]$
- Repeat K times to generate synthetic training

$$\mathbf{S} = \{S_{1,1}, S_{1,2}, \ldots, S_{1K}, S_{2,1}, \ldots, S_{N,K}\}$$

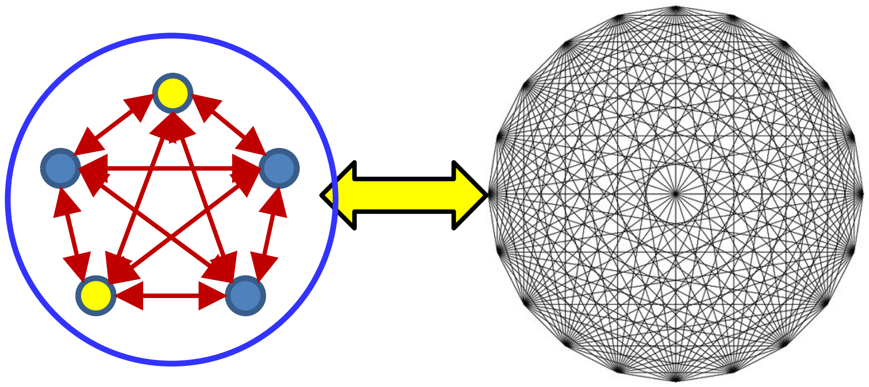
Step 2



- Now *unclamp* the visible units and let the entire network evolve several times to generate

$$\mathbf{S}_{simul} = \{S_{simul,1}, S_{simul,1=2}, \dots, S_{simul,M}\}$$

Gradients

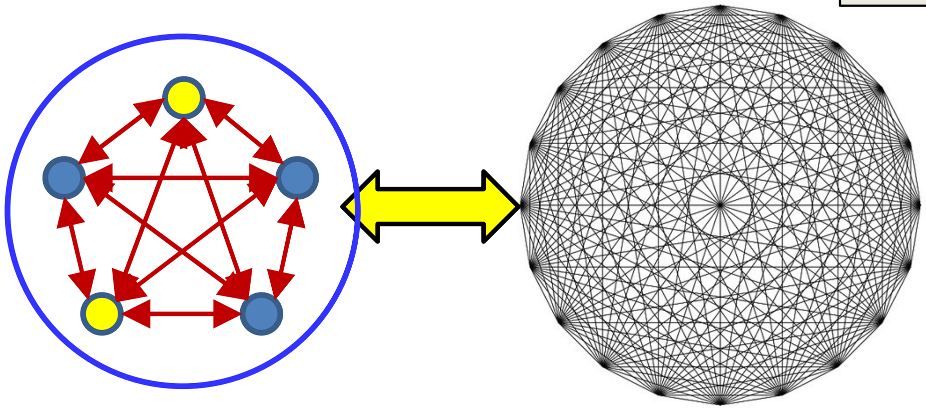


$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{NK}\sum_{\mathbf{S}} s_i s_j - \frac{1}{M}\sum_{S' \in \mathbf{S}_{simul}} s_i' s_j'$$

- Gradients are computed as before, except that the first term is now computed over the *expanded* training data

Overall Training

$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{NK}\sum_{\mathbf{S}} s_i s_j - \frac{1}{M}\sum_{S' \in \mathbf{S}_{simul}} s_i' s_j'$$



$$\boxed{w_{ij} = w_{ij} - \eta \frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}}}$$

- Initialize weights
- Run simulations to get clamped and unclamped training samples
- Compute gradient and update weights
- Iterate

Boltzmann machines

- Stochastic extension of Hopfield nets
- Enables storage of many more patterns than Hopfield nets
- But also enables computation of probabilities of patterns, and completion of pattern

