Neural Networks

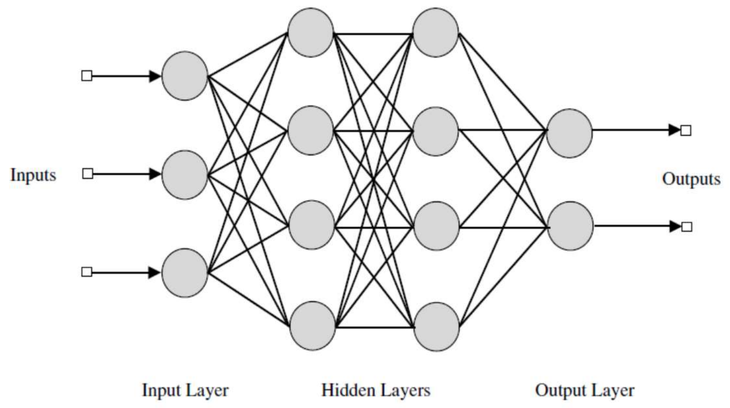
Hopfield Nets and Auto Associators

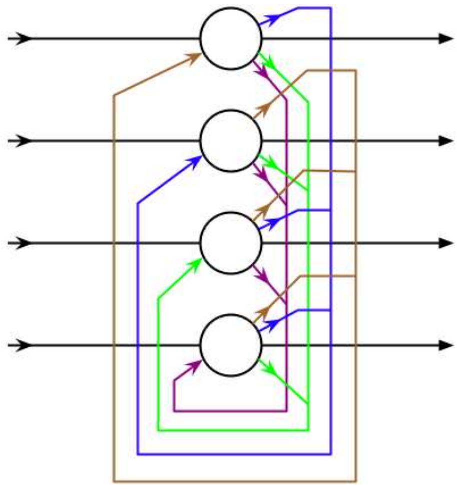
Spring 2021

Story so far

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

- But what about..

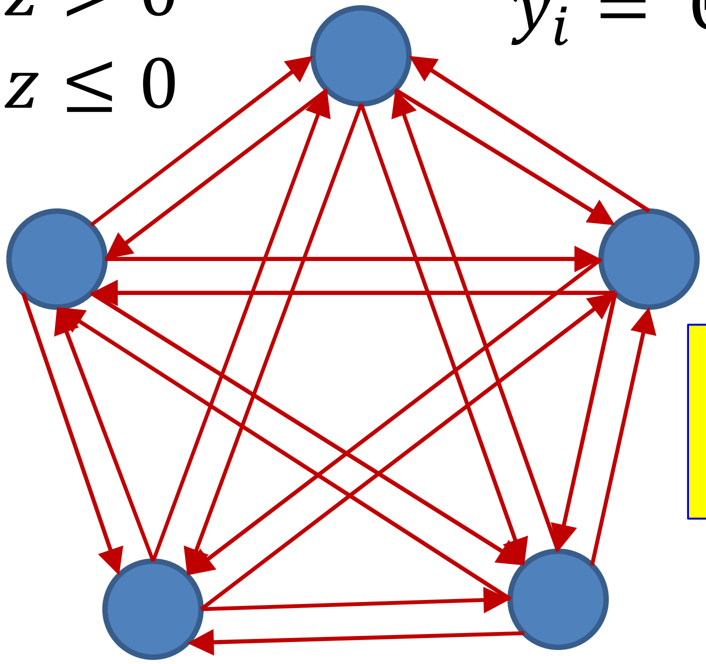




Consider this loopy network

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

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



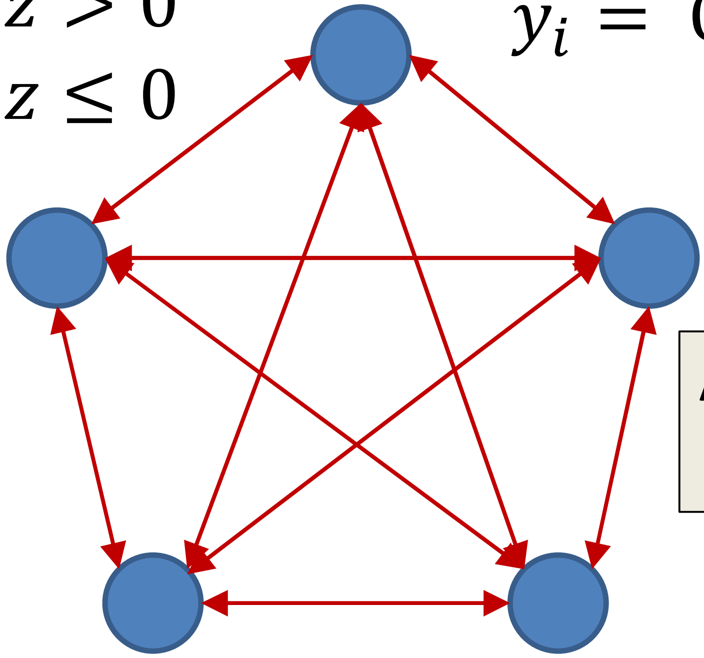
The output of a neuron affects the input to the neuron

- 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

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

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



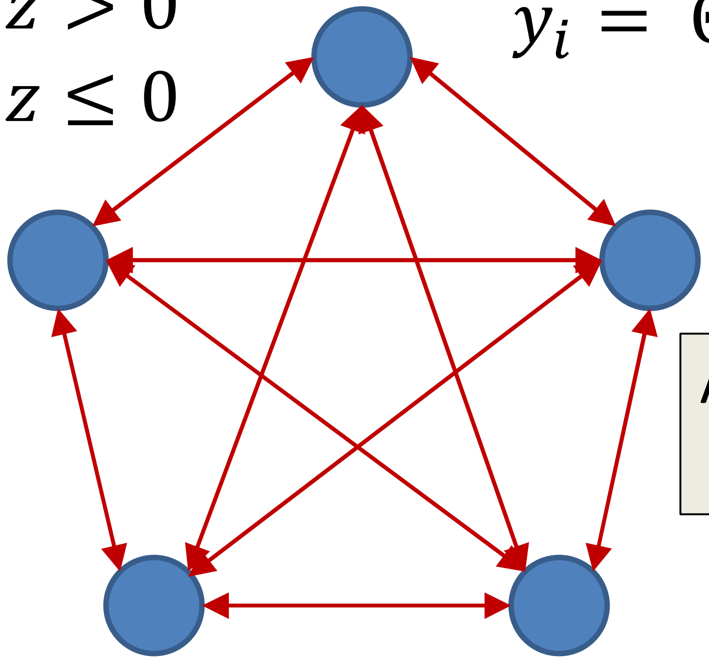
A symmetric network:
$w_{ij} = w_{ji}$

- 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

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

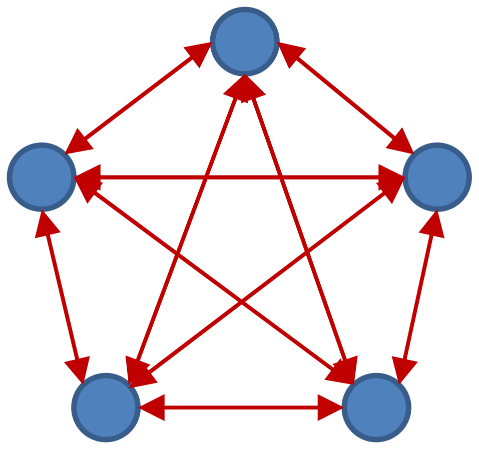
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$



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

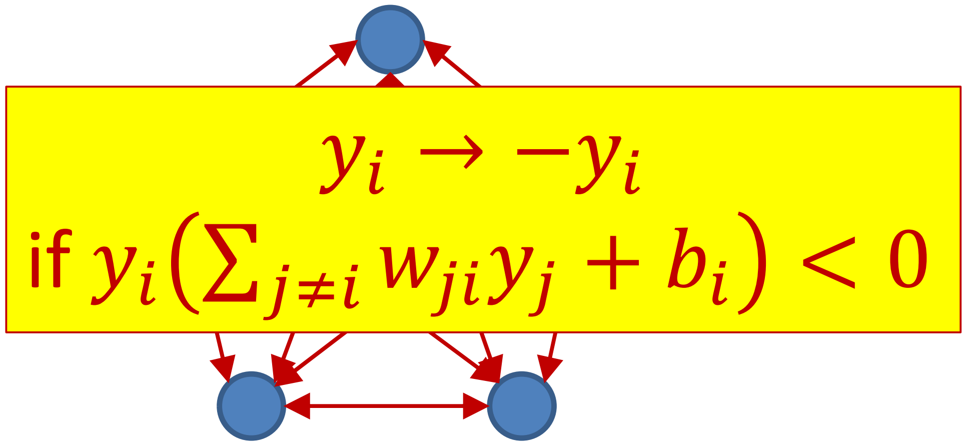


$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

- 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



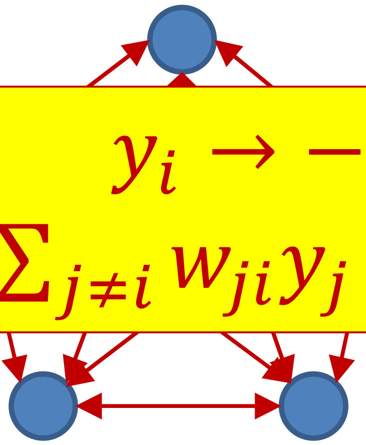
$$y_i \rightarrow -y_i$$
if $y_i\left(\sum_{j\neq i} w_{ji}y_j + b_i\right) < 0$

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

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

- 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



$$y_i \to -y_i$$

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

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

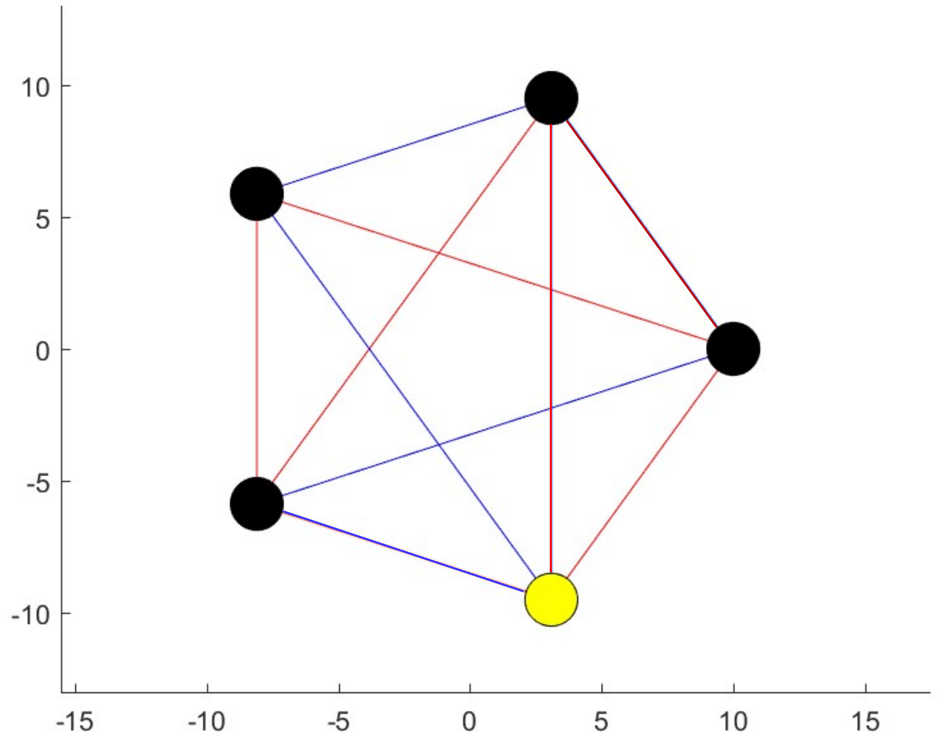
$$\Theta(z) = \begin{cases} +1 \; if \; z > 0 \\ -1 \; if \; z \leq 0 \end{cases}$$

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!

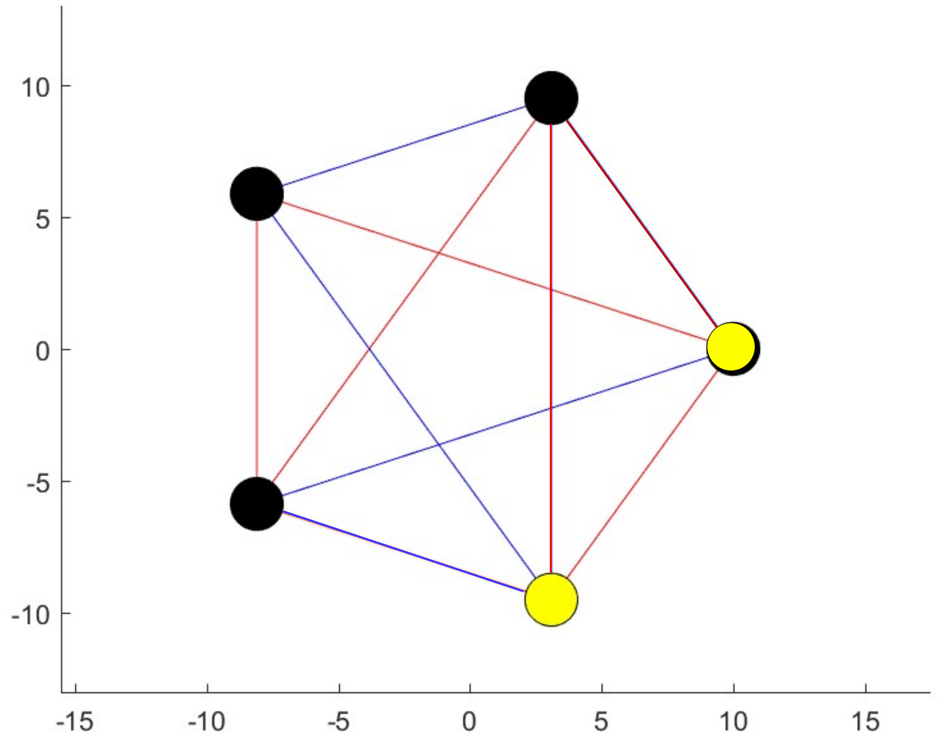
- es a "field" $\sum_{j\neq i} w_{ji}y_j + b_i$
- s own sign, it does not respond
- If the sign of the field opposes its own sign, it "flips" to match the sign of the field

Example



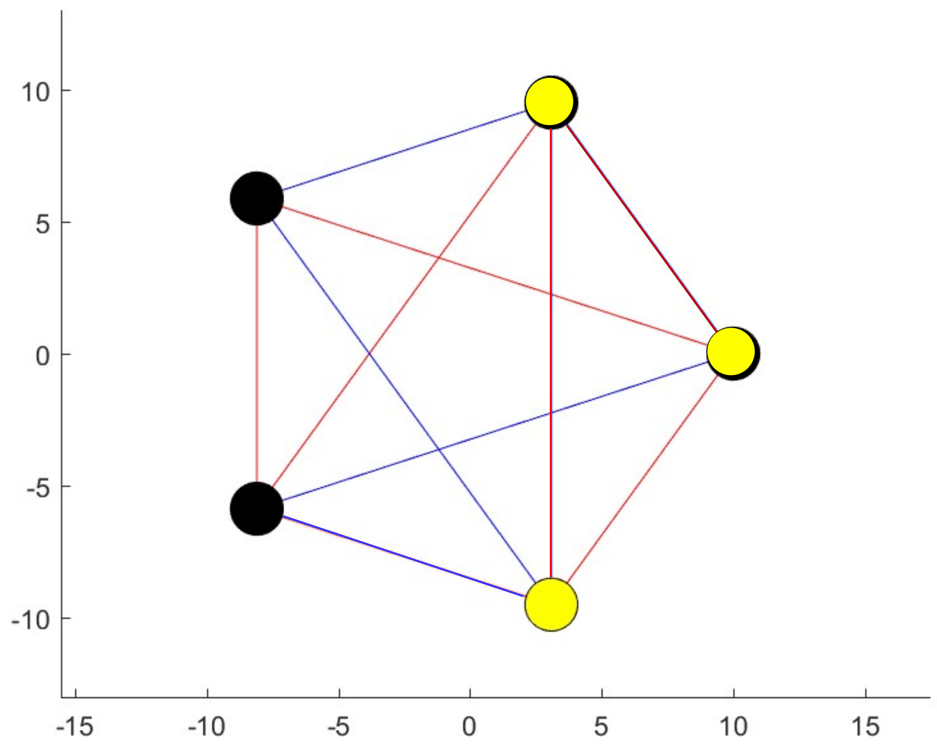
- Red edges are +1, blue edges are -1
- Yellow nodes are -1, black nodes are +1

Example



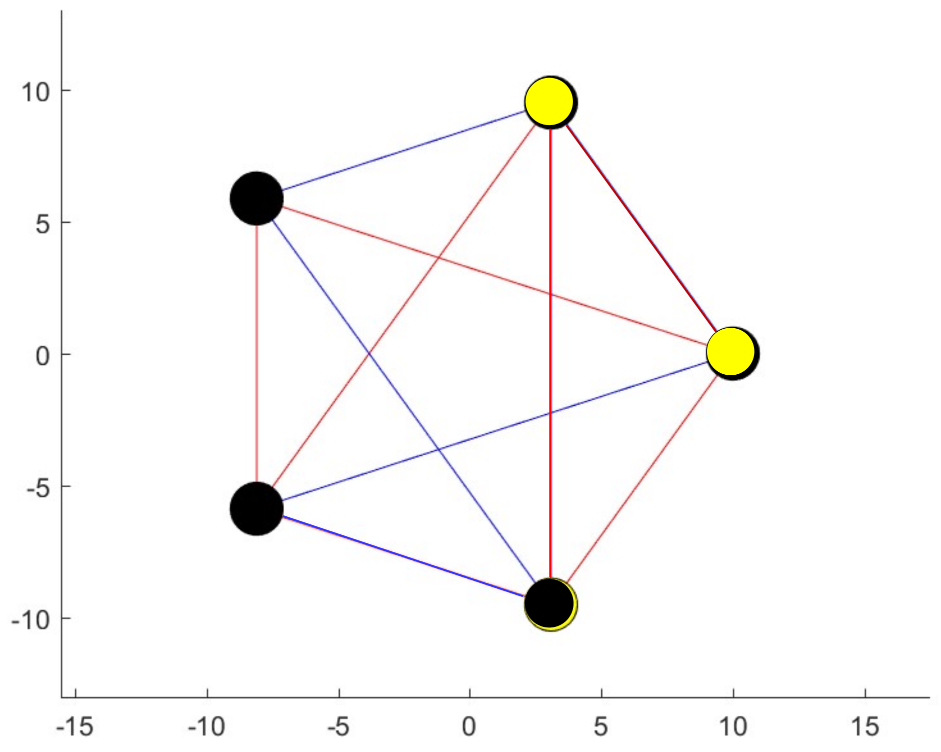
- Red edges are +1, blue edges are -1
- Yellow nodes are -1, black nodes are +1

Example



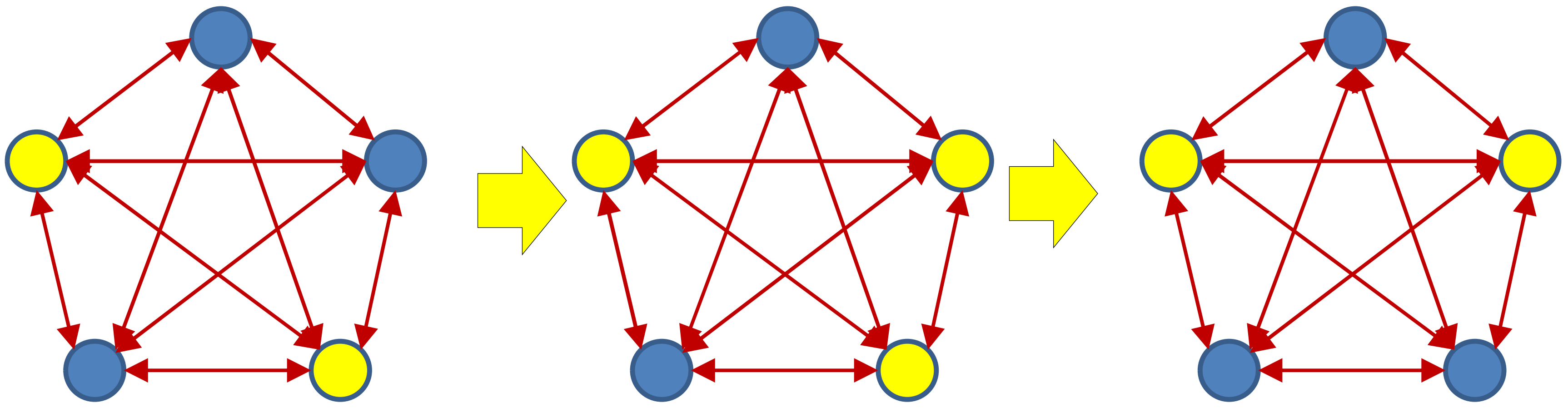
- Red edges are +1, blue edges are -1
- Yellow nodes are -1, black nodes are +1

Example



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

Loopy network

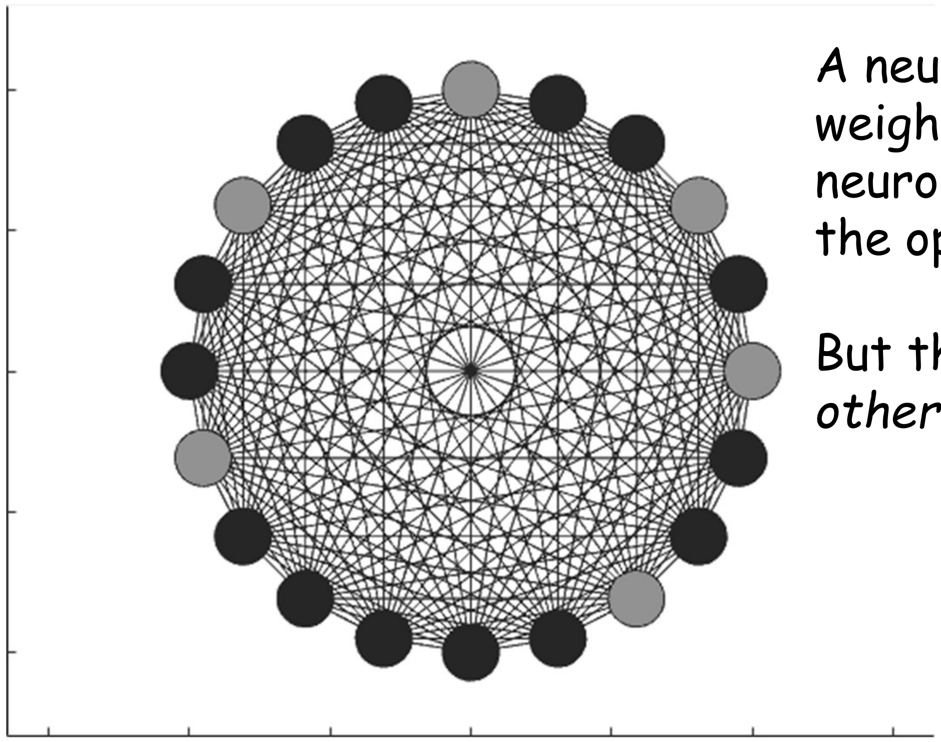


- If the sign of the field at any neuron opposes its own sign, it “flips” to match the field
  - Which will change the field at other nodes
    - Which may then flip
      - Which may cause other neurons including the first one to flip…
        - » And so on…

20 evolutions of a loopy net

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

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

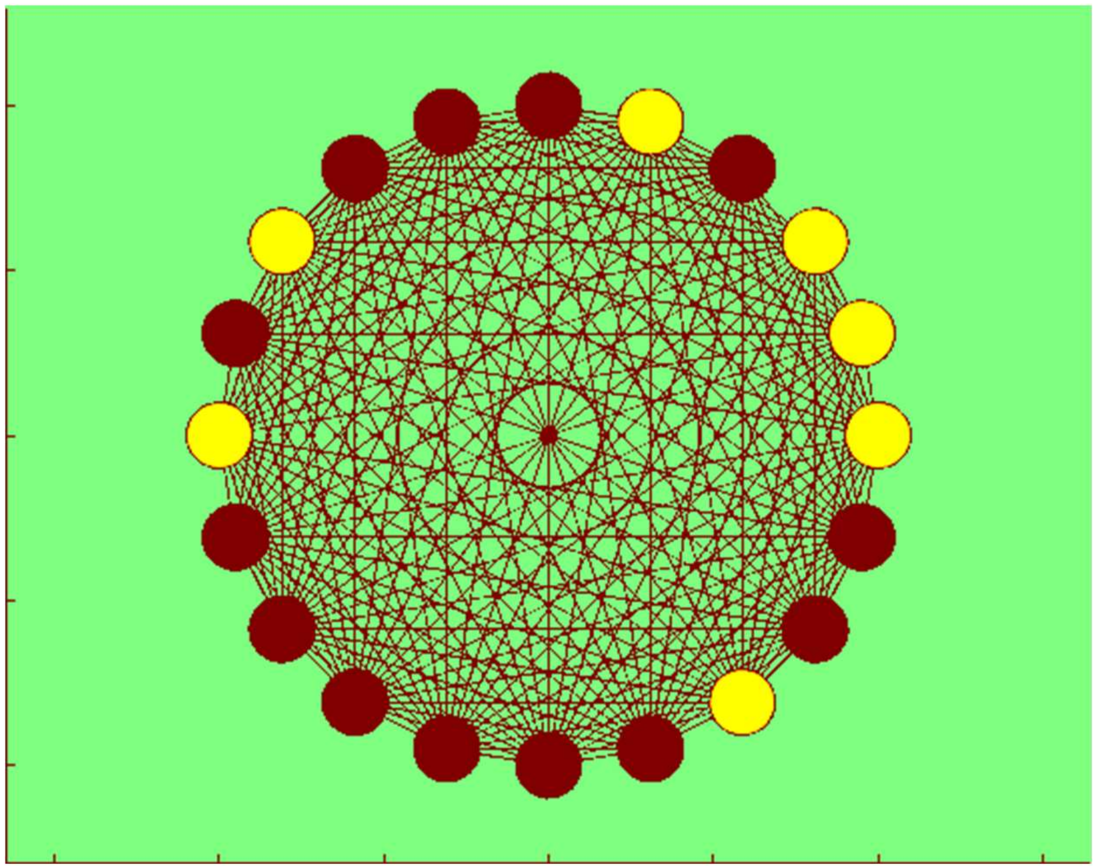


A neuron "flips" if weighted sum of other neuron's outputs is of the opposite sign

But this may cause *other* neurons to flip!

