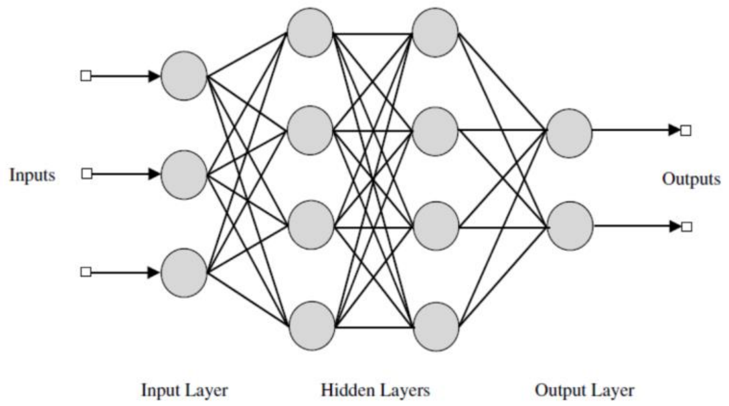
Neural Networks

Hopfield Nets and Auto Associators

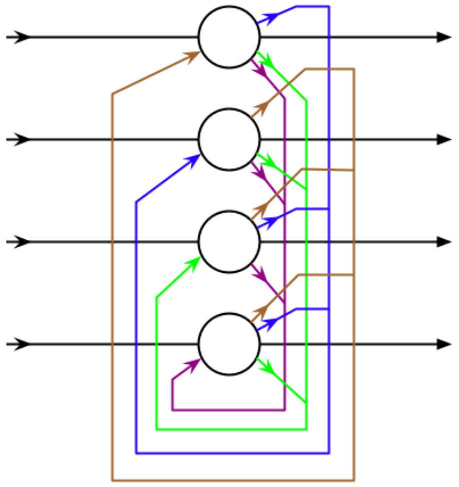
Fall 2017

Story so far

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



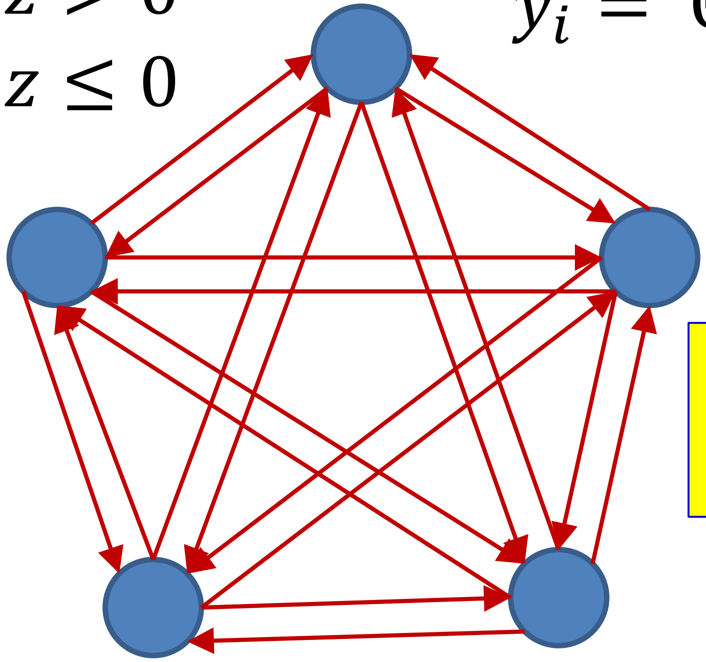
- But what about..



Loopy network

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

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



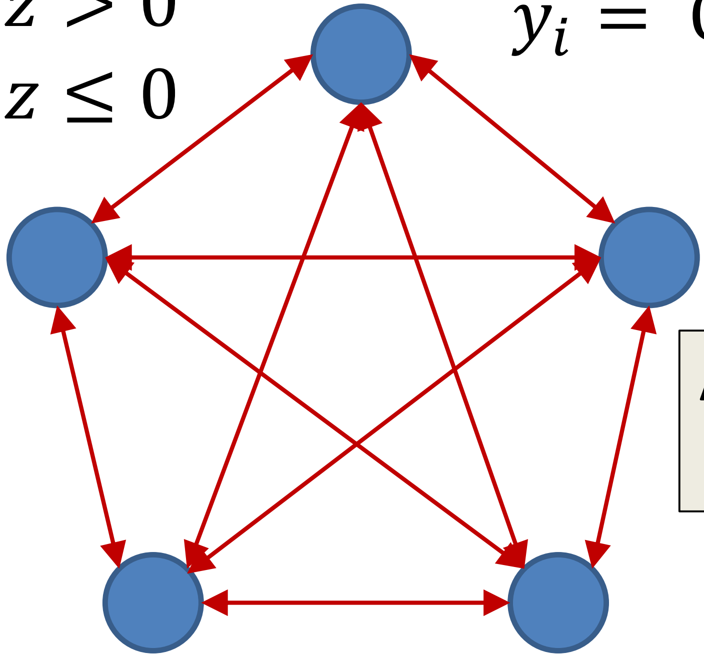
The output of a neuron affects the input to the neuron

- 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

Loopy network

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

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



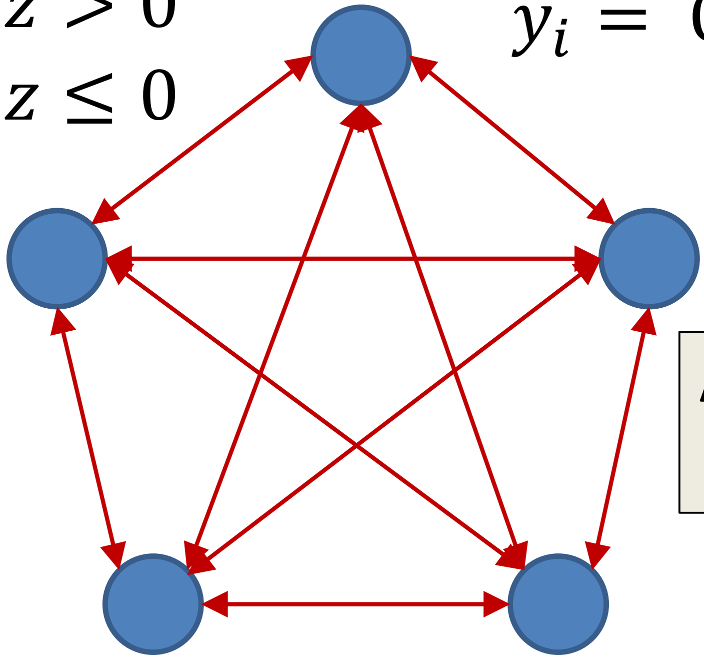
A symmetric network: $w_{ij} = w_{ji}$

- 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

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

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



A symmetric network:

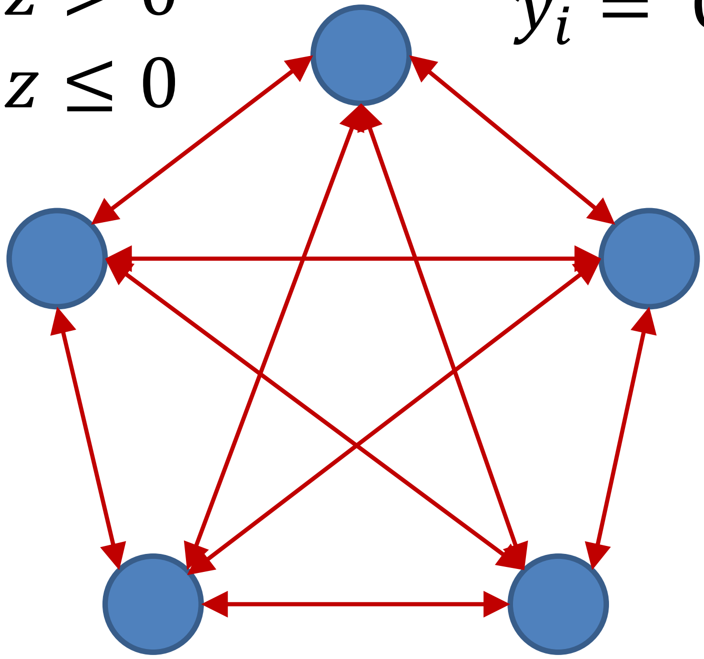
$w_{ij} = w_{ji}$

- 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

Loopy network

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

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

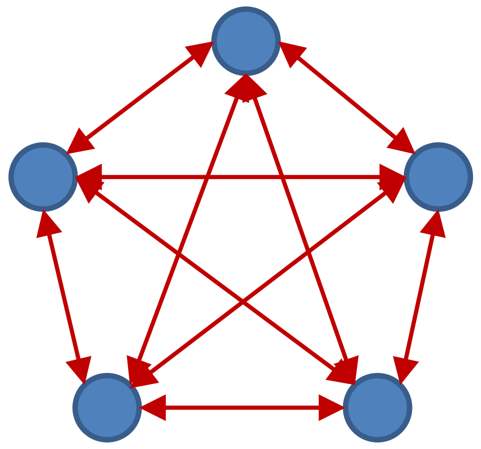


A neuron "flips" if weighted sum of other neuron's outputs is of the opposite sign

But this may cause *other* neurons to flip!

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

Loopy network

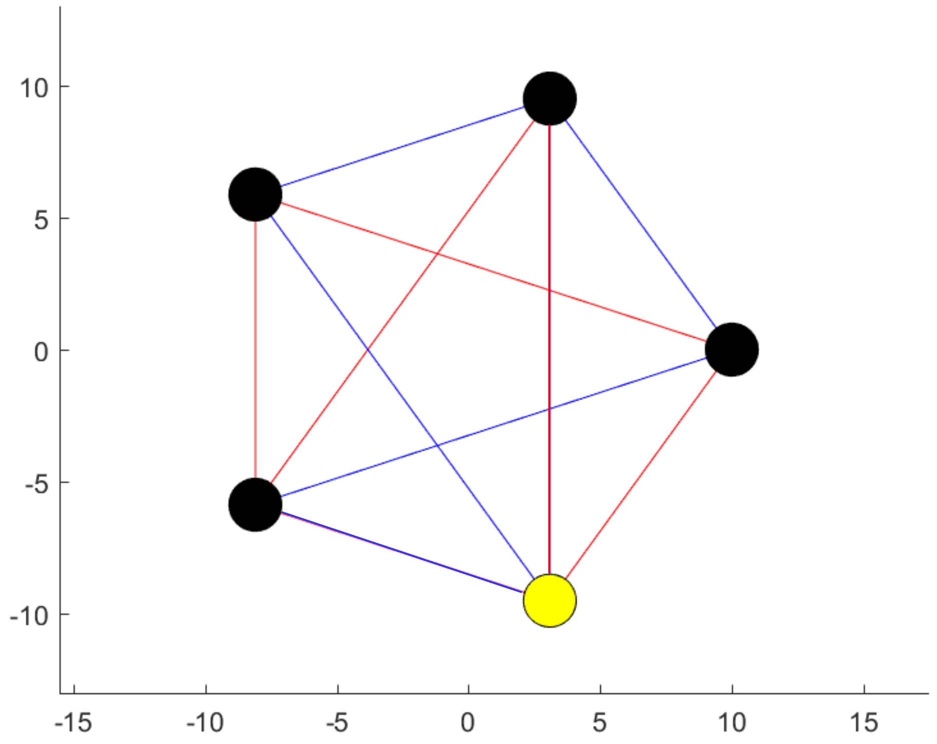


$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

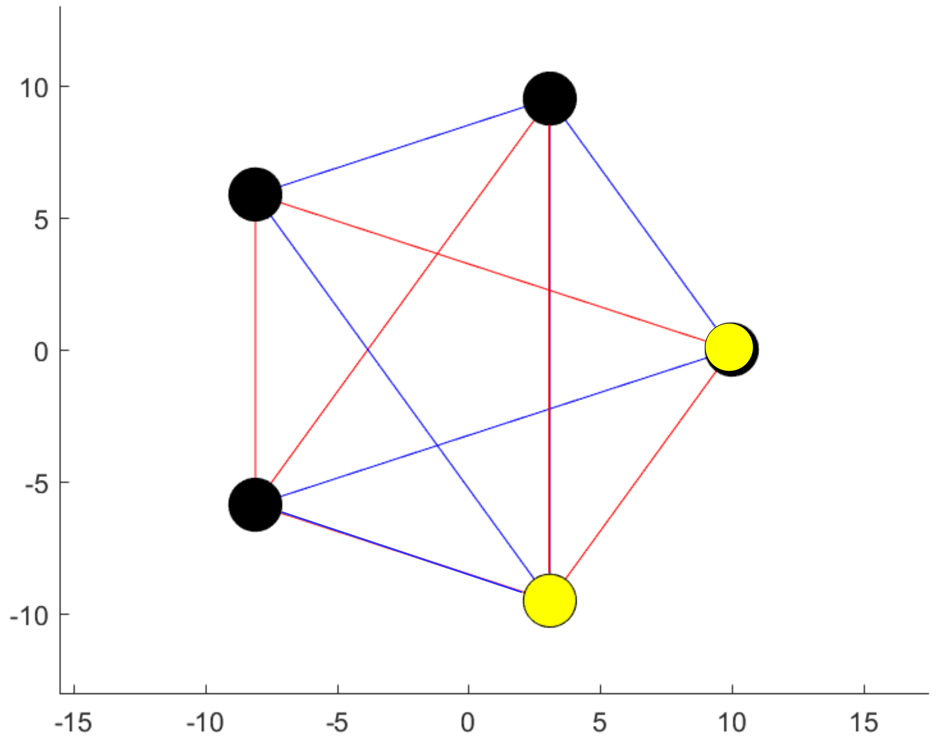
- At each time each neuron receives a "field" $\sum_{j \neq i} w_{ji} y_j + b_i$
- If the sign of the field matches its own sign, it does not respond
- If the sign of the field opposes its own sign, it "flips" to match the sign of the field

Example



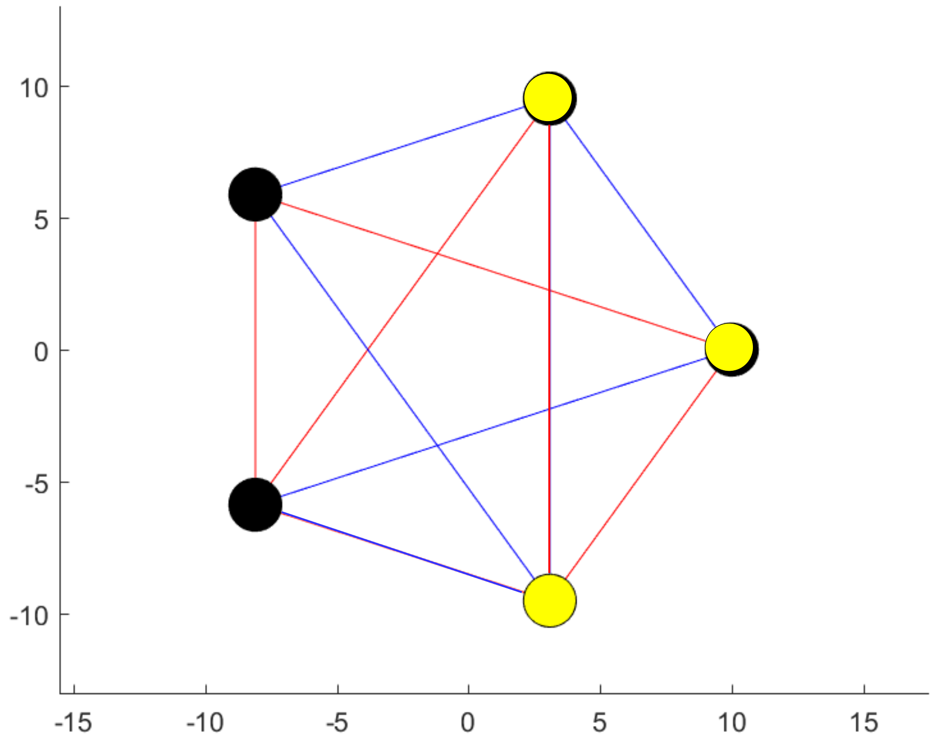
- Red edges are -1, blue edges are +1
- Yellow nodes are +1, black nodes are -1