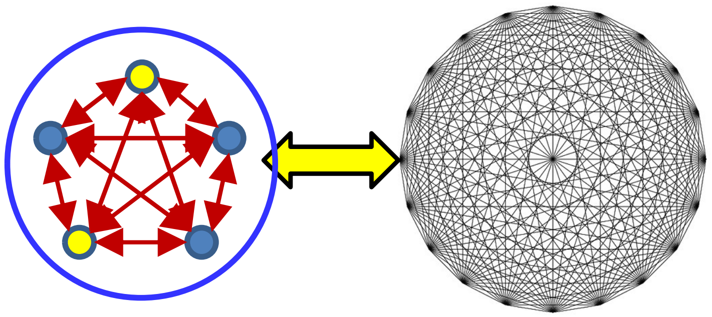
Boltzmann machines: Overall

$$z_i = \sum_j w_{ji} s_i + b_i$$

$$P(s_i = 1) = \frac{1}{1 + e^{-z_i}}$$

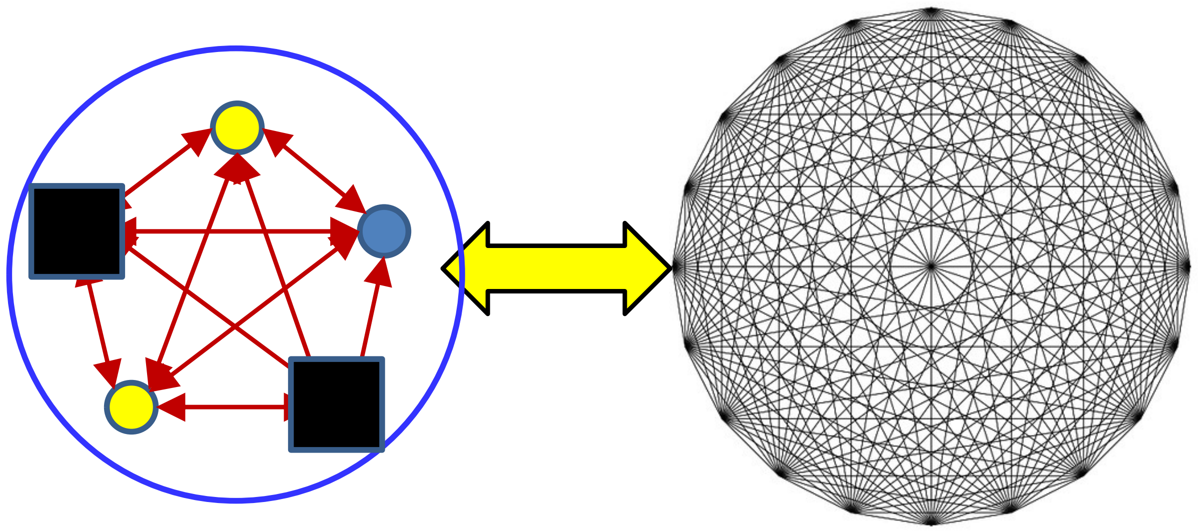
$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{NK} \sum_{\mathbf{S}} s_i s_j - \frac{1}{M} \sum_{S' \in \mathbf{S}_{simul}} s_i' s_j'$$

$$w_{ij} = w_{ij} - \eta \frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}}$$



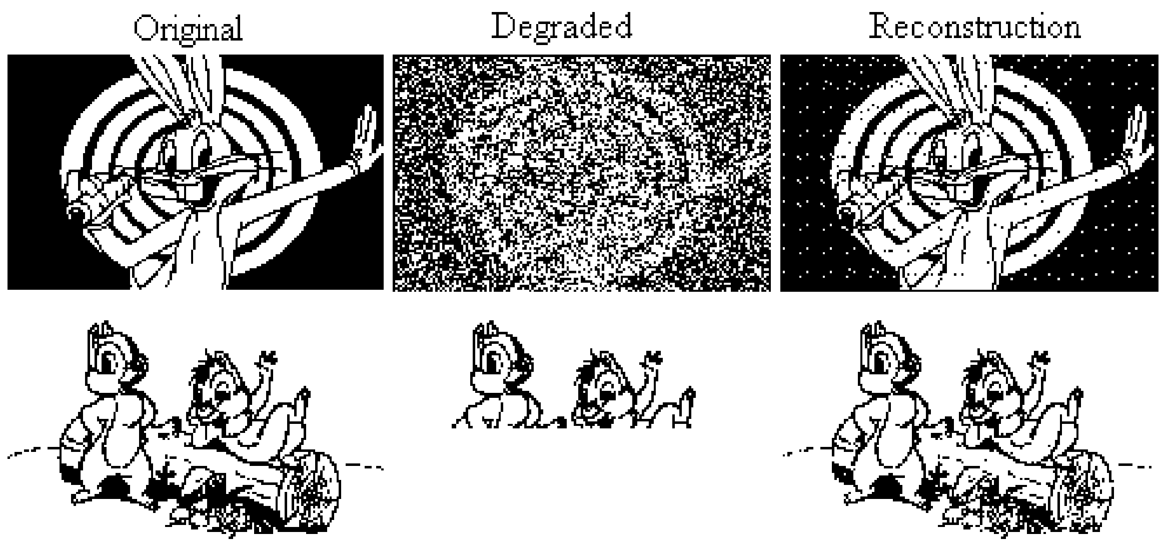
- **Training:** Given a set of training patterns
  - Which could be repeated to represent relative probabilities
- Initialize weights
- Run simulations to get clamped and unclamped training samples
- Compute gradient and update weights
- Iterate

Boltzmann machines: Overall



- Running: Pattern completion
  - “Anchor” the *known* visible units
  - Let the network evolve
  - Sample the unknown visible units
    - Choose the most probable value