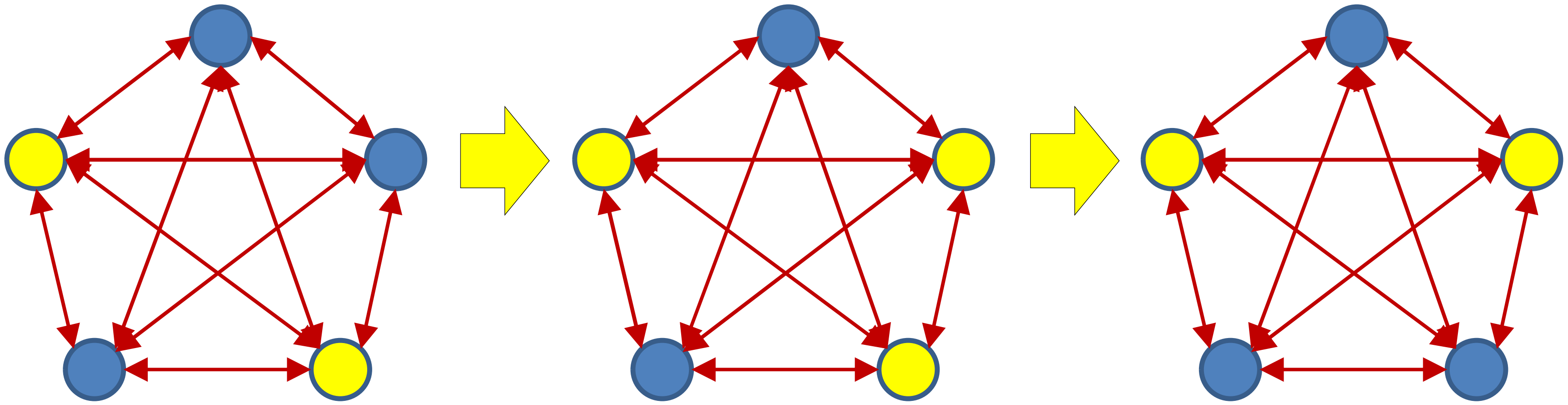
- All neurons which do not "align" with the local field "flip"

120 evolutions of a loopy net



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

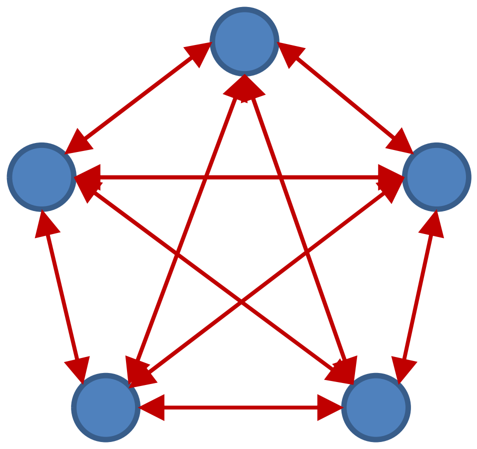
Loopy network



- 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



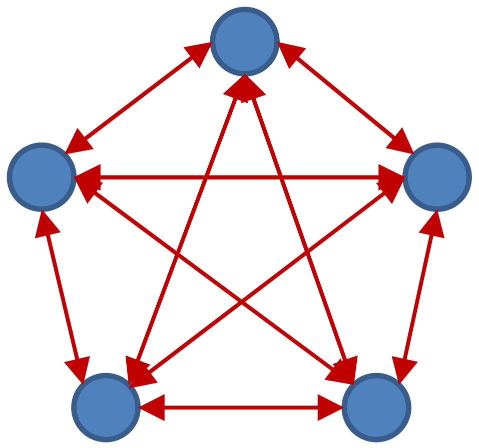
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

- 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

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

Upon flipping a single unit

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

- Expanding

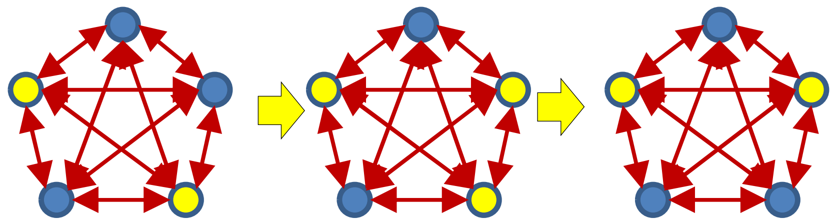
$$\Delta D(y_k) = (y_k^+ - y_k^-)\left(\sum_{j \neq k} w_{jk} y_j + b_k\right)$$

  – 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

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

- 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

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

  – 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

The Energy of a Hopfield Net

- Define the *Energy* of the network as

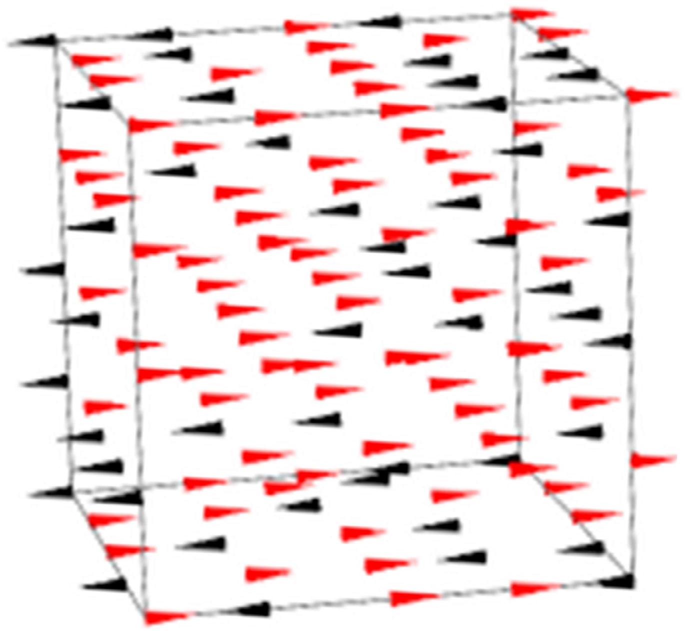
$$E = -\frac{1}{2}\left(\sum_{i,j\neq i} w_{ij} y_i y_j - \sum_i b_i y_i\right)$$

  - Just 0.5 times the negative of D

- The evolution of a Hopfield network constantly decreases its energy

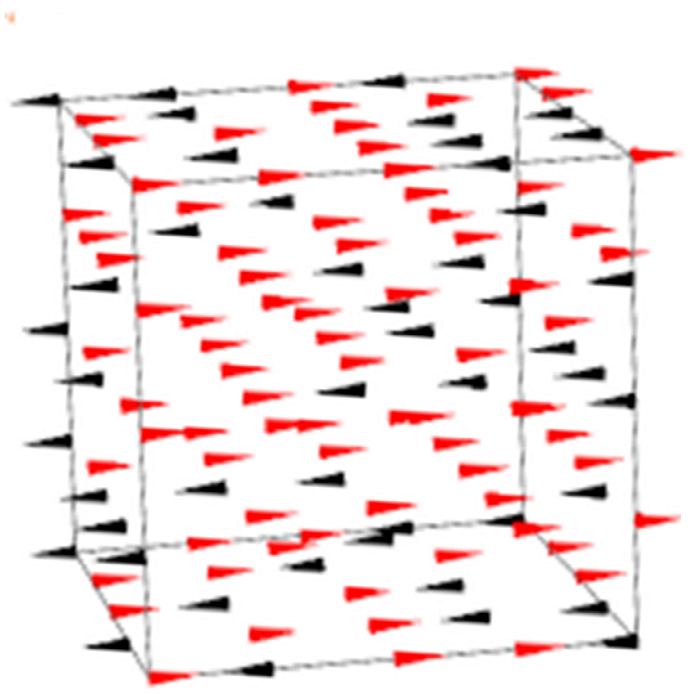
- Where did this "energy" concept suddenly sprout from?

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:

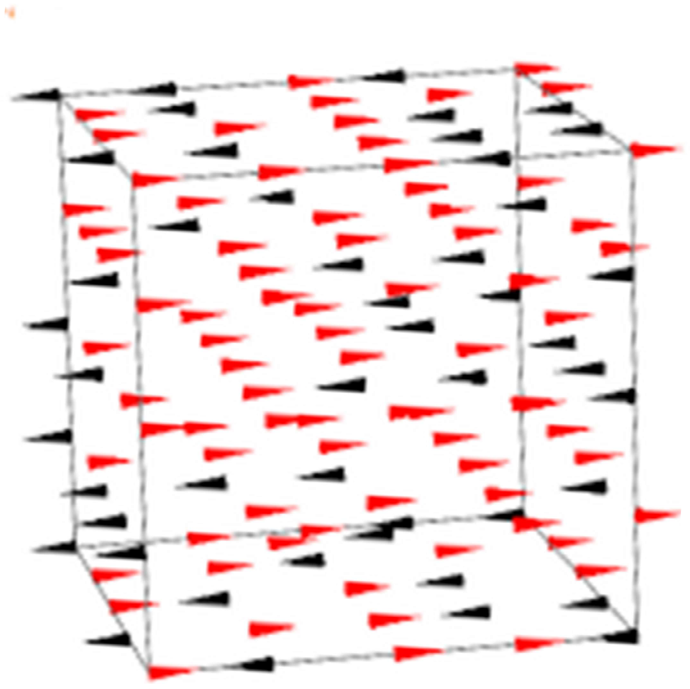
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

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:

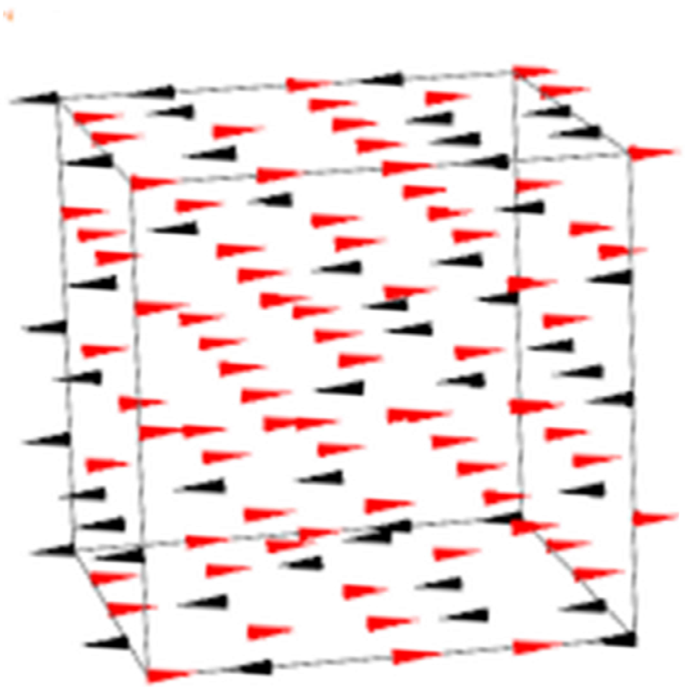
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

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:

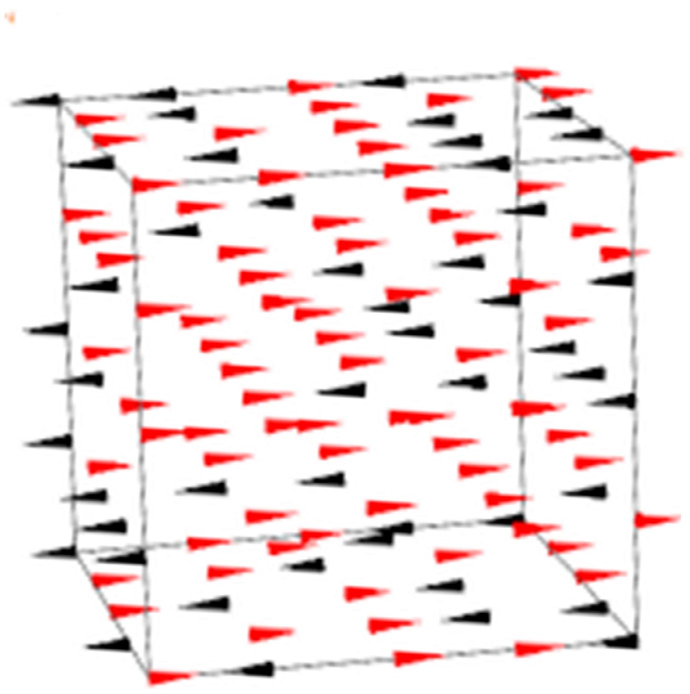
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

Response of current dipole

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

- 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???

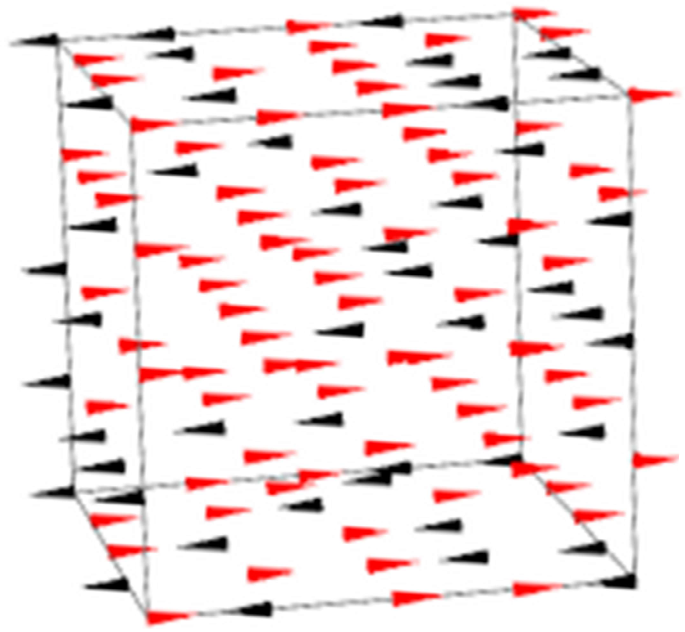
Total field at current dipole:

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

Response of current dipole

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

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

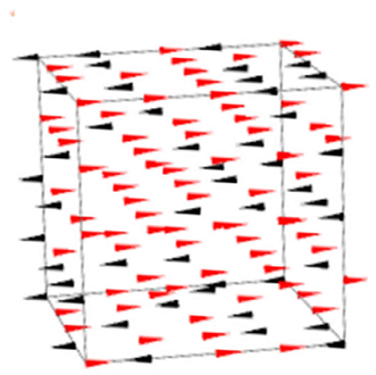
$$x_i = \begin{cases} x_i \ if \ sign\big(x_i \ f(p_i)\big) = 1 \\ -x_i \ otherwise \end{cases}$$

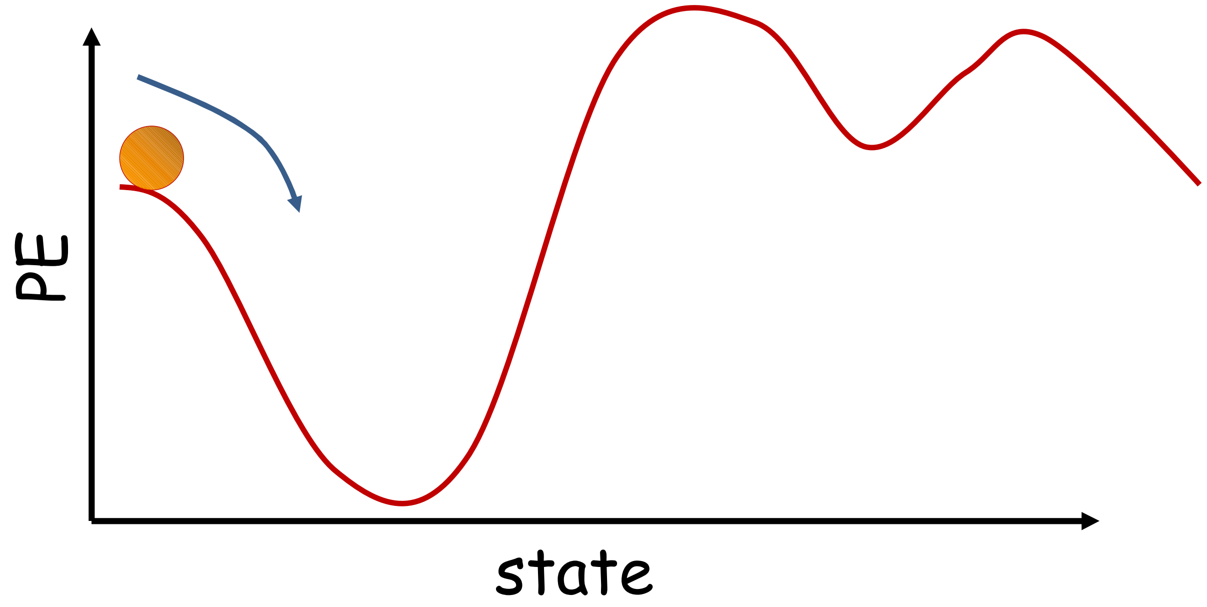
- The "Hamiltonian" (total energy) of the system

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

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

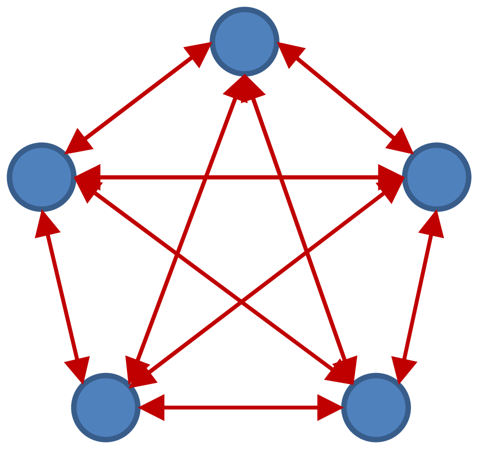
Spin Glasses





- 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



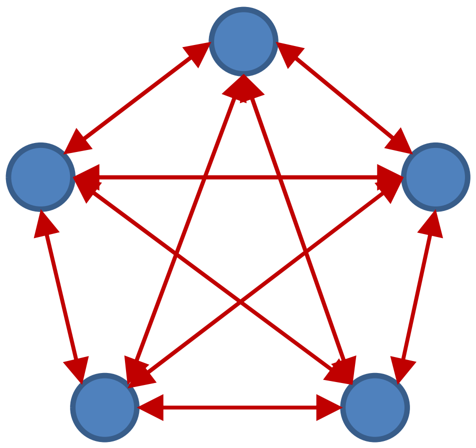
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

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

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

Hopfield Network



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

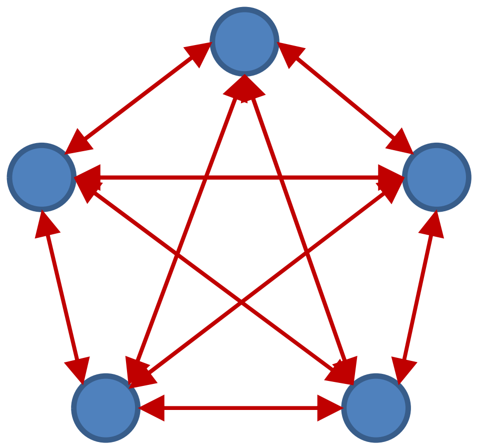
$$\Theta(z) = \begin{cases} +1 \; if \; z > 0 \\ -1 \; if \; z \leq 0 \end{cases}$$

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

We will bring it back later in the discussion when needed

Hopfield Network



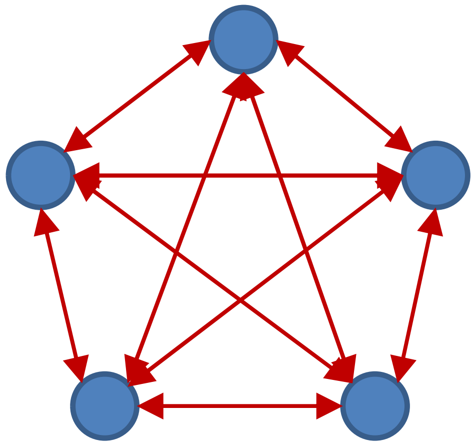
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j\right)$$

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

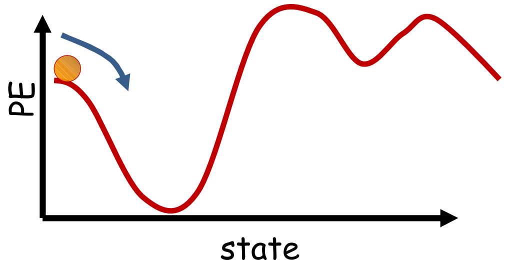
$$E = -\frac{1}{2} \sum_{i,j<i} w_{ij} y_i y_j$$

- 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

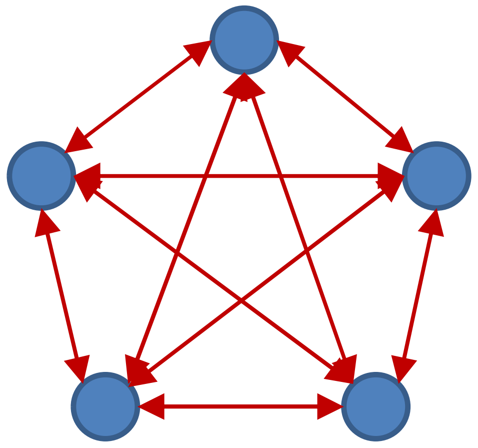


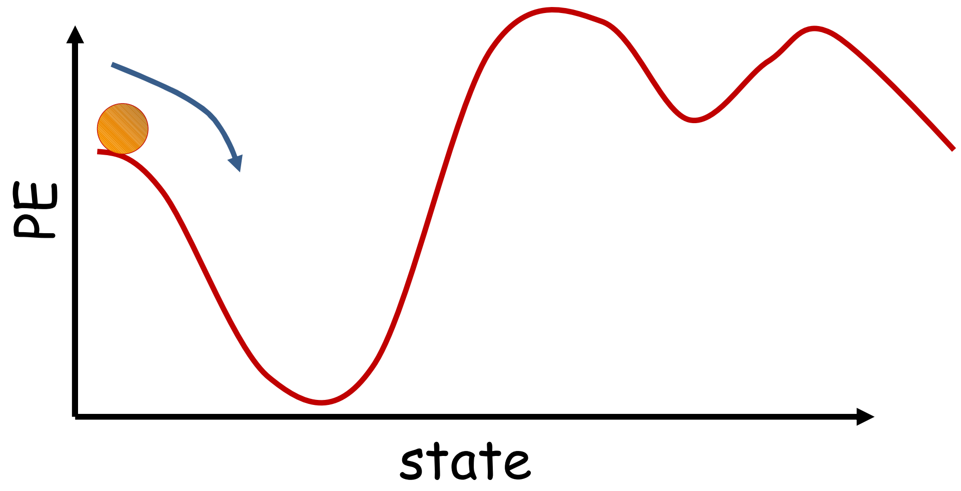
$$E = -\frac{1}{2}\sum_{i,j<i} w_{ij} y_i y_j$$



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

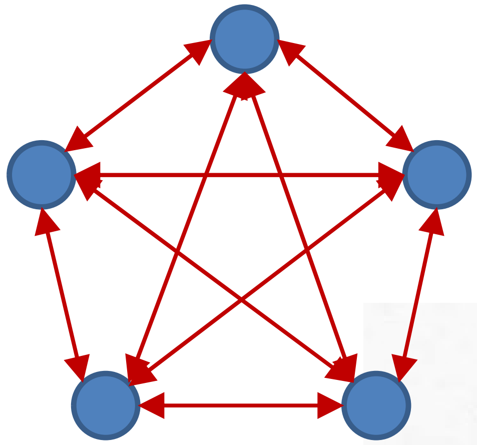
Content-addressable memory





- 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



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

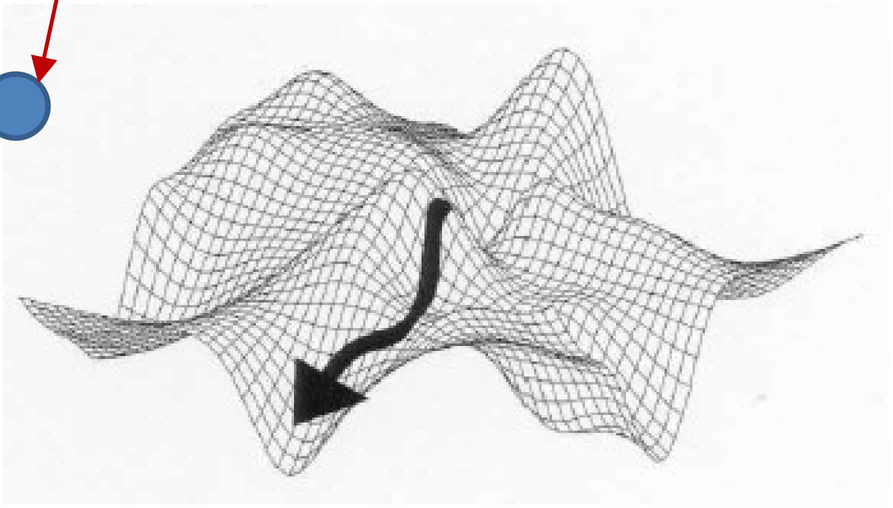
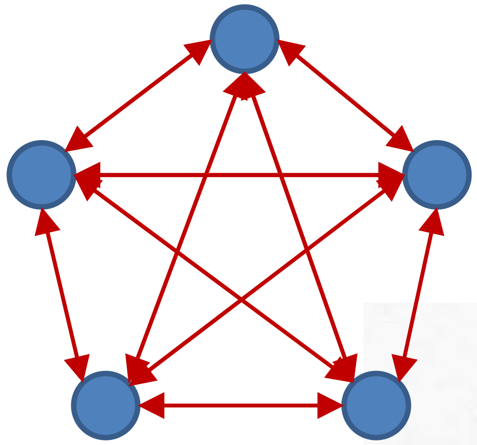