Example



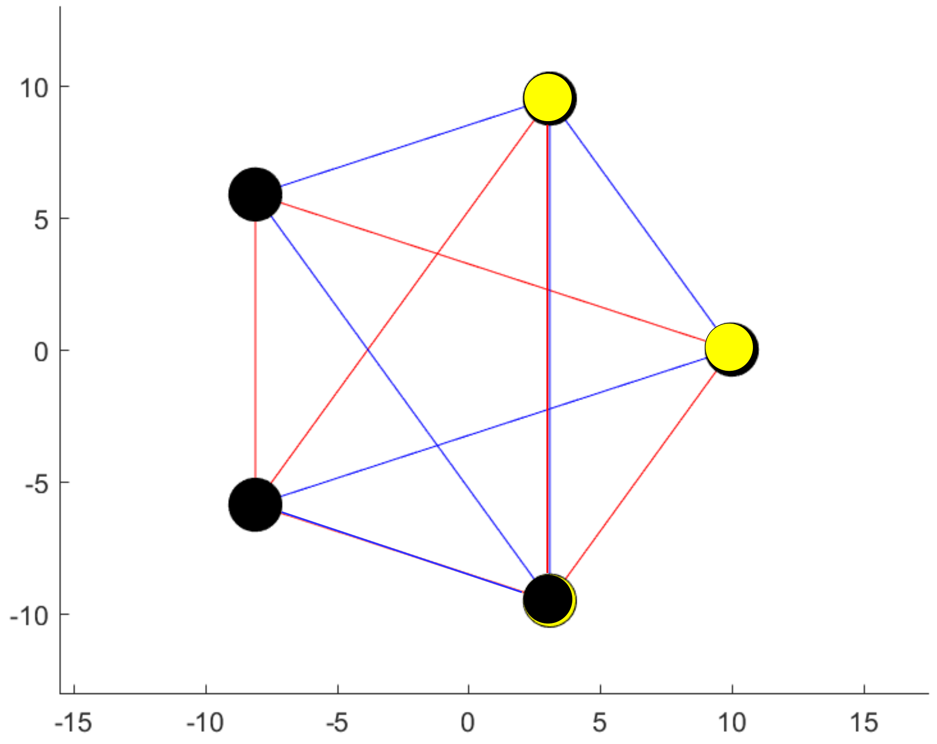
- Red edges are -1, blue edges are +1
- Yellow nodes are +1, black nodes are -1

Example



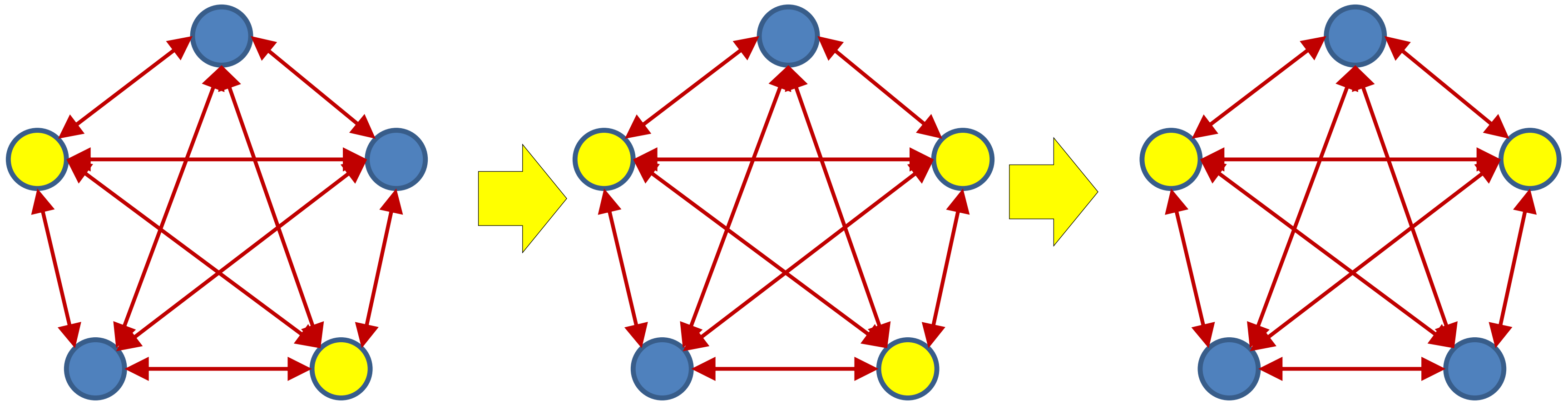
- Red edges are -1, blue edges are +1
- Yellow nodes are +1, black nodes are -1

Example



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

Loopy network

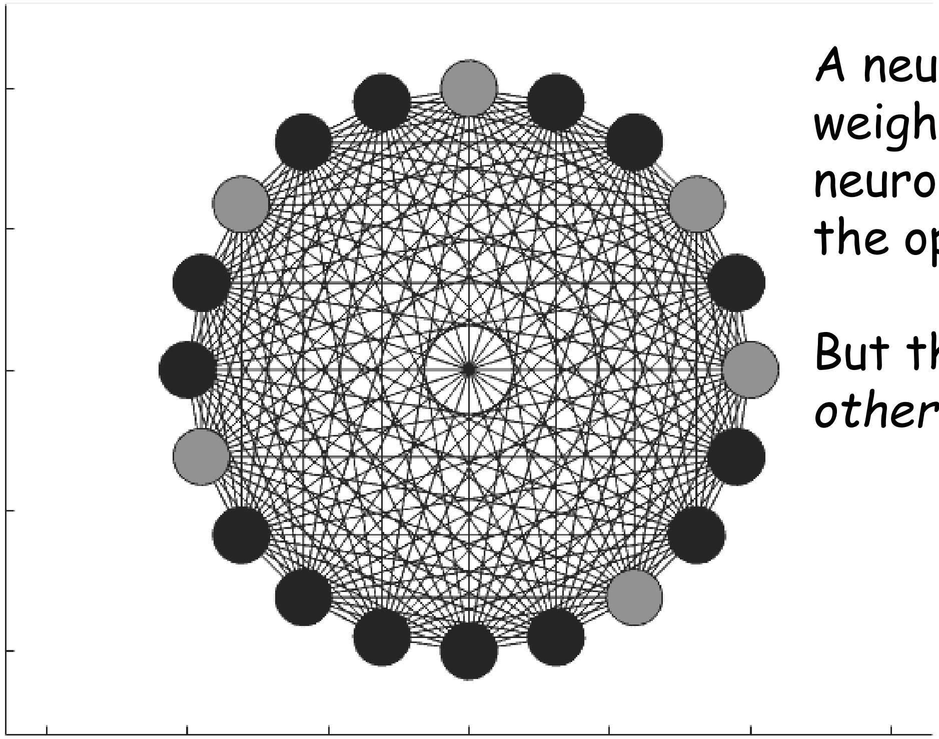


- If the sign of the field at any neuron opposes its own sign, it “flips” to match the field
  - Which will change the field at other nodes
    - Which may then flip
      - Which may cause other neurons including the first one to flip…
        - And so on…

20 evolutions of a loopy net

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

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

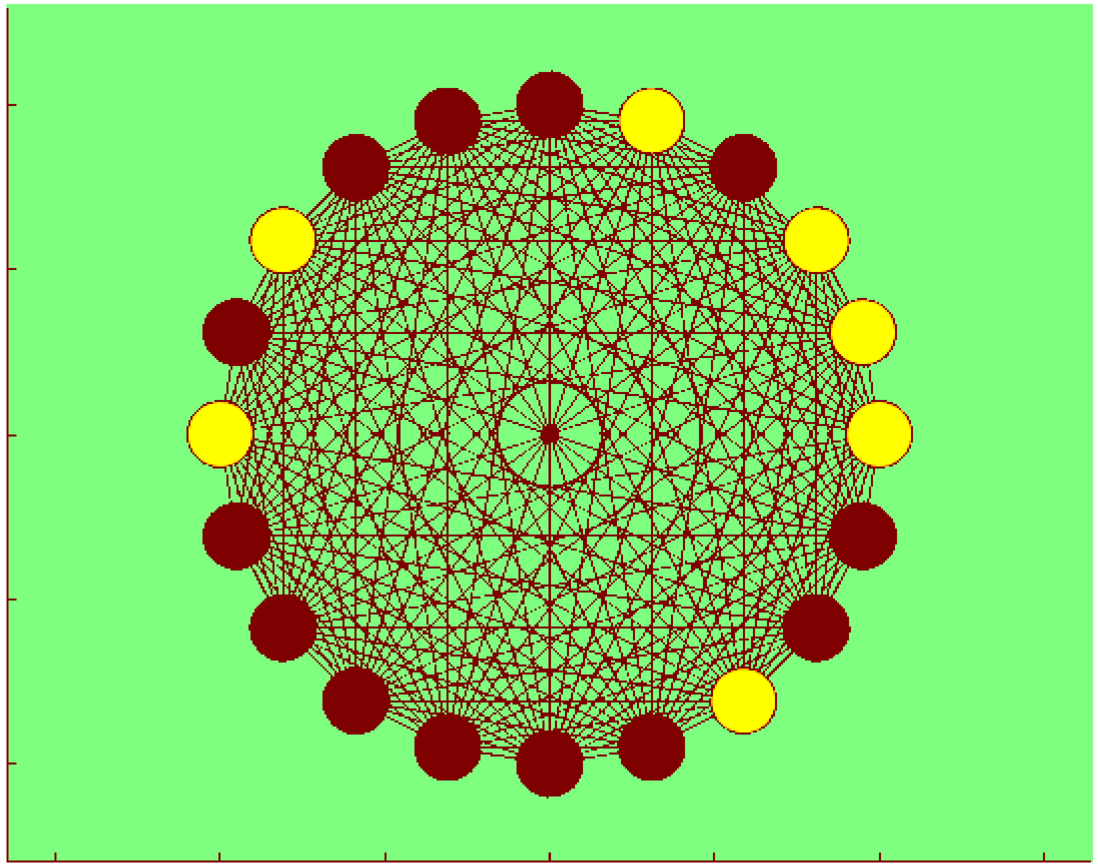


A neuron "flips" if weighted sum of other neuron's outputs is of the opposite sign

But this may cause *other* neurons to flip!

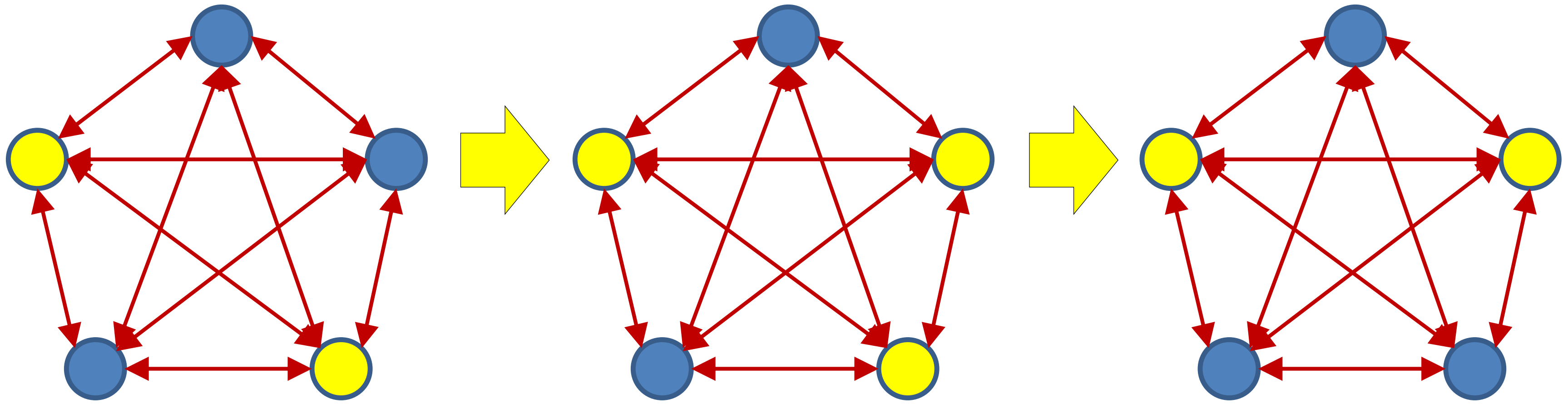
- All neurons which do not "align" with the local field "flip"

120 evolutions of a loopy net



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

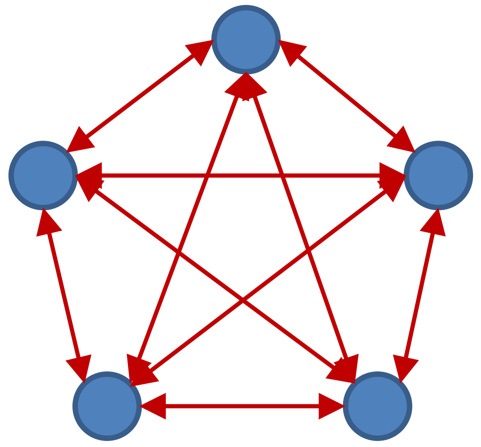
Loopy network



- 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



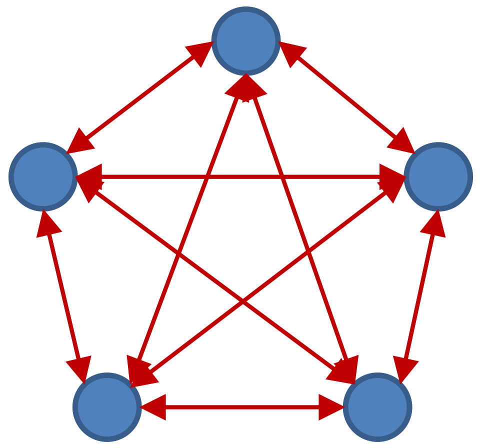
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

- 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

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

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

  - Definition same as earlier, but avoids double counting and assumes $w_{ii} = 0$

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

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

Upon flipping a single unit

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

- Expanding

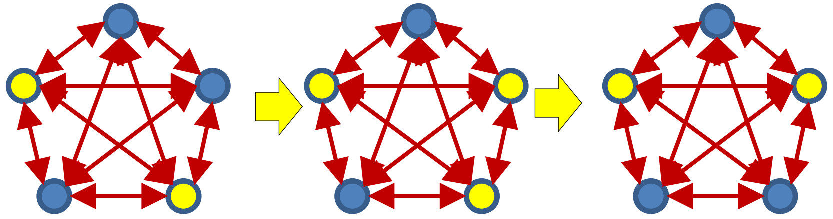
$$\Delta D(y_k) = (y_k^+ - y_k^-) \sum_{j \neq i} w_{jk} y_j + (y_k^+ - y_k^-) b_k$$

  – All other terms that do not include y_k cancel out

- This is always positive!

- *Every flip of a unit results in an increase in* D

Hopfield Net



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

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

- D is bounded

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

- 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

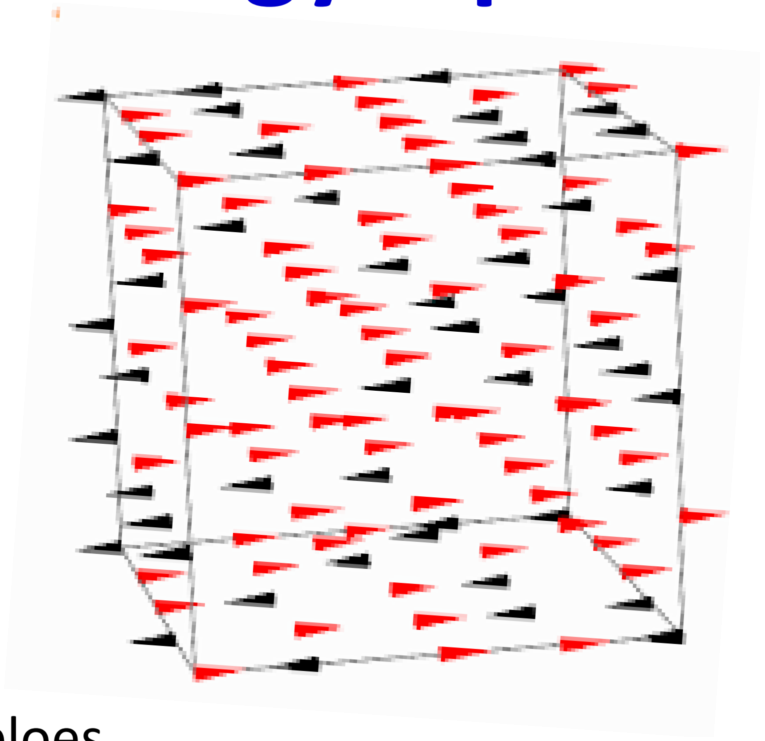
- Define the *Energy* of the network as

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

  – Just the negative of D

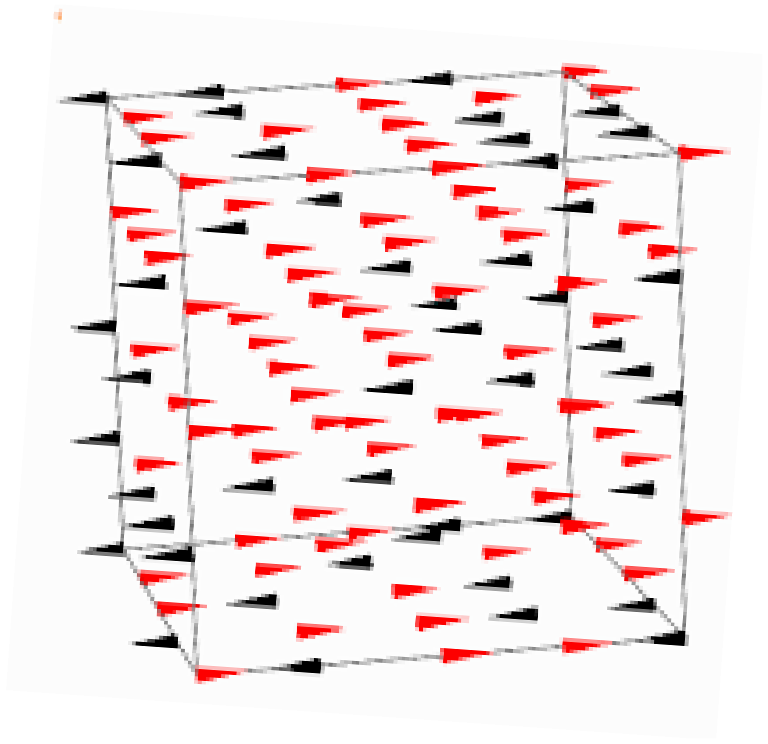
- The evolution of a Hopfield network constantly decreases its energy
- Where did this "energy" concept suddenly sprout from?