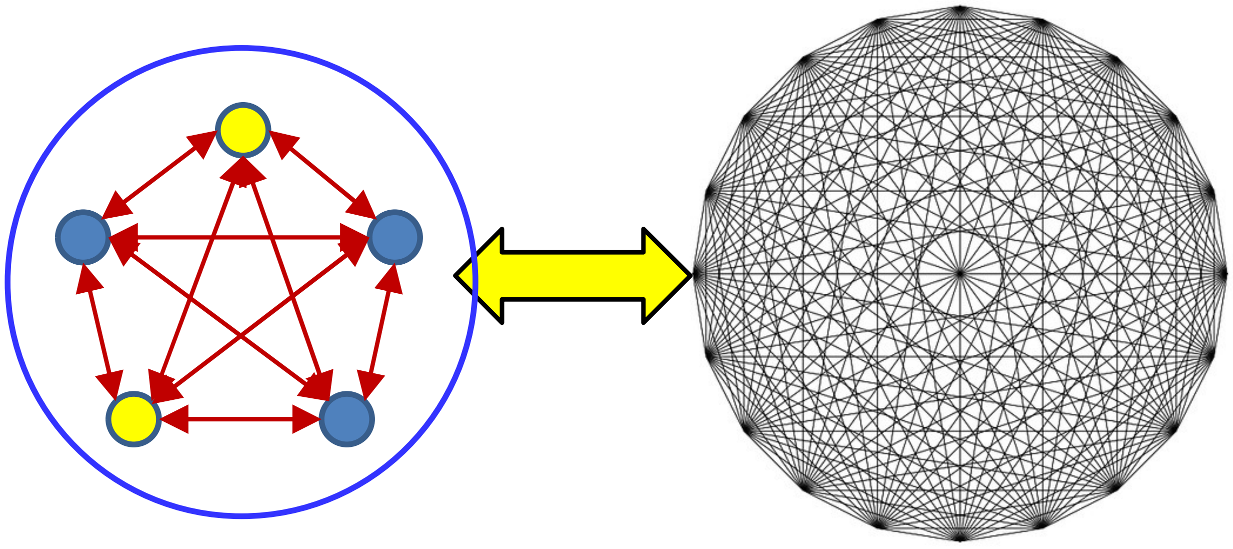
Applications



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

- Filling out patterns
- Denoising patterns
- *Computing conditional probabilities of patterns*
- ***Classification!!***
  - *How?*

Boltzmann machines for classification

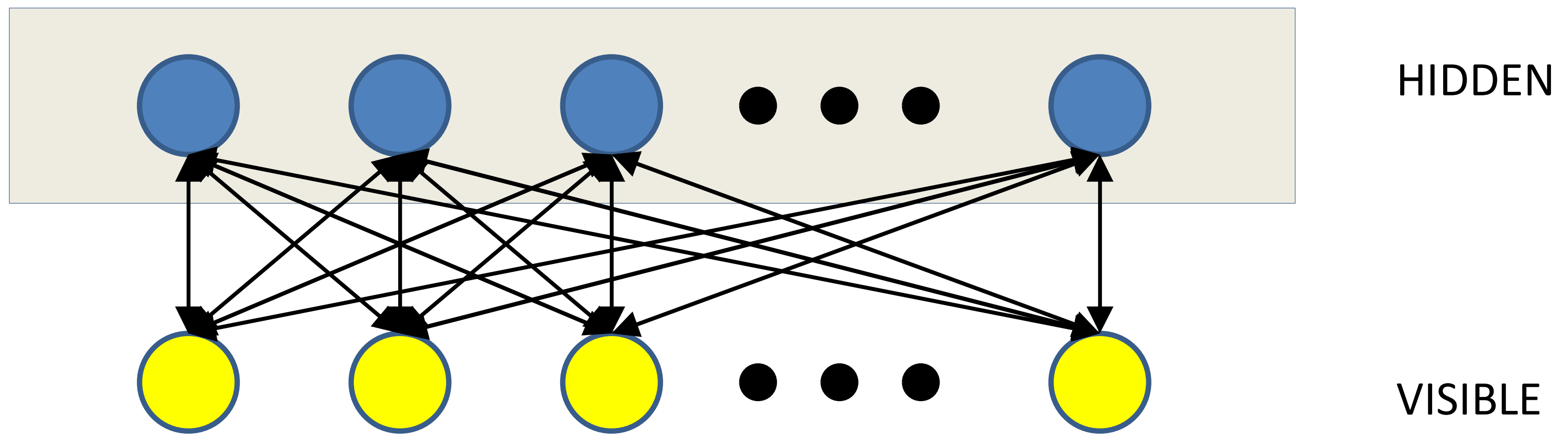


- Training patterns:
  - [f_1, f_2, f_3, …. , class]
  - Features can have binarized or continuous valued representations
  - Classes have “one hot” representation
- Classification:
  - Given features, anchor features, estimate a posteriori probability distribution over classes
    - Or choose most likely class