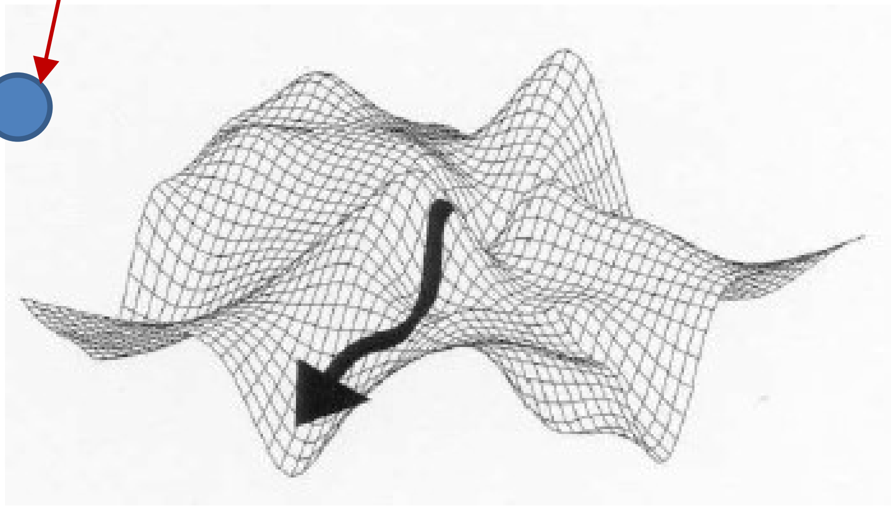
Image pilfered from unknown source

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

Evolution

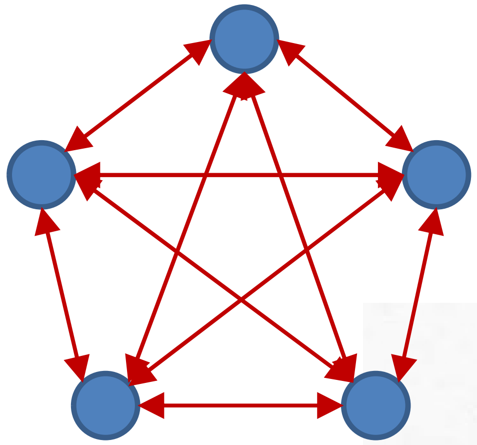


$$E = -\frac{1}{2}\sum_{i,j<i} w_{ij}y_iy_j$$



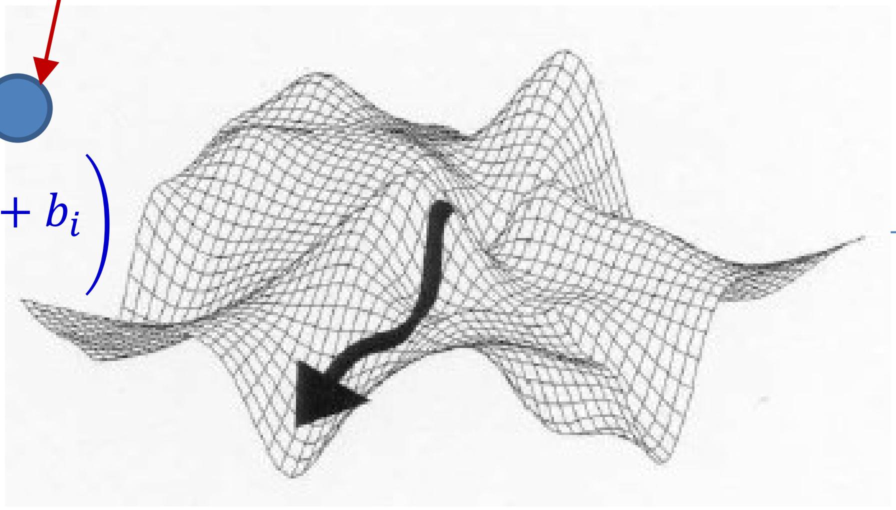
- The network will evolve until it arrives at a local minimum in the energy contour
- We proved that *every* change in the network will result in *decrease* in energy
  - So path to energy minimum is monotonic

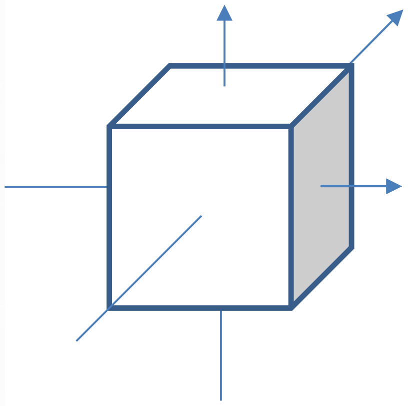
Evolution



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

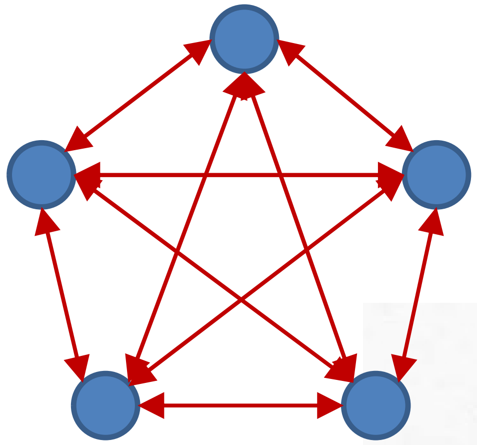
$$E = -\frac{1}{2} \sum_{i,j<i} w_{ij} y_i y_j$$





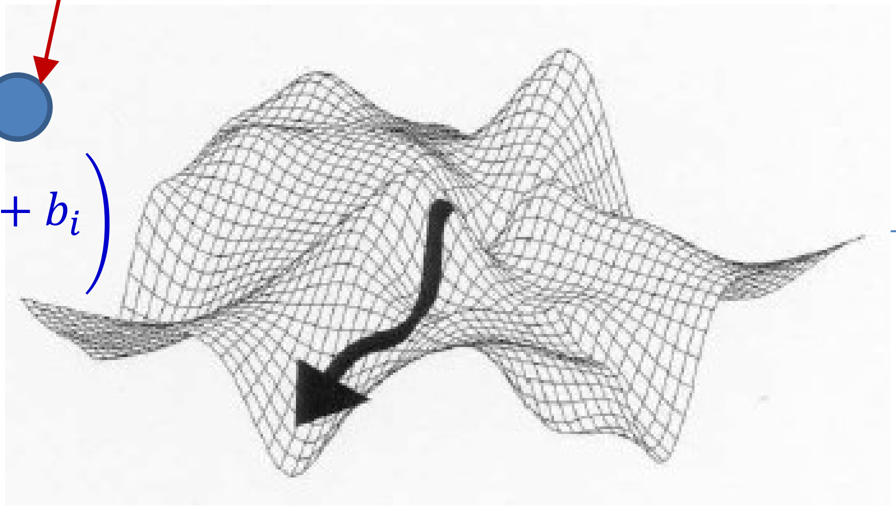
- For threshold activations the energy contour is only defined on a lattice
  - Corners of a unit cube on [-1,1]^N

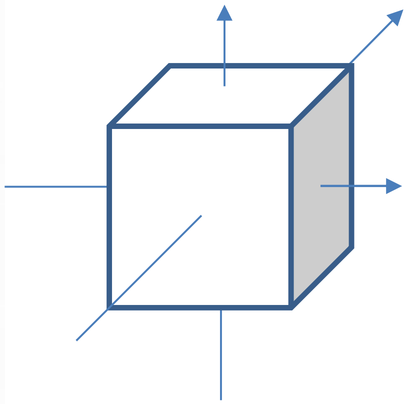
Evolution



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

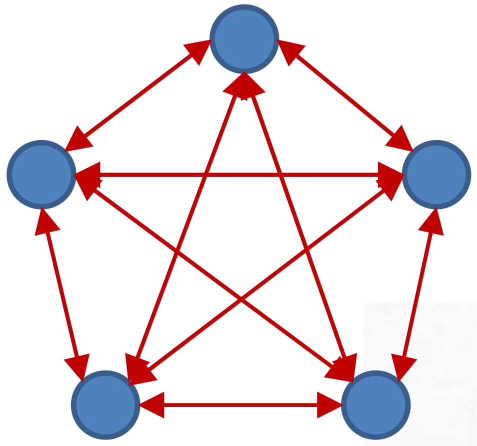
$$E = -\frac{1}{2}\sum_{i,j<i} w_{ij} y_i y_j$$



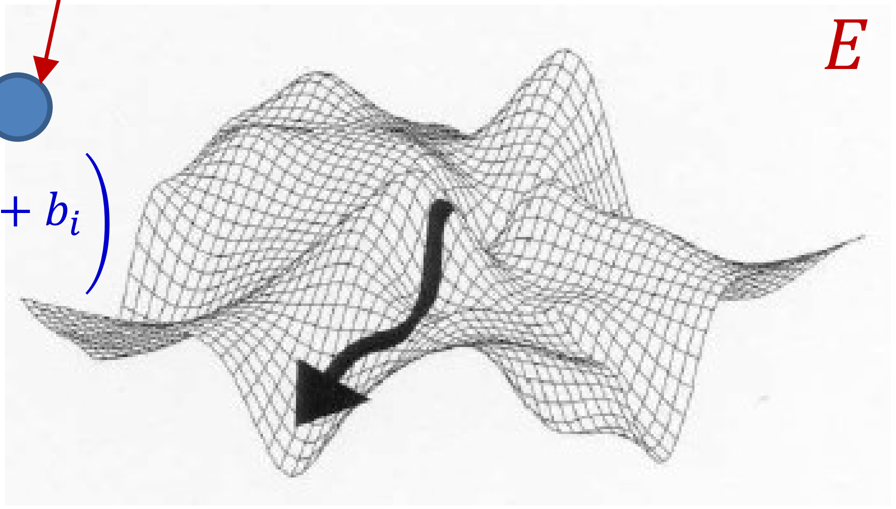


- For threshold activations the energy contour is only defined on a lattice
  - Corners of a unit cube on [-1,1]^N
- For tanh activations it will be a continuous function

Evolution



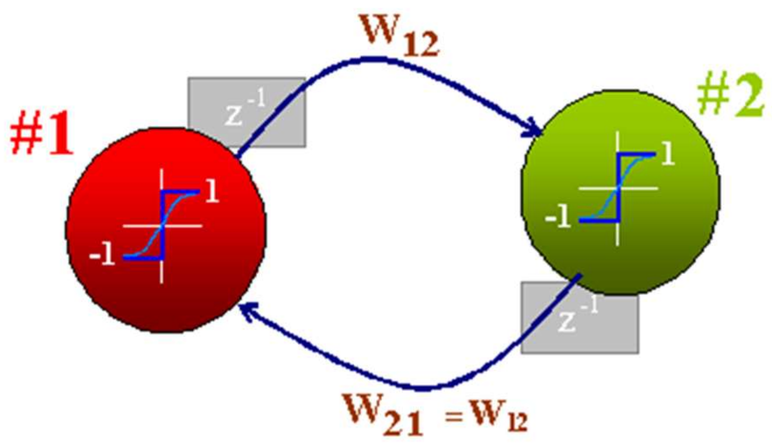
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

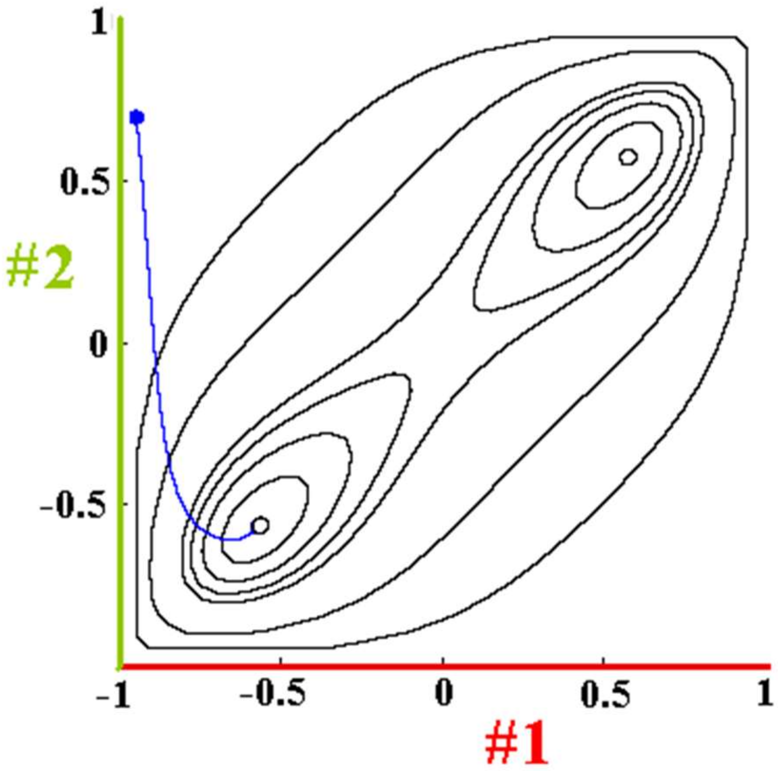


$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$

- 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
  - With output in [-1 1]

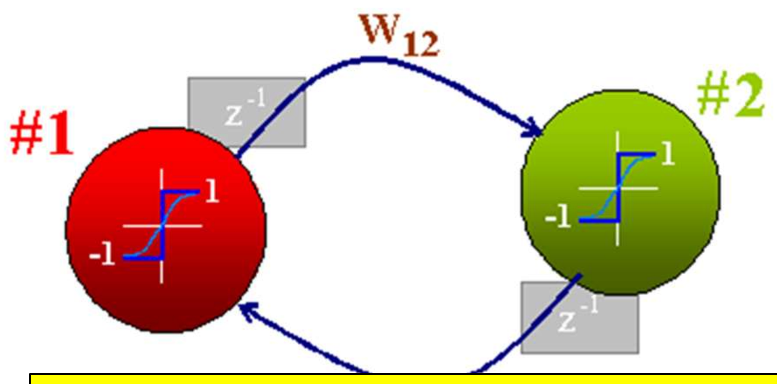
“Energy”contour for a 2-neuron net

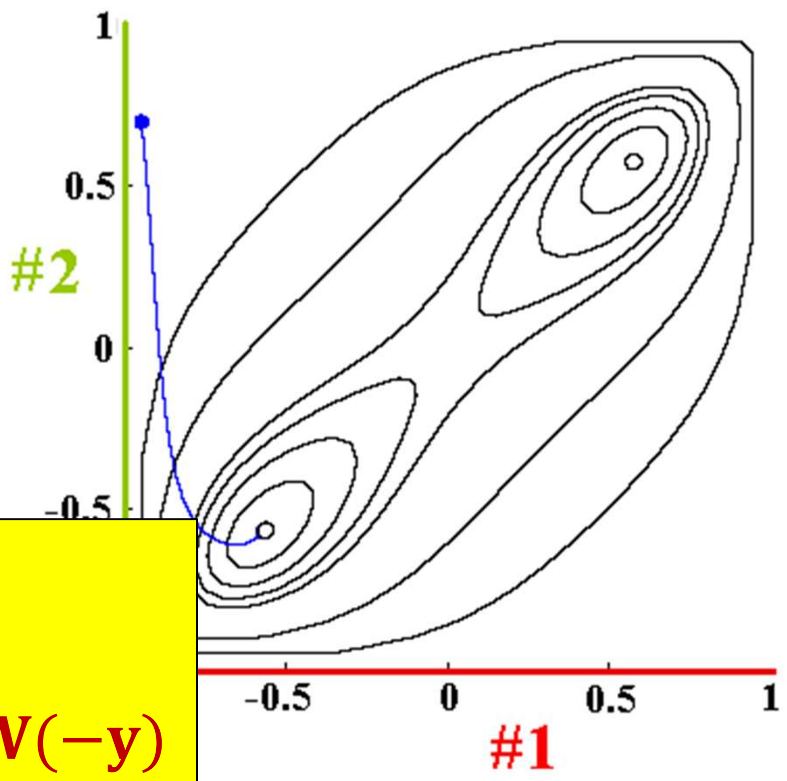




- Two stable states (tanh activation)
  - Symmetric, not at corners
  - Blue arc shows a typical trajectory for tanh activation

"Energy"contour for a 2-neuron net





Why symmetric?

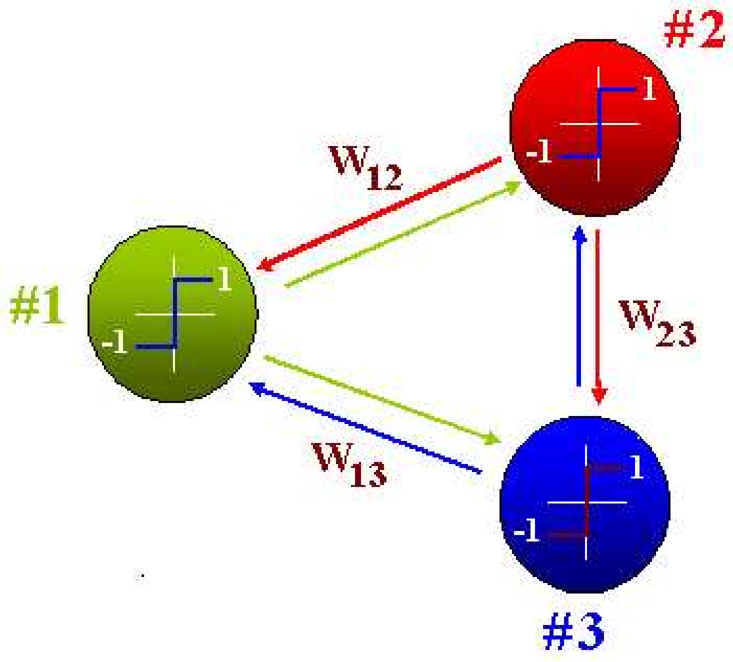
Because $-\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} = -\frac{1}{2}(-\mathbf{y})^T\mathbf{W}(-\mathbf{y})$

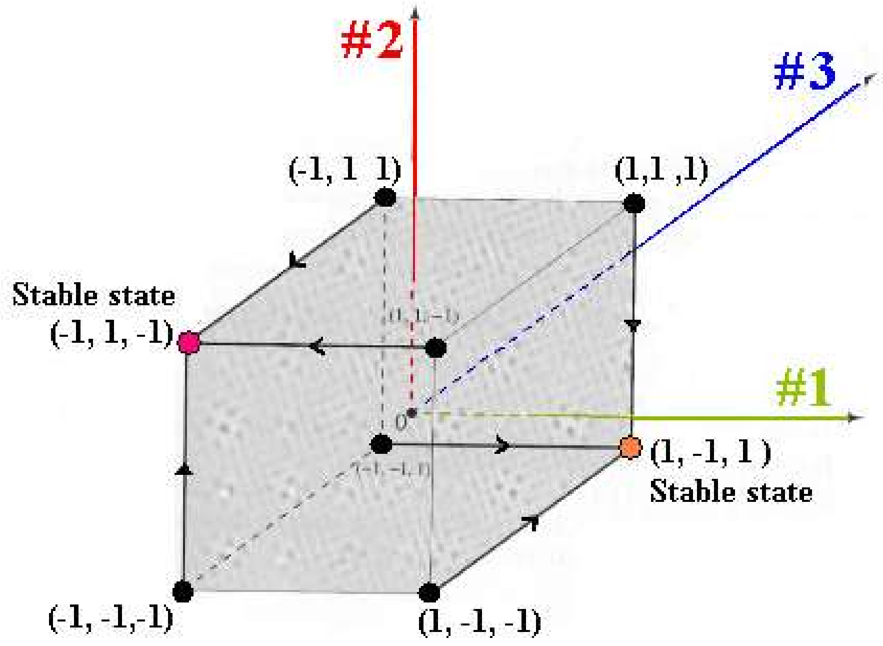
If $\hat{\mathbf{y}}$ is a local minimum, so is $-\hat{\mathbf{y}}$

•

– Blue arc shows a typical trajectory for sigmoid activation

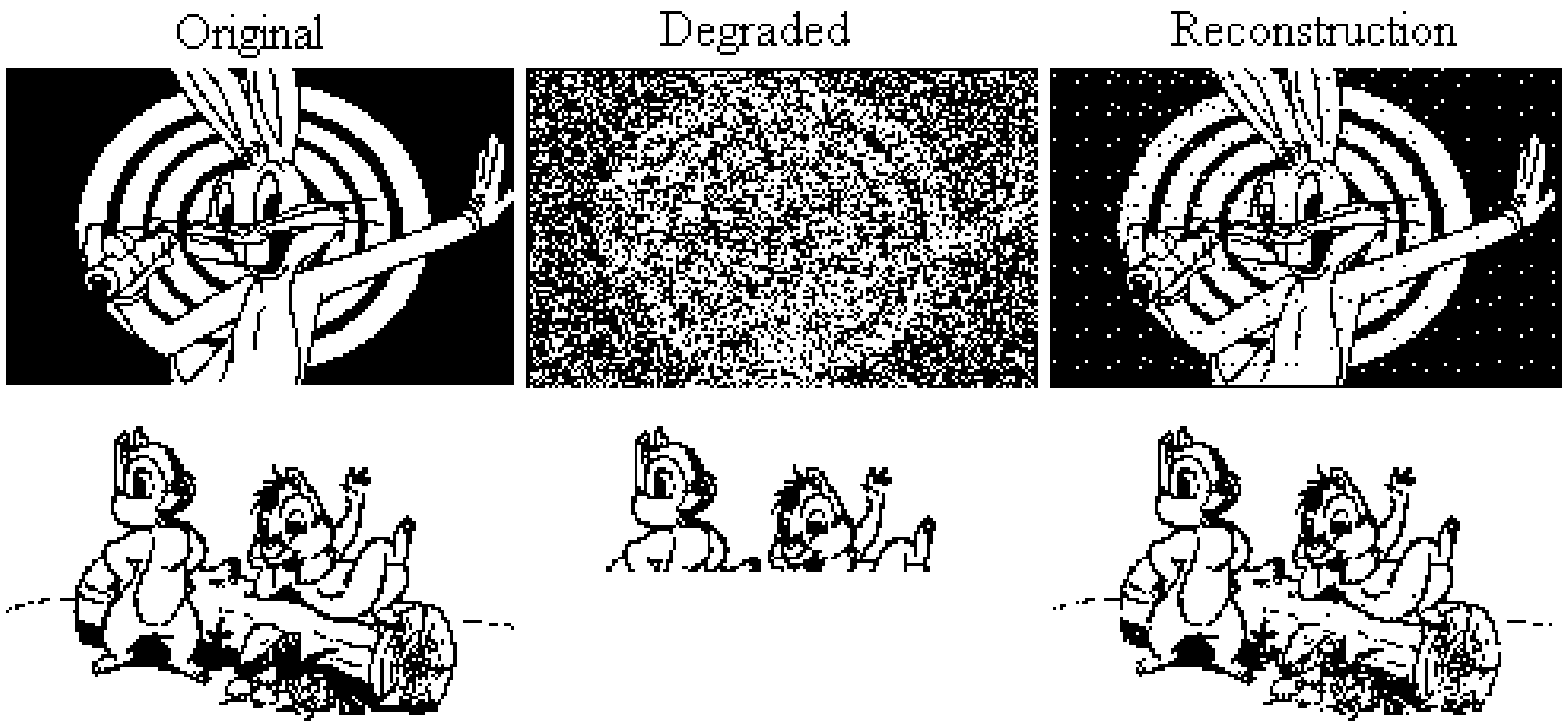
3-neuron net





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

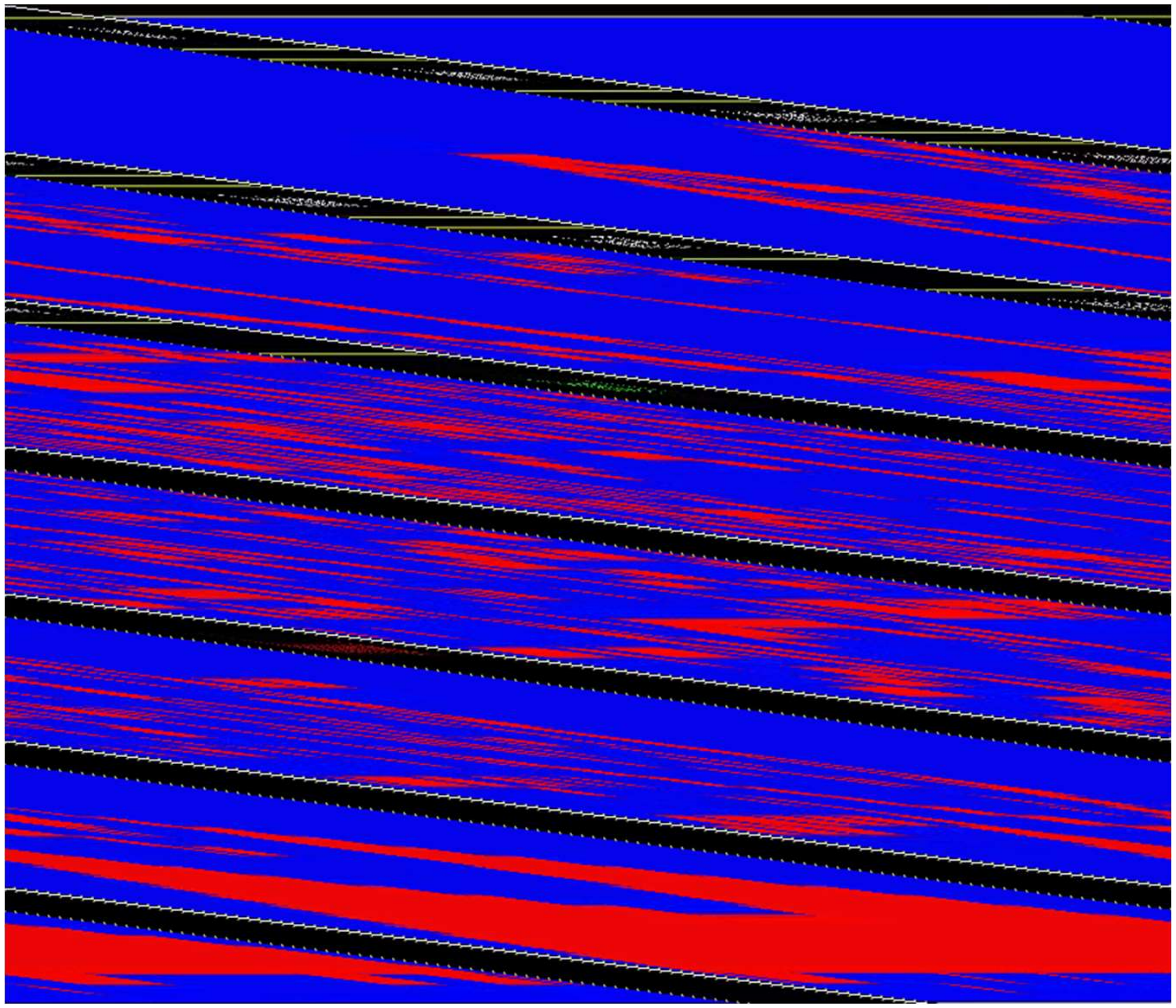
Examples: Content addressable memory



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

Computational algorithm

1. Initialize network with initial pattern

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

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

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?

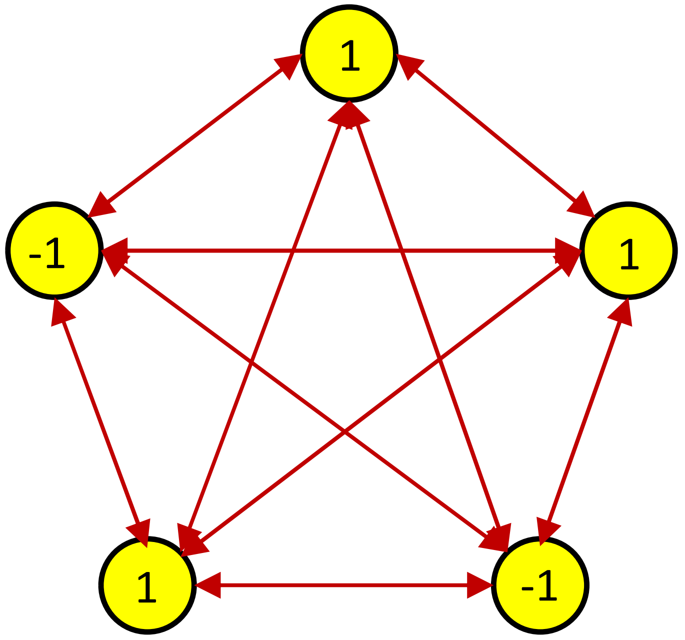
- How to “retrieve” patterns better..

