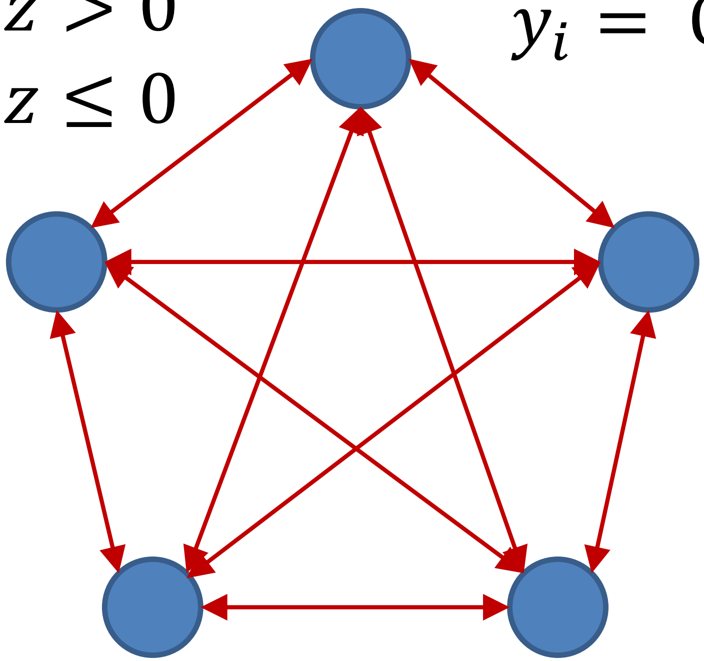
Neural Networks

Hopfield Nets and Boltzmann Machines

Fall 2017

Recap: Hopfield network

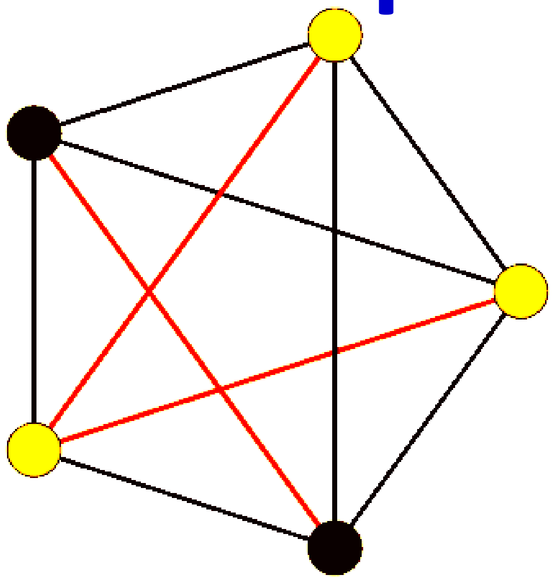
$$\Theta(z) = \begin{cases} +1 \; if \; z > 0 \\ -1 \; if \; z \leq 0 \end{cases}$$



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

- ***Symmetric loopy network***
- 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

Recap: Hopfield network

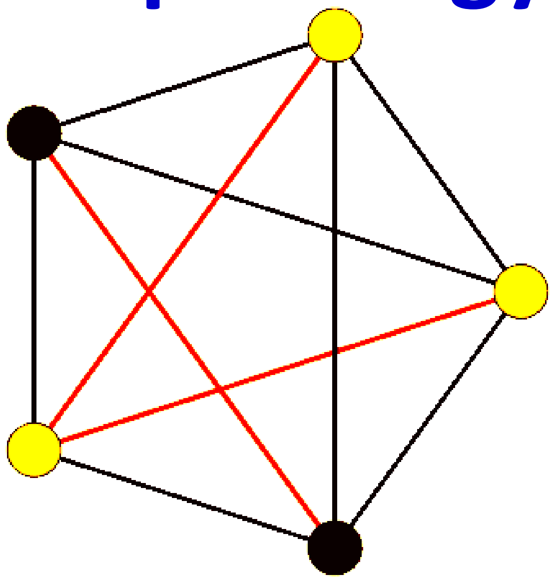


$$y_i = \Theta\left(\sum_{j\neq i} w_{ji}y_j + b_i\right)$$

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

- 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

Recap: Energy of a Hopfield Network



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

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

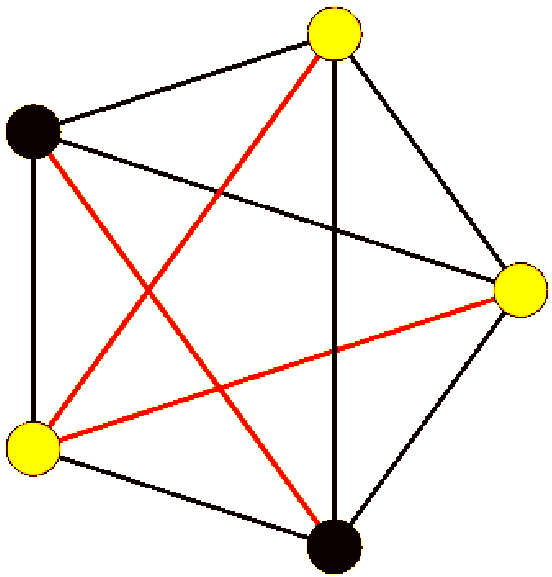
Not assuming node bias

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

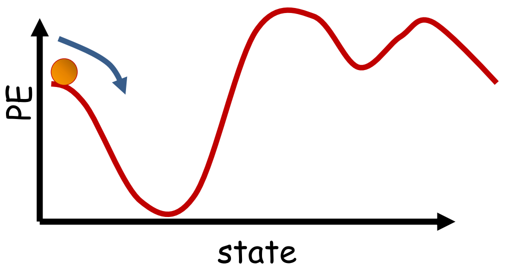
- The system will evolve until the energy hits a local minimum
- In vector form, including a bias term (not used in Hopfield nets)