Analogy: Spin Glasses



- Magnetic diploes
- 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

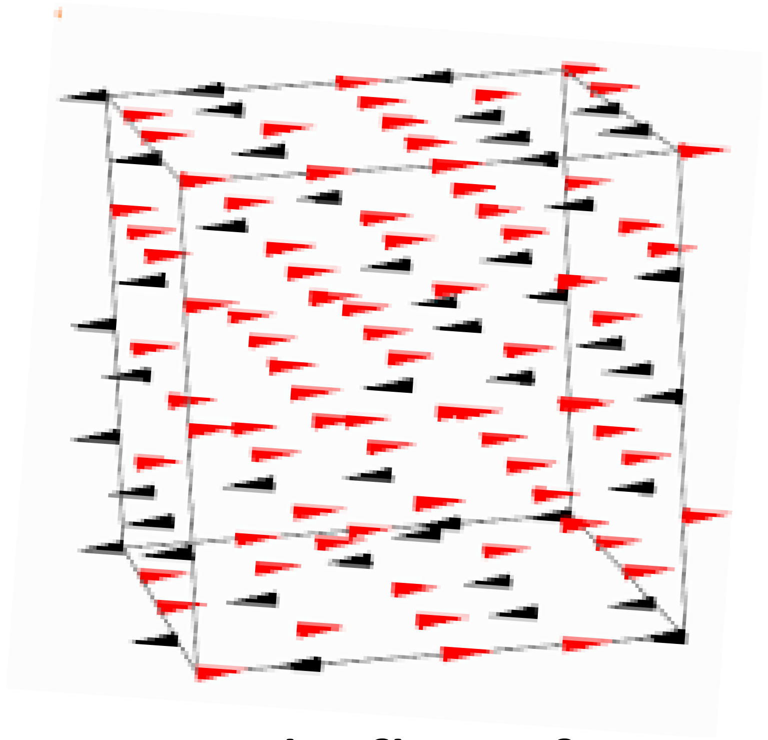


Total field at current dipole:

$$f(p_i) = \underbrace{\sum_{j \neq i} \frac{r x_j}{\|p_i - p_j\|^2}}_{\text{intrinsic}} + \underbrace{b_i}_{\text{external}}$$

- 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 falls off inversely with square of distance

Analogy: Spin Glasses



Total field at current dipole:

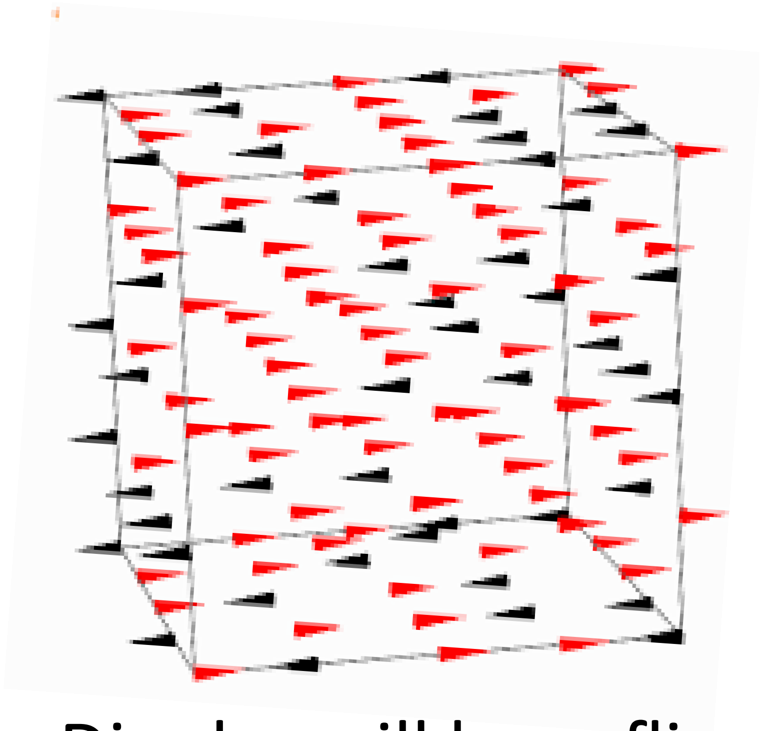
$$f(p_i) = \sum_{j \neq i} \frac{r x_j}{\left\| p_i - p_j \right\|^2} + b_i$$

Response of current diplose

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

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

Analogy: Spin Glasses



Total field at current dipole:

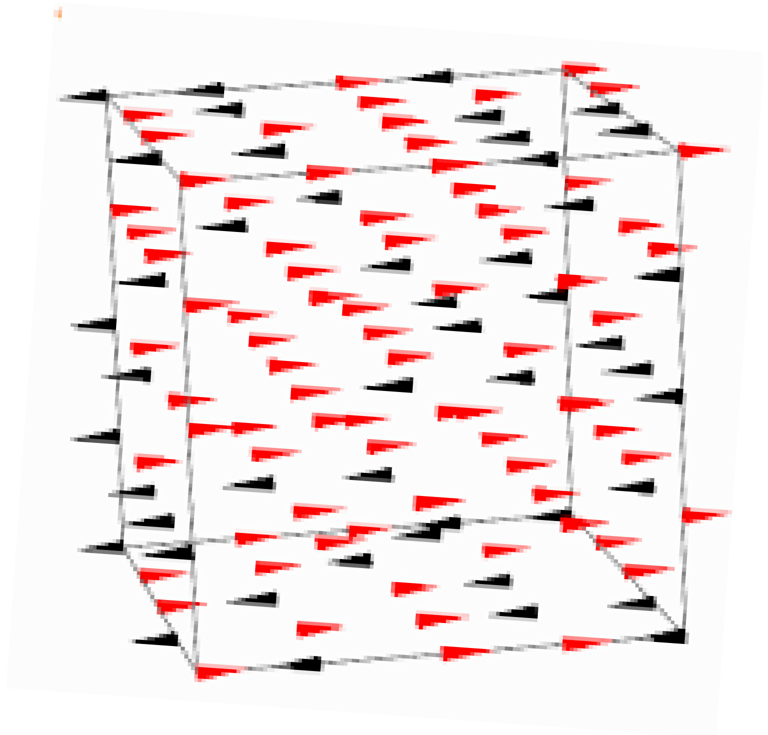
$$f(p_i) = \sum_{j \neq i} \frac{r x_j}{\|p_i - p_j\|^2} + b_i$$

Response of current diplose

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

- 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



- When will it stop???

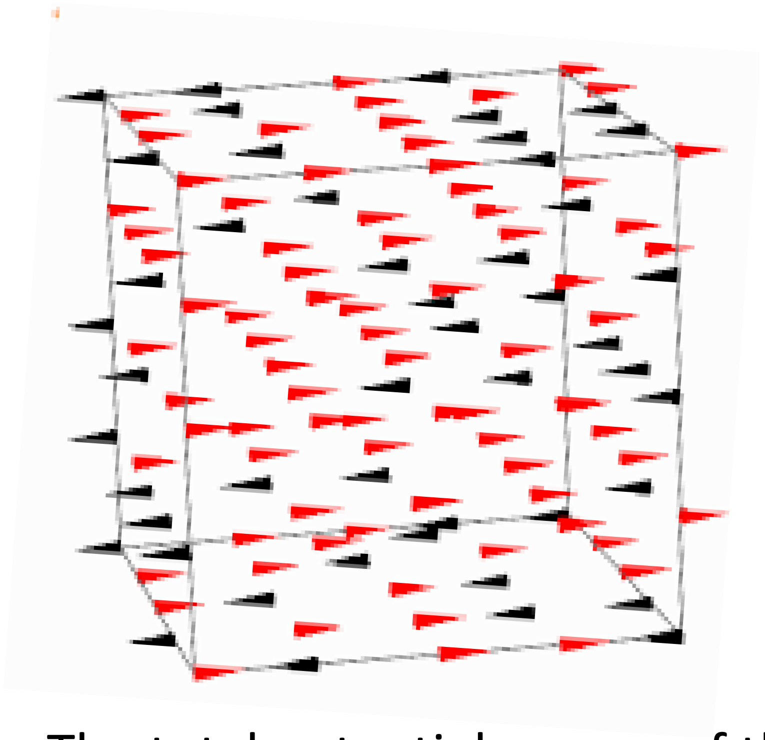
Total field at current dipole:

$$f(p_i) = \sum_{j \neq i} \frac{r x_j}{\left\|p_i - p_j\right\|^2} + b_i$$

Response of current diplose

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

Analogy: Spin Glasses



Total field at current dipole:

$$f(p_i) = \sum_{j \neq i} \frac{r x_j}{\left\| p_i - p_j \right\|^2} + b_i$$

Response of current diplose

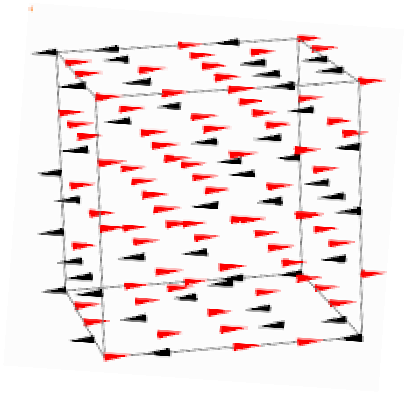
$$x_i = \begin{cases} x_i \; if \; sign\left(x_i \, f(p_i)\right) = 1 \\ -x_i \; otherwise \end{cases}$$

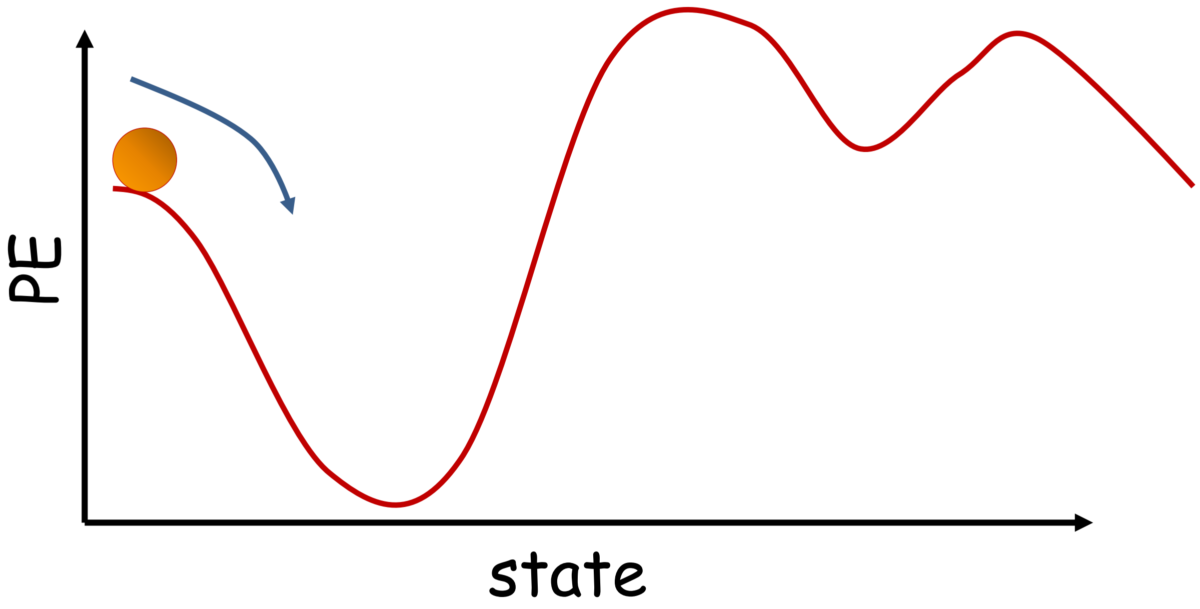
- The total potential energy of the system

$$E = C - \frac{1}{2} \sum_i x_i f(p_i) = C - \sum_i \sum_{j>i} \frac{r x_i x_j}{\left\| p_i - p_j \right\|^2} - \sum_i b_i x_j$$

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

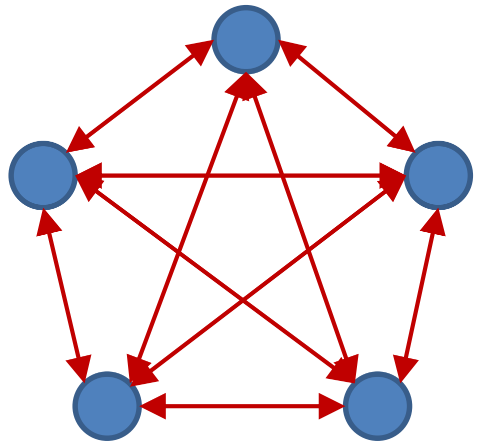
Spin Glasses





- The system stops at one of its *stable* configurations
  - Where PE 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



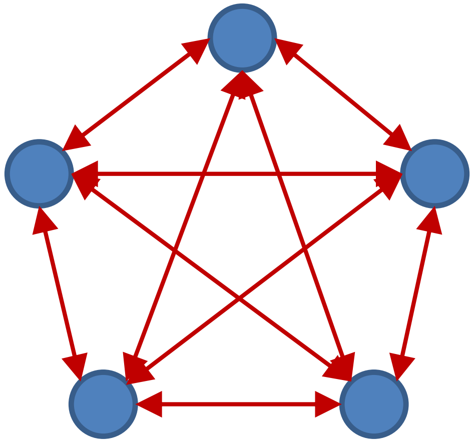
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

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

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

Hopfield Network



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

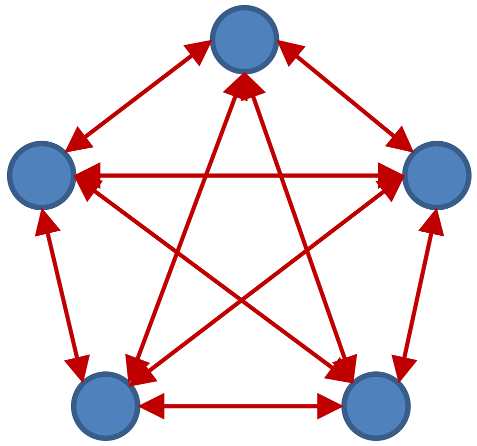
$$\Theta(z) = \begin{cases} +1 \text{ } if \text{ } z > 0 \\ -1 \text{ } if \text{ } z \leq 0 \end{cases}$$

Typically will not utilize bias: The bias is similar to having a single extra neuron that is pegged to 1.0

Removing the bias term does not affect the rest of the discussion in any manner

But not RIP, we will bring it back later in the discussion

Hopfield Network



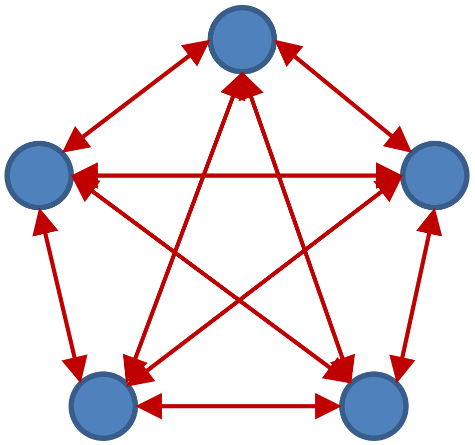
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j\right)$$