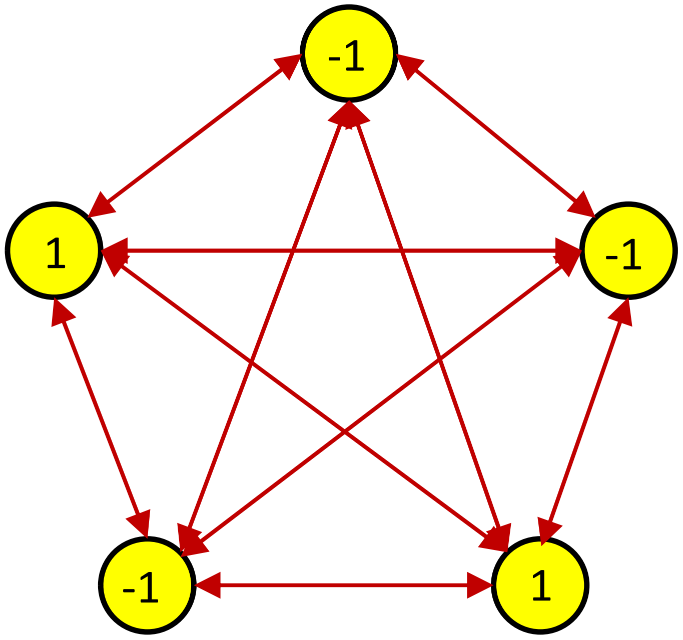
How do we remember a *specific* pattern?

- 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 patterns: Training a network

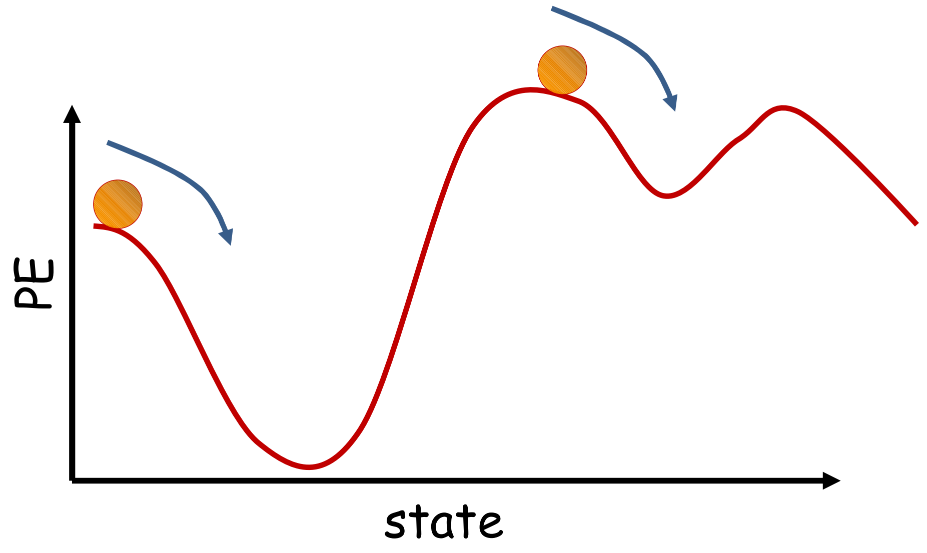


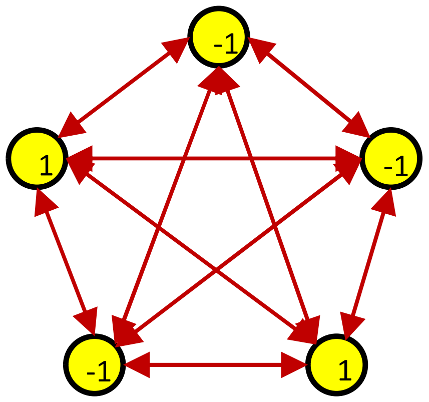


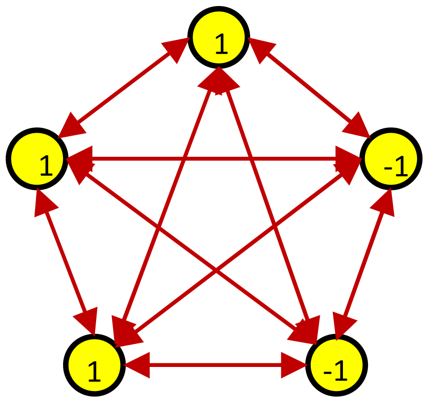
- A network that stores pattern P also naturally stores – P
  - Symmetry $E(P) = E(-P)$ since E is a function of $y_i y_j$

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

A network can store *multiple* patterns

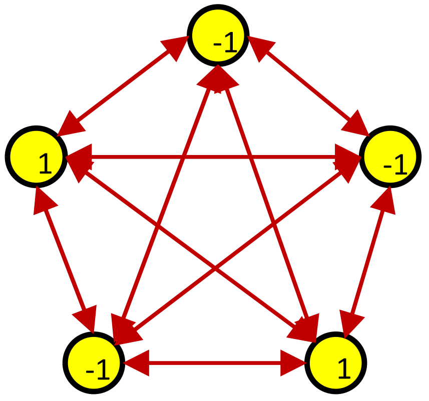


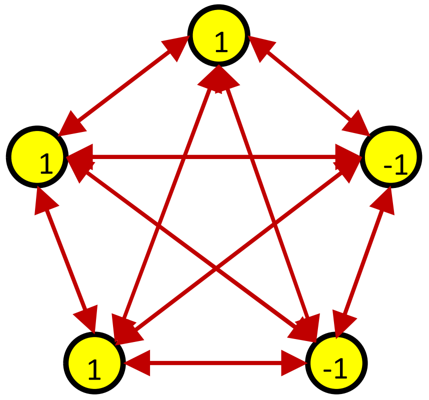




- Every stable point is a stored pattern
- So we could design the net to store multiple patterns
  - Remember that every stored pattern P is actually *two* stored patterns, P and $-P$

Storing a pattern

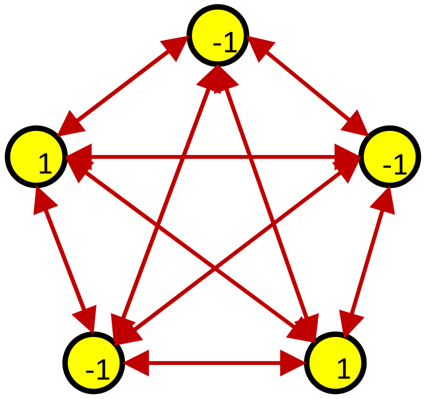




$$E = -\sum_{i}\sum_{j<i} w_{ji} y_j y_i$$

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

Storing specific patterns

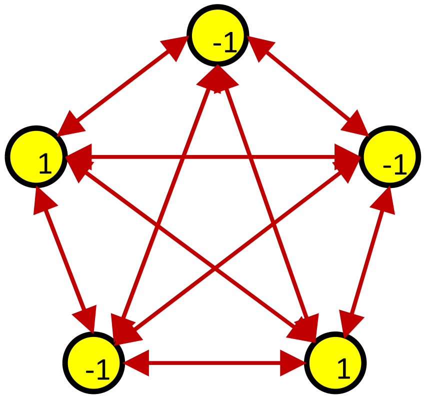


- Storing 1 pattern: We want

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

- This is a stationary pattern

Storing specific patterns



HEBBIAN LEARNING:

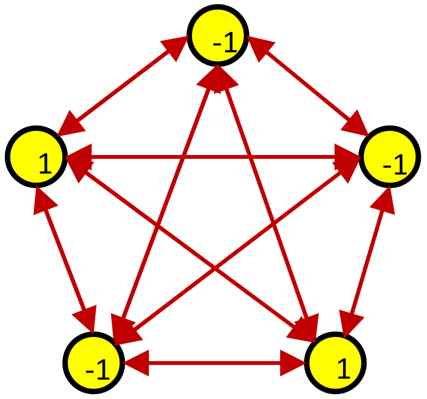
$$w_{ji} = y_j y_i$$

- 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



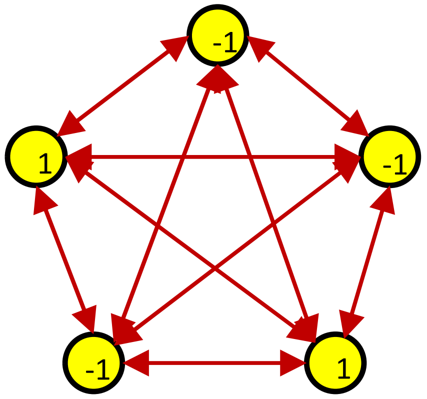
HEBBIAN LEARNING:

$$w_{ji} = y_j y_i$$

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

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

Storing specific patterns



HEBBIAN LEARNING:

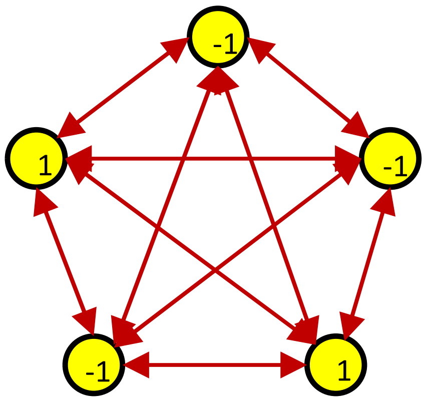
$$w_{ji} = y_j y_i$$

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:

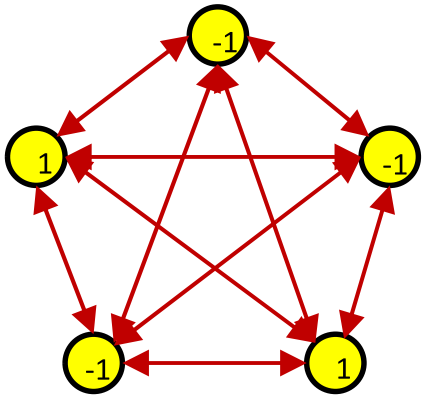
$$w_{ji} = y_j y_i$$

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

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

- This is the lowest possible energy value for the network

Storing specific patterns



HEBBIAN LEARNING:

$$w_{ji} = y_j y_i$$

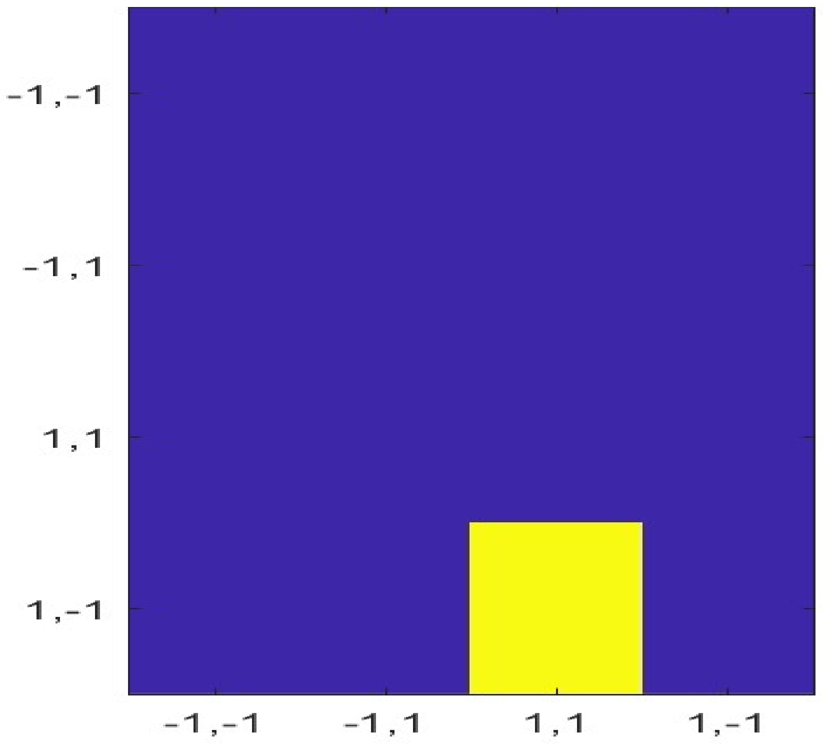
The pattern is *STABLE*

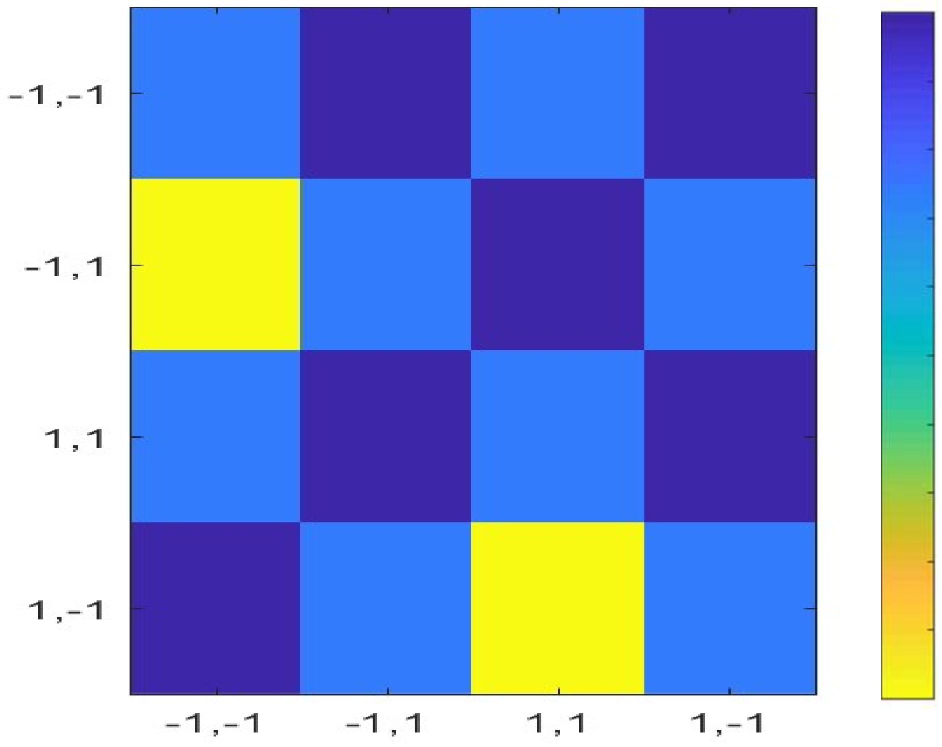
$$E = -\sum_i \sum_{j<i} w_{ji} y_j y_i = -\sum_i \sum_{j<i} y_i^2 y_j^2$$

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

- This is the lowest possible energy value for the network

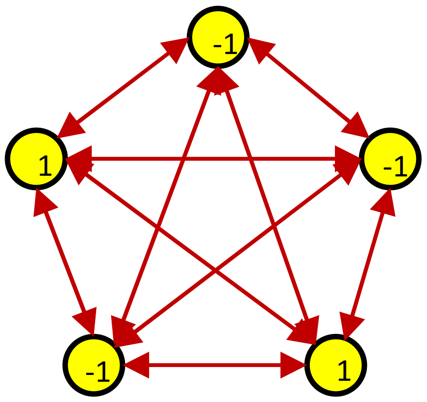
Hebbian learning: Storing a 4-bit pattern

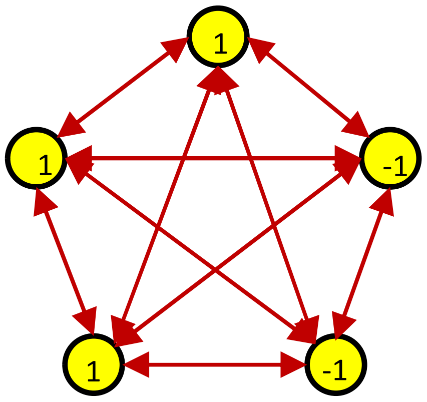




- Left: Pattern stored.   Right: Energy map
- Stored pattern has lowest energy
- Gradation of energy ensures stored pattern (or its ghost) is recalled from everywhere

Storing multiple patterns



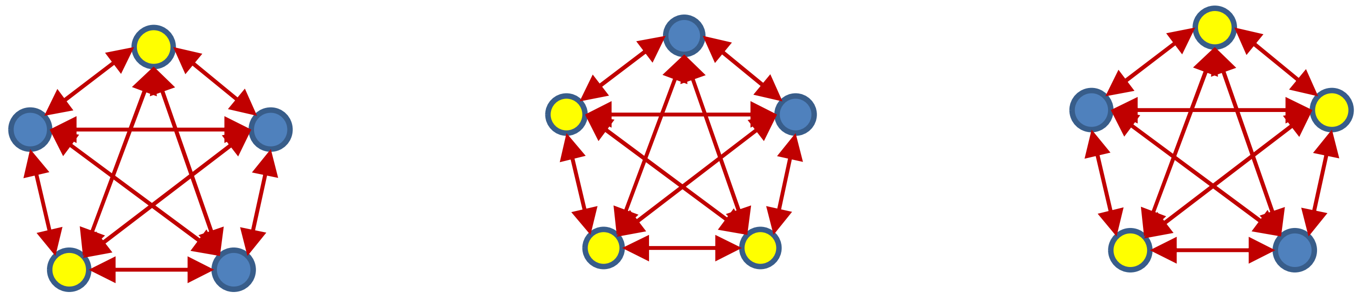


- To store *more* than one pattern

$$w_{ji} = \sum_{\mathbf{y}_p \in \{\mathbf{y}_p\}} y_i^p y_j^p$$

- $\{\mathbf{y}_p\}$ is the set of patterns to store
- Super/subscript p represents the specific pattern

How many patterns can we store?



- **Hopfield**: For a network of N neurons can store up to ~$0.15N$ random patterns through Hebbian learning
  - Provided they are "far" enough
- Where did this number come from?

The limits of Hebbian Learning

- Consider the following: We must store K N-bit patterns of the form

$$\mathbf{y}_k = [y_1^k, y_2^k, \dots, y_N^k], k = 1 \dots K$$

- Hebbian learning (scaling by $\frac{1}{N}$ for normalization, this does not affect actual pattern storage):

$$w_{ij} = \frac{1}{N}\sum_k y_i^k y_j^k$$

- **For any pattern $\mathbf{y}_p$ to be stable:**

$$y_i^p \sum_j w_{ij} y_j^p > 0 \quad \forall i$$

$$y_i^p \frac{1}{N}\sum_j \sum_k y_i^k y_j^k \, y_j^p > 0 \quad \forall i$$

The limits of Hebbian Learning

- **For any pattern $\mathbf{y}_p$ to be stable:**

$$y_i^p \frac{1}{N} \sum_j \sum_k y_i^k y_j^k \, y_j^p > 0 \quad \forall i$$

$$y_i^p \frac{1}{N} \sum_j y_i^p y_j^p y_j^p + y_i^p \frac{1}{N} \sum_j \sum_{k \neq p} y_i^k y_j^k \, y_j^p > 0 \quad \forall i$$

- Note that the first term equals 1 (because $y_j^p y_j^p = y_i^p y_i^p = 1$)
  - i.e. for $\mathbf{y}_p$ to be stable the requirement is that the second *crosstalk term*:

$$y_i^p \frac{1}{N} \sum_j \sum_{k \neq p} y_i^k y_j^k \, y_j^p > -1 \quad \forall i$$

- The pattern will *fail* to be stored if the *crosstalk*

$$y_i^p \frac{1}{N} \sum_j \sum_{k \neq p} y_i^k y_j^k \, y_j^p < -1 \quad for\ any\ i$$

The limits of Hebbian Learning

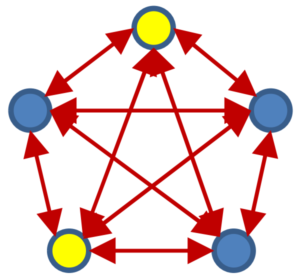
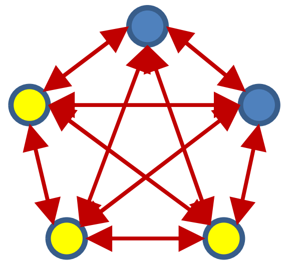
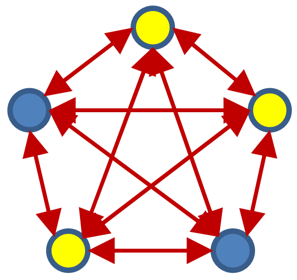
- For any random set of K patterns to be stored, the probability of the following must be low

$$\left(C_i^p = \frac{1}{N}\sum_j \sum_{k\neq p} y_i^p y_i^k y_j^k\, y_j^p\right) < -1$$

- For large N and K the probability distribution of C_i^p approaches a Gaussian with 0 mean, and variance K/N
  – Considering that individual bits $y_i^l \in \{-1, +1\}$ and have variance 1
- For a Gaussian, $C \sim N(0, K/N)$
  – $P(C < -1 \mid \mu = 0, \sigma^2 = K/N) <\ 0.004$ for $K/N < 0.14$

- I.e. To have less than 0.4% probability that stored patterns will *not* be stable, $K < 0.14N$

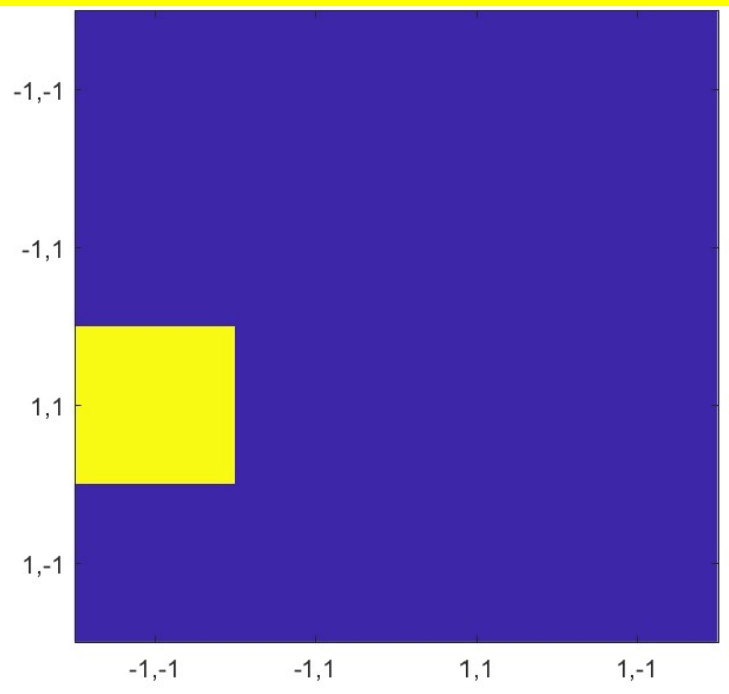
How many patterns can we store?

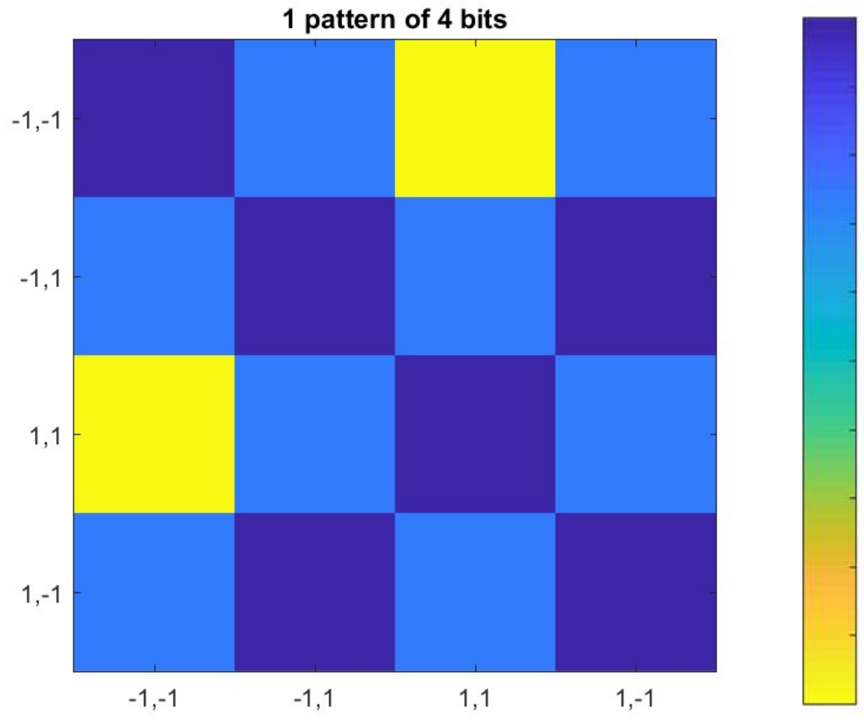




- A network of N neurons trained by Hebbian learning can store up to ~0.14N random patterns with low probability of error
  - Computed assuming $prob(bit = 1) = 0.5$
    - On average no. of matched bits in any pair = no. of mismatched bits
      - Patterns are "orthogonal" – maximally distant – from one another
  - Expected behavior for *non-orthogonal* patterns?
- To get some insight into what is stored, lets see some examples

Hebbian learning: One 4-bit pattern

Topological representation on a Karnaugh map





- Left: Pattern stored. Right: Energy map
- Note: Pattern is an energy well, but there are other local minima
  - Where?
  - Also note "shadow" pattern

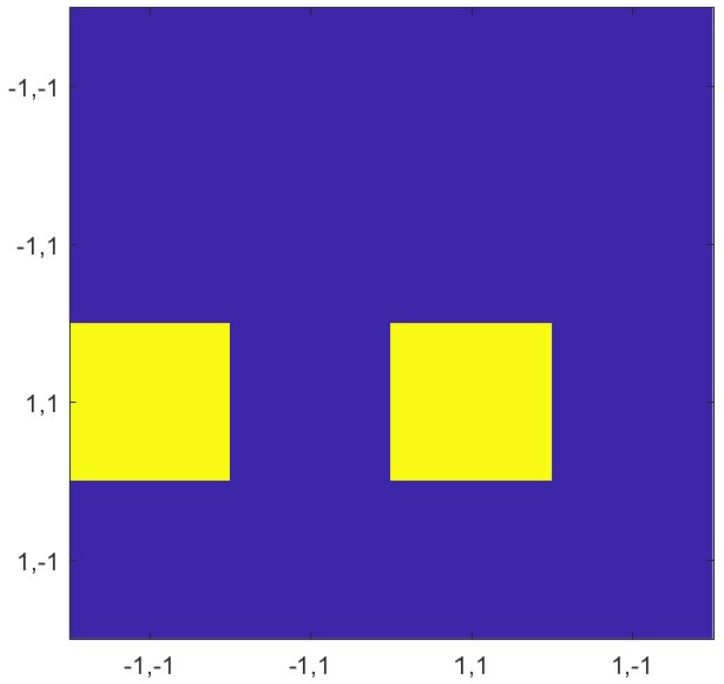
Storing multiple patterns: Orthogonality