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

Recap: Evolution

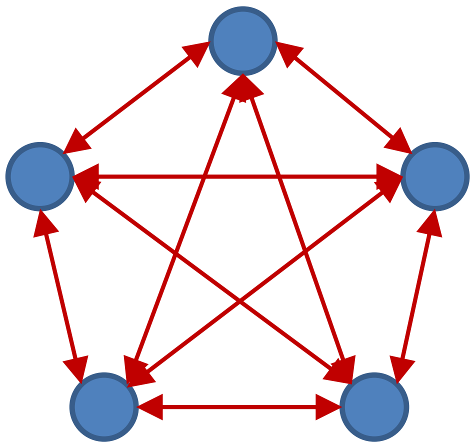


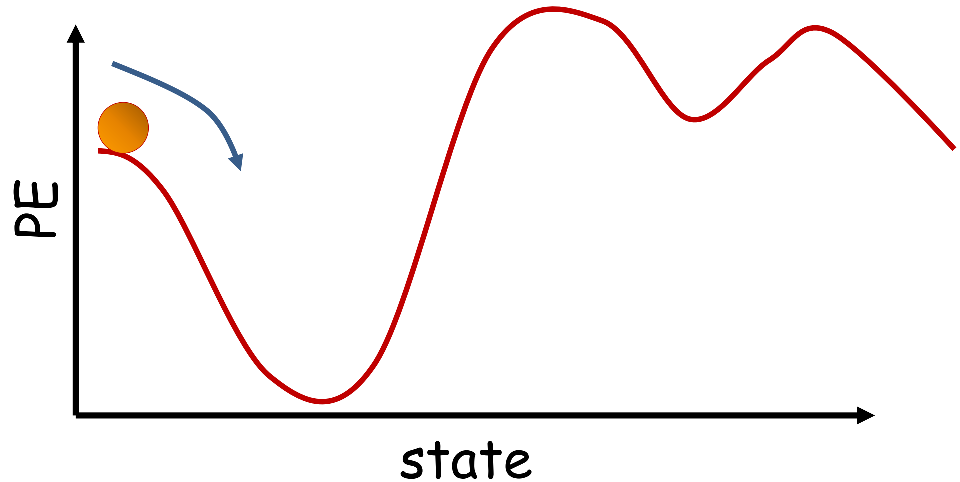
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$



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

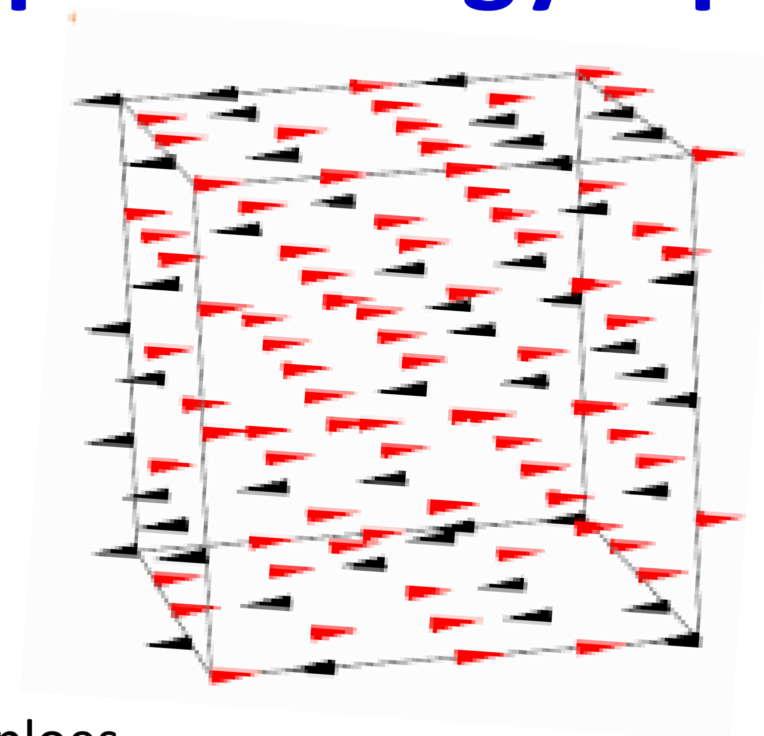
Recap: Content-addressable memory





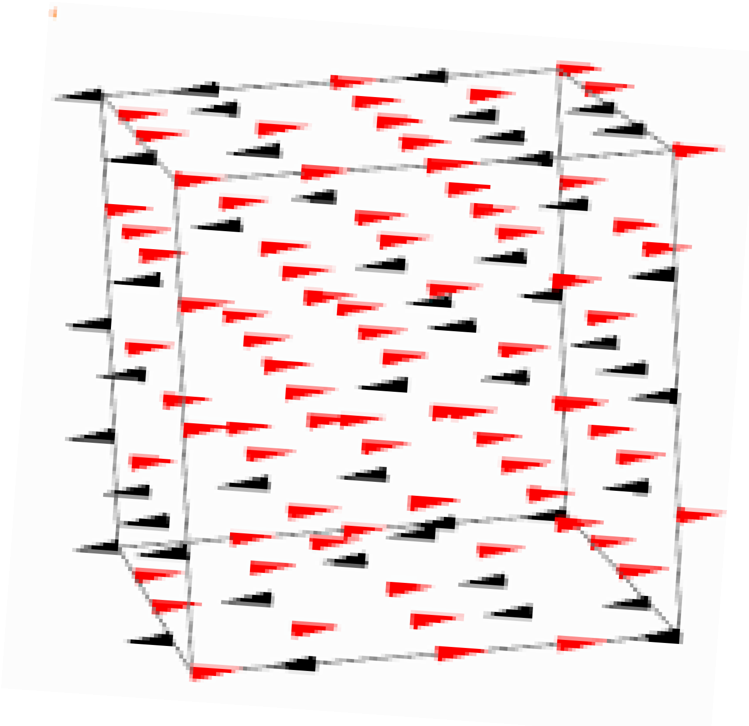
- Each of the minima is a "stored" pattern
  – If the network is initialized close to a stored pattern, it will inevitably evolve to the pattern
- **This is a *content addressable memory***
  – Recall memory content from partial or corrupt values
- Also called ***associative memory***

Recap – 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…

Recap – 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

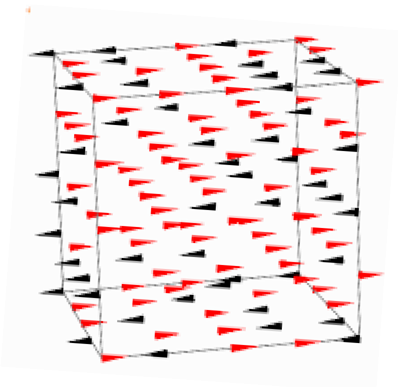
$$x_i = \begin{cases} x_i \ if \ sign\left(x_i \ f(p_i)\right) = 1 \\ -x_i \ otherwise \end{cases}$$