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

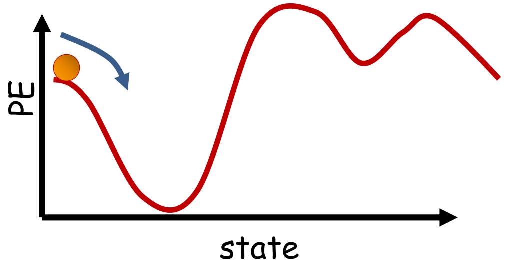
$$E = -\sum_{i,j<i} w_{ij} y_i y_j$$

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

Evolution

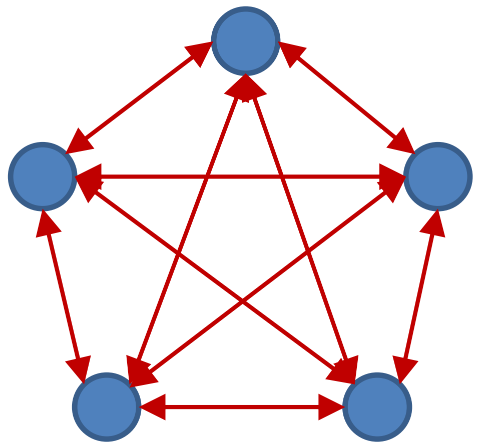


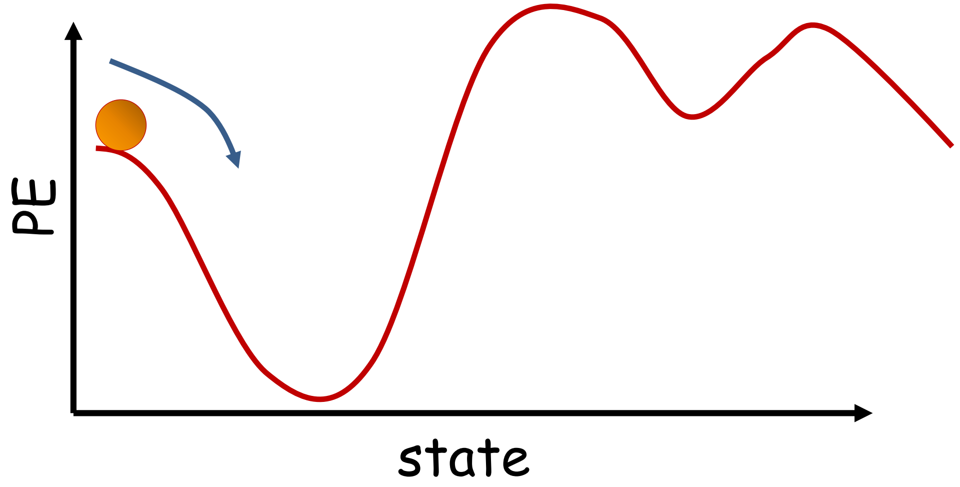
$$E = -\sum_{i,j<i} w_{ij} y_i y_j$$



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

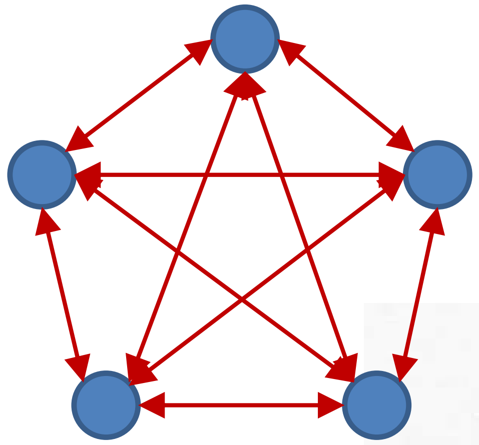
Content-addressable memory





- 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***

Evolution



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

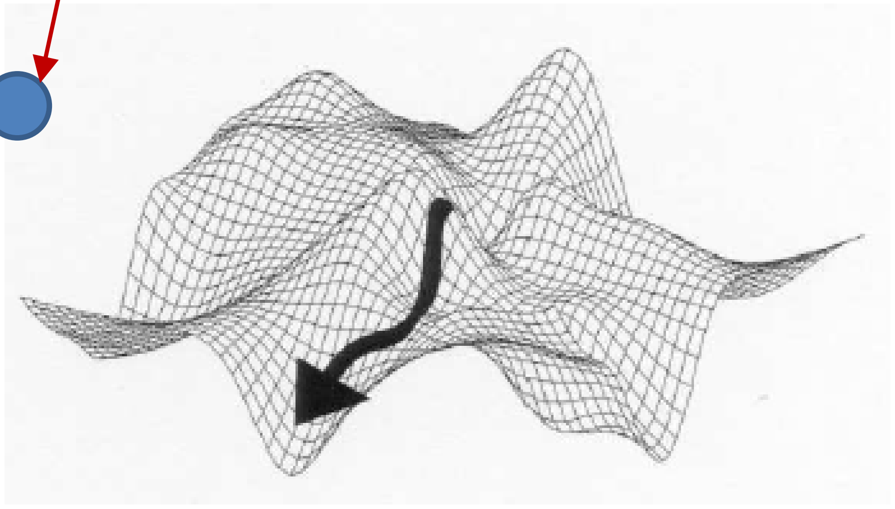
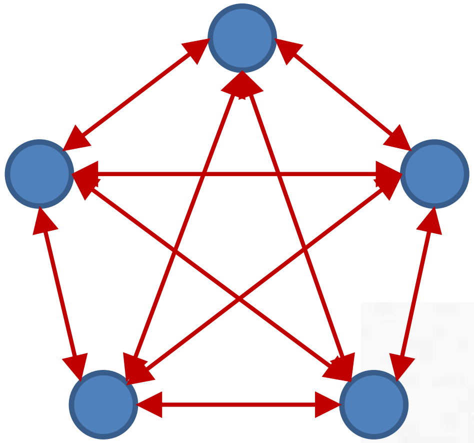


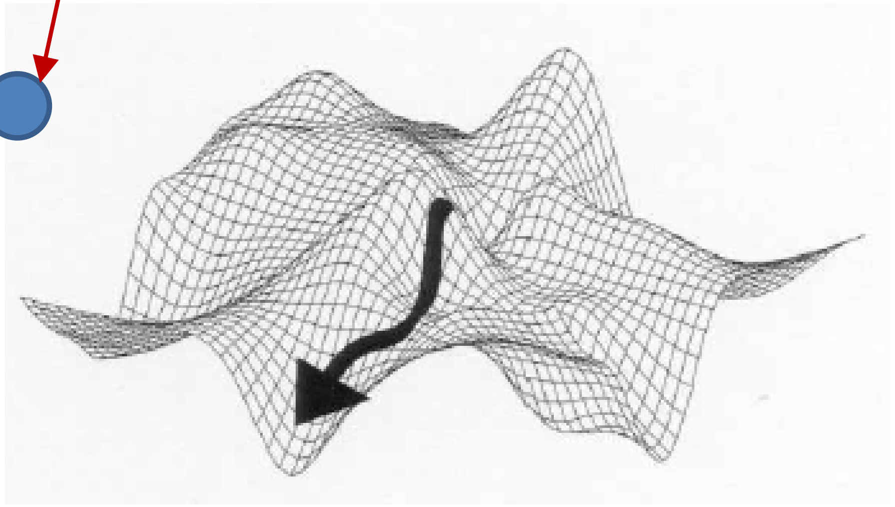
Image pilfered from unknown source

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

Evolution

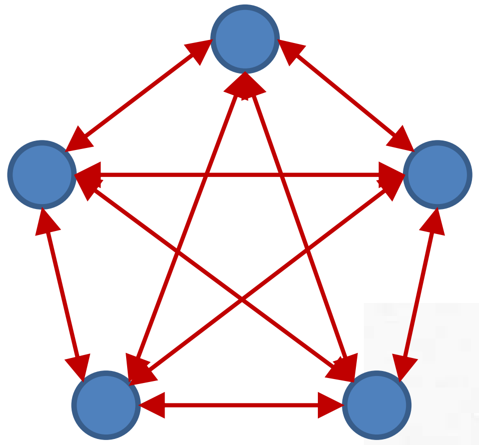


$$E = -\sum_{i,j<i} w_{ij} y_i y_j$$

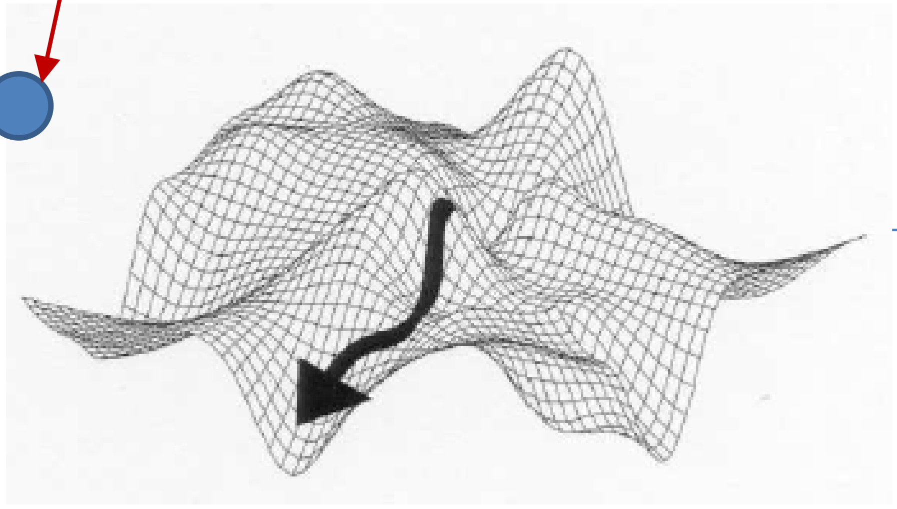


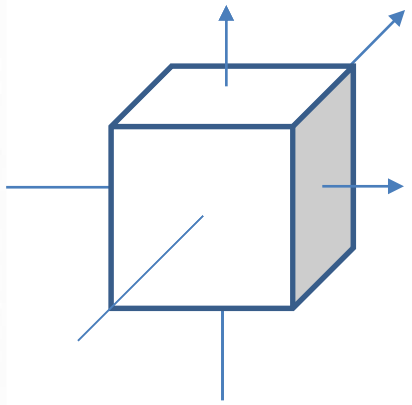
- The network will evolve until it arrives at a local minimum in the energy contour
- We proved that *every* change in the network will result in *decrease* in energy
  - So path to energy minimum is monotonic

Evolution



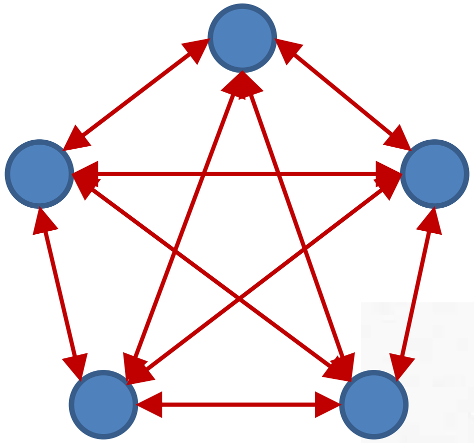
$$E = -\sum_{i,j<i} w_{ij} y_i y_j$$



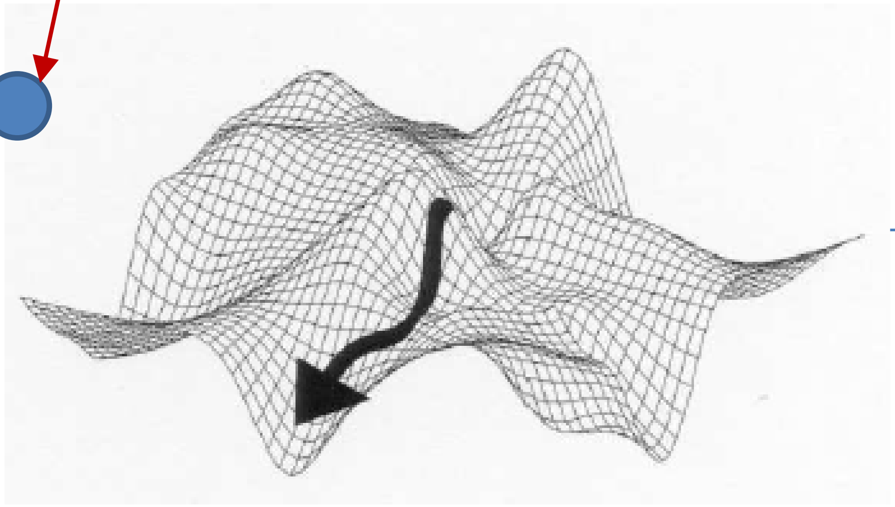


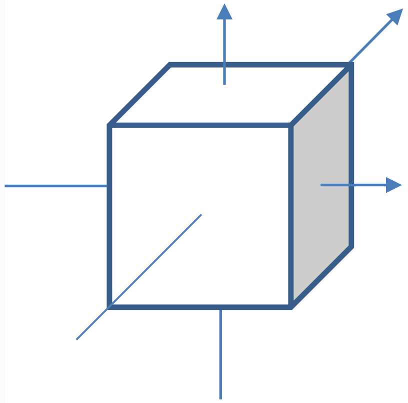
- For threshold activations the energy contour is only defined on a lattice
  - Corners of a unit cube on $[-1,1]^N$

Evolution



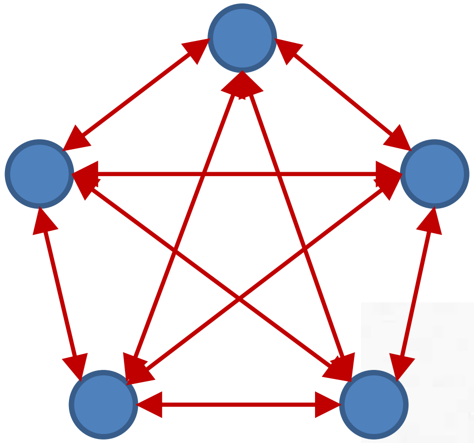
$$E = -\sum_{i,j<i} w_{ij} y_i y_j$$

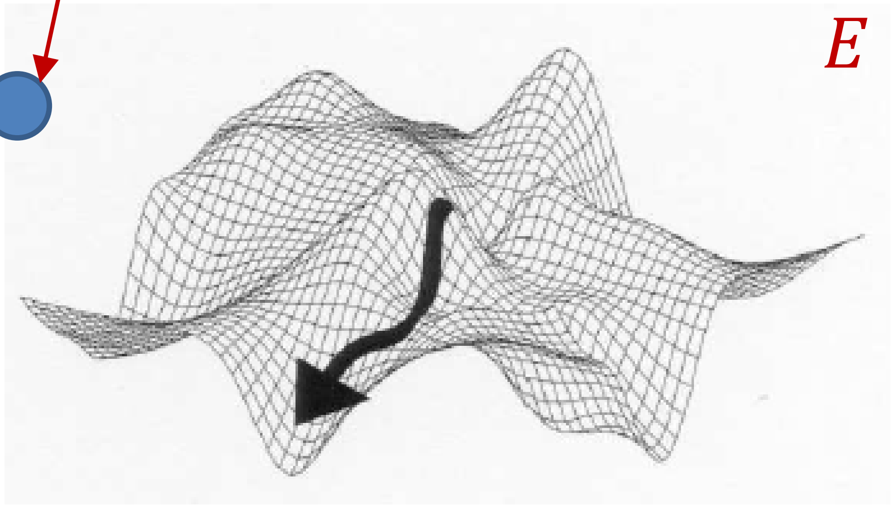




- For threshold activations the energy contour is only defined on a lattice
  - Corners of a unit cube on $[-1,1]^N$
- For tanh activations it will be a continuous function

Evolution





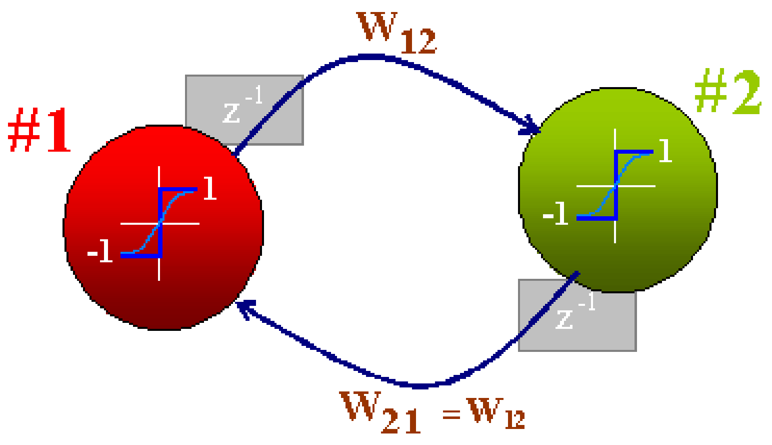
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$

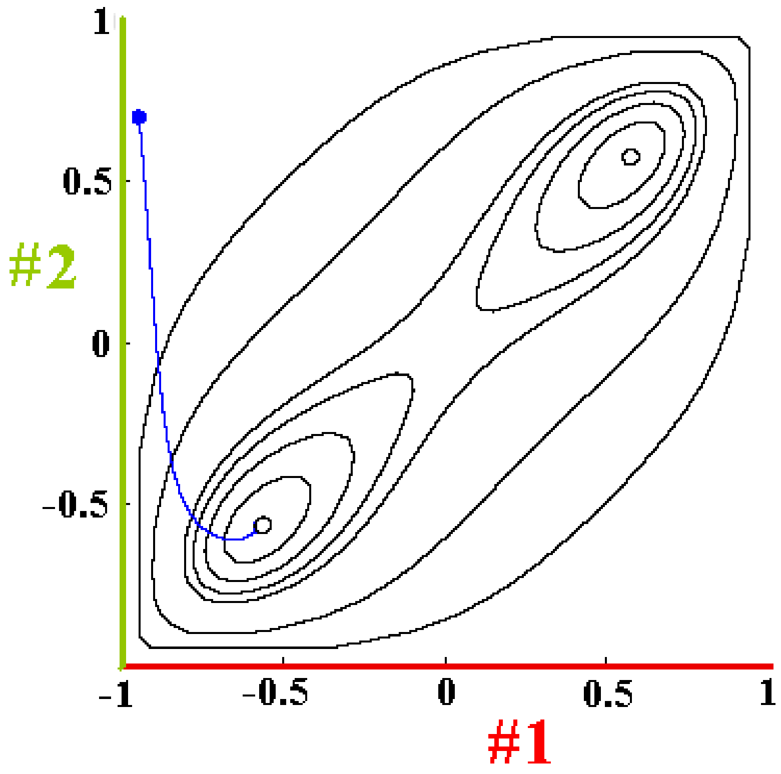
In matrix form

Note the 1/2

- For threshold activations the energy contour is only defined on a lattice
  - Corners of a unit cube
- For tanh activations it will be a continuous function