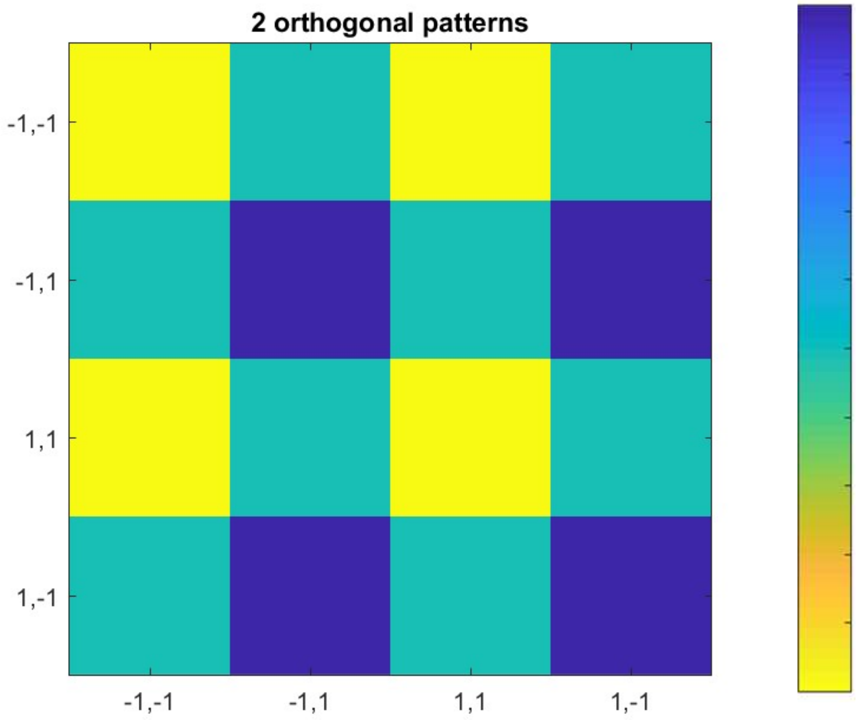
- The maximum Hamming distance between two N-bit patterns is $N/2$
  - Because any pattern $Y = -Y$ for our purpose

- Two patterns $y_1$and y_2 that differ in $N/2$ bits are *orthogonal*
  - Because $y_1^T y_2 = 0$

- For $N = 2^M L$, where L is an odd number, there are at most 2^M orthogonal binary patterns
  - Others may be *almost* orthogonal

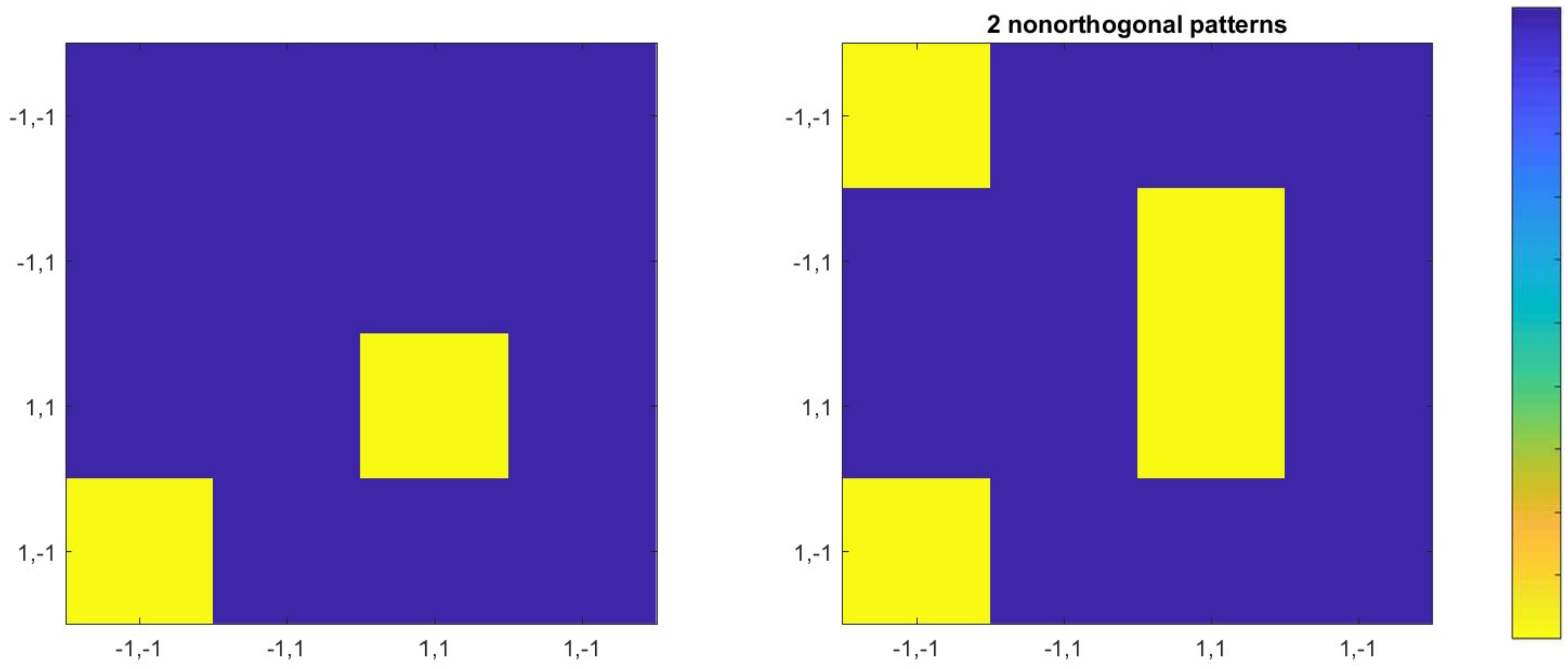
Two orthogonal 4-bit patterns





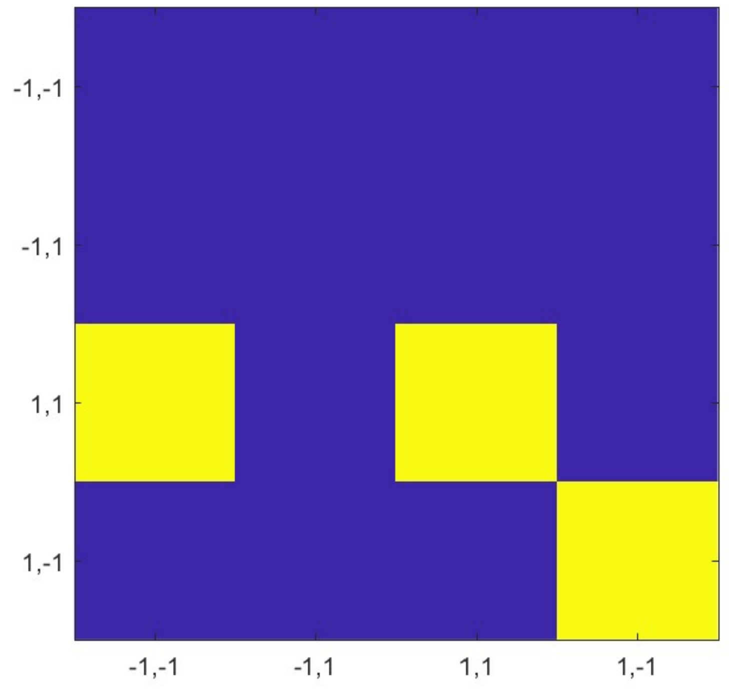
- Patterns are local minima (stationary and stable)
  - No other local minima exist
  - But patterns perfectly confusable for recall

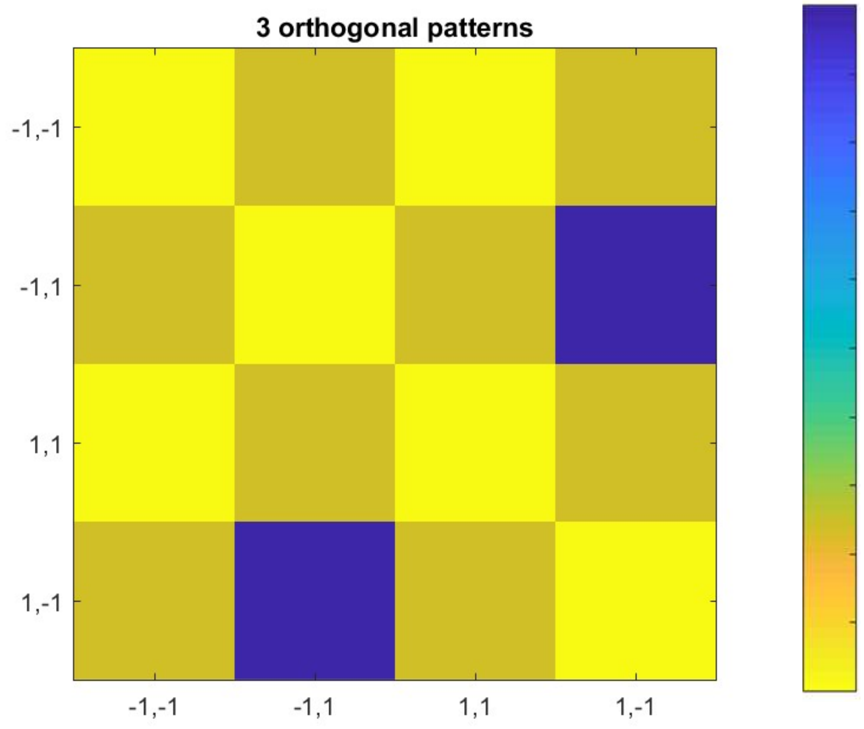
Two *non*-orthogonal 4-bit patterns



- Patterns are local minima (stationary and stable)
  - No other local minima exist
  - Actual *wells* for patterns
    - Patterns may be perfectly recalled!
  - Note K > 0.14 N

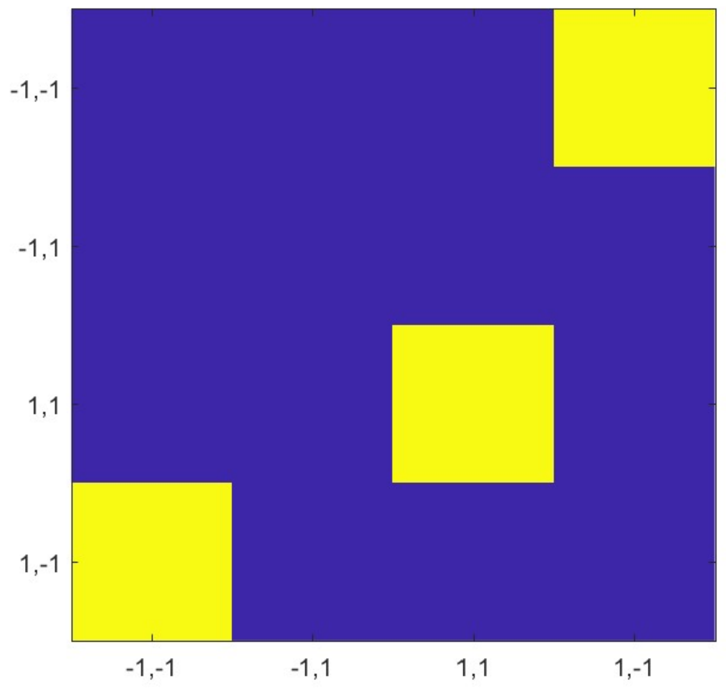
Three orthogonal 4-bit patterns

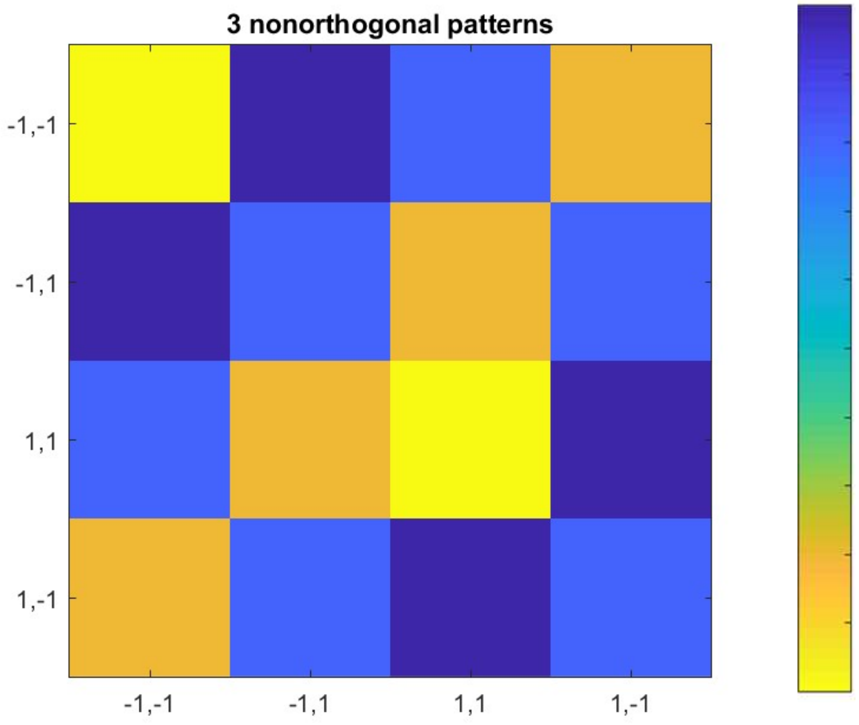




- All patterns are local minima (stationary)
  - – But recall from perturbed patterns is random

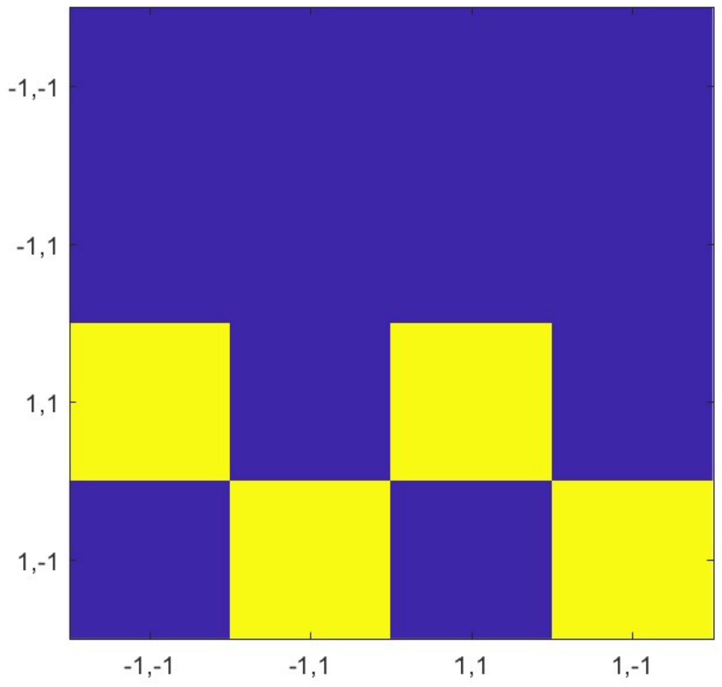
Three *non*-orthogonal 4-bit patterns

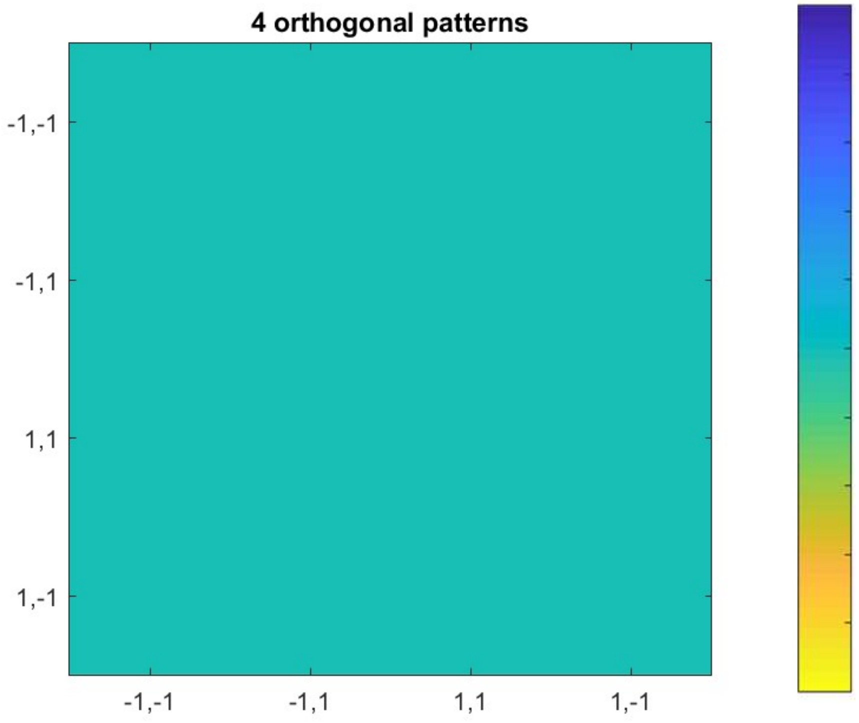




- Patterns in the corner are not recalled
  - They end up being attracted to the -1,-1 pattern
  - Note some "ghosts" ended up in the "well" of other patterns
    - So one of the patterns has stronger recall than the other two

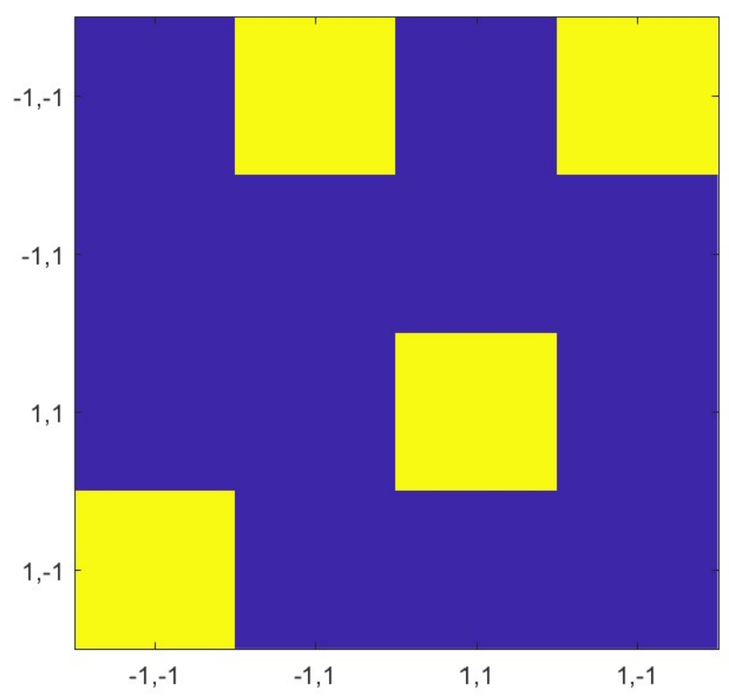
Four orthogonal 4-bit patterns

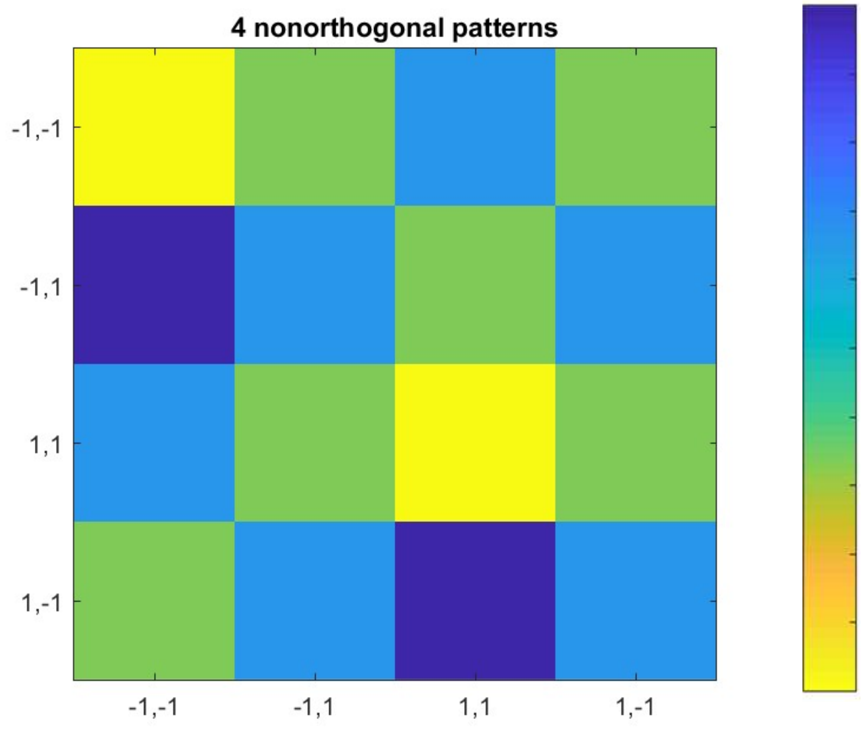




- All patterns are stationary, but none are stable
  - Total wipe out

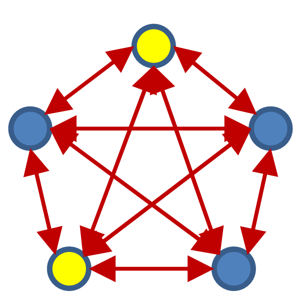
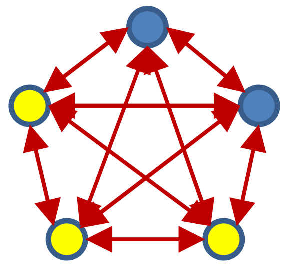
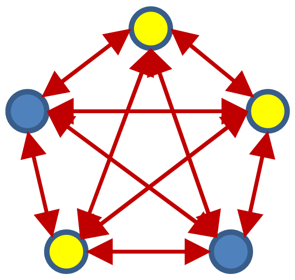
*Four non*orthogonal 4-bit patterns





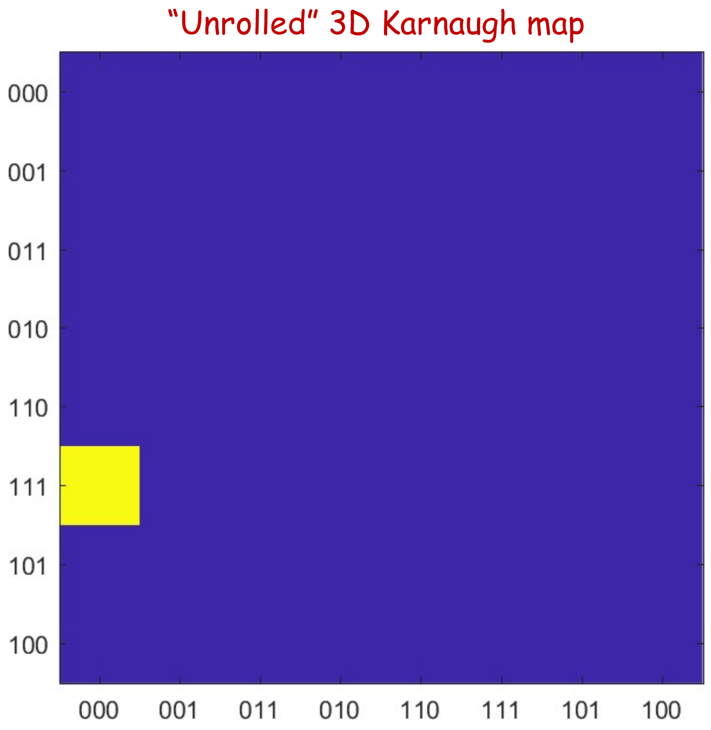
- One stable pattern
  - "Collisions" when the ghost of one pattern occurs next to another

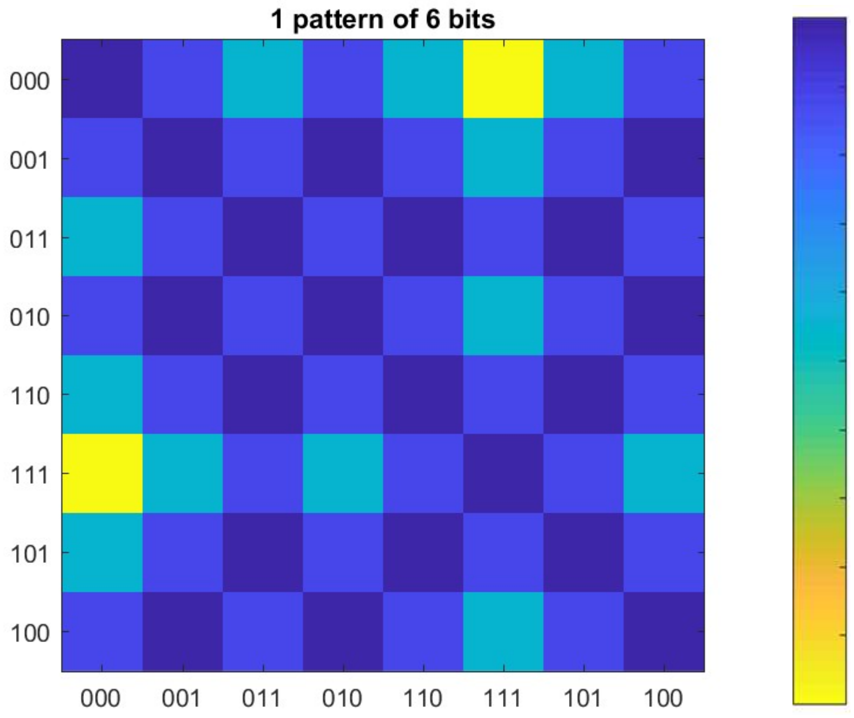
How many patterns can we store?