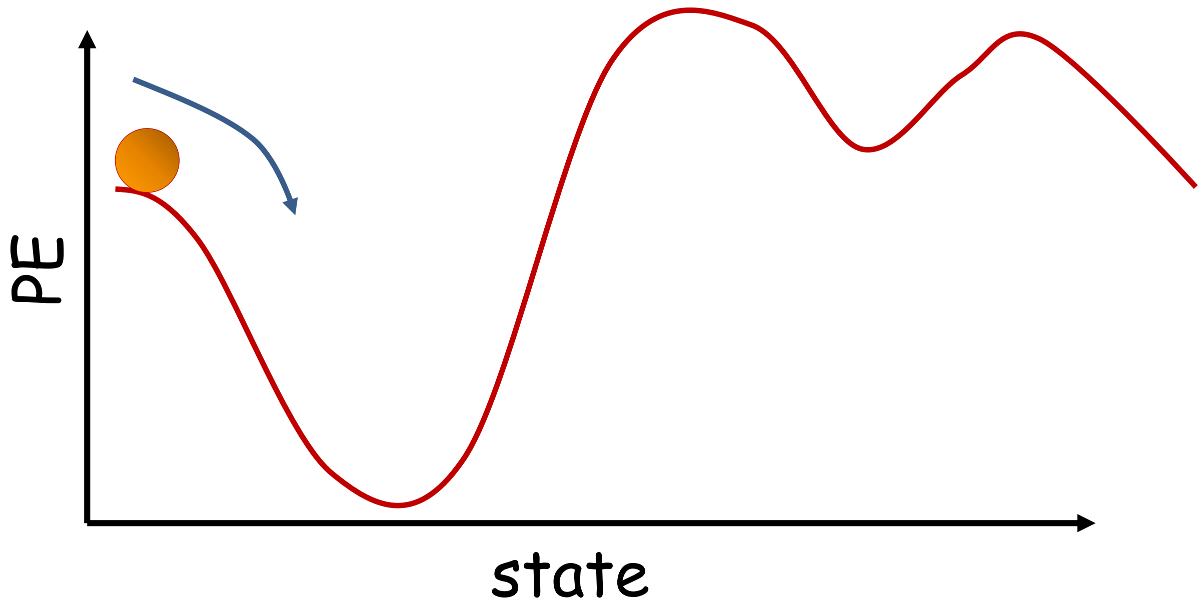
- The total potential energy of the system

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

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

Recap : 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

Recap: Hopfield net computation

1. Initialize network with initial pattern

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

2. Iterate until convergence

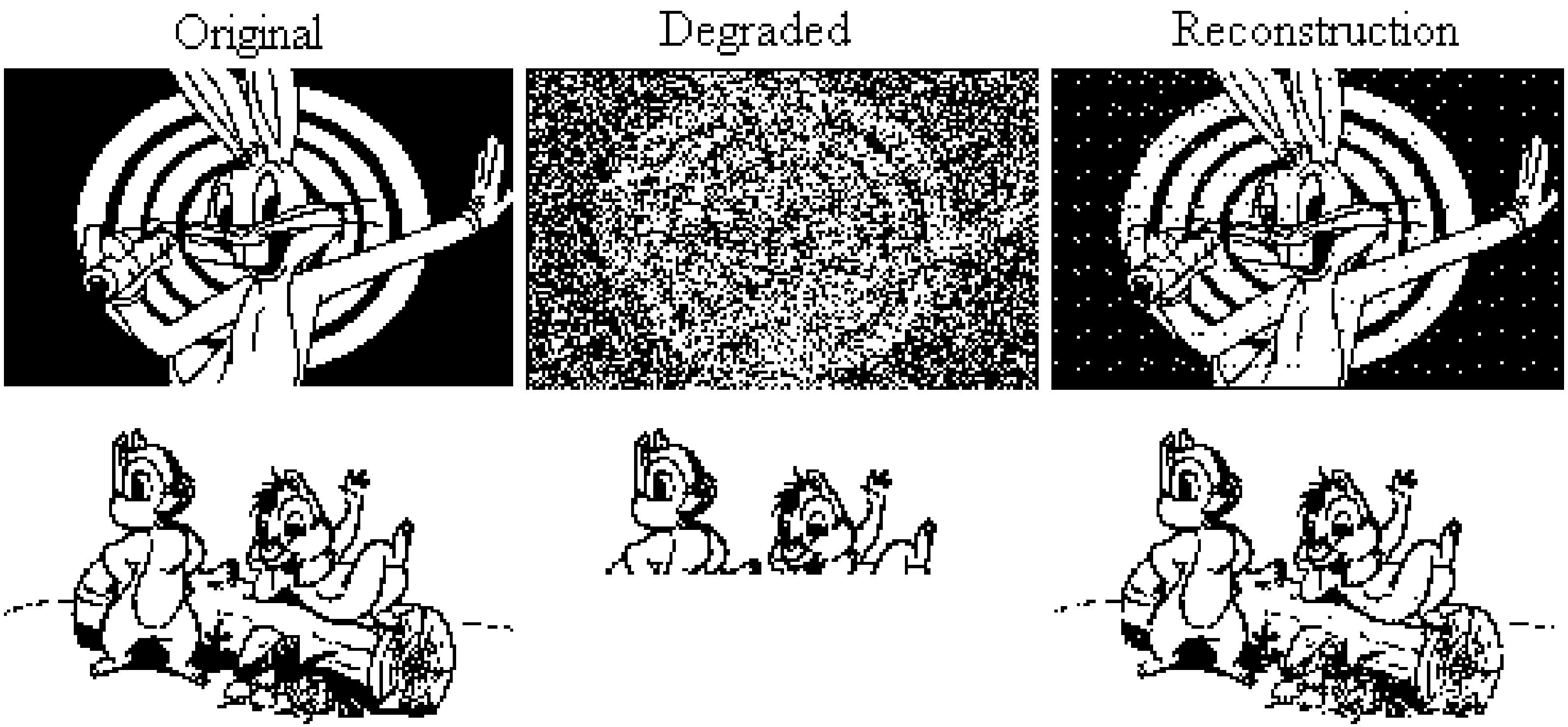
$$y_i(t+1) = \Theta\left(\sum_{j \ne i} w_{ji} y_j\right), \qquad 0 \le i \le N-1$$

- 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

Examples: Content addressable memory



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

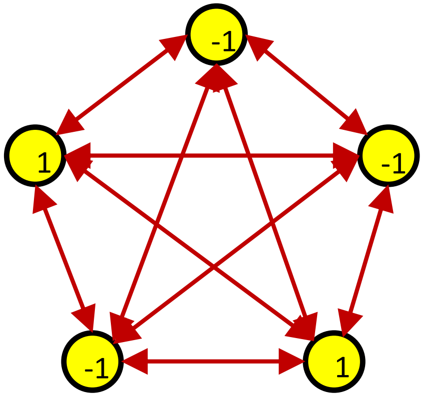
- http://staff.itee.uq.edu.au/janetw/cmc/chapters/Hopfield/

“Training” the network

- How do we make the network store *a specific* pattern or set of patterns?
  - Hebbian learning
  - Geometric approach
  - Optimization

- Secondary question
  - How many patterns can we store?