Boltzmann machines: Issues

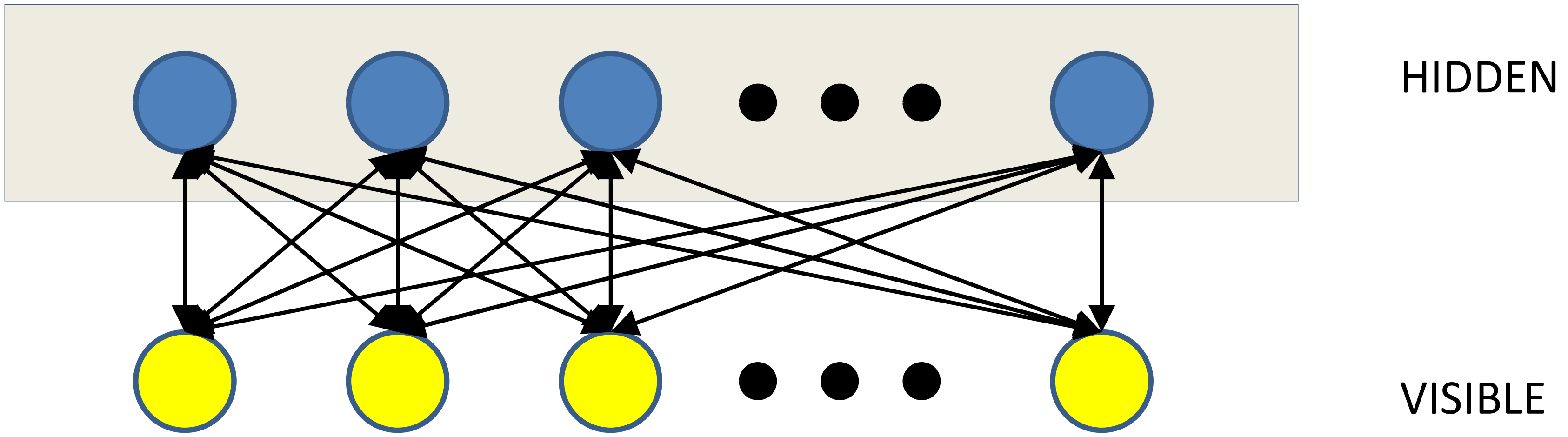
- Training takes for ever
- Doesn’t really work for large problems
  - A small number of training instances over a small number of bits

Solution: *Restricted* Boltzmann Machines



- Partition visible and hidden units
  - Visible units ONLY talk to hidden units
  - Hidden units ONLY talk to visible units
- Restricted Boltzmann machine..
  - **Originally proposed as “Harmonium Models” by Paul Smolensky**

Solution: *Restricted* Boltzmann Machines

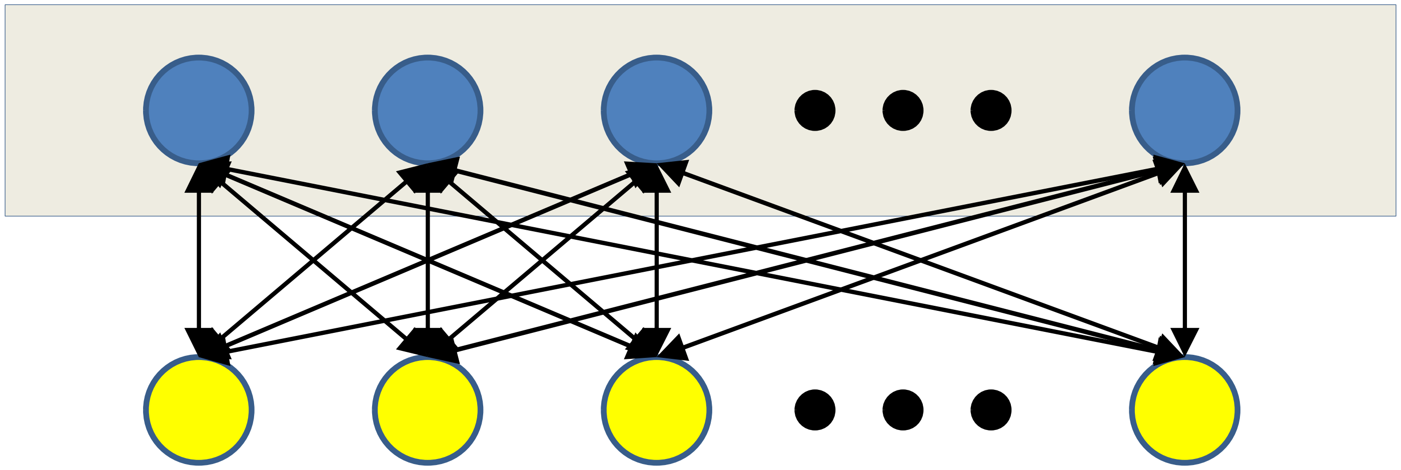


$$z_i = \sum_j w_{ji} s_i + b_i$$

$$\boxed{P(s_i = 1) = \frac{1}{1 + e^{-z_i}}}$$

- Still obeys the same rules as a regular Boltzmann machine
- But the modified structure adds a big benefit..