- Hopfield: For a network of N neurons can store up to $0.14N$ random patterns
- Apparently a fuzzy statement
  - What does it really mean to say "stores" 0.14N random patterns?
    - Stationary? Stable? No other local minima?
  - What if the patterns to store are not random?
- N=4 may not be a good case (N too small)

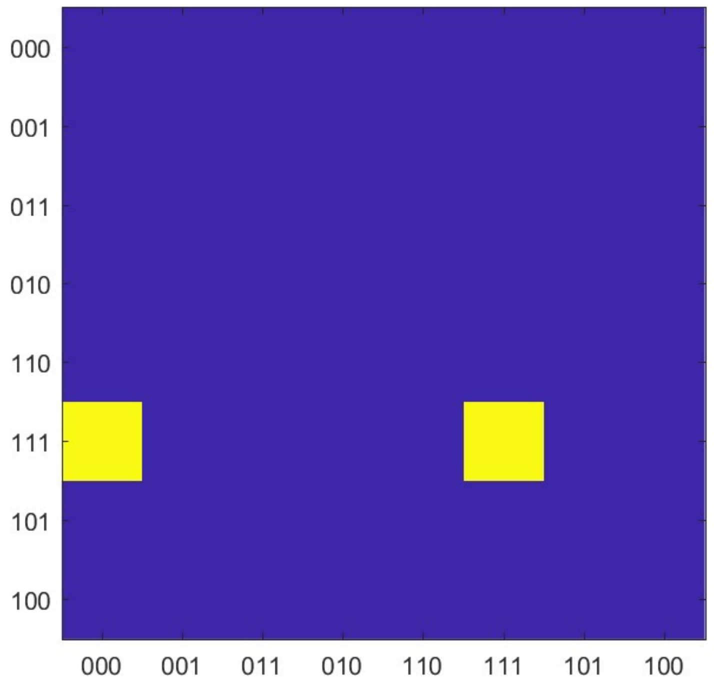
A 6-bit pattern

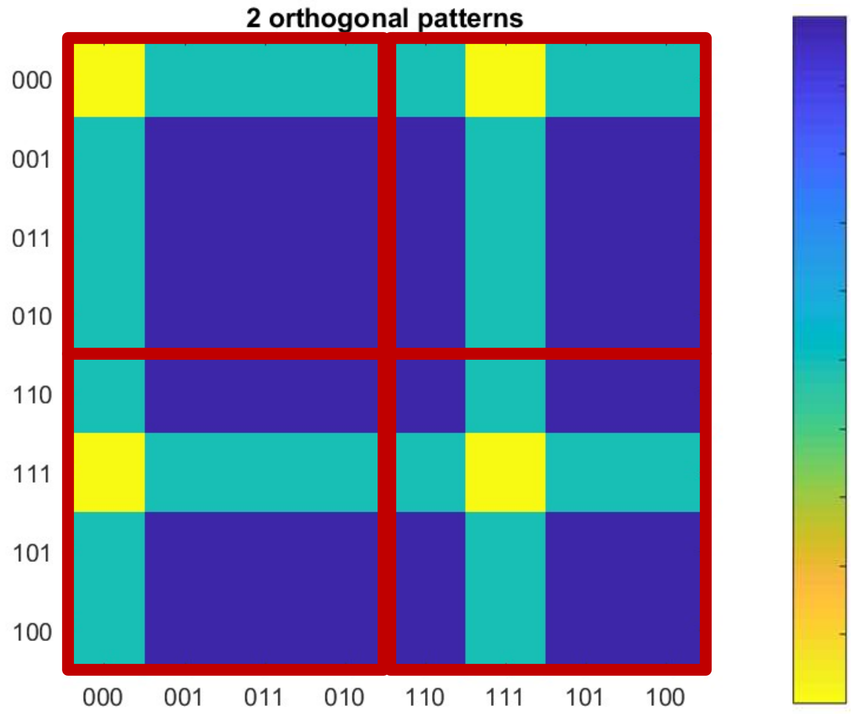




- Perfectly stationary and stable
- But many spurious local minima..
  - Which are "fake" memories

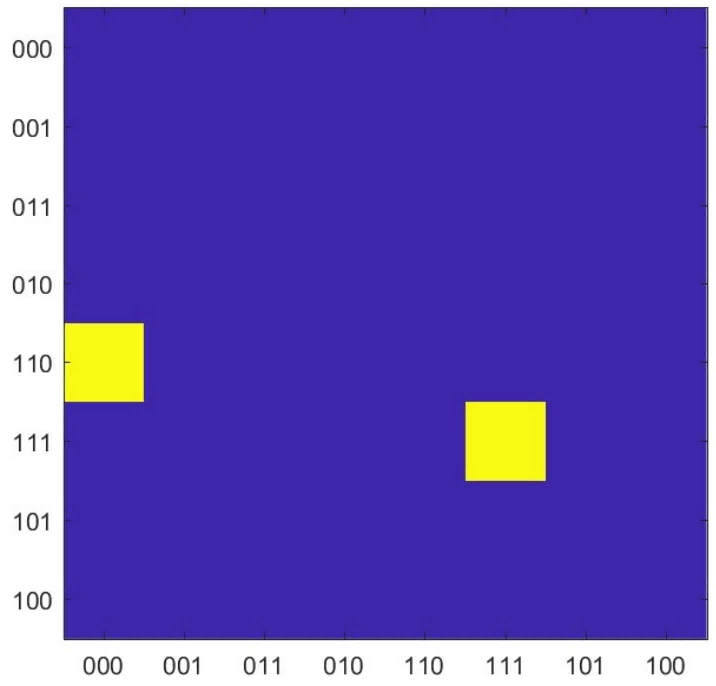
Two orthogonal 6-bit patterns

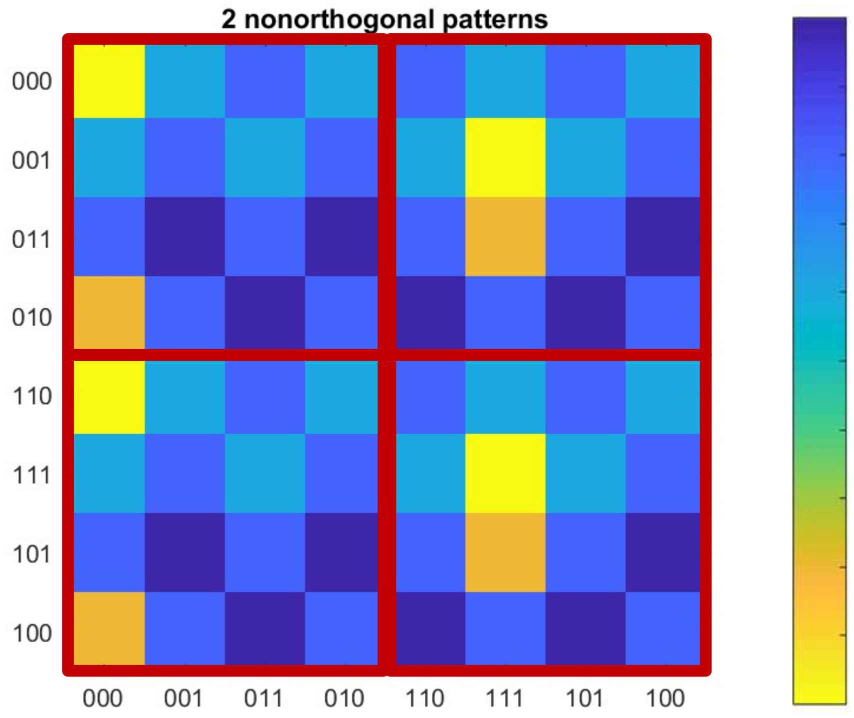




- Perfectly stationary and stable
- Several spurious "fake-memory" local minima..
  - Figure overstates the problem: actually a 3-D Kmap

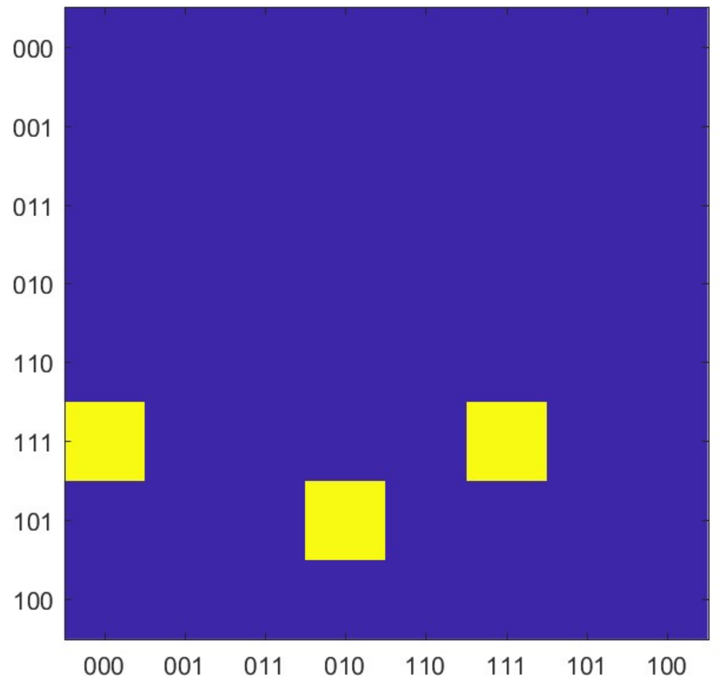
Two non-orthogonal 6-bit patterns

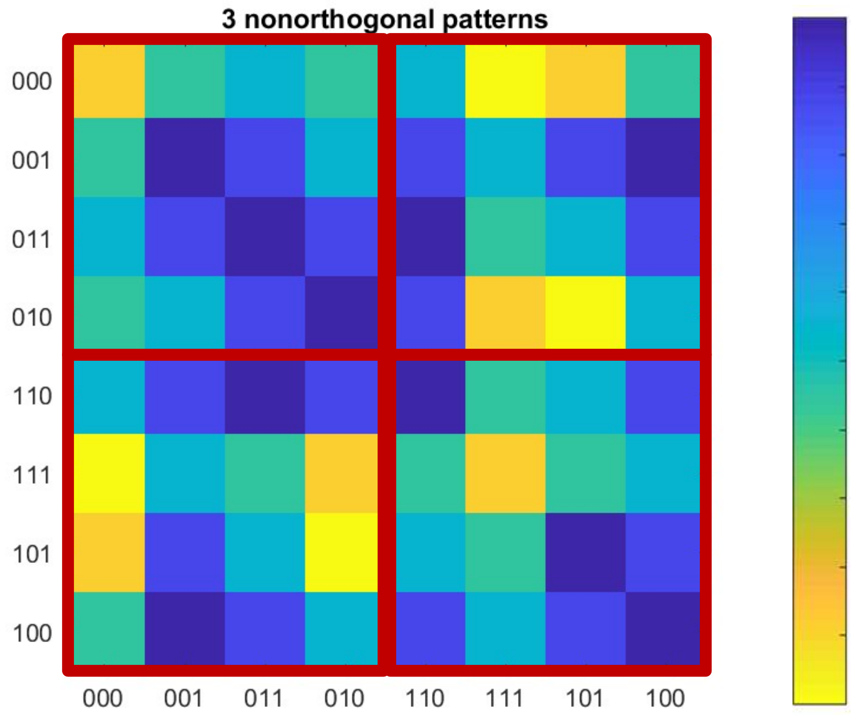




- Perfectly stationary and stable
- Some spurious "fake-memory" local minima..
  - But every stored pattern has "bowl"
  - *Fewer* spurious minima than for the orthogonal case

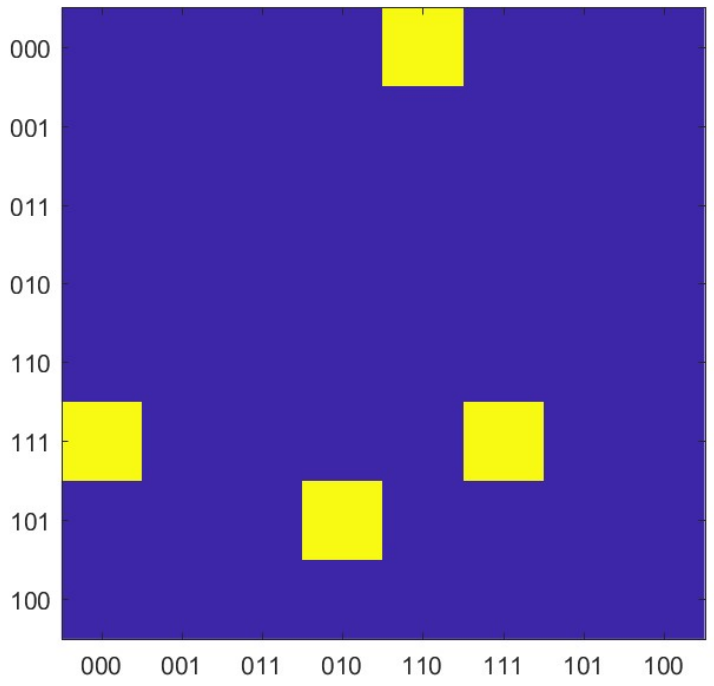
Three *non*-orthogonal 6-bit patterns

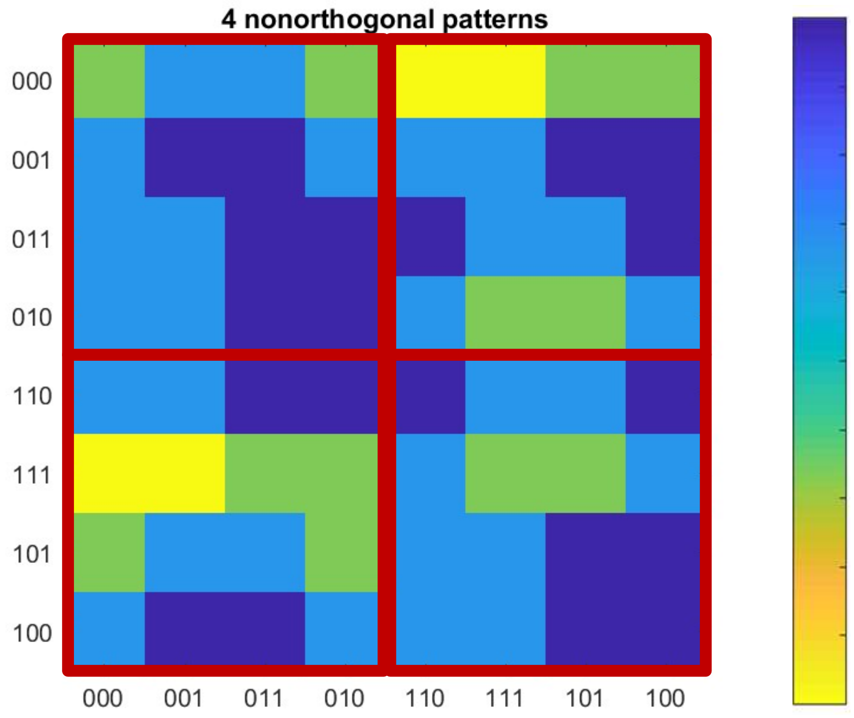




- Note: Cannot have 3 or more orthogonal 6-bit patterns..
- Patterns are perfectly stationary and stable (K > 0.14N)
- Some spurious “fake-memory” local minima..
  - But every stored pattern has “bowl”
  - *Fewer* spurious minima than for the orthogonal 2-pattern case

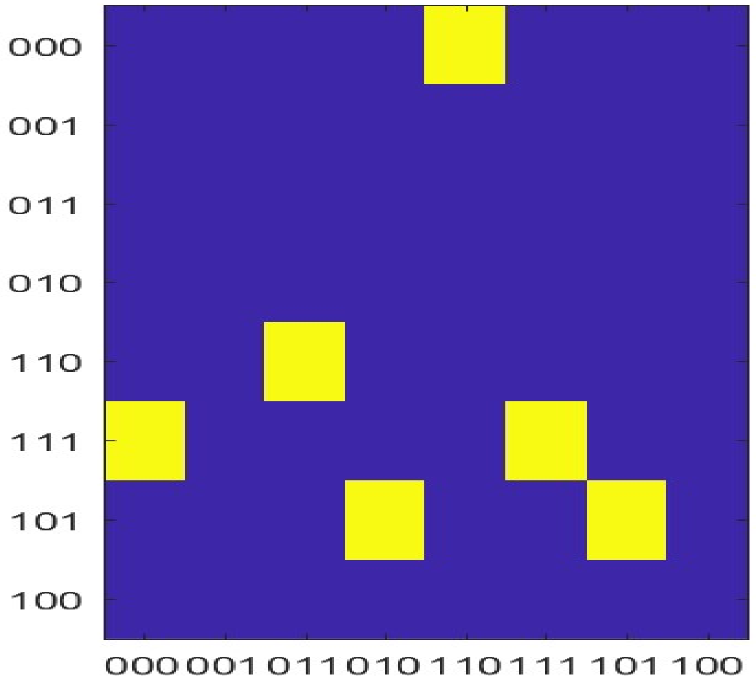
Four *non*-orthogonal 6-bit patterns

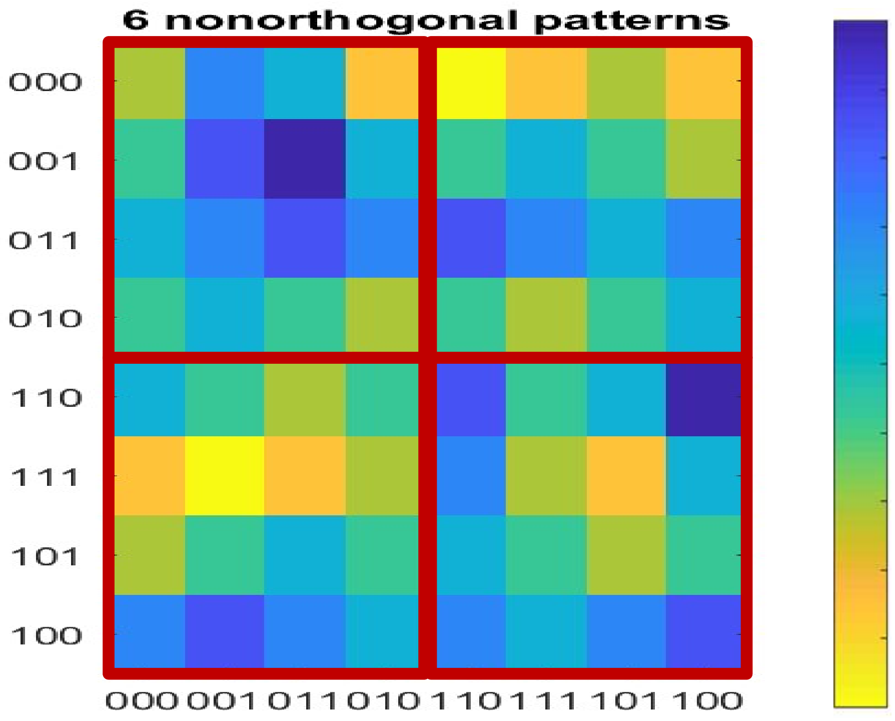




- Patterns are perfectly stationary for K > 0.14N
- *Fewer* spurious minima than for the orthogonal 2-pattern case
  - Most fake-looking memories are in fact ghosts..

Six *non*-orthogonal 6-bit patterns



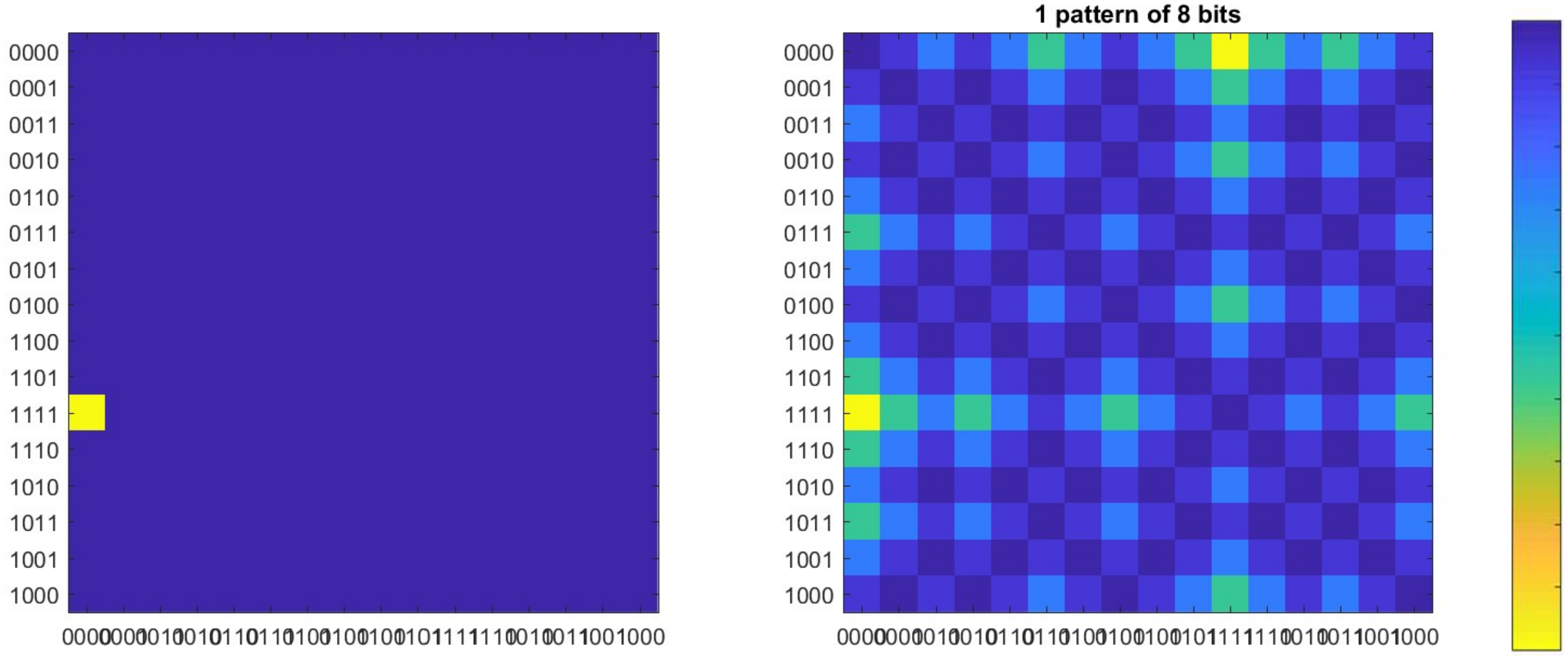


- Breakdown largely due to interference from “ghosts”
- But multiple patterns are stationary, and often stable
  - For K >> 0.14N

More visualization..

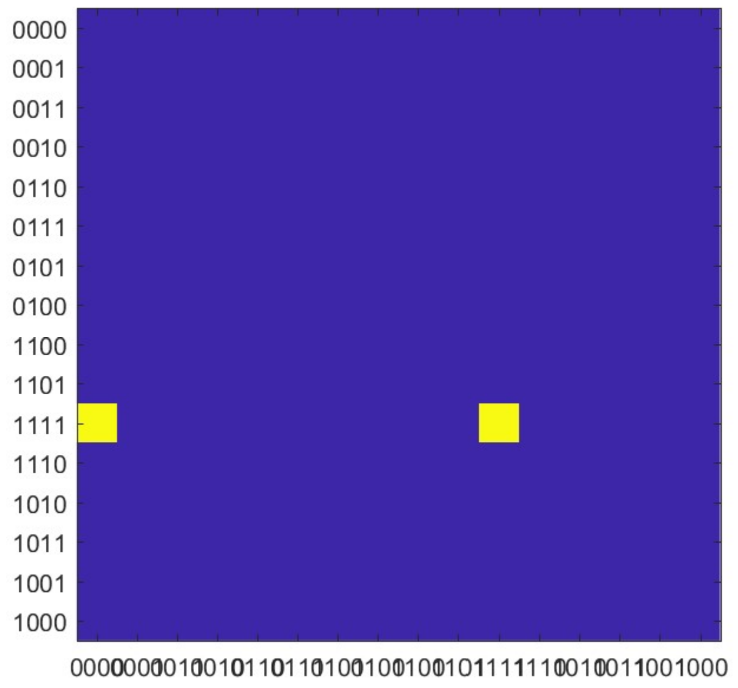
- Lets inspect a few 8-bit patterns
  - Keeping in mind that the Karnaugh map is now a 4-dimensional tesseract

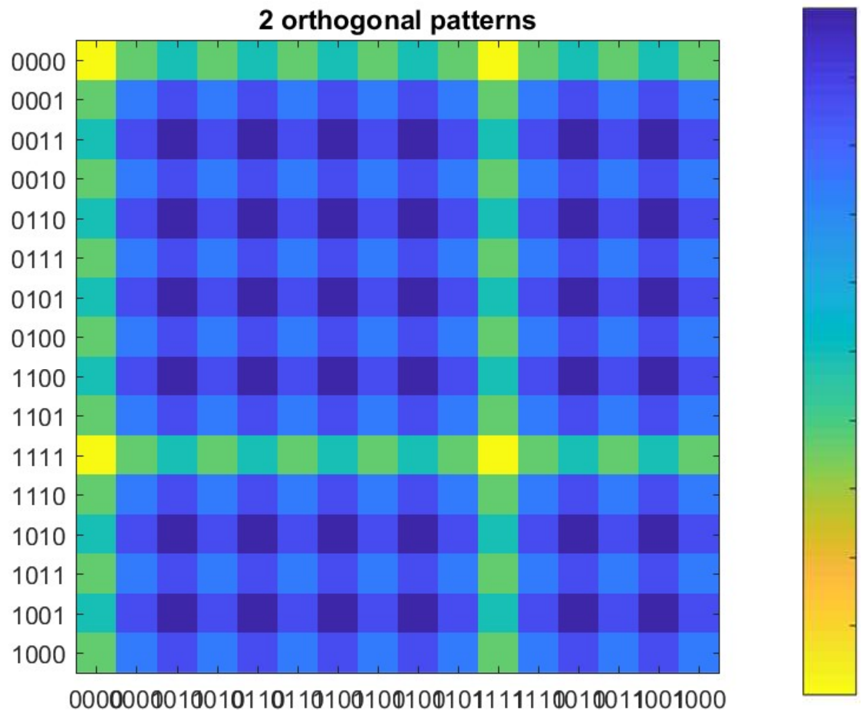
One 8-bit pattern



- Its actually cleanly stored, but there are a few spurious minima

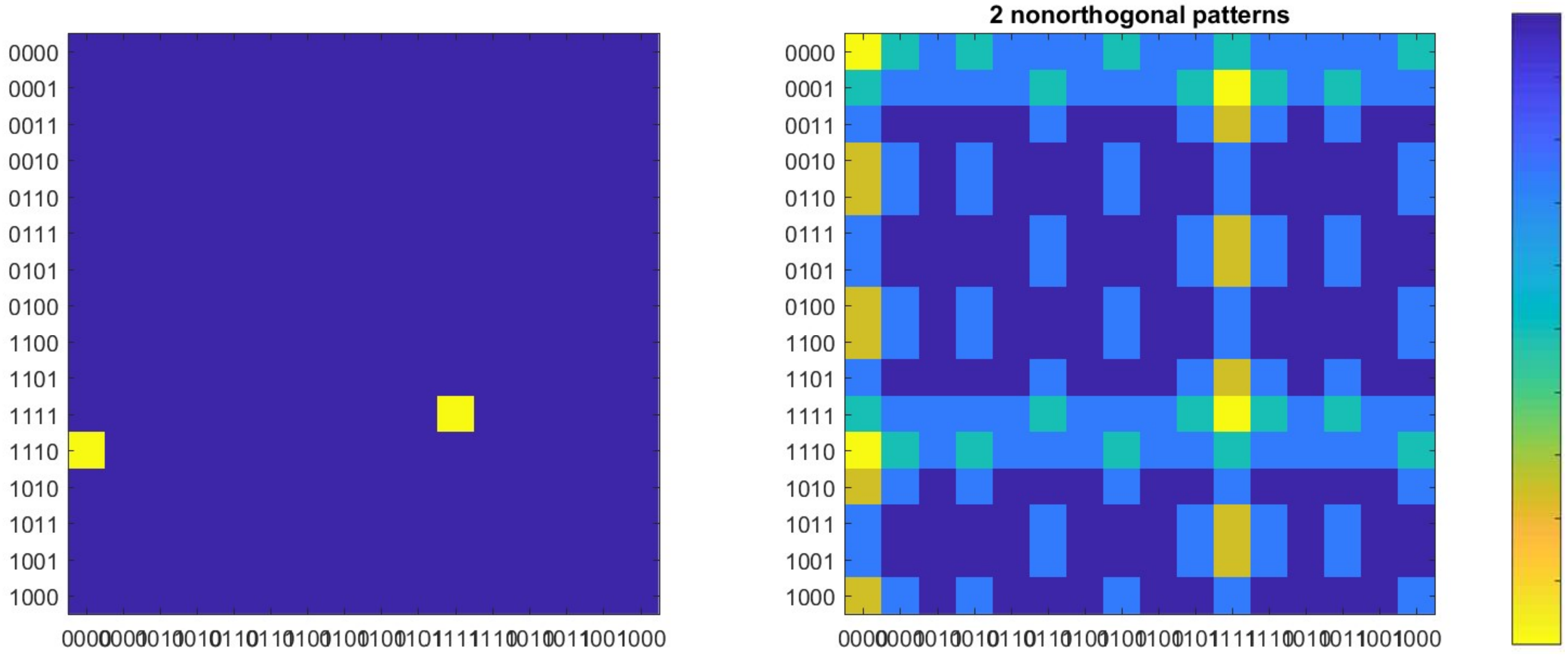
Two orthogonal 8-bit patterns





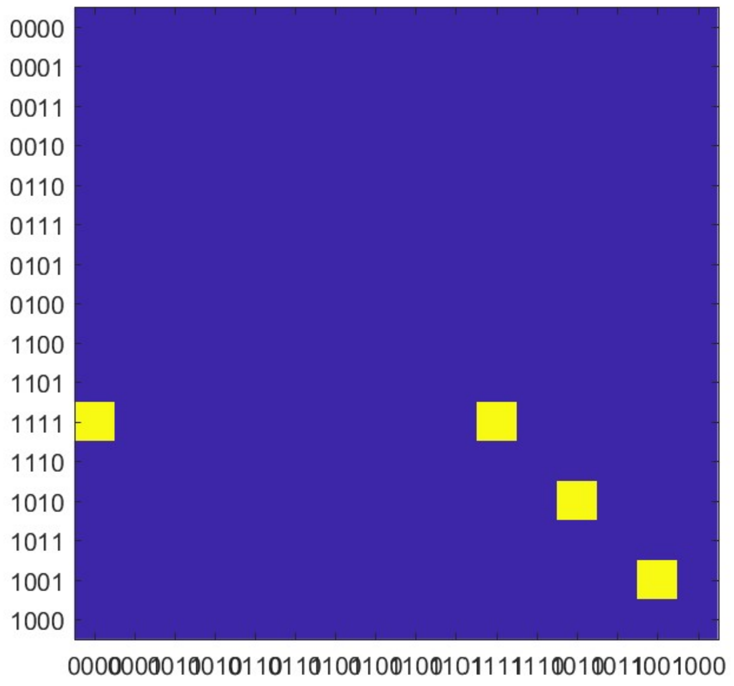
- Both have regions of attraction
- Some spurious minima