Recap: Hebbian Learning to Store a Specific Pattern



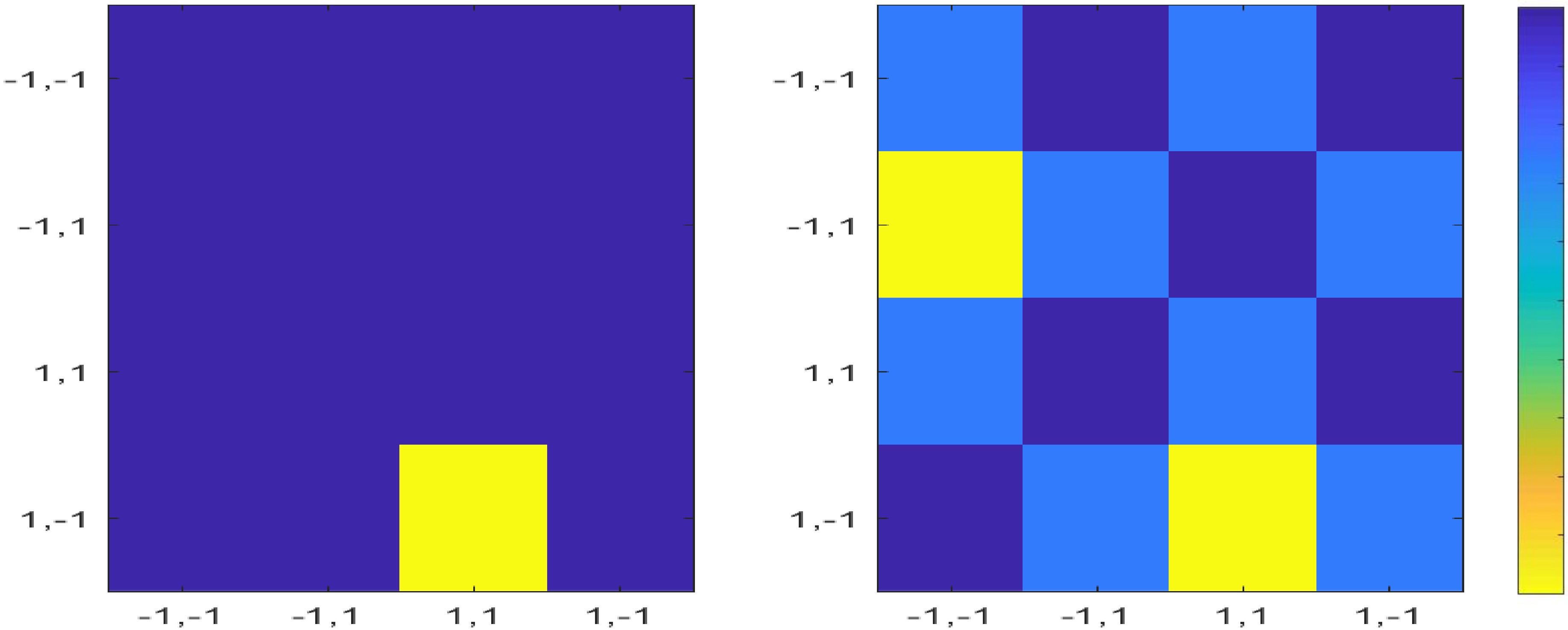
HEBBIAN LEARNING:

$$w_{ji} = y_j y_i$$

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

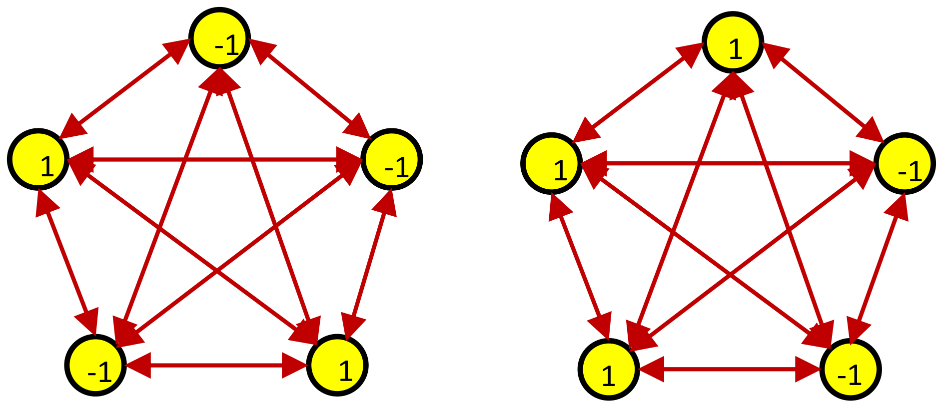
- For a single stored pattern, Hebbian learning results in a network for which the target pattern is a global minimum

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

Recap: Hebbian Learning to Store Multiple Patterns

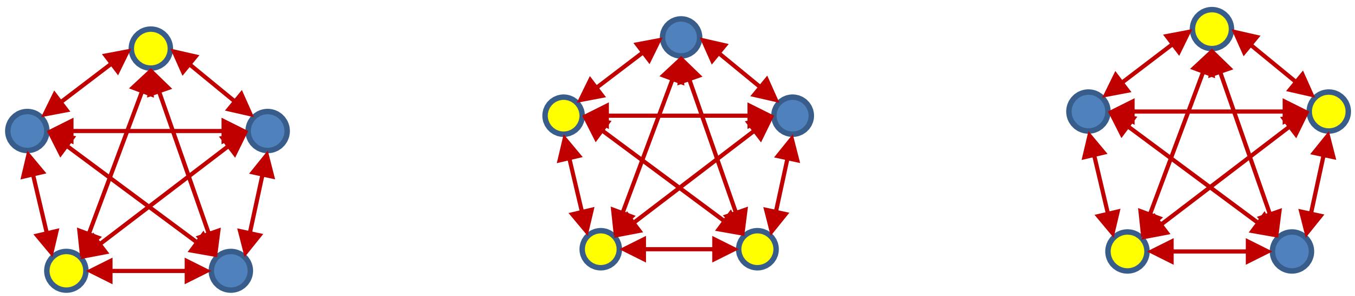


$$w_{ji} = \sum_{p \in \{p\}} y_i^p y_j^p$$

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

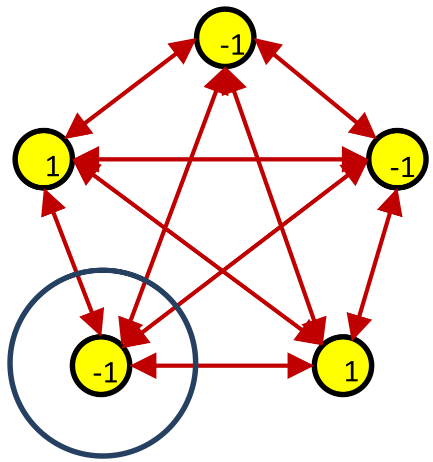
- {p} is the set of patterns to store
  - Superscript p represents the specific pattern
- N_p is the number of patterns to store

How many patterns can we store?