Solution: *Restricted* Boltzmann Machines



HIDDEN

VISIBLE

HIDDEN

$$z_i = \sum_j w_{ji} v_i + b_i$$

$$\boxed{P(h_i = 1) = \frac{1}{1 + e^{-z_i}}}$$

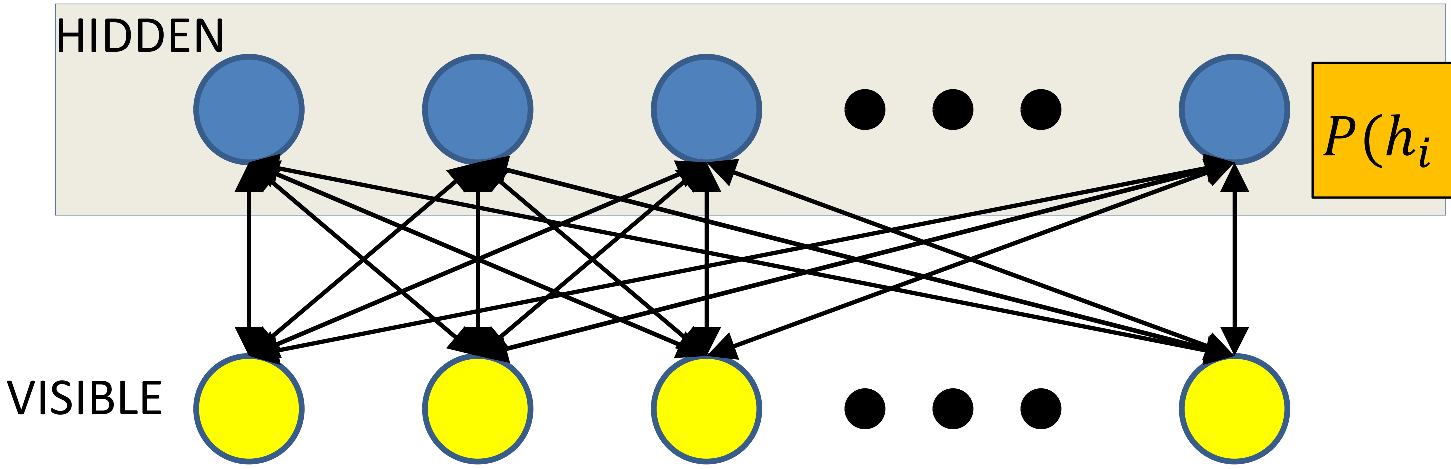
VISIBLE

$$y_i = \sum_j w_{ji} h_i + b_i$$

$$\boxed{P(v_i = 1) = \frac{1}{1 + e^{-y_i}}}$$

Sampling: Restricted Boltzmann machine

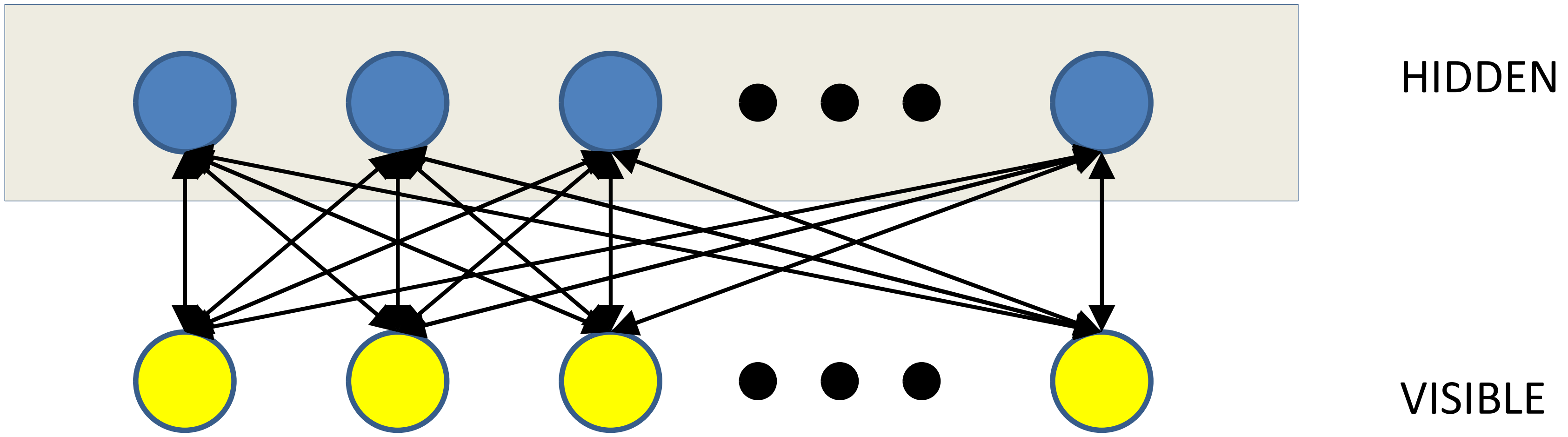
$$z_i = \sum_j w_{ji} v_i + b_i$$



$$\boxed{P(h_i = 1) = \frac{1}{1 + e^{-z_i}}}$$

- For each sample:
  - Anchor visible units
  - Sample from hidden units
  - No looping!!