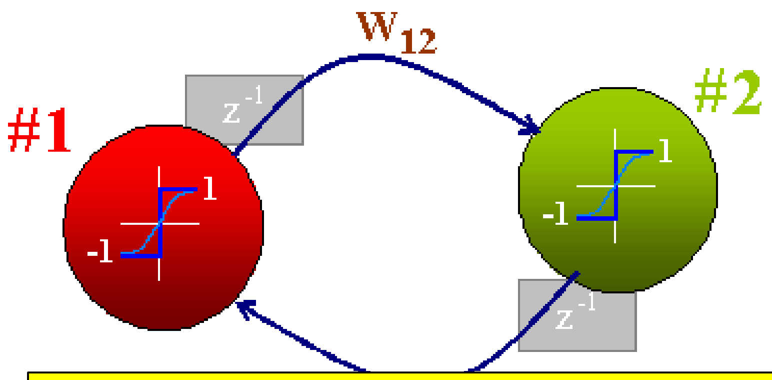
“Energy”contour for a 2-neuron net

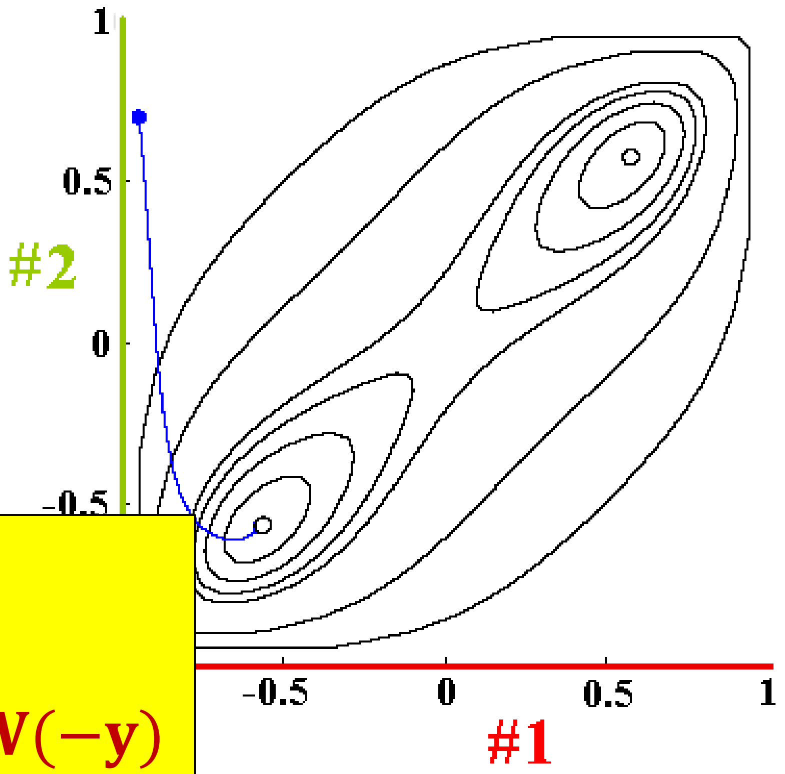




- Two stable states (tanh activation)
  - Symmetric, not at corners
  - Blue arc shows a typical trajectory for sigmoid activation

"Energy"contour for a 2-neuron net





Why symmetric?

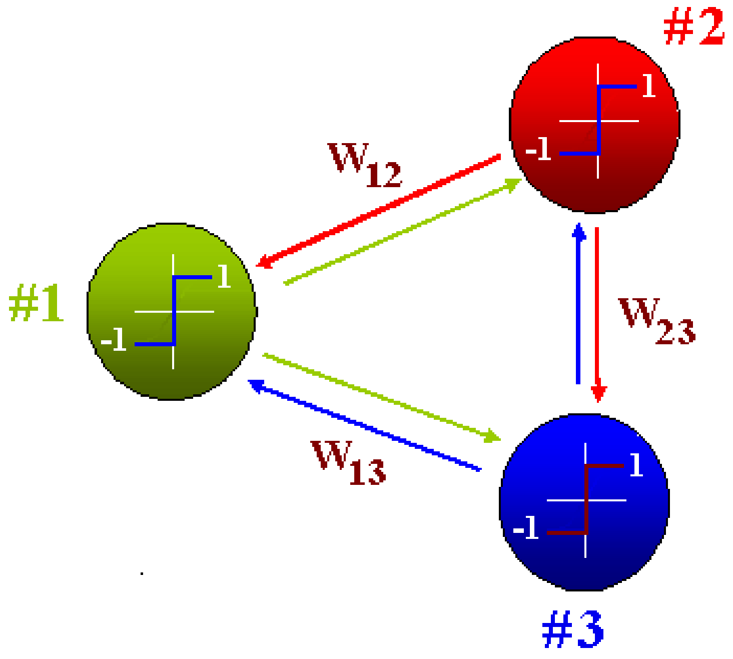
Because $-\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} = -\frac{1}{2}(-\mathbf{y})^T\mathbf{W}(-\mathbf{y})$

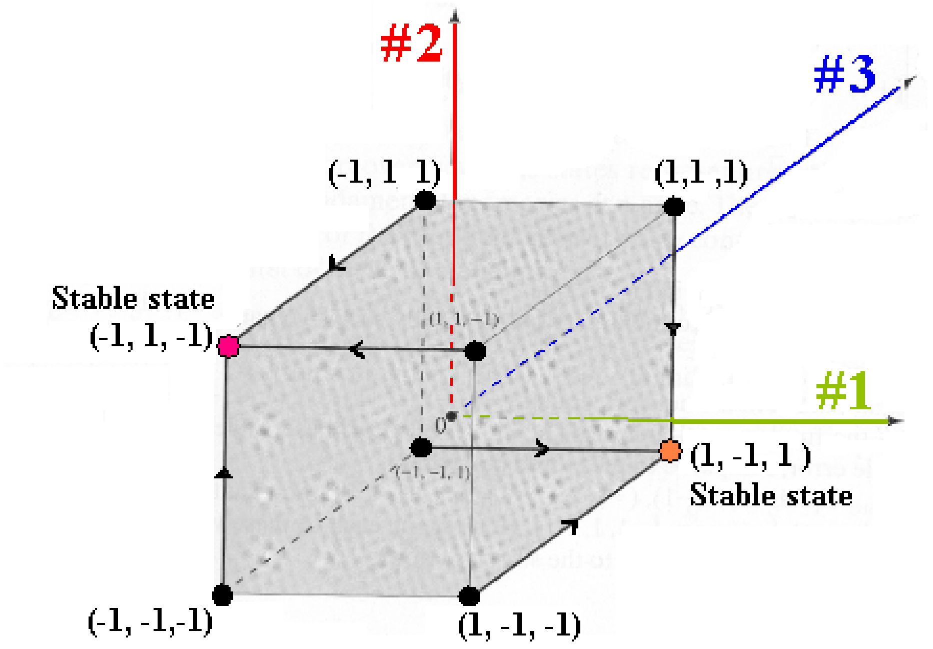
If $\hat{\mathbf{y}}$ is a local minimum, so is $-\hat{\mathbf{y}}$

•

– Blue arc shows a typical trajectory for sigmoid activation

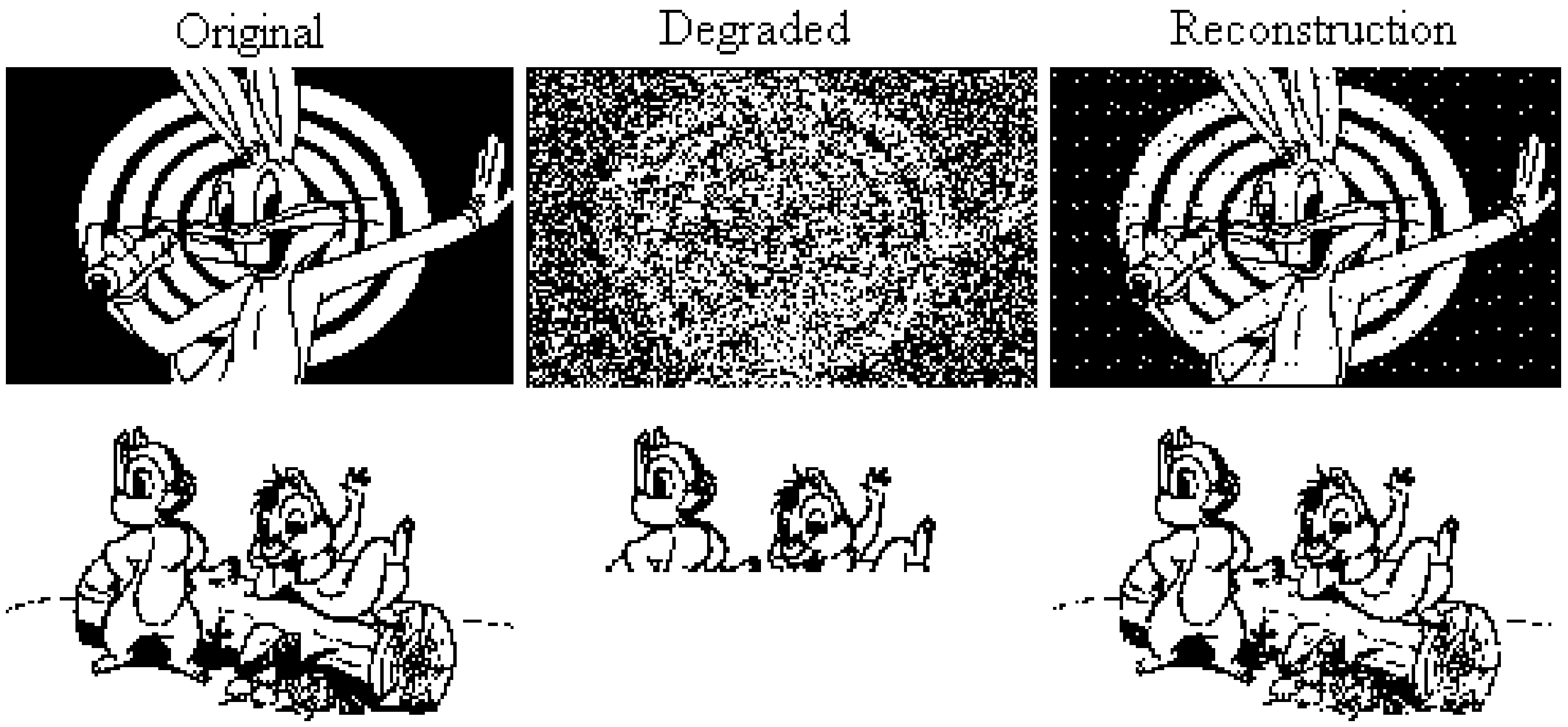
3-neuron net





- 8 possible states
- 2 stable states (hard thresholded network)

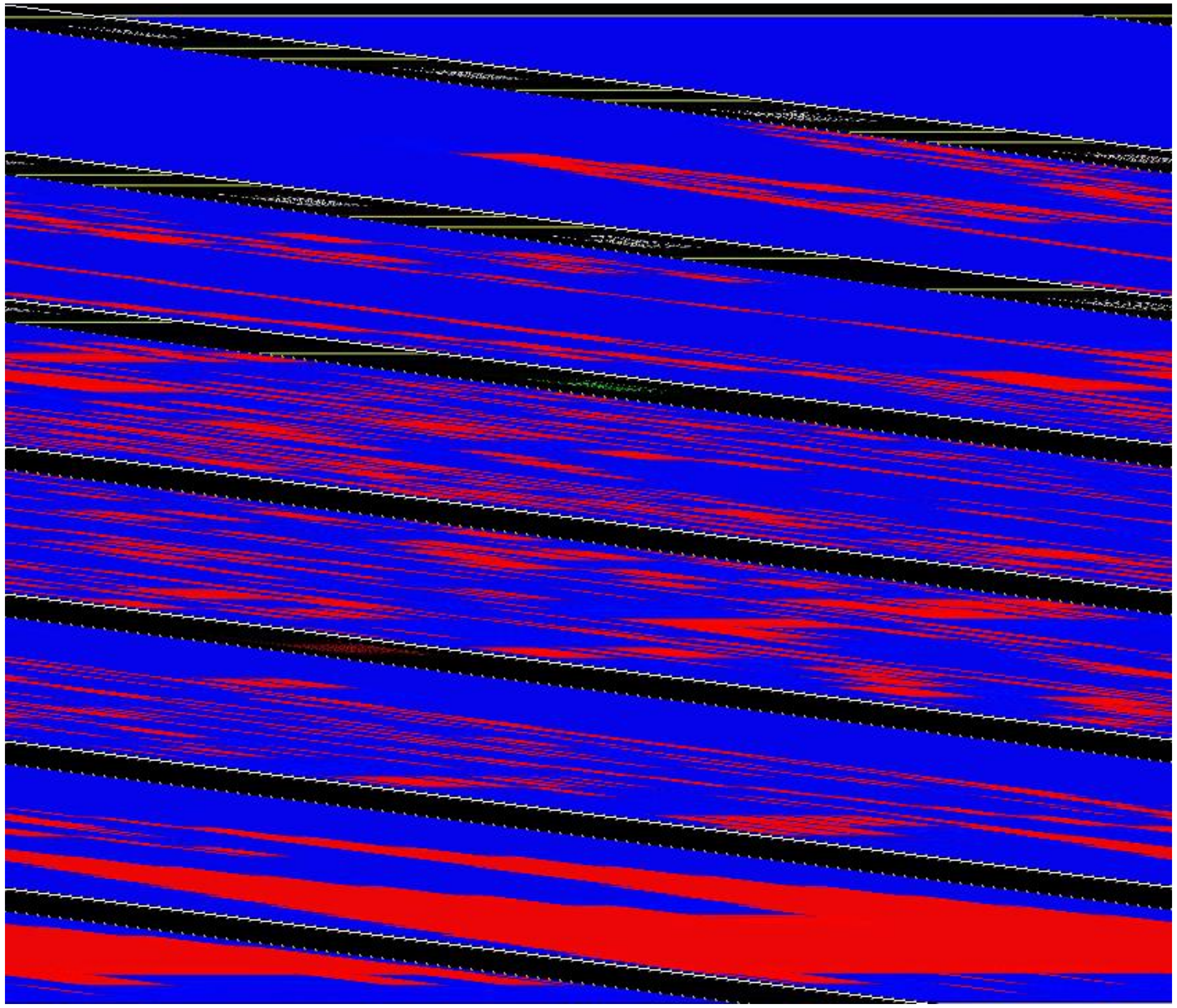
Examples: Content addressable memory



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

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

- 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

Issues

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

- How many patterns can we store?

Issues

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

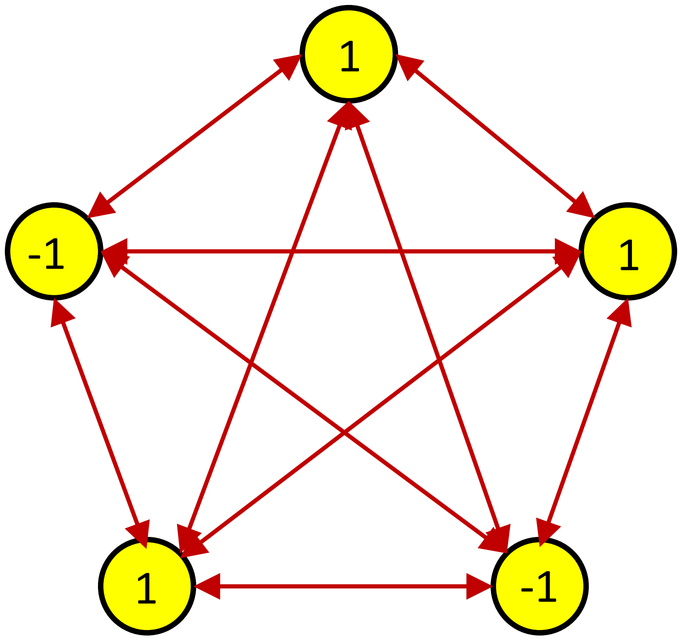
- How many patterns can we store?