- Hopfield: For a network of N neurons can store up to $0.14N$ patterns

Recap: Hebbian Learning to Store a Specific Pattern



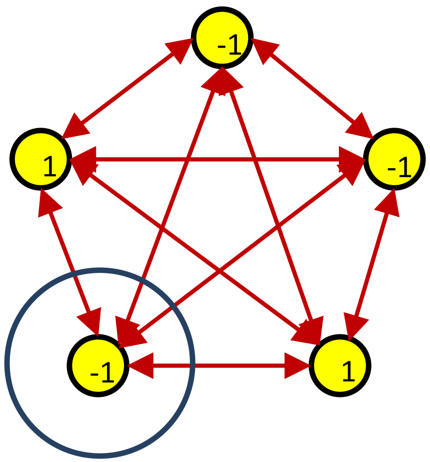
$$w_{ji} = \sum_{p \in \{p\}} y_i^p y_j^p$$

- Consider that the network is in any stored state $y^{p'}$
- At any node k the field we obtain is

$$h_k^{p'} = \sum_j y_k^{p'} y_j^{p'} y_j^{p'} + \sum_{p \neq p'} \sum_j y_k^p y_j^p y_j^{p'} = (N-1) y_k^{p'} + \sum_{p \neq p'} \sum_j y_k^p y_j^p y_j^{p'}$$

- If the second "crosstalk" term sums to less than $N-1$, the symbol will not flip

Recap: Hebbian Learning to Store a Specific Pattern



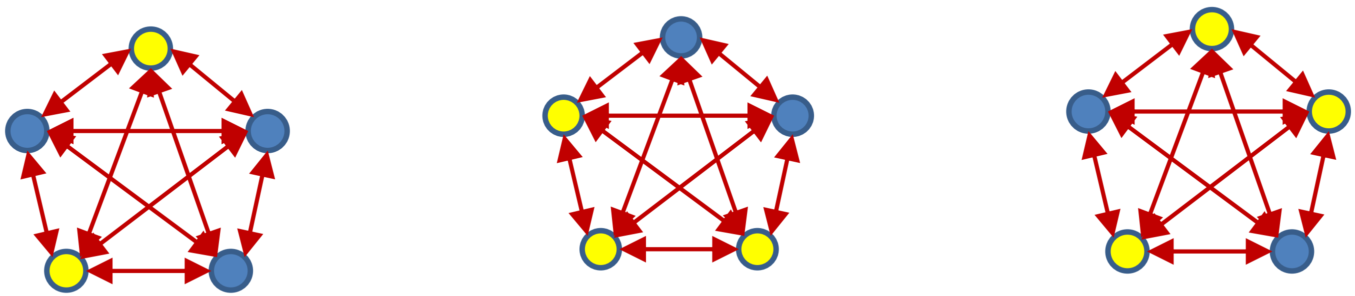
$$w_{ji} = \sum_{p \in \{p\}} y_i^p y_j^p$$

$$h_k^{p'} = \sum_j y_k^{p'} y_j^{p'} y_j^{p'} + \sum_{p \neq p'} \sum_j y_k^p y_j^p y_j^{p'} = (N-1) y_k^{p'} + \sum_{p \neq p'} \sum_j y_k^p y_j^p y_j^{p'}$$

- If $y_k^{p'} \sum_{p \neq p'} \sum_j y_k^p y_j^p y_j^{p'}$ is positive, then $\sum_{p \neq p'} \sum_j y_k^p y_j^p \, y_j^{p'}$ is the same sign as $y_k^{p'}$, and it will not flip

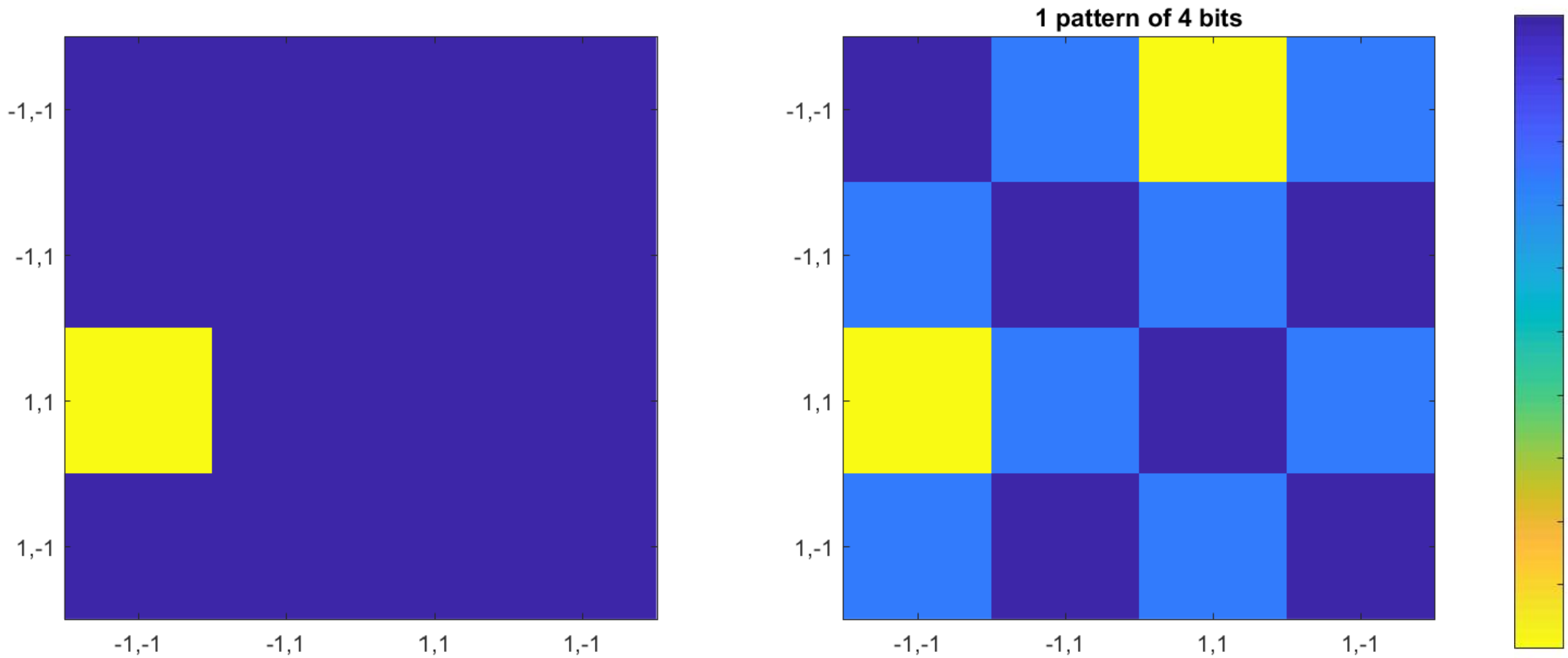
- If we choose P patterns at random, what is the probability that $y_k^{p'} \sum_{p \neq p'} \sum_j y_k^p y_j^p y_j^{p'}$ will be positive for all symbols for all P of them?