Sampling: Restricted Boltzmann machine



$$z_i = \sum_j w_{ji} v_i + b_i$$

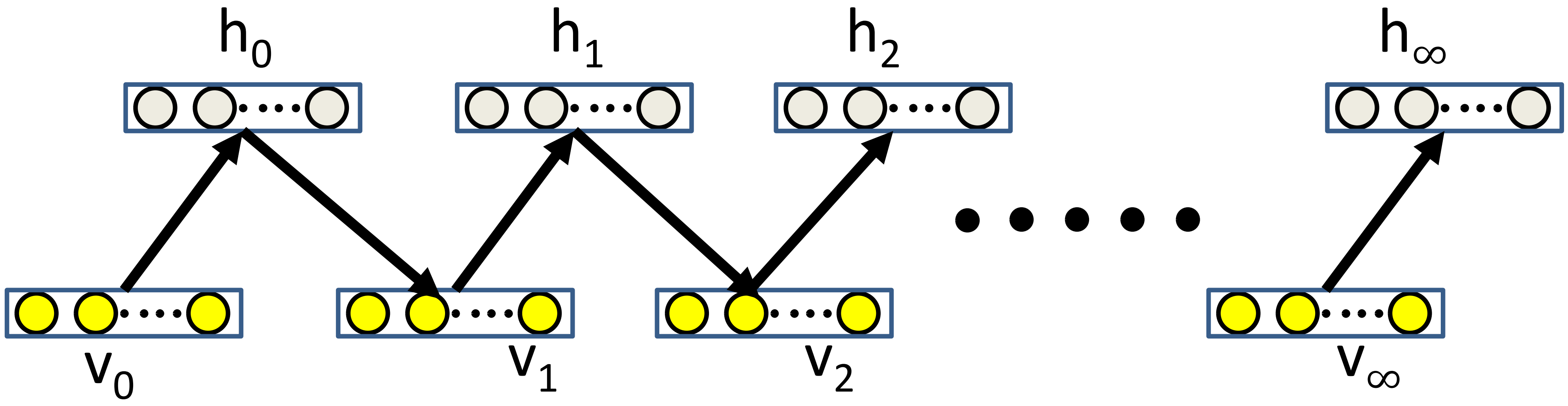
$$\boxed{P(h_i = 1) = \frac{1}{1 + e^{-z_i}}}$$

$$y_i = \sum_j w_{ji} h_i + b_i$$

$$\boxed{P(v_i = 1) = \frac{1}{1 + e^{-y_i}}}$$

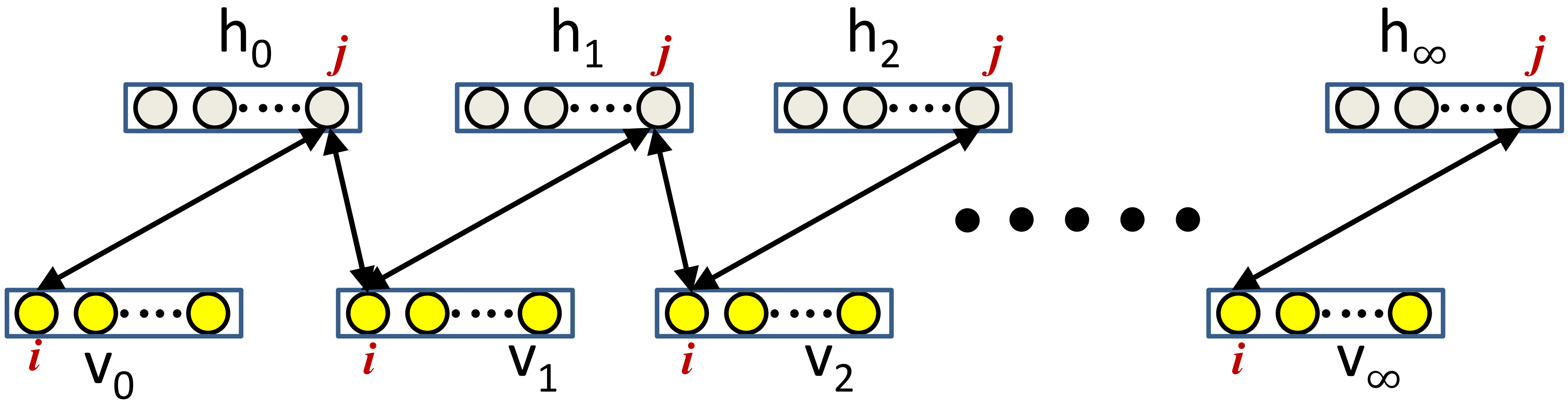
- For each sample:
  - Iteratively sample hidden and visible units for a long time
  - Draw final sample of both hidden and visible units

Pictorial representation of RBM training



- For each sample:
  - Initialize V_0 (visible) to training instance value
  - Iteratively generate hidden and visible units
    - For a very long time