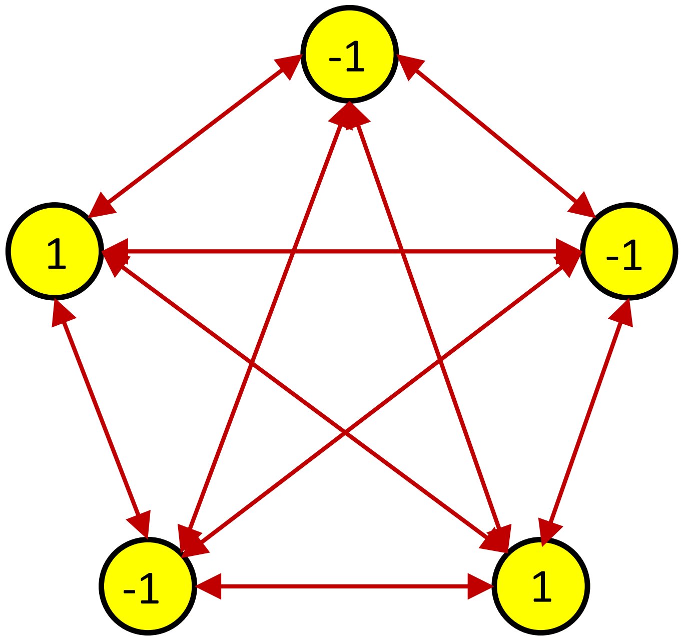
How do we remember a *specific* pattern?

- 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 patterns: Training a network

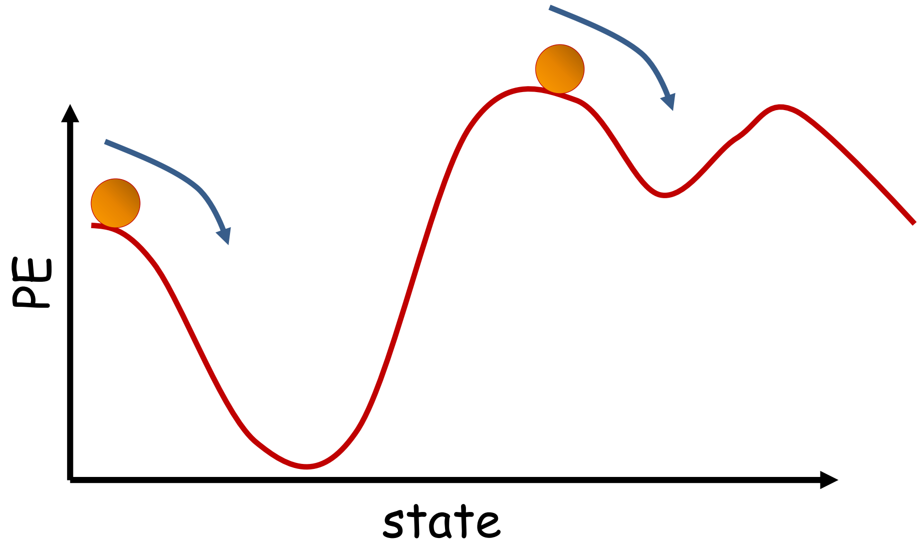


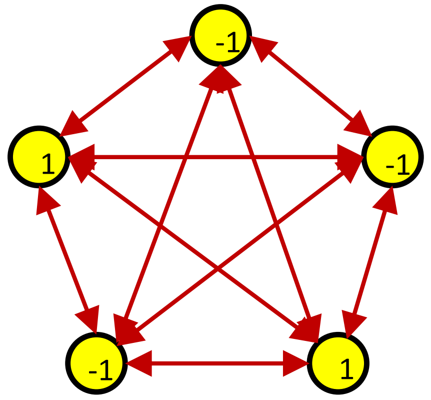


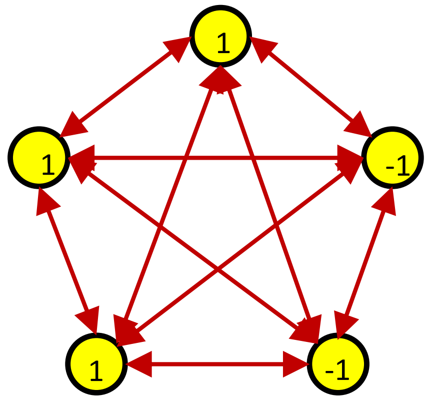
- A network that stores pattern P also naturally stores – P
  - Symmetry $E(P) = E(-P)$ since E is a function of $y_i y_j$

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

A network can store *multiple* patterns

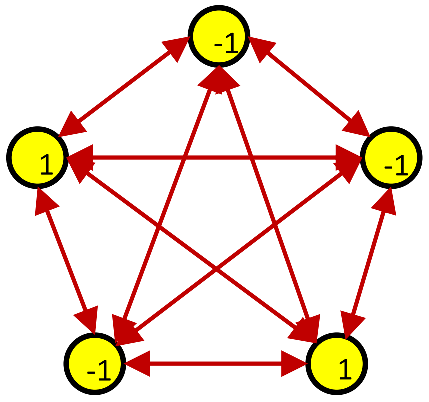


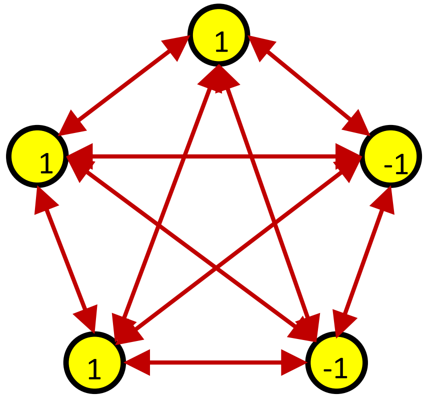




- Every stable point is a stored pattern
- So we could design the net to store multiple patterns
  - Remember that every stored pattern P is actually *two* stored patterns, P and $-P$

Storing a pattern

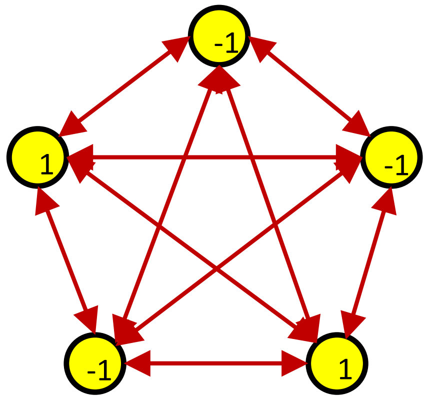




$$E = -\sum_i \sum_{j<i} w_{ji} y_j y_i$$

- Design $\{w_{ij}\}$ such that the energy is a local minimum at the desired $P = \{y_i\}$

Storing specific patterns

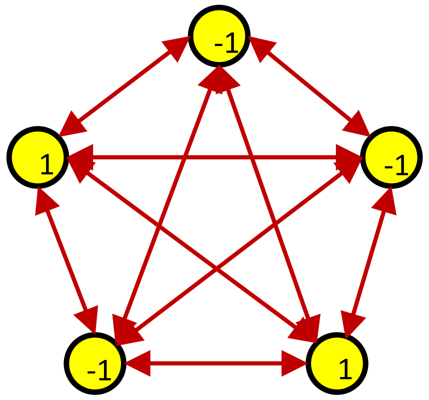


- Storing 1 pattern: We want

$$sign\left(\sum_{j\neq i} w_{ji}y_j\right) = y_i \quad \forall\, i$$

- This is a stationary pattern

Storing specific patterns



HEBBIAN LEARNING:

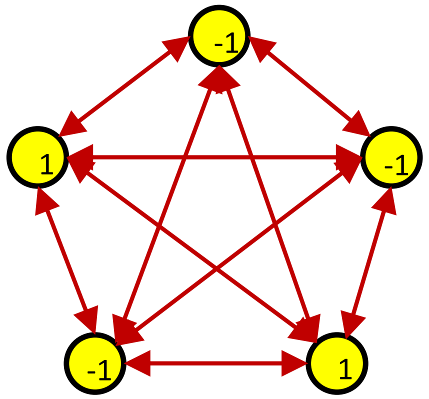
$$w_{ji} = y_j y_i$$

- Storing 1 pattern: We want

$$sign\left(\sum_{j \neq i} w_{ji} y_j\right) = y_i \quad \forall\, i$$

- This is a stationary pattern

Storing specific patterns



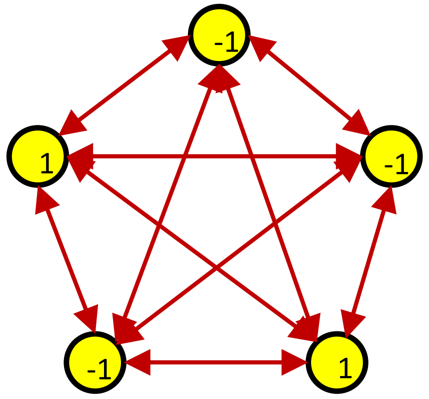
HEBBIAN LEARNING:

$$w_{ji} = y_j y_i$$

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

$$= sign\left(\sum_{j \neq i} y_j^2 y_i\right) = sign(y_i) = y_i$$

Storing specific patterns



HEBBIAN LEARNING:

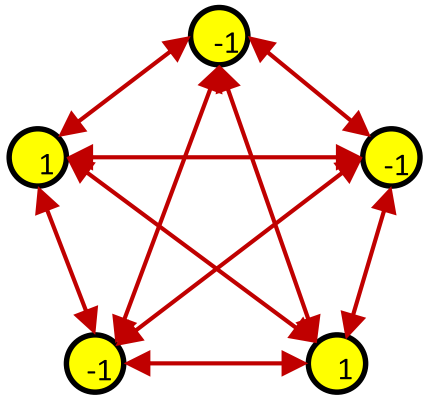
$$w_{ji} = y_j y_i$$

The pattern is stationary

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

$$= sign\left(\sum_{j\neq i} y_j^2 y_i\right) = sign(y_i) = y_i$$

Storing specific patterns



HEBBIAN LEARNING:

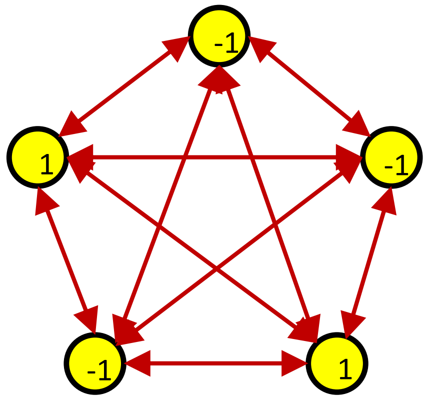
$$w_{ji} = y_j y_i$$

$$E = -\sum_i \sum_{j<i} w_{ji} y_j y_i = -\sum_i \sum_{j<i} y_i^2 y_j^2$$

$$= -\sum_i \sum_{j<i} 1 = -0.5N(N-1)$$

- This is the lowest possible energy value for the network

Storing specific patterns



HEBBIAN LEARNING:

$$w_{ji} = y_j y_i$$

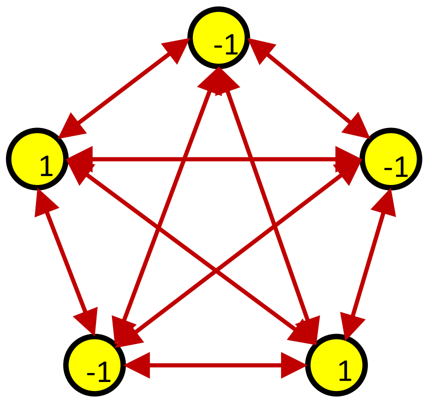
The pattern is *STABLE*

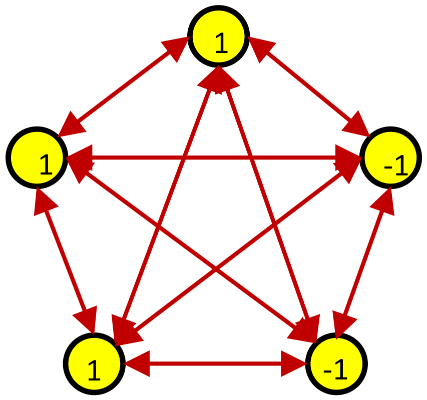
$$E = -\sum_i \sum_{j<i} w_{ji} y_j y_i = -\sum_i \sum_{j<i} y_i^2 y_j^2$$

$$= -\sum_i \sum_{j<i} 1 = -0.5N(N-1)$$

- This is the lowest possible energy value for the network

Storing multiple patterns

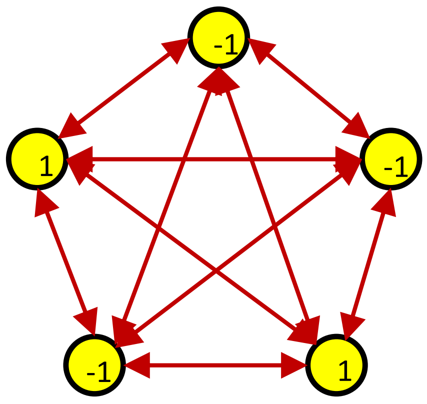


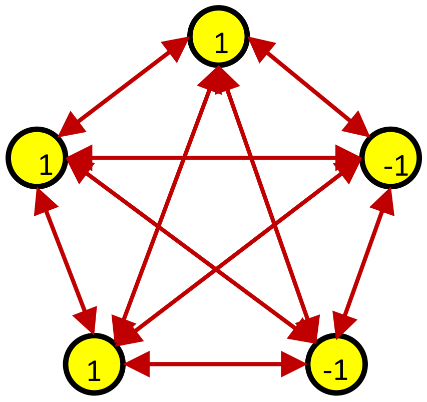


$$w_{ji} = \sum_{p \in \{y_p\}} y_i^p y_j^p$$

- $\{y_p\}$ is the set of patterns to store
- Superscript p represents the specific pattern

Storing multiple patterns





- Let $\mathbf{y}_p$ be the vector representing p-th pattern
- Let $\mathbf{Y} = [\mathbf{y}_1\ \mathbf{y}_2\ \ \ldots]$ be a matrix with all the stored pattern
- Then..

$$\mathbf{W} = \sum_{p} (\mathbf{y}_p \mathbf{y}_p^T - \mathrm{I}) = \mathbf{Y}\mathbf{Y}^T - N_p \mathbf{I}$$

Number of patterns

Storing multiple patterns

- Note behavior of $\mathbf{E(y)} = \mathbf{y}^T\mathbf{W}\mathbf{y}$ with

$$\mathbf{W} = \mathbf{Y}\mathbf{Y}^T - N_p\mathbf{I}$$

Energy landscape only differs by an additive constant

- Is identical to behavior with

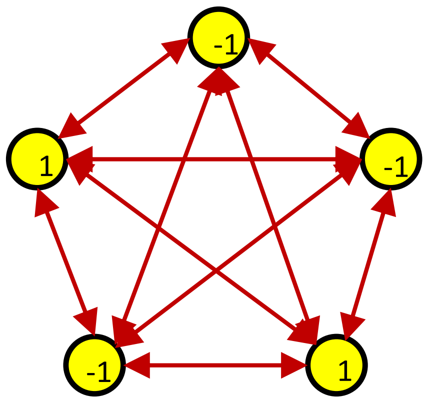
$$\mathbf{W} = \mathbf{Y}\mathbf{Y}^T$$

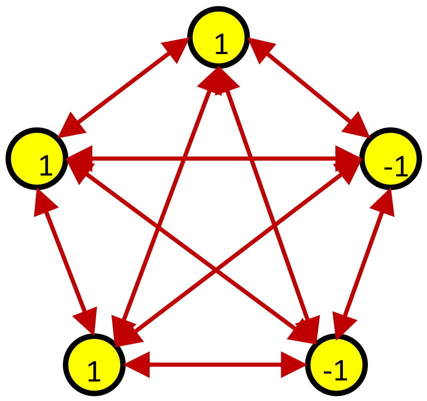
- Since

$$\mathbf{y}^T\left(\mathbf{Y}\mathbf{Y}^T - N_p\mathbf{I}\right)\mathbf{y} = \mathbf{y}^T\mathbf{Y}\mathbf{Y}^T\mathbf{y} - NN_p$$

- But the latter $\mathbf{W} = \mathbf{Y}\mathbf{Y}^T$ is easier to analyze. Hence in the following slides we will use $\mathbf{W} = \mathbf{Y}\mathbf{Y}^T$

Storing multiple patterns





- Let $\mathbf{y}_p$ be the vector representing p-th pattern
- Let $\mathbf{Y} = [\mathbf{y}_1\ \mathbf{y}_2\ \ \dots]$ be a matrix with all the stored pattern
- Then..

$$\mathbf{W} = \mathbf{Y}\mathbf{Y}^T$$

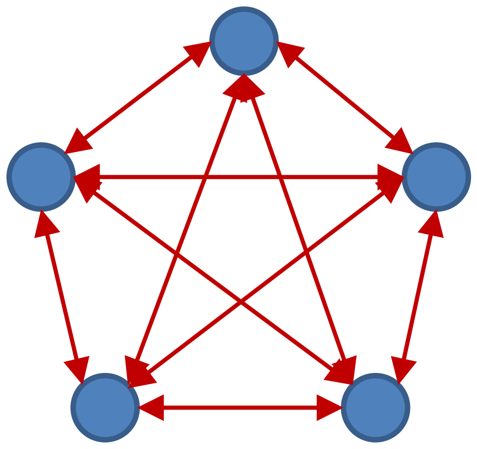
Positive semidefinite!