How many patterns can we store?



- Hopfield: For a network of N neurons can store up to $0.14N$ patterns
- What does this really mean?
  – Lets look at some examples

Hebbian learning: One 4-bit pattern

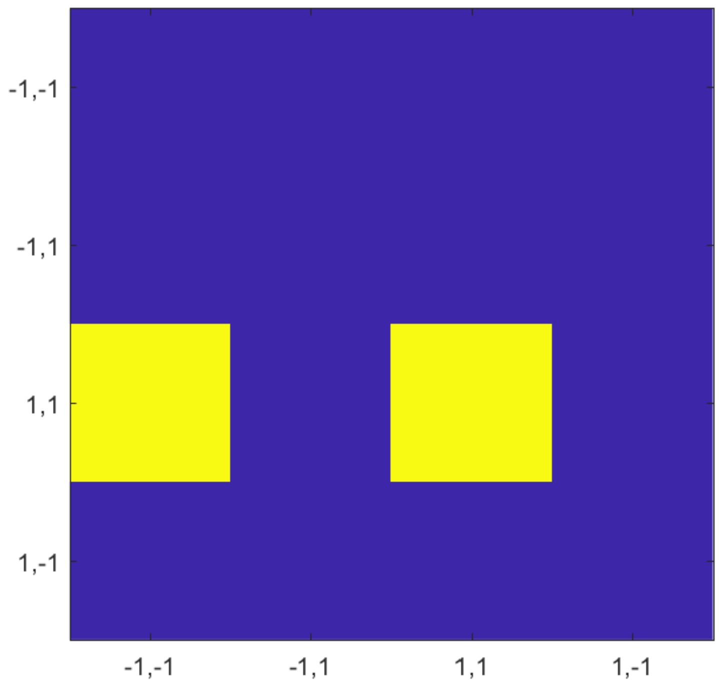


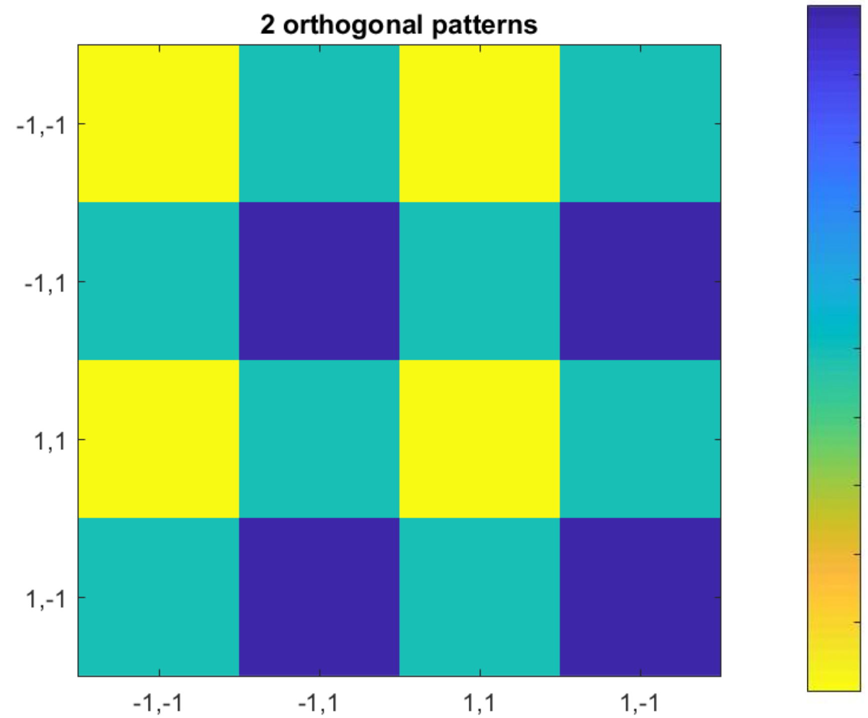
- Left: Pattern stored. Right: Energy map
- Note: Pattern is an energy well, but there are other local minima
  - Where?
  - Also note “shadow” pattern