Pictorial representation of RBM training



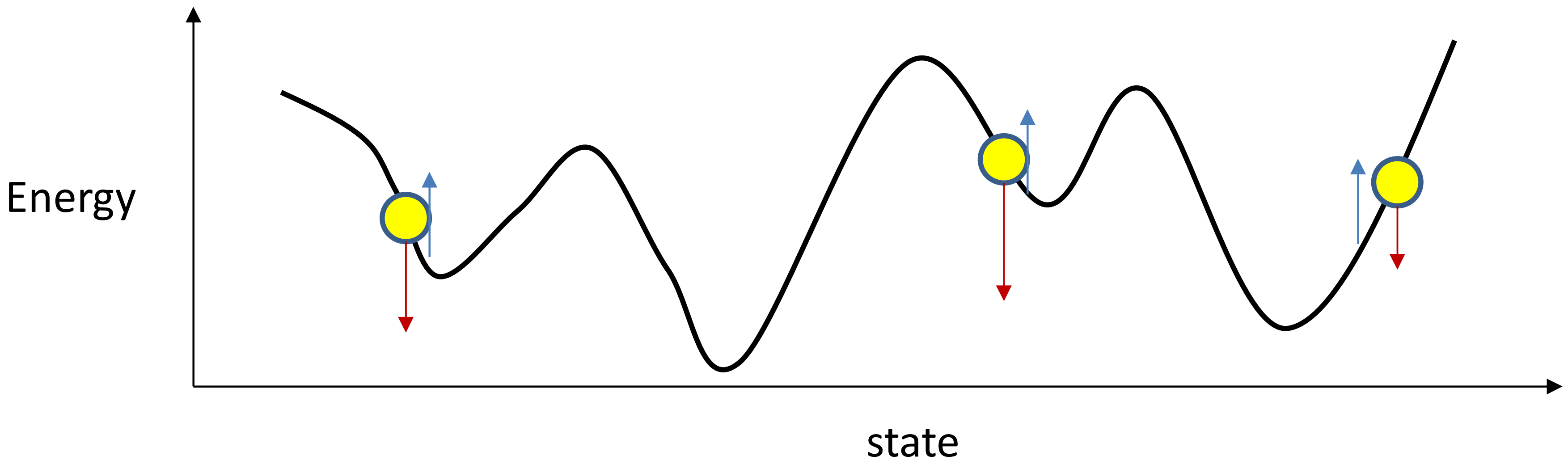
- Gradient (showing only one edge from visible node i to hidden node j)

$$\frac{\partial \log p(v)}{\partial w_{ij}} = \langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^\infty$$

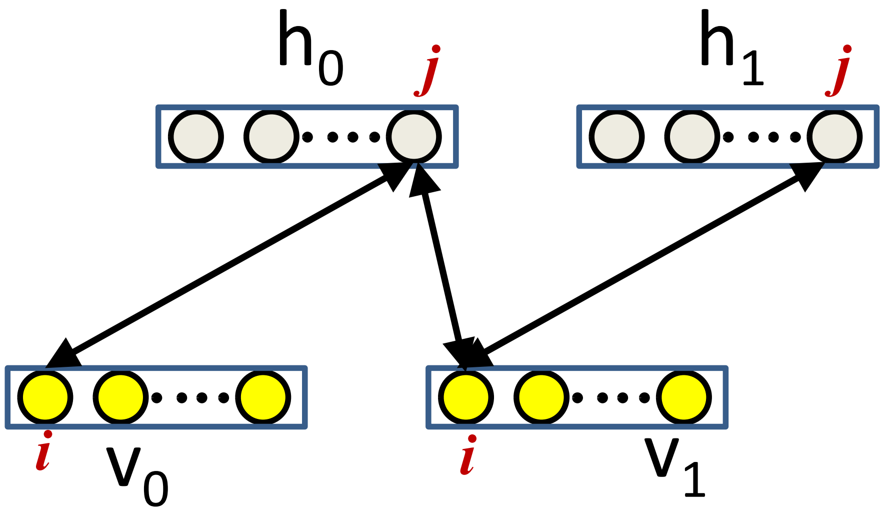
- $\langle v_i, h_j \rangle$ represents average over many generated training samples

Recall: Hopfield Networks

- Really no need to raise the entire surface, or even every valley
- Raise the *neighborhood* of each target memory
  - Sufficient to make the memory a valley
  - The broader the neighborhood considered, the broader the valley



A Shortcut: Contrastive Divergence



- Sufficient to run one iteration!