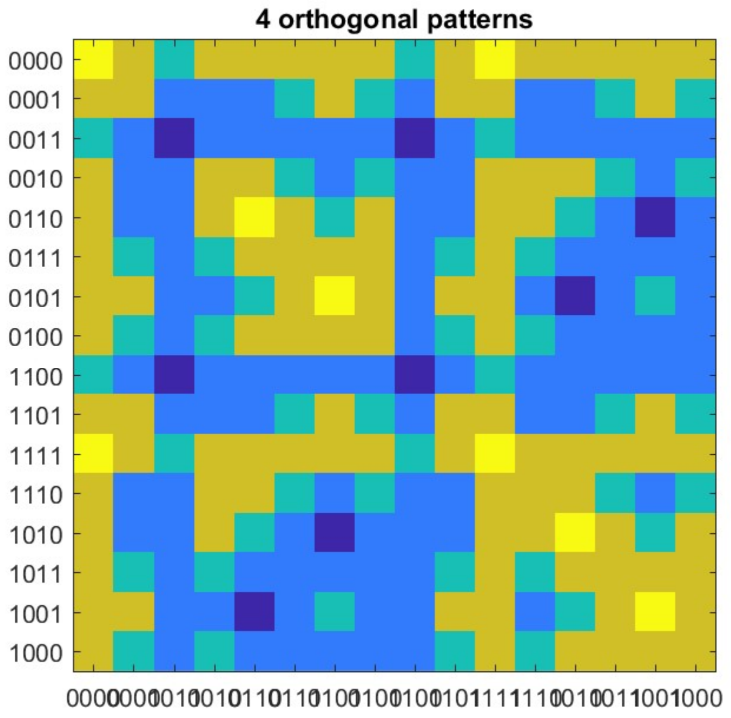
Two non-orthogonal 8-bit patterns



- Actually have fewer spurious minima
  - Not obvious from visualization..

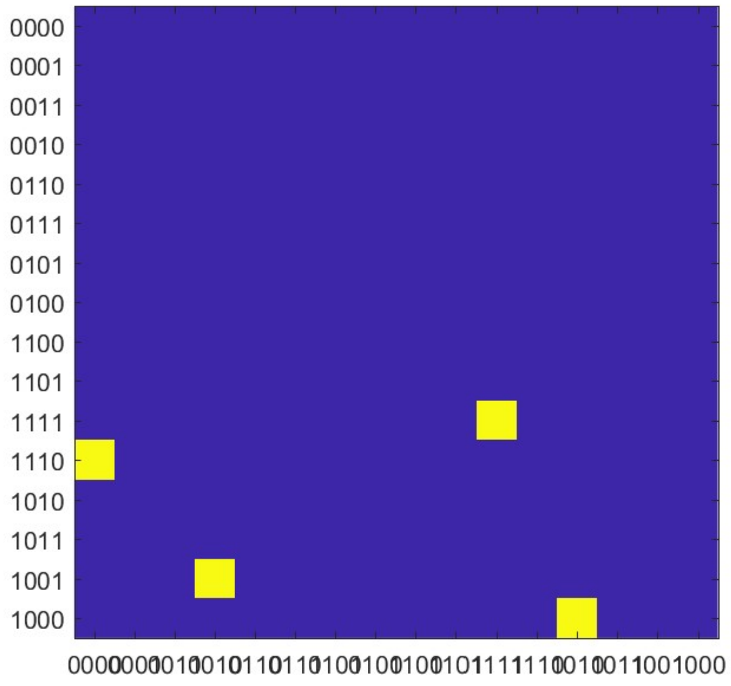
Four orthogonal 8-bit patterns

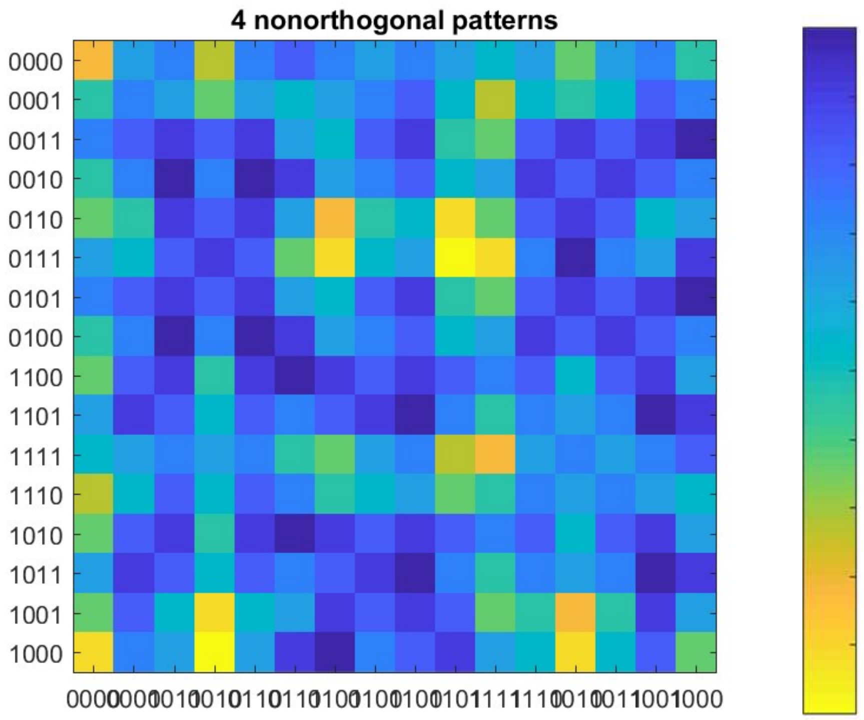




- Successfully stored

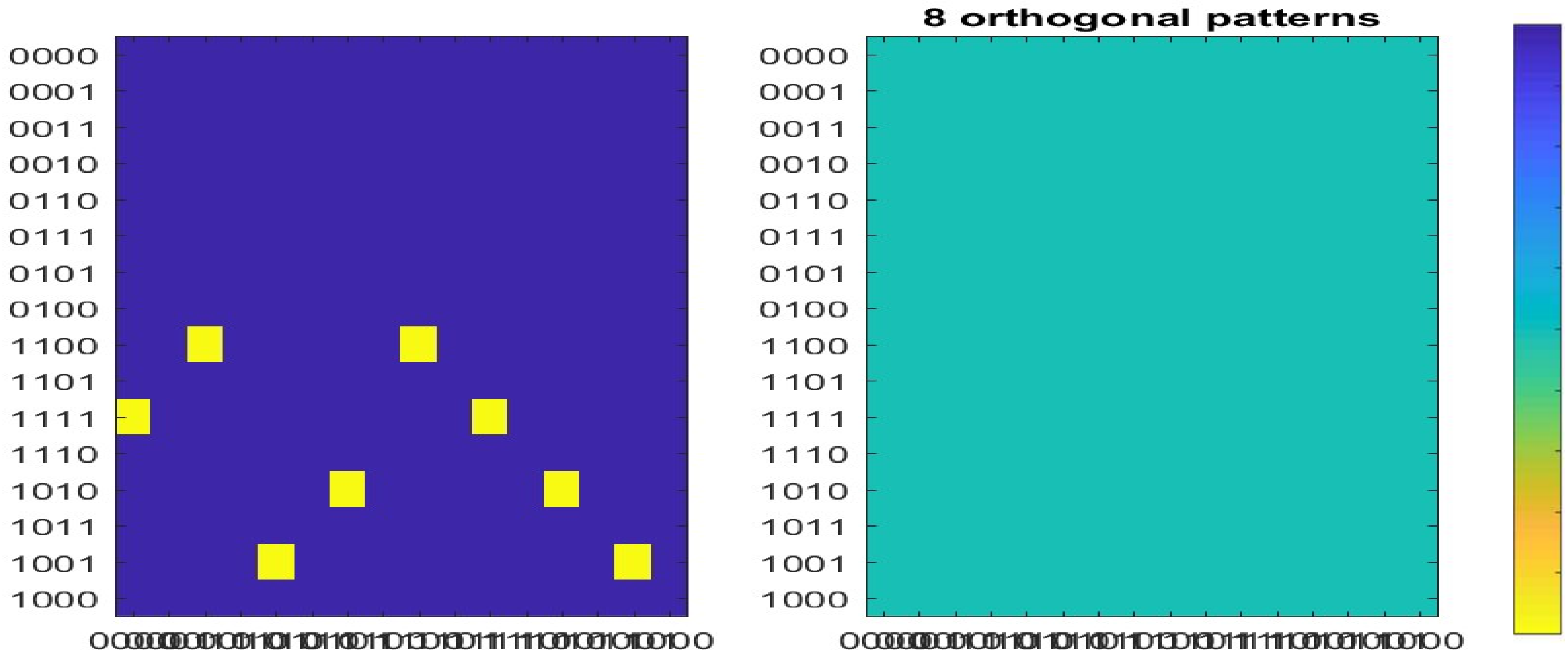
Four non-orthogonal 8-bit patterns





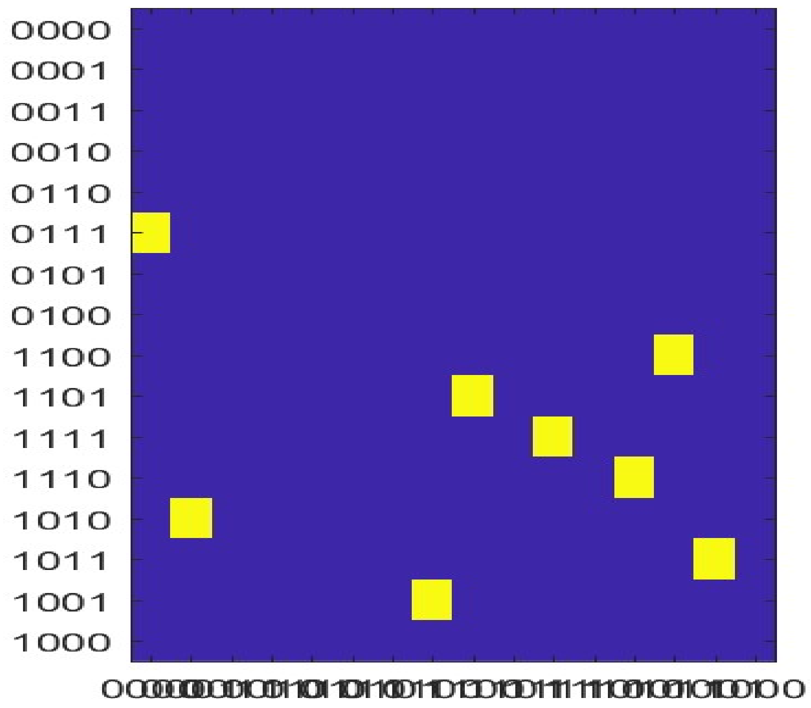
- Stored with interference from ghosts..

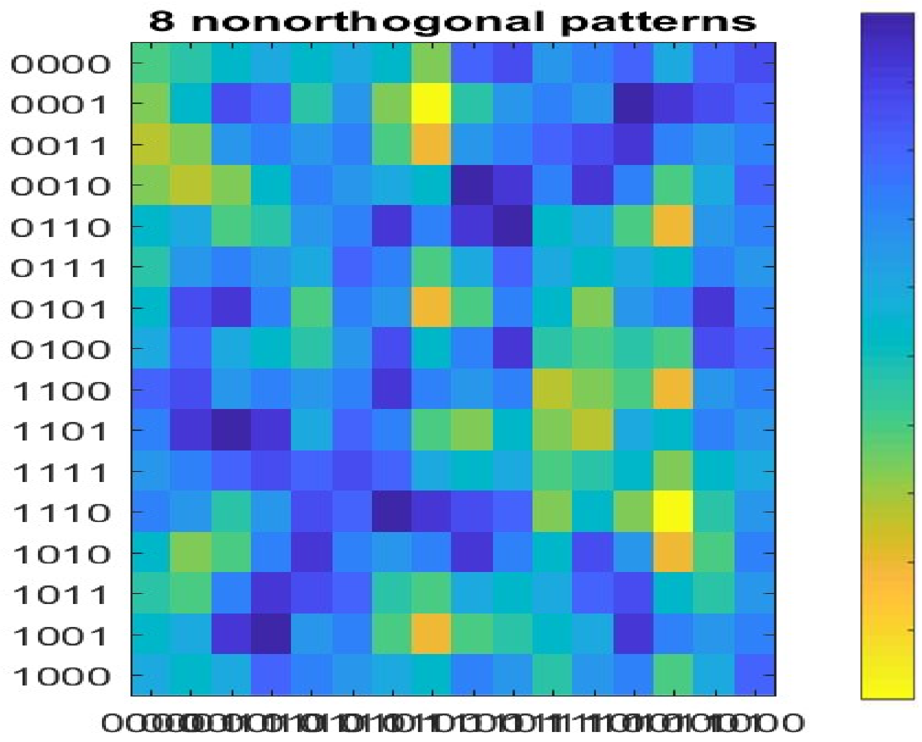
Eight orthogonal 8-bit patterns



- Wipeout

Eight non-orthogonal 8-bit patterns





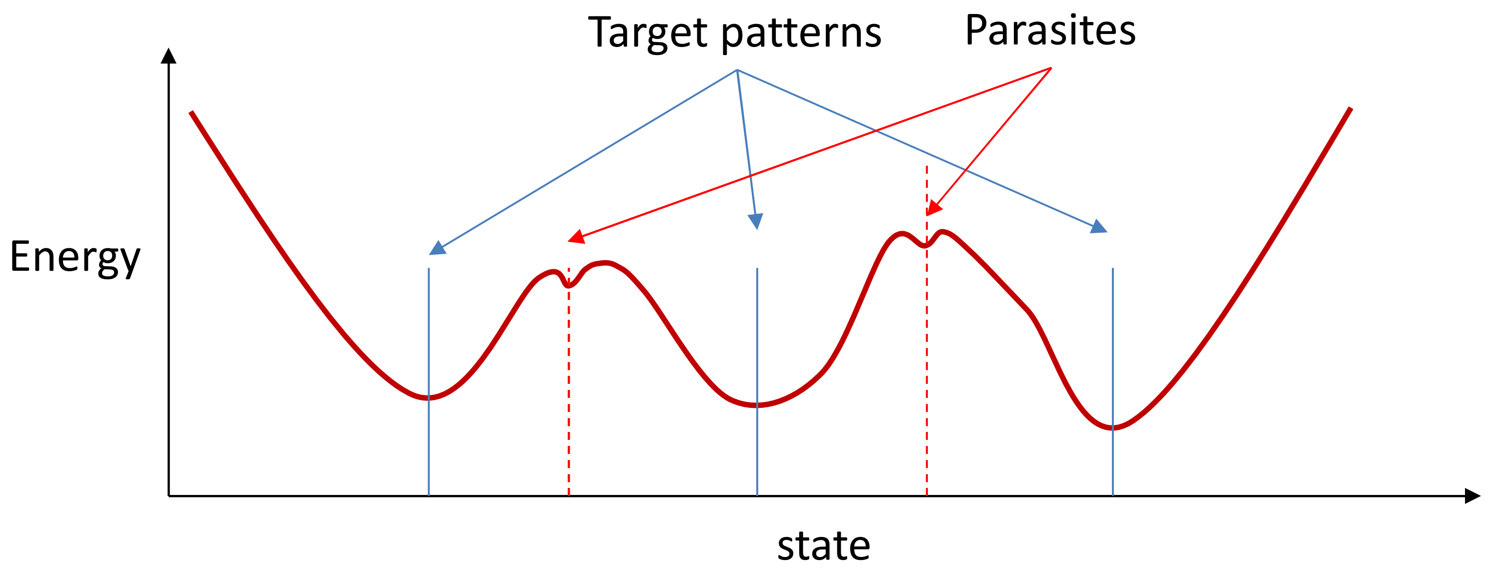
- Nothing stored
  - Neither stationary nor stable

Observations

- Many “parasitic” patterns
  - Undesired patterns that also become stable or attractors

- Apparently a capacity to store *more* than 0.14N patterns

Parasitic Patterns



- Parasitic patterns can occur because sums of odd numbers of stored patterns are also stable for Hebbian learning:
  - $\mathbf{y}_{parasite} = sign(\mathbf{y}_a + \mathbf{y}_b + \mathbf{y}_c)$
- They are also from other random local energy minima from the weights matrices themselves

Capacity

- Seems possible to store K > 0.14N patterns
  - i.e. obtain a weight matrix W such that K > 0.14N patterns are stationary
  - Possible to make more than 0.14N patterns at-least 1-bit stable

- Patterns that are *non-orthogonal* easier to remember
  - I.e. patterns that are *closer* are easier to remember than patterns that are farther!!

- Can we attempt to get greater control on the process than Hebbian learning gives us?
  - Can we do *better* than Hebbian learning?
    - Better capacity and fewer spurious memories?

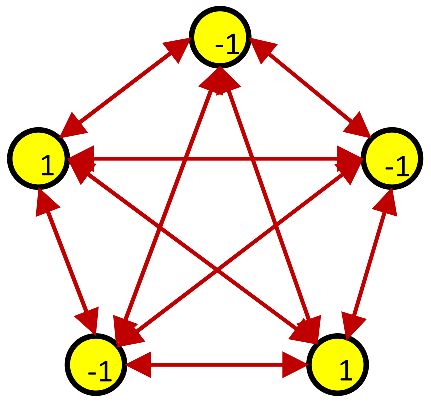
Story so far

- A Hopfield network is a loopy binary net 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 network acts as a *content-addressable* memory
    - Given a damaged memory, it can evolve to recall the memory fully
- The network must be designed to store the desired memories
  - Memory patterns must be *stationary* and *stable* on the energy contour
- Network memory can be trained by Hebbian learning
  - Guarantees that a network of N bits trained via Hebbian learning can store 0.14N random patterns with less than 0.4% probability that they will be unstable
- However, empirically it appears that we may sometimes be able to store *more than* 0.14N patterns

Bold Claim

- I can *always* store (upto) N orthogonal patterns such that they are stationary!
  - Why?
- I can avoid spurious memories by adding some noise during recall!

Recap: Hebbian Learning to Store a Specific Pattern



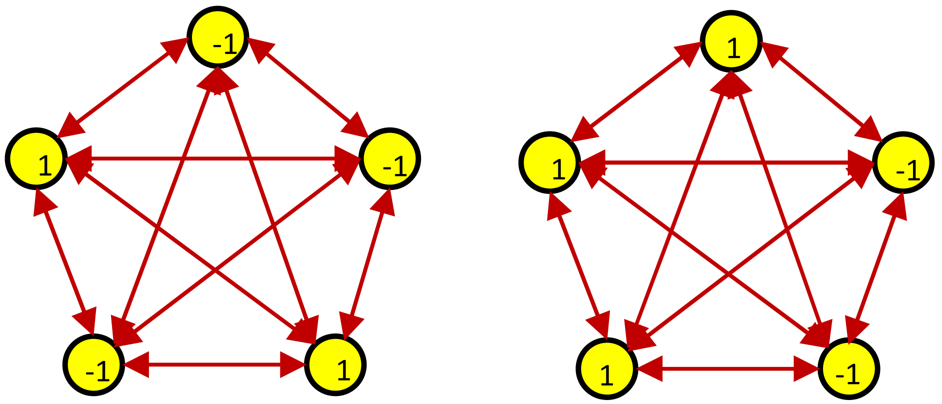
HEBBIAN LEARNING:

$$w_{ji} = y_j y_i$$

$$\mathbf{W} = \mathbf{y}_p \mathbf{y}_p^T - \mathrm{I}$$

- For a single stored pattern, Hebbian learning results in a network for which the target pattern is a global minimum

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 patterns
- Then..

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

Number of patterns

A minor adjustment

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

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

- Is identical to behavior with

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

Energy landscape only differs by an additive constant

Gradients and location of minima remain same

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

A minor adjustment

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

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

Both have the
same Eigen vectors

behavior with

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

Energy landscape
only differs by
an additive constant

Gradients and location
of minima remain same

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

A minor adjustment

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

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

behavior with

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

Both have the
same Eigen vectors

NOTE: This
is a positive
semidefinite matrix

Energy landscape
only differs by
an additive constant

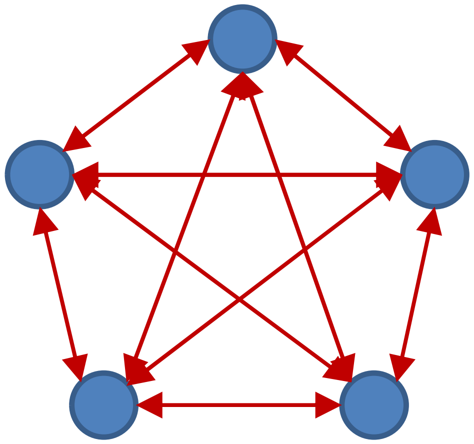
Gradients and location
of minima remain same

- S

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

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

Consider the energy function



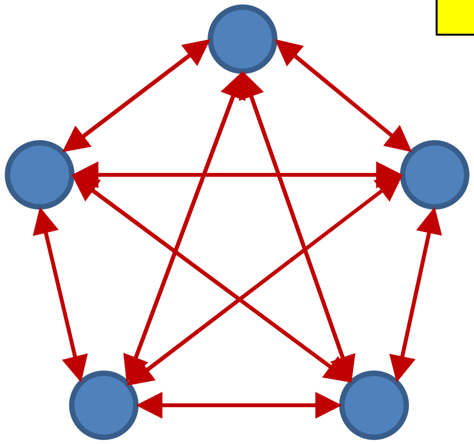
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$

Consider the energy function

This is a quadratic!