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

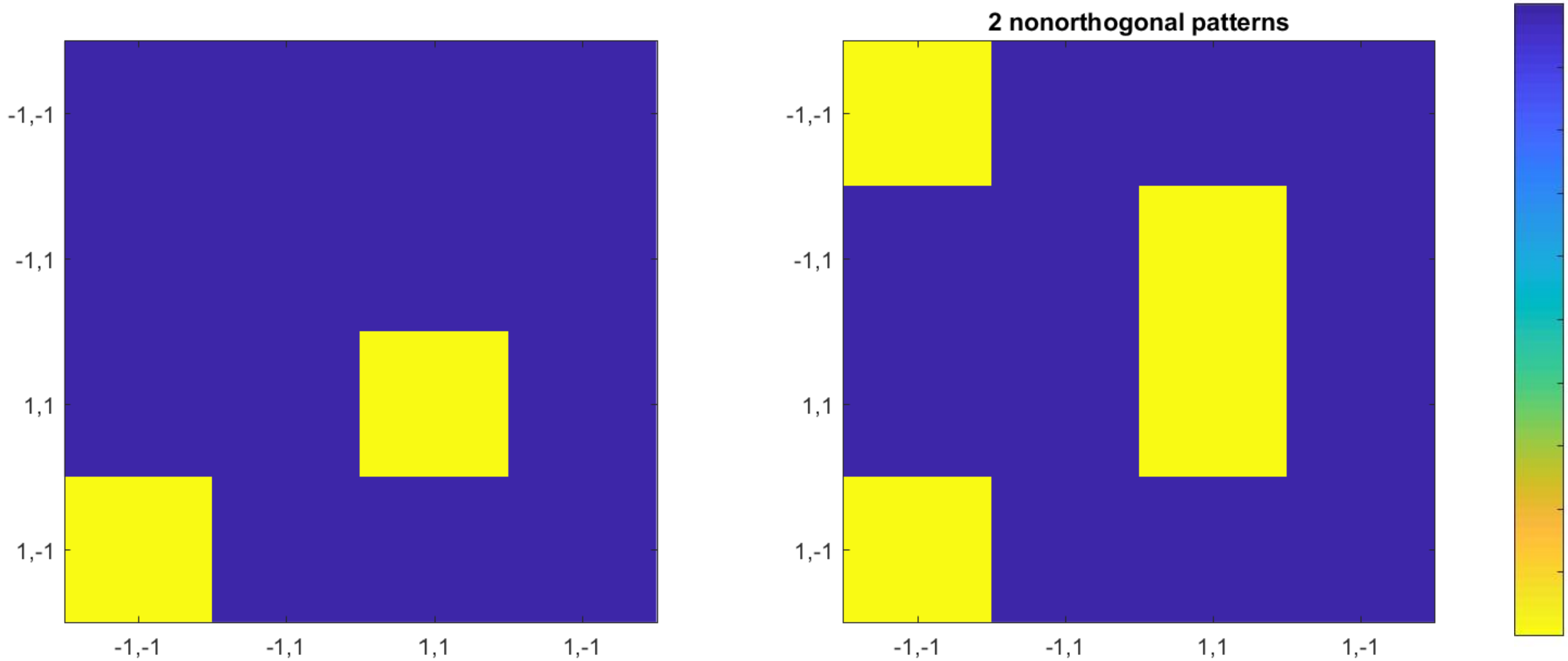
Two orthogonal 4-bit patterns





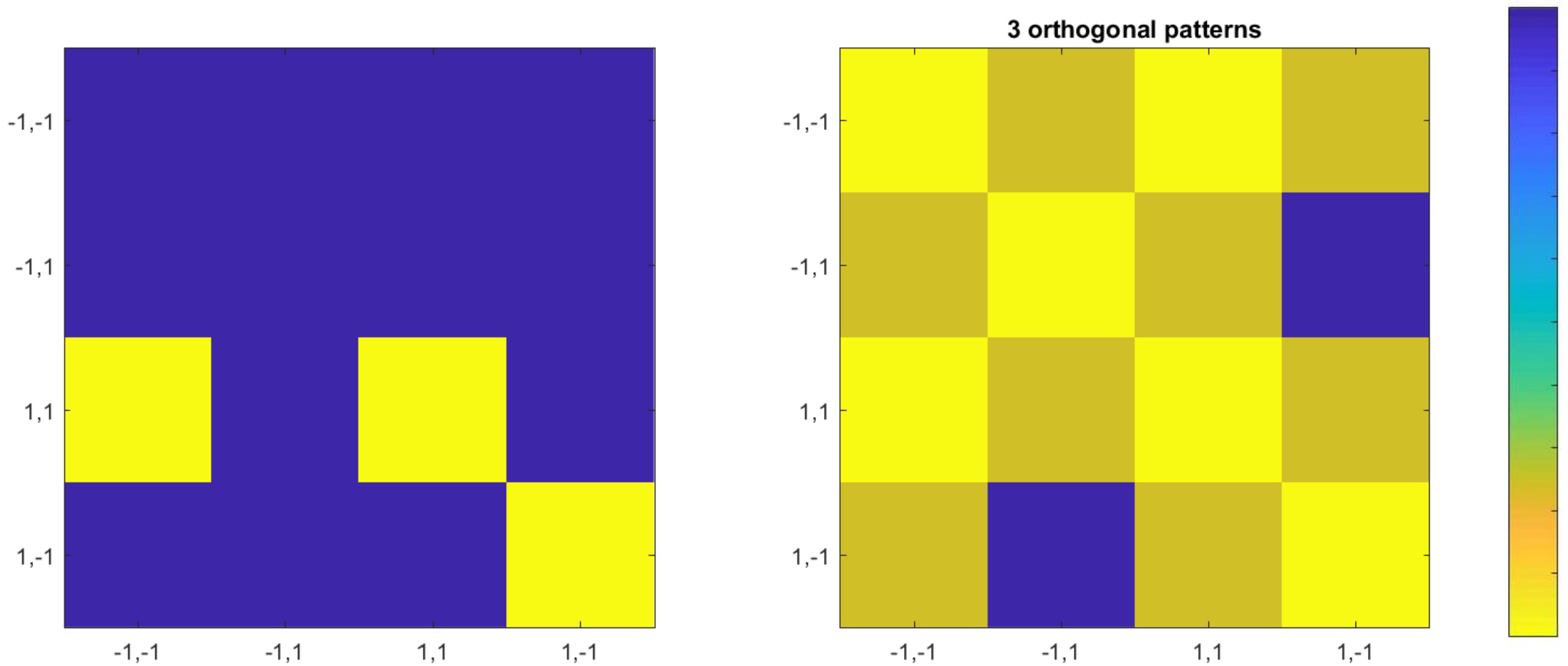
- Patterns are local minima (stationary and stable)
  - No other local minima exist
  - But patterns perfectly confusable for recall