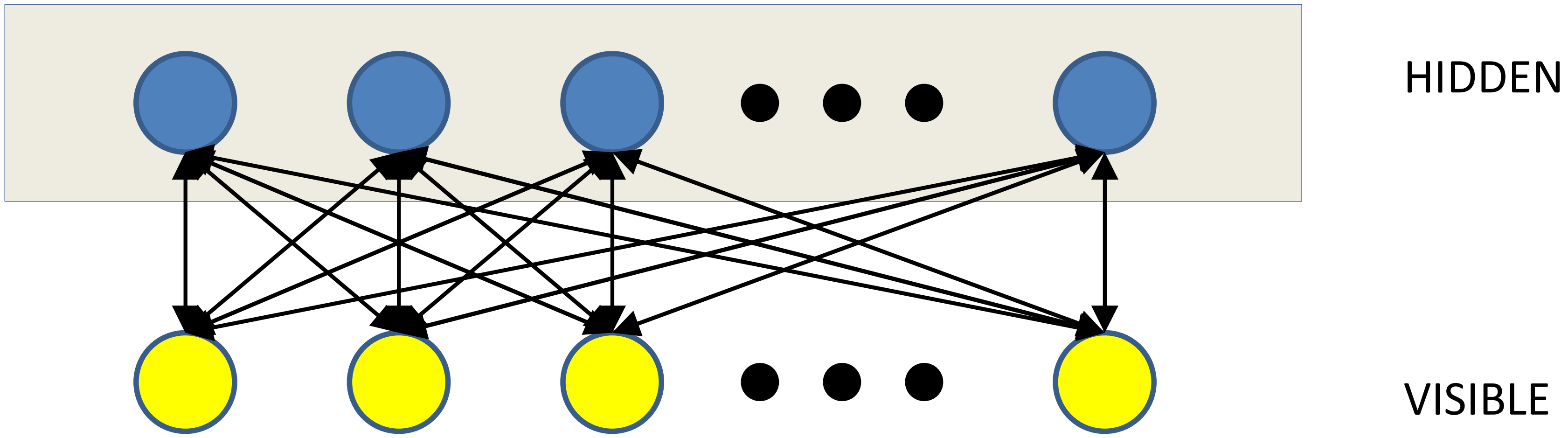
$$\frac{\partial \log p(v)}{\partial w_{ij}} = <v_i h_j>^0 - <v_i h_j>^1$$

- This is sufficient to give you a good estimate of the gradient

Restricted Boltzmann Machines

- Excellent generative models for binary (or binarized) data
- Can also be extended to continuous-valued data
  - "Exponential Family Harmoniums with an Application to Information Retrieval", Welling et al., 2004
- Useful for classification and regression
  - How?
  - More commonly used to *pretrain* models

Continuous-values RBMs

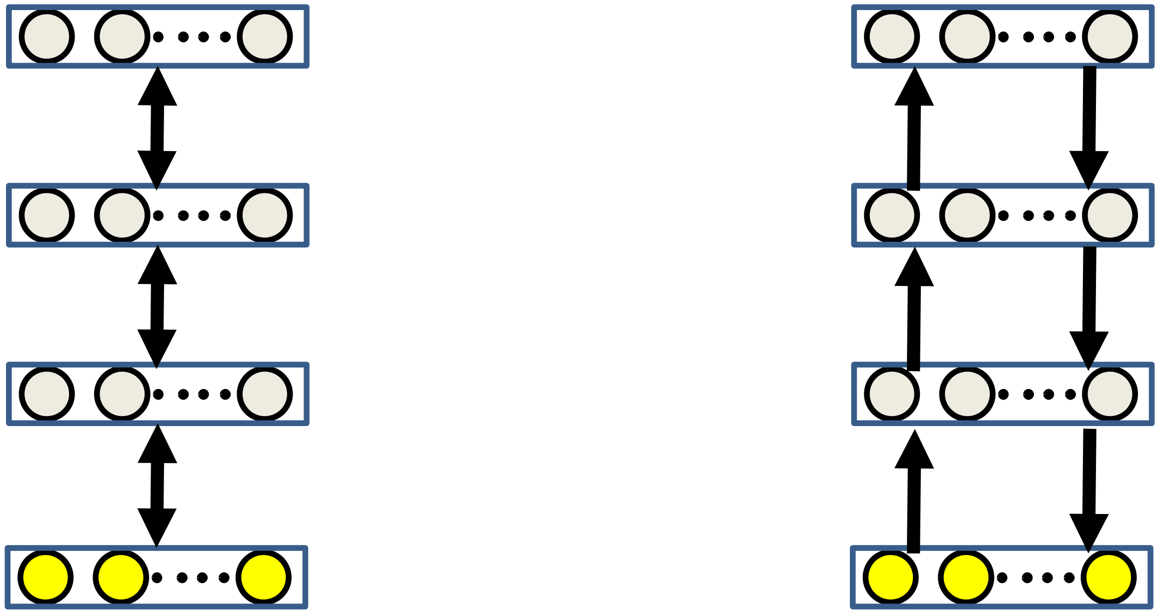


HIDDEN $$z_i = \sum_j w_{ji} v_i + b_i$$ $$\boxed{P(h_i = 1) = \frac{1}{1 + e^{-z_i}}}$$

VISIBLE $$y_i = \sum_j w_{ji} h_i + b_i$$ $$\boxed{P(v_i) = r(y_i) exp(y_i)}$$

Hidden units may also be continuous values

Other variants



- Left: "Deep" Boltzmann machines
- Right: Helmholtz machine
  - Trained by the "wake-sleep" algorithm

Topics missed..

- Other algorithms for Learning and Inference over RBMs
  - Mean field approximations
- RBMs as feature extractors
  - Pre training
- RBMs as generative models
- More structured DBMs
- …