Issues

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

- How many patterns can we store?

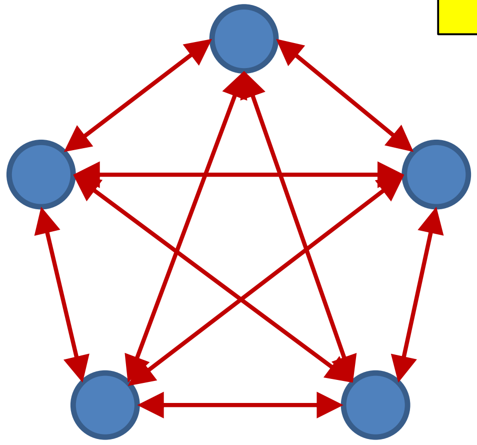
Consider the energy function



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

- Reinstating the bias term for completeness sake
  - Remember that we don’t actually use it in a Hopfield net

Consider the energy function



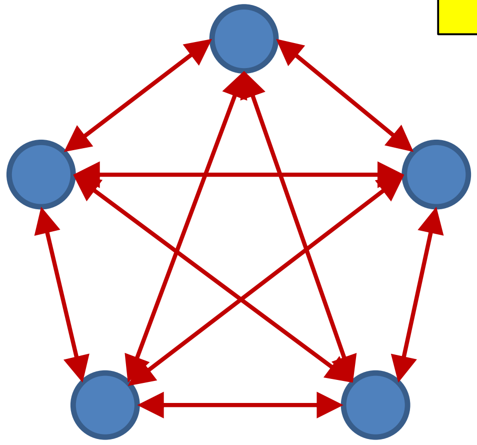
This is a quadratic!

W is positive semidefinite

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

- Reinstating the bias term for completeness sake
  - Remember that we don’t actually use it in a Hopfield net

Consider the energy function



This is a quadratic!

For Hebbian learning
W is positive semidefinite

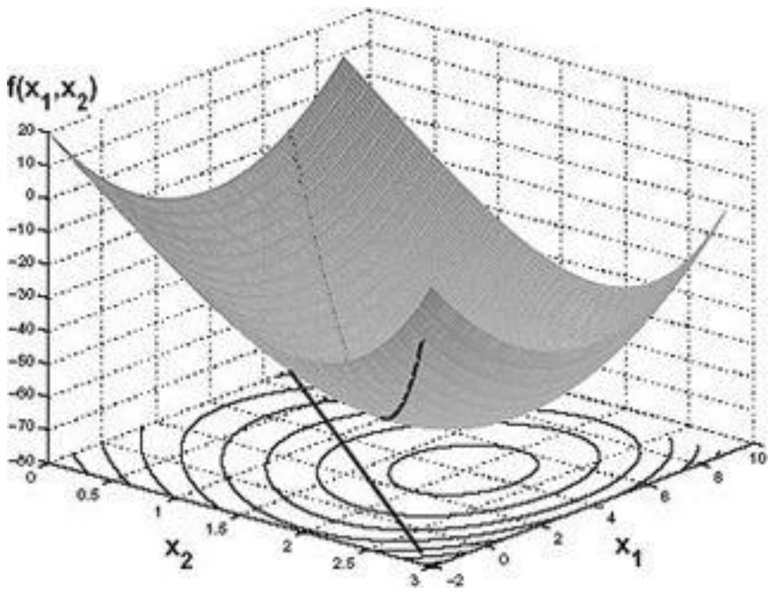
E is convex

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

- Reinstating the bias term for completeness sake
  - Remember that we don’t actually use it in a Hopfield net

The energy function

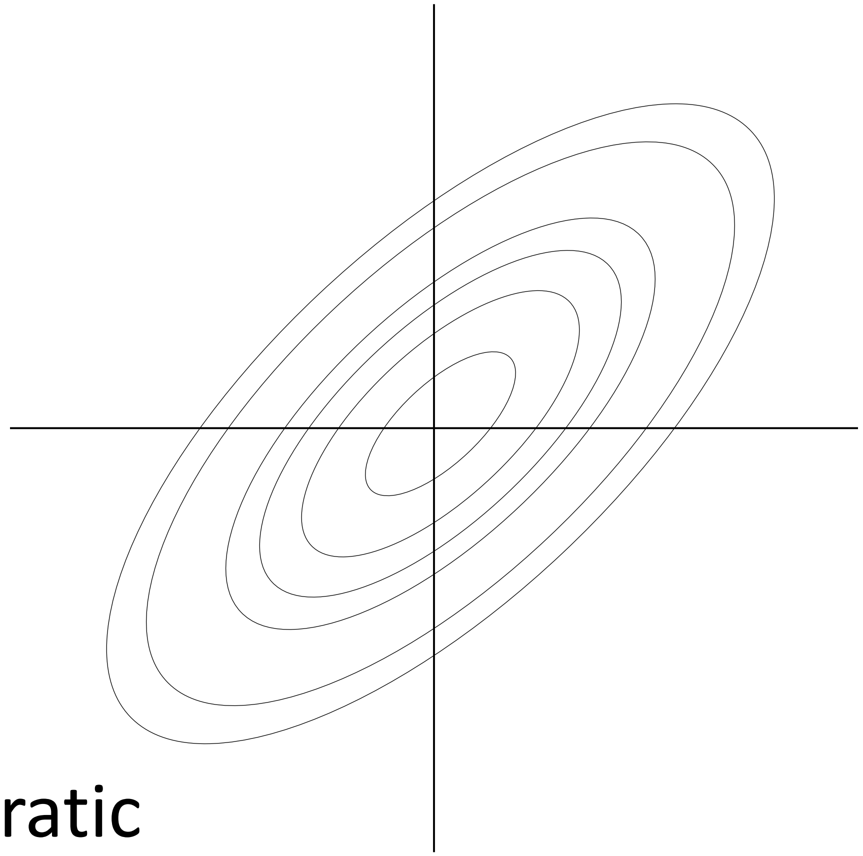
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y}$$



- E is a convex quadratic

The energy function

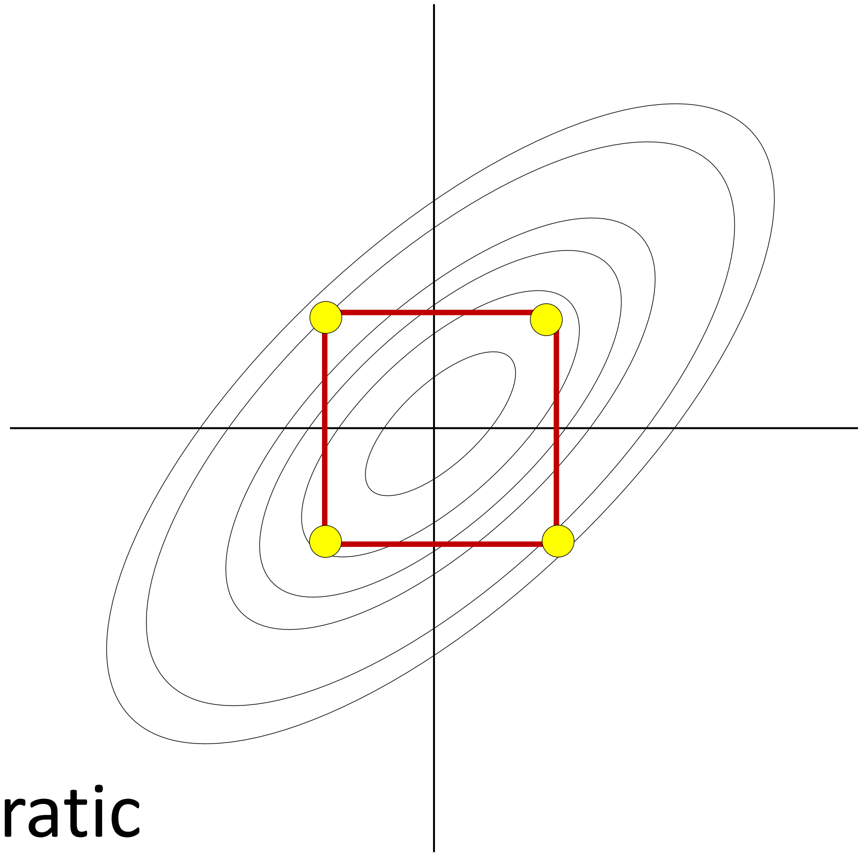
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y}$$



- E is a convex quadratic
  - Shown from above (assuming 0 bias)

The energy function

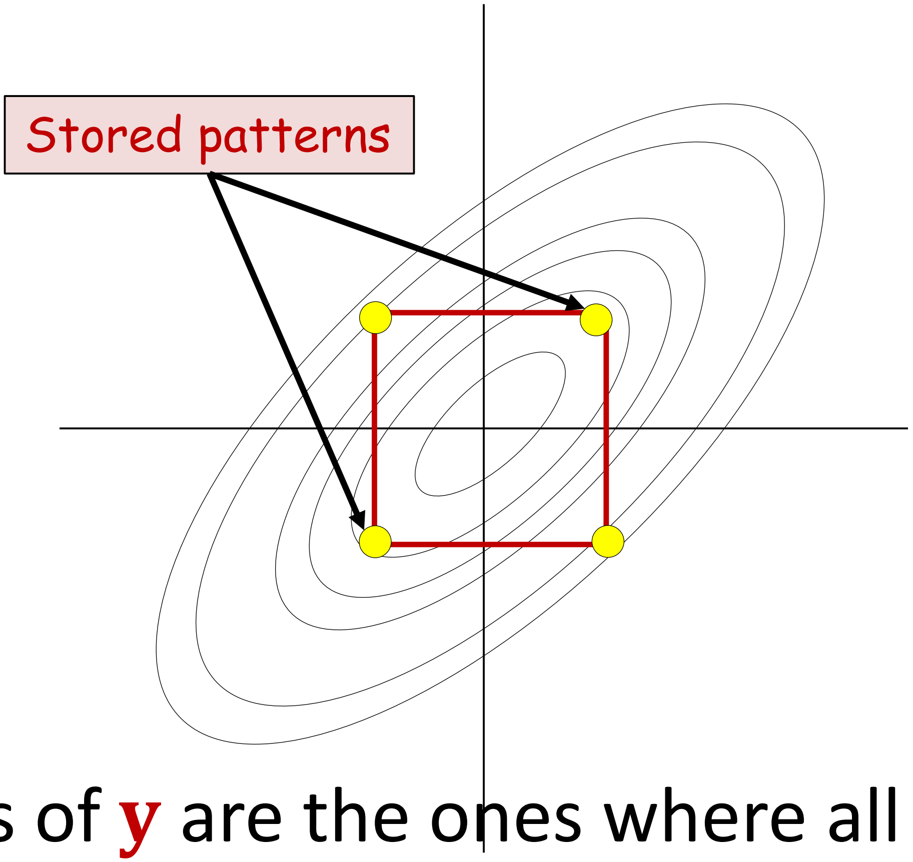
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y}$$



- E is a convex quadratic
  - Shown from above (assuming 0 bias)
- But components of y can only take values ± 1
  - I.e y lies on the corners of the unit hypercube

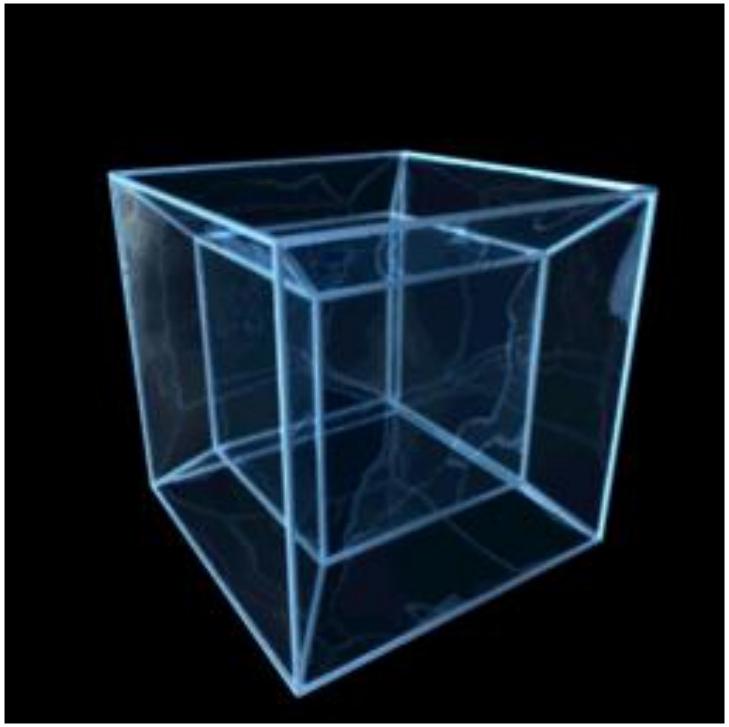
The energy function

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



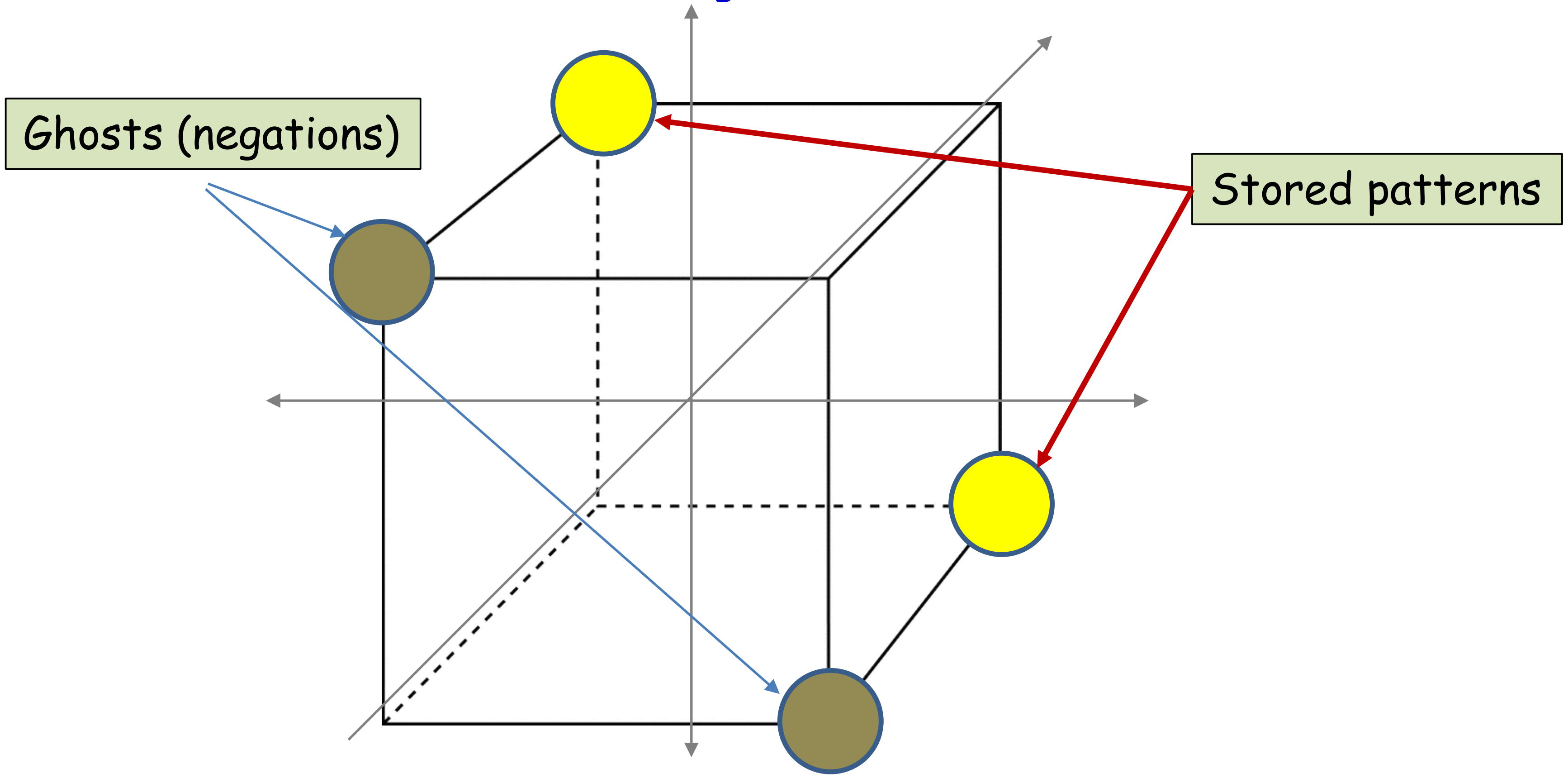
- The stored values of $\mathbf{y}$ are the ones where all adjacent corners are higher on the quadratic
  - Hebbian learning attempts to make the quadratic steep in the vicinity of stored patterns

Patterns you can store



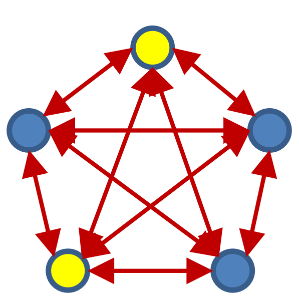
- 4-bit patterns
- Stored patterns would be the corners where the value of the quadratic energy is lowest

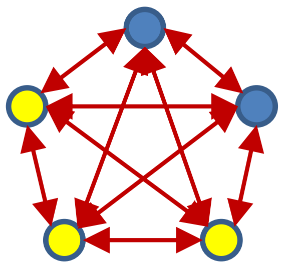
Patterns you can store

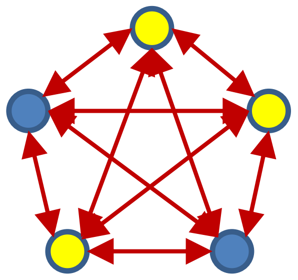


- Ideally must be maximally separated on the hypercube
  - The number of patterns we can store depends on the actual distance between the patterns

How many patterns can we store?