For Hebbian learning
W is positive semidefinite

E is concave

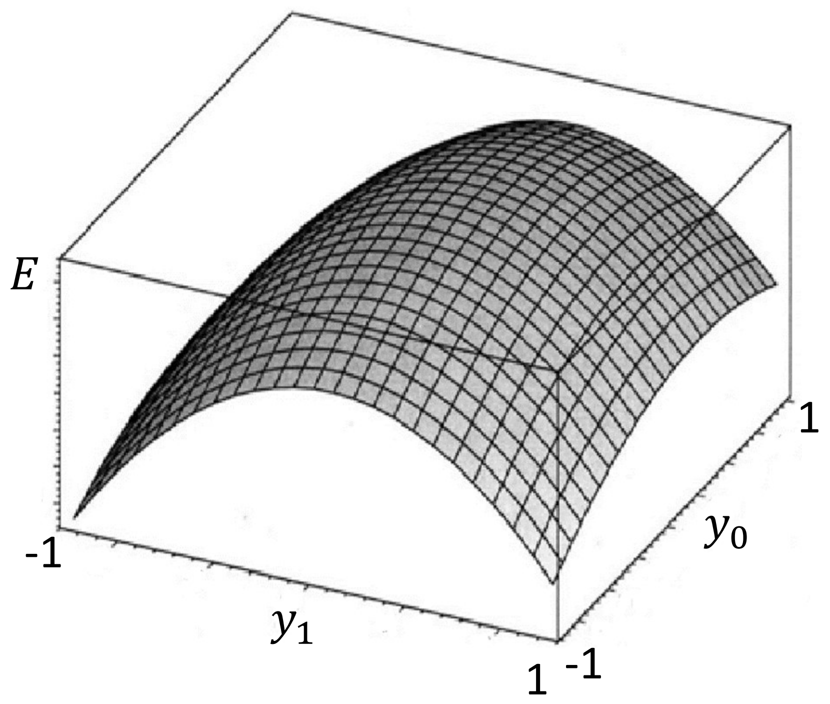


$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$

- The Energy function is concave if $\mathbf{W}$ is positive (semi) definite

The Energy function

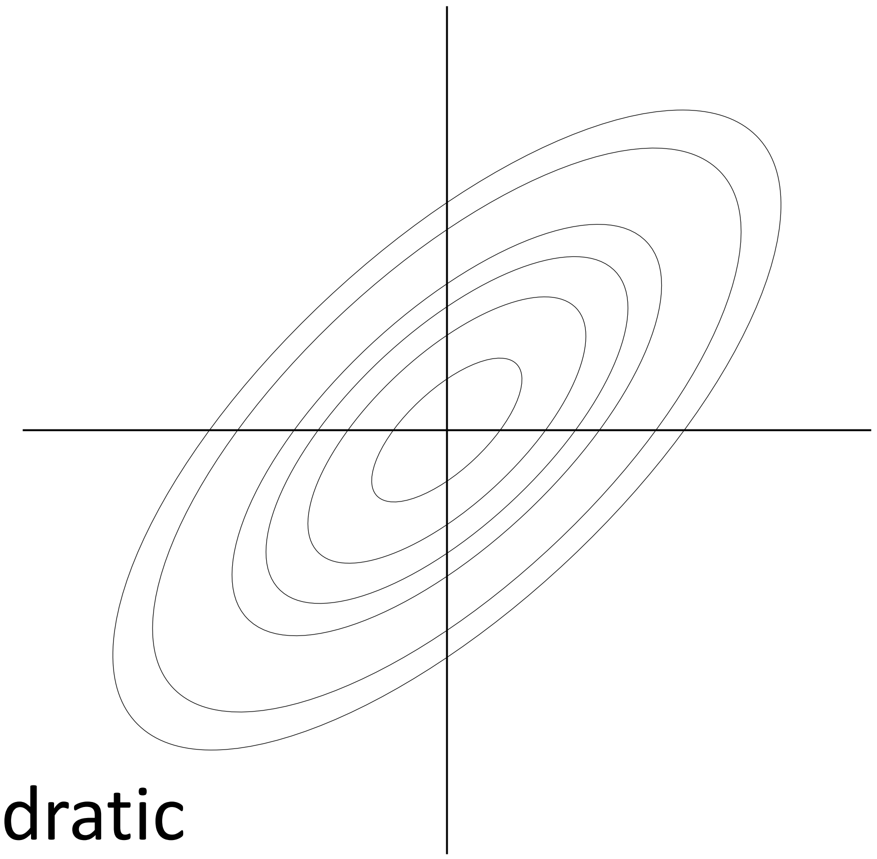
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$



- E is a concave quadratic

The Energy function

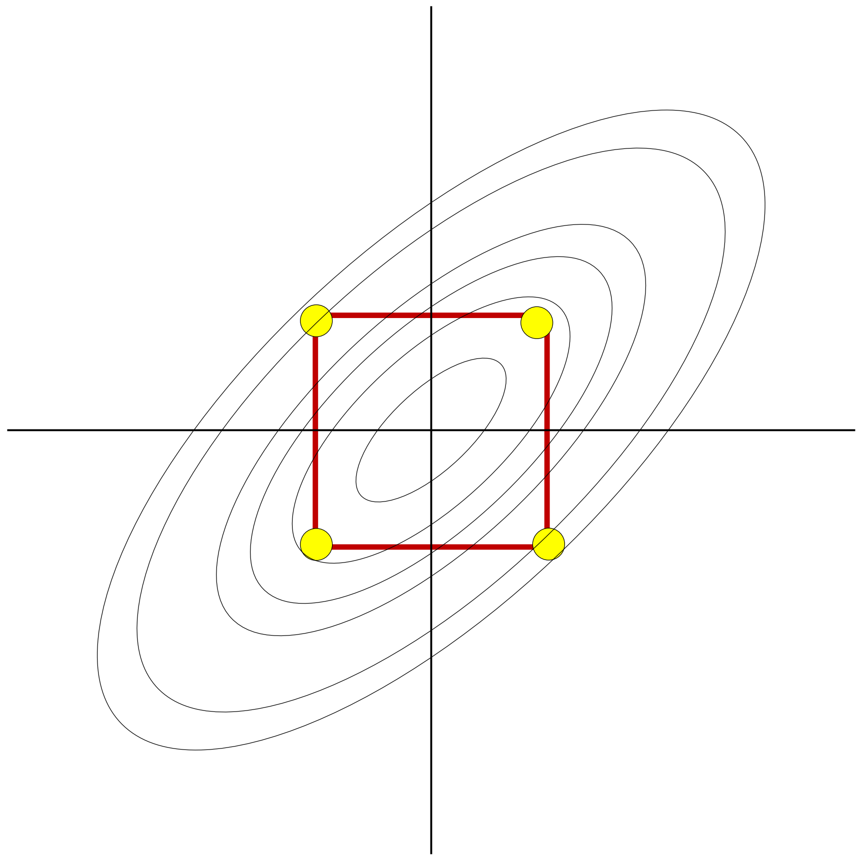
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$



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

The energy function

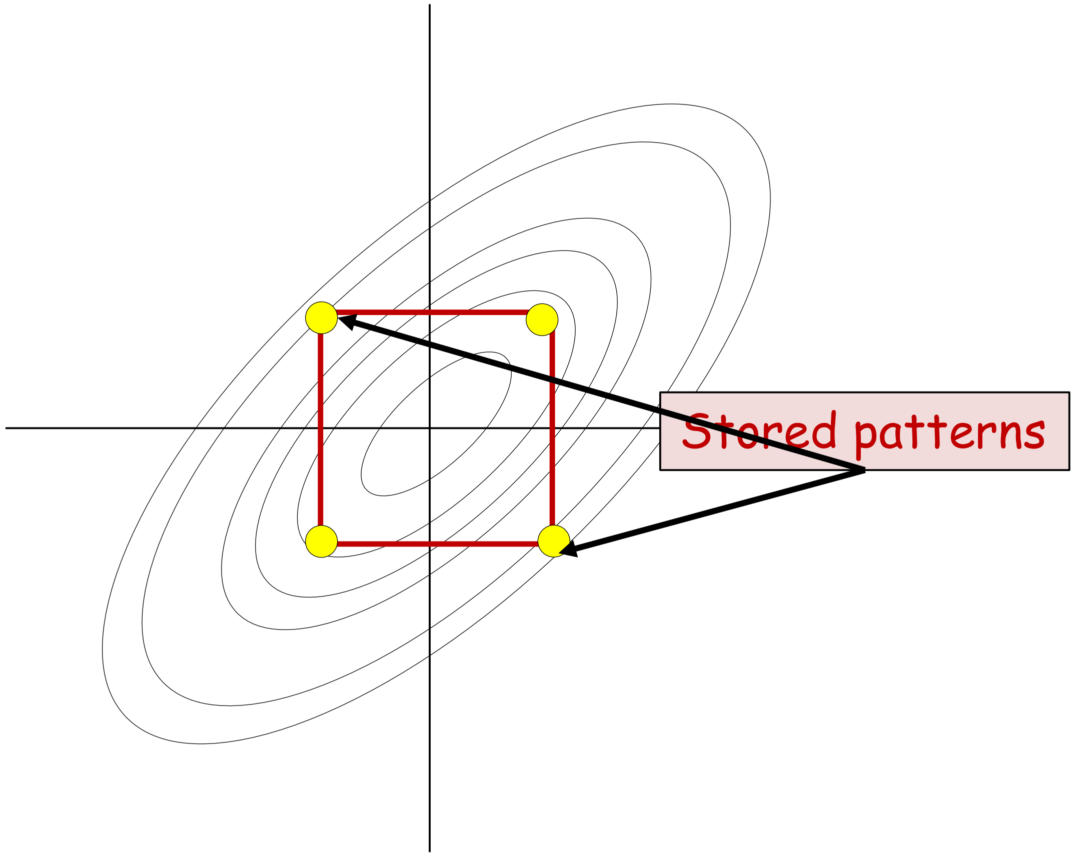
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$



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

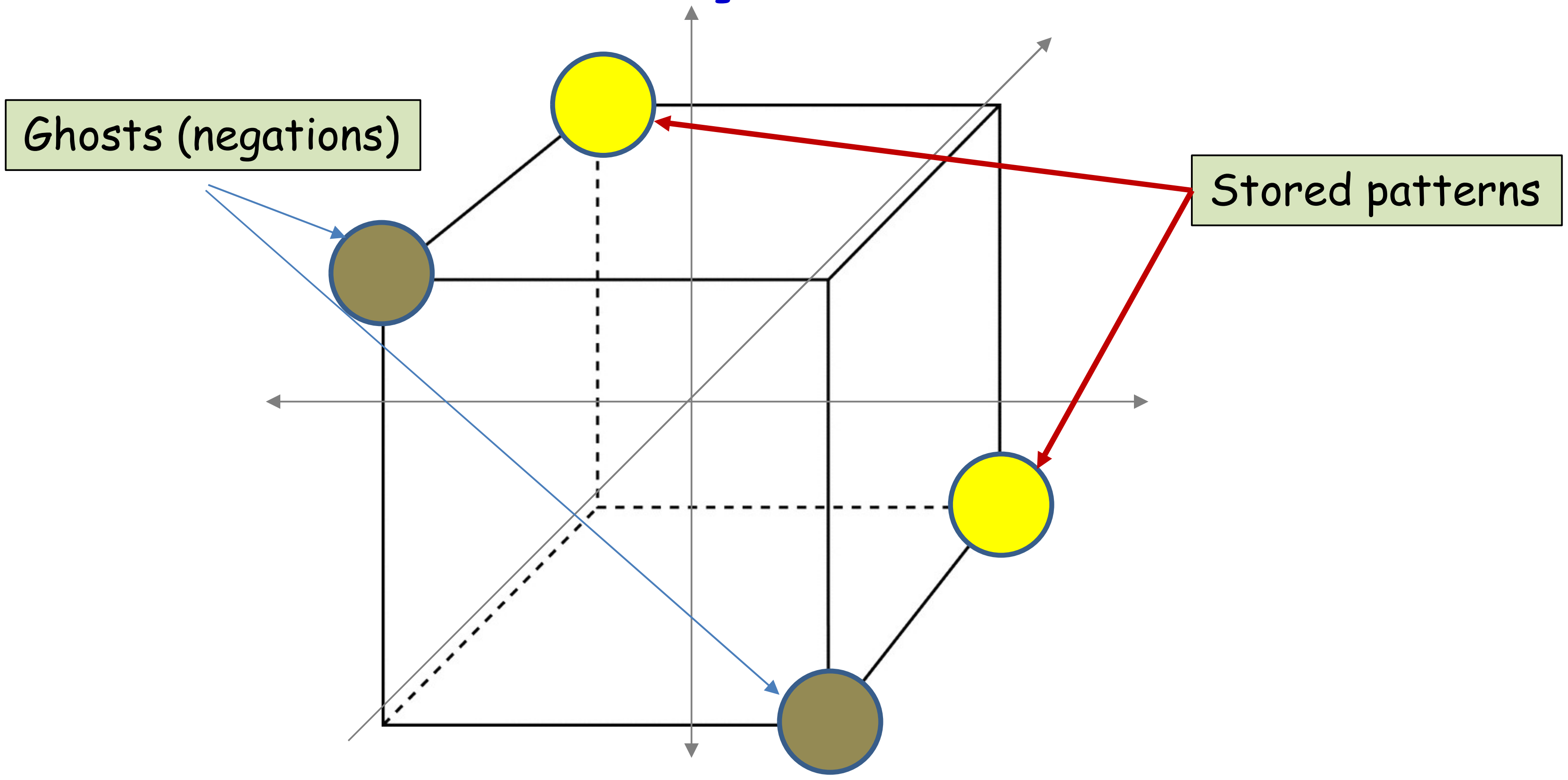
The energy function

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



- The stored values of $\mathbf{y}$ are the ones where all adjacent corners are lower on the quadratic

Patterns you can store



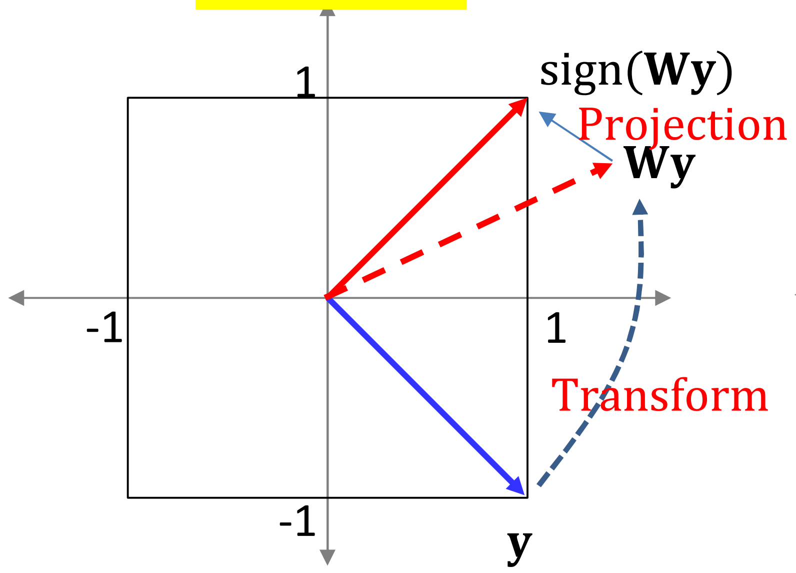
- All patterns are on the corners of a hypercube
  - If a pattern is stored, it's "ghost" is stored as well
  - Intuitively, patterns must ideally be maximally far apart
    - Though this doesn't seem to hold for Hebbian learning

Evolution of the network

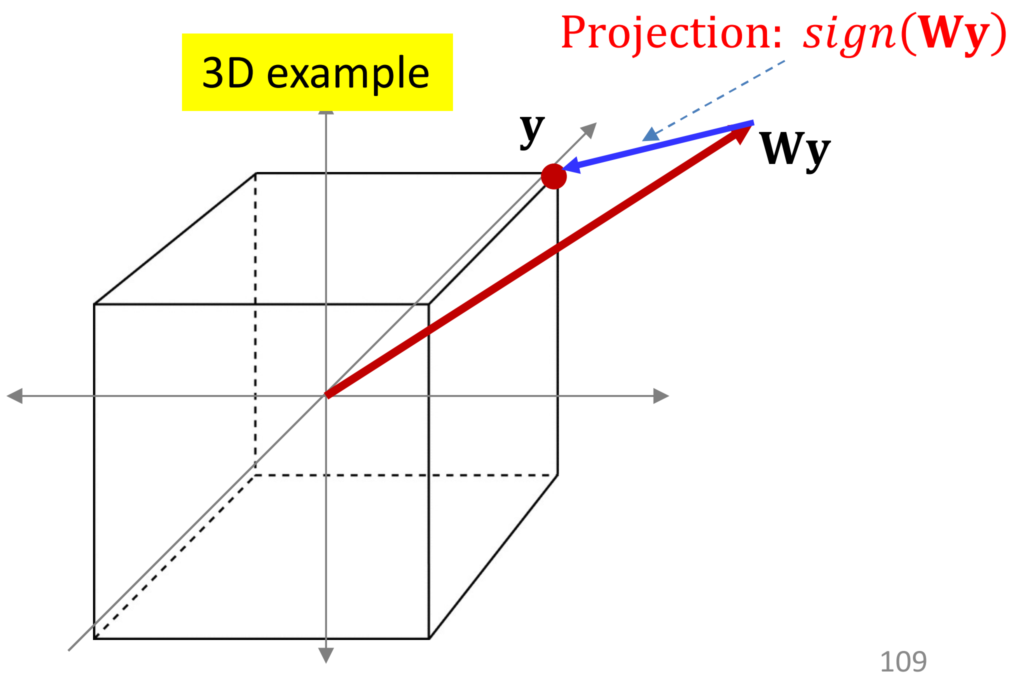
- Note: for real vectors $sign(\mathbf{y})$is a projection
  - Projects $\mathbf{y}$ onto the nearest corner of the hypercube
  - It "quantizes" the space into orthants

- Response to field: $\mathbf{y} \leftarrow sign(\mathbf{W}\mathbf{y})$
  - Each step rotates the vector $\mathbf{y}$ and then projects it onto the nearest corner

2D example



3D example



Storing patterns

- A pattern $\mathbf{y}_P$ is stored if:
  - $sign(\mathbf{W}\mathbf{y}_p) = \mathbf{y}_p$ for all target patterns
- Training: Design $\mathbf{W}$ such that this holds
- Simple solution: $\mathbf{y}_p$ is an Eigenvector of $\mathbf{W}$
  - And the corresponding Eigenvalue is positive

$$\mathbf{W}\mathbf{y}_p = \lambda\mathbf{y}_p$$

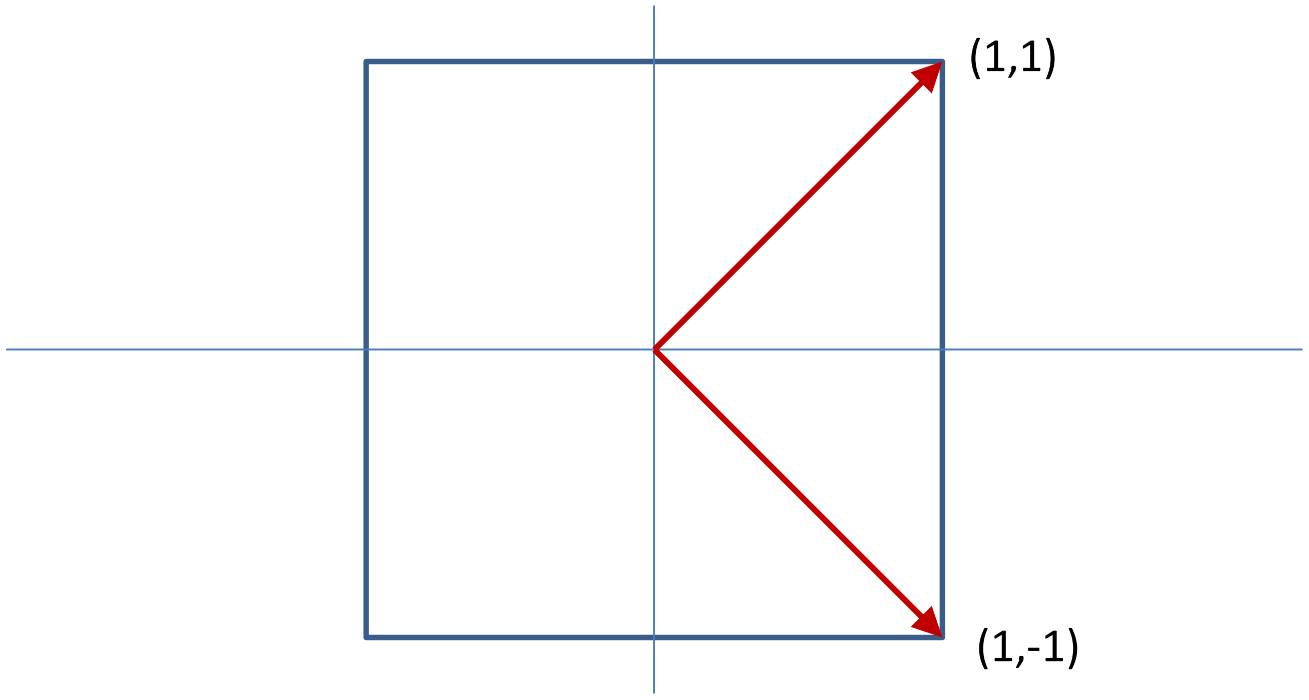
  - More generally orthant($\mathbf{W}\mathbf{y}_p$) = orthant($\mathbf{y}_p$)
- How many such $\mathbf{y}_p$ can we have?

Random fact that should interest you

- Number of ways of selecting two N-bit binary patterns $\boldsymbol{y}_1$and $\boldsymbol{y}_2$ such that they differ from one another in exactly $N/2$ bits is $\mathcal{O}\left(2^{\frac{3N}{2}}\right)$

- The size of the largest set of N-bit binary patterns $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots\}$ that *all* differ from one another in exactly $N/2$ bits is at most N
  - Trivial proof.. ☺

Only N patterns?



- Symmetric weight matrices have orthogonal Eigen vectors
- You can have max N orthogonal vectors in an N-dimensional space

random fact that should interest you

- The Eigenvectors of any symmetric matrix $\mathbf{W}$ are orthogonal
- The Eigen*values* may be positive or negative

Storing more than one pattern

- Requirement: Given $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_P$
  - 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

- What is the largest number of patterns that can be stored?

Storing K orthogonal patterns

- Simple solution: Design $\mathbf{W}$ such that $\mathbf{y}_1$, $\mathbf{y}_2, \dots, \mathbf{y}_K$ are the Eigen vectors of $\mathbf{W}$
  - Let $\mathbf{Y} = [\mathbf{y}_1\ \mathbf{y}_2 \dots \mathbf{y}_K]$

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

  - $\lambda_1, \dots, \lambda_K$ are positive
  - For $\lambda_1 = \lambda_2 = \lambda_K = 1$ this is exactly the Hebbian rule
- The patterns are provably stationary

Hebbian rule

- In reality
  - Let $\mathbf{Y} = [\mathbf{y}_1\ \mathbf{y}_2 \ldots \mathbf{y}_K\ \mathbf{r}_{K+1}\ \mathbf{r}_{K+2} \ldots \mathbf{r}_N]$

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

  - $\mathbf{r}_{K+1}\ \mathbf{r}_{K+2} \ldots \mathbf{r}_N$ are orthogonal to $\mathbf{y}_1\ \mathbf{y}_2 \ldots \mathbf{y}_K$
  - $\lambda_1 = \lambda_2 = \lambda_K = 1$
  - $\lambda_{K+1}, \ldots, \lambda_N = 0$

Storing N orthogonal patterns

- When we have N orthogonal (or near orthogonal) patterns $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N$
  - $Y = [\mathbf{y}_1\ \mathbf{y}_2 \ldots \mathbf{y}_N]$

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

  - $\lambda_1 = \lambda_2 = \lambda_N = 1$
- The Eigen vectors of $\mathbf{W}$ span the space
- Also, for any $\mathbf{y}_k$

$$\mathbf{W}\mathbf{y}_k = \mathbf{y}_k$$

Storing N orthogonal patterns

- The N orthogonal patterns $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N$ *span the space*
- Any pattern $\mathbf{y}$ can be written as

$$\mathbf{y} = a_1\mathbf{y}_1 + a_2\mathbf{y}_2 + \cdots + a_N\mathbf{y}_N$$

$$\mathbf{Wy} = a_1\mathbf{Wy}_1 + a_2\mathbf{Wy}_2 + \cdots + a_N\mathbf{Wy}_N$$

$$= a_1\mathbf{y}_1 + a_2\mathbf{y}_2 + \cdots + a_N\mathbf{y}_N = \mathbf{y}$$

- *All patterns are stable*
  - Remembers everything
  - ***Completely useless network***

Storing *K* orthogonal patterns

- Even if we store fewer than N patterns
  - Let $Y = [\mathbf{y}_1\ \mathbf{y}_2 \dots \mathbf{y}_K\ \mathbf{r}_{K+1}\ \mathbf{r}_{K+2} \dots \mathbf{r}_N]$

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

  - $\mathbf{r}_{K+1}\ \mathbf{r}_{K+2} \dots \mathbf{r}_N$ are orthogonal to $\mathbf{y}_1\ \mathbf{y}_2 \dots \mathbf{y}_K$
  - $\lambda_1 = \lambda_2 = \lambda_K = 1$
  - $\lambda_{K+1}, \dots, \lambda_N = 0$
- Any pattern that is *entirely* in the subspace spanned by $\mathbf{y}_1\ \mathbf{y}_2 \dots \mathbf{y}_K$ is also stable (same logic as earlier)
- Only patterns that are *partially* in the subspace spanned by $\mathbf{y}_1\ \mathbf{y}_2 \dots \mathbf{y}_K$ are unstable
  - Get projected onto subspace spanned by $\mathbf{y}_1\ \mathbf{y}_2 \dots \mathbf{y}_K$

Problem with Hebbian Rule

- Even if we store fewer than N patterns
  - Let $Y = [\mathbf{y}_1\ \mathbf{y}_2 \ldots \mathbf{y}_K\ \mathbf{r}_{K+1}\ \mathbf{r}_{K+2} \ldots \mathbf{r}_N]$

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

  - $\mathbf{r}_{K+1}\ \mathbf{r}_{K+2} \ldots \mathbf{r}_N$ are orthogonal to $\mathbf{y}_1\ \mathbf{y}_2 \ldots \mathbf{y}_K$
  - $\lambda_1 = \lambda_2 = \lambda_K = 1$
- Problems arise because Eigen values are all 1.0
  - Ensures stationarity of vectors in the subspace
  - All stored patterns are equally important
  - What if we get rid of this requirement?

Hebbian rule and general (non-orthogonal) vectors

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

- What happens when the patterns are *not* orthogonal
- What happens when the patterns are presented *more* than once
  - Different patterns presented different numbers of times
  - Equivalent to having unequal Eigen values..
- Can we predict the evolution of any vector $\mathbf{y}$
  - Hint: For real valued vectors, use Lanczos iterations
    - Can write $\mathbf{Y}_P = \mathbf{U}_P \Lambda \mathbf{V}_p^T, \rightarrow \mathbf{W} = \mathbf{U}_P \Lambda^2 \mathbf{U}_p^T$
  - Tougher for binary vectors (NP)

The bottom line

- With a network of N units (i.e. N-bit patterns)
- The maximum number of stationary patterns is actually *exponential* in N
  - McElice and Posner, 84'
  - E.g. when we had the Hebbian net with N orthogonal base patterns, *all* patterns are stationary

- For a *specific* set of K patterns, we can *always* build a network for which all K patterns are stable provided $K \leq N$
  - Mostafa and St. Jacques 85'
    - For large N, the upper bound on K is actually N/4logN
      - McElice et. Al. 87'
  - **But this may come with many "parasitic" memories**

The bottom line

- With an network of N units (i.e. N-bit patterns)
- The maximum number of stable patterns is actually *exponential* in N
  - McElice and Posner, 84'
  - E.g. when we had the H   se patterns, *all* patterns are stable

How do we find this network?

- For a *specific* set of K patterns, we can *always* build a network for which all K patterns are stable provided $K \leq N$
  - Mostafa and St. Jacques 85'
    - For large N, the upper bound on K is actually N/4logN
      - McElice et. Al. 87'
  - **But this may come with many "parasitic" memories**

The bottom line

- With an network of N units (i.e. N-bit patterns)
- The maximum number of stable patterns is actually *exponential* in N
  - McElice and Posner, 84'
  - E.g. when we had the H ... se patterns, *all* patterns are stable

How do we find this network?

- For a *specific* set of K patterns, we can *always* build a network for which all K patterns are stable provided $K \leq N$
  - Mostafa and St. Jacques 85'
    - For large N, the upper bound on K is actually N
      - McElice et. Al. 87'
  - **But this may come with many "parasitic" memories**

Can we do something about this?

Story so far

- Hopfield nets with N neurons can store up to 0.14N random patterns through Hebbian learning with 0.996 probability of recall
  - The recalled patterns are the Eigen vectors of the weights matrix with the highest Eigen values
- Hebbian learning assumes all patterns to be stored are equally important
  - For orthogonal patterns, the patterns are the Eigen vectors of the constructed weights matrix
  - All Eigen values are identical
- In theory the number of stationary states in a Hopfield network can be exponential in N
- The number of *intentionally* stored patterns (stationary *and* stable) can be as large as N
  - But comes with many parasitic memories