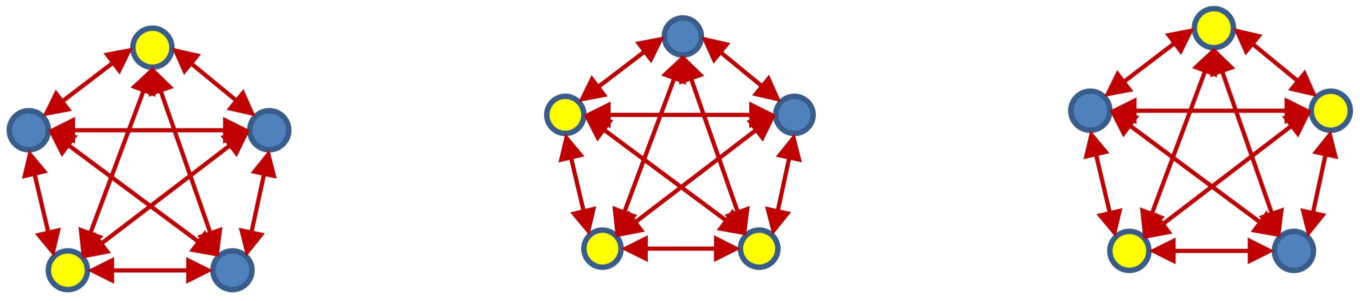






- Hopfield: For a network of N neurons can store up to $0.15N$ patterns
  - *Provided* the patterns are random and “far apart”

How many patterns can we store?



- Problem with Hebbian learning: Focuses on patterns that must be stored
  - What about patterns that must *not* be stored?

- More recent work: can actually store up to N patterns
    - Non Hebbian learning
    - $\boldsymbol{W}$ loses positive semi-definiteness

Storing N patterns

- Non Hebbian
- Requirement: Given $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_P$
  - Design $\mathbf{W}$ such that
    - $sign(\mathbf{W}\mathbf{y}_p) = \mathbf{y}_p$ for all target patterns
    - There are no other *binary* vectors for which this holds

- I.e. $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_P$ are the only *binary* Eigenvectors of $\mathbf{W}$, and the corresponding eigen values are positive

Storing N patterns

- Simple solution: Design $\mathbf{W}$ such that $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_P$ are the Eigen vectors of $\mathbf{W}$
- Easily achieved if $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_P$ are orthogonal to one another
  - Let $Y = [\mathbf{y}_1\ \mathbf{y}_2 \ldots \mathbf{y}_P\ \ \mathbf{r}_{P+1} \ldots \mathbf{r}_N]$
  - N is the number of bits
  - $\mathbf{r}_{P+1} \ldots \mathbf{r}_N$ are "synthetic" non-binary vectors
  - $\mathbf{y}_1\ \mathbf{y}_2 \ldots \mathbf{y}_P\ \ \mathbf{r}_{P+1} \ldots \mathbf{r}_N$ are all orthogonal to one another

$$W = Y\Lambda Y^T$$

- Eigen values λ in diagonal matrix Λ determine the steepness of the energy function around the stored values
  - What must the Eigen values corresponding to the $\mathbf{y}_P$s be?
  - What must the Eigen values corresponding to the "$\mathbf{r}$"s be?
- Under no condition can more than N values be stored

Storing N patterns

- For non-orthogonal $\mathbf{W}$, solution is less simple
- $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_P$ can no longer be Eigen values, but now represent *quantized* Eigen directions

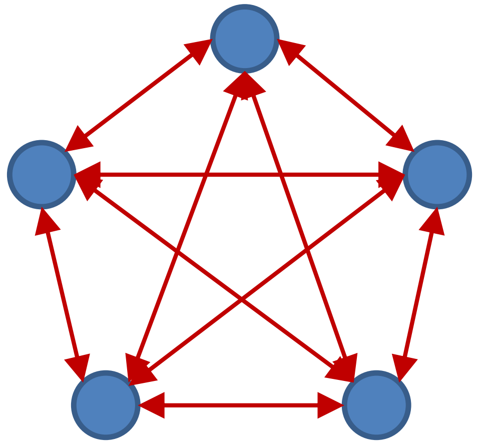
$$sign\left(\mathbf{W}\mathbf{y}_p\right) = \mathbf{y}_p$$

  – Note that this is not an exact Eigen value equation

- Optimization algorithms can provide $\mathbf{W}$s for many patterns
- Under no condition can we store more than N patterns

Alternate Approach to Estimating the Network

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



- Estimate $\mathbf{W}$ (and $\mathbf{b}$) such that
  - E is minimized for $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_P$
  - E is maximized for all other $\mathbf{y}$
- We will encounter this solution again soon
- Once again, cannot store more than N patterns

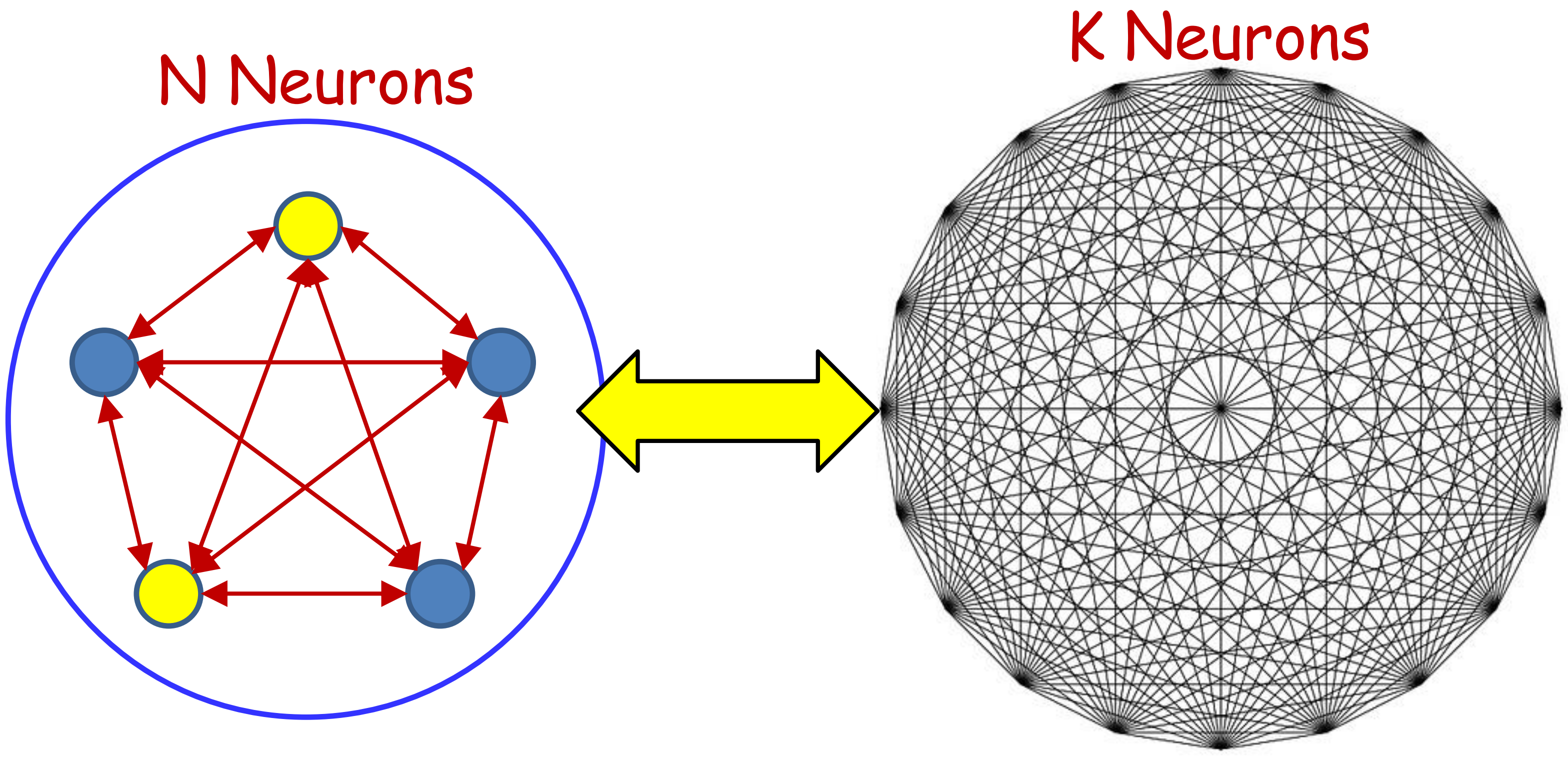
Storing more than N patterns

- How do we even solve the problem of storing N patterns in the first place
- How do we increase the capacity of the network
  - Store more patterns
- Common answer to both problems..

Lookahead..

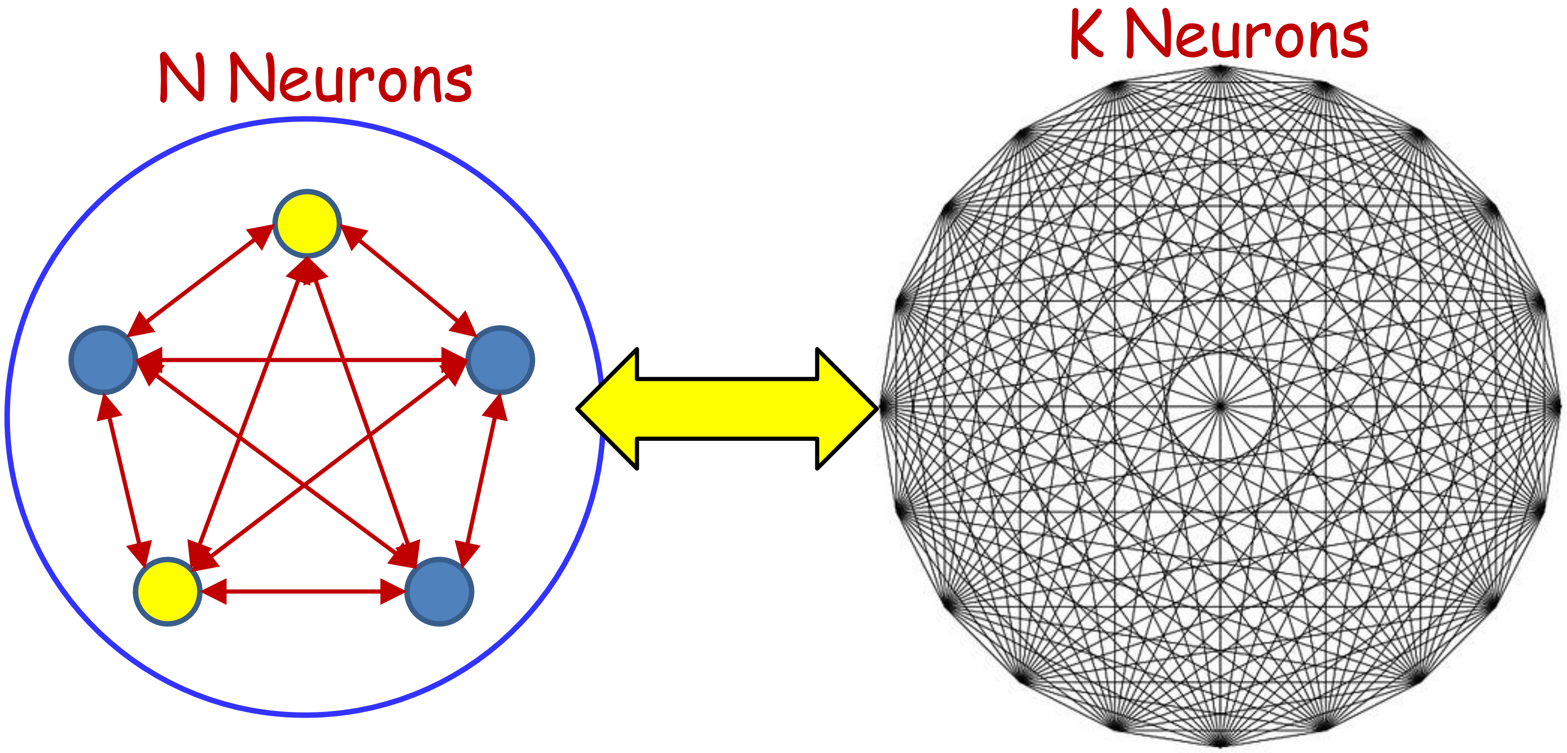
- Adding capacity to a Hopfield network

Expanding the network



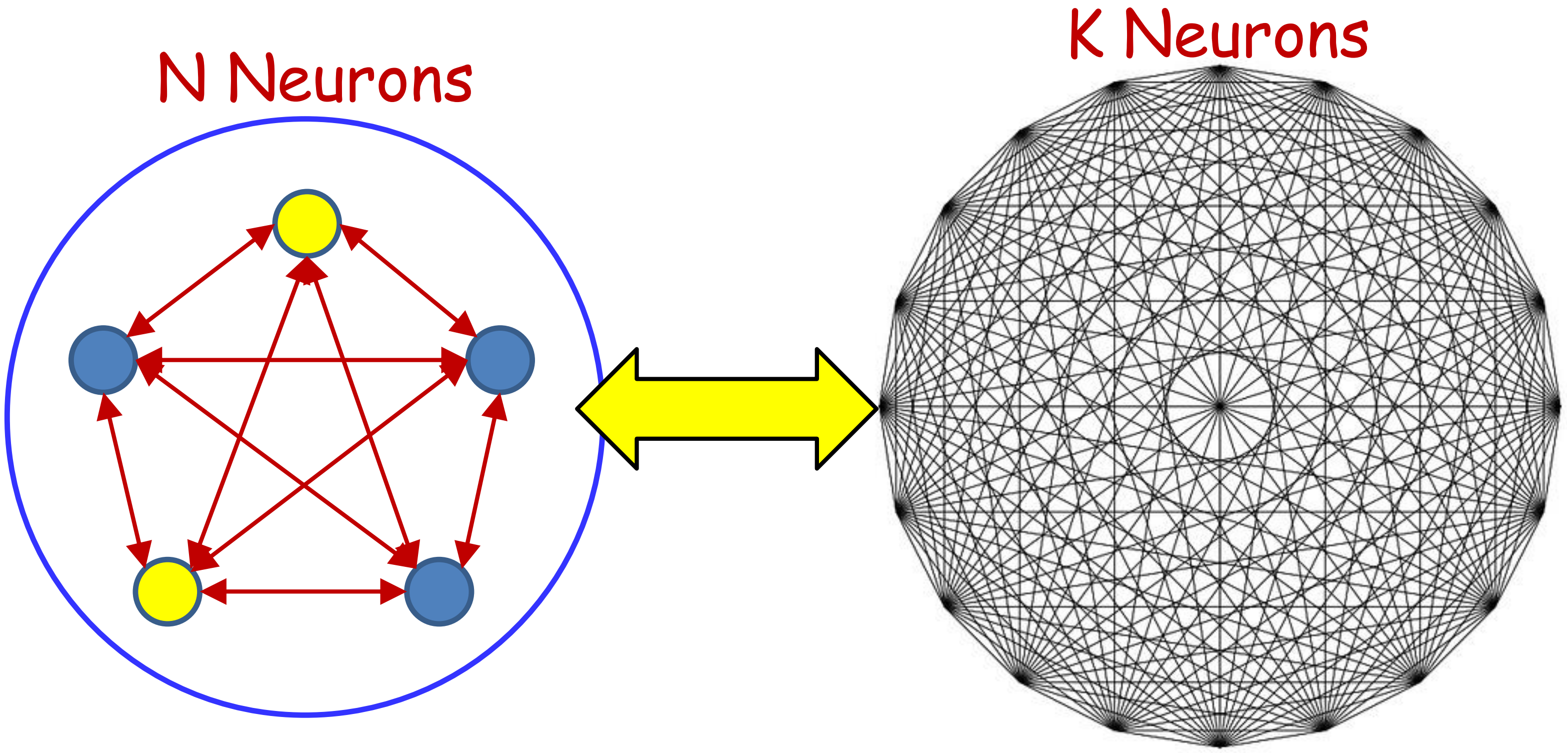
- Add a large number of neurons whose actual values you don't care about!

Expanded Network



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

Introducing…



- The Boltzmann machine…
- Next regular class…