Two *non*-orthogonal 4-bit patterns



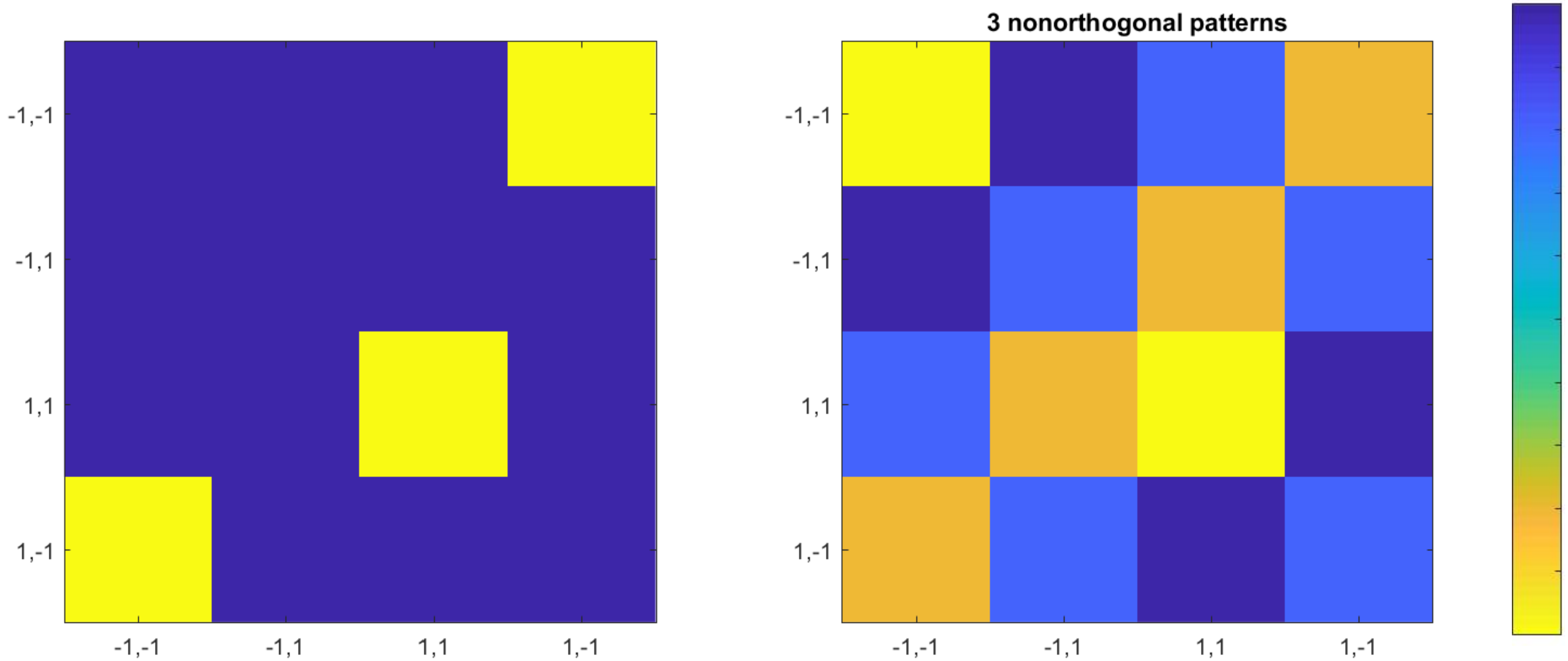
- Patterns are local minima (stationary and stable)
  - No other local minima exist
  - Actual *wells* for patterns
    - Patterns may be perfectly recalled!
  - Note K > 0.14 N

Three orthogonal 4-bit patterns



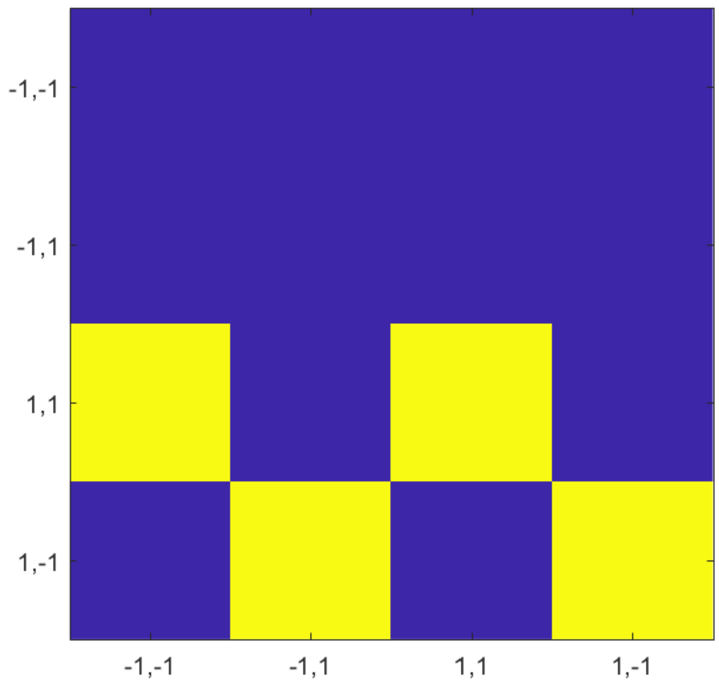
- All patterns are local minima (stationary and stable)
  - But recall from perturbed patterns is random

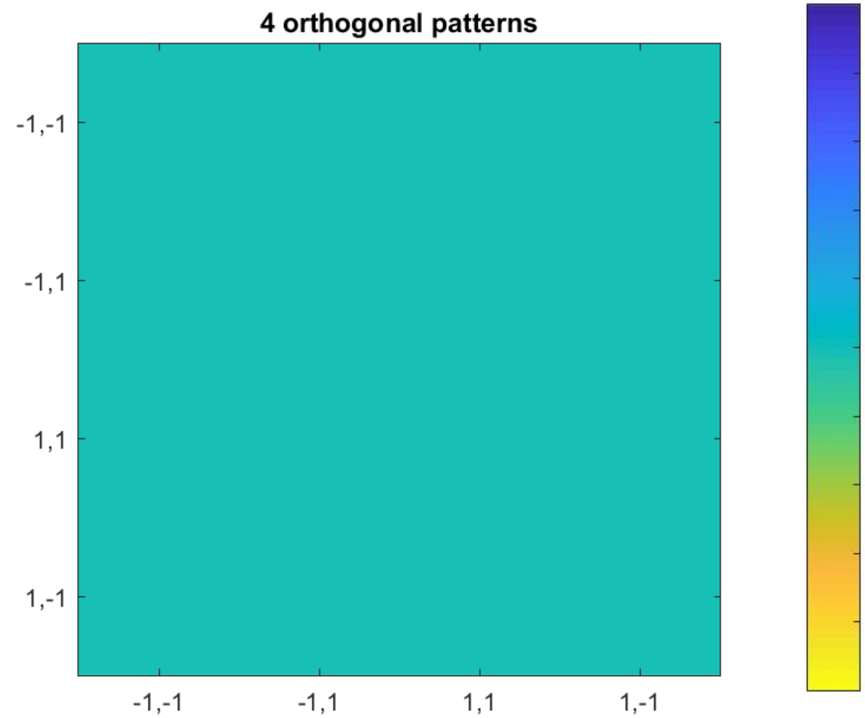
Three *non*-orthogonal 4-bit patterns



- All patterns are local minima and recalled
  - Note K > 0.14 N
  - Note some "ghosts" ended up in the "well" of other patterns
    - So one of the patterns has stronger recall than the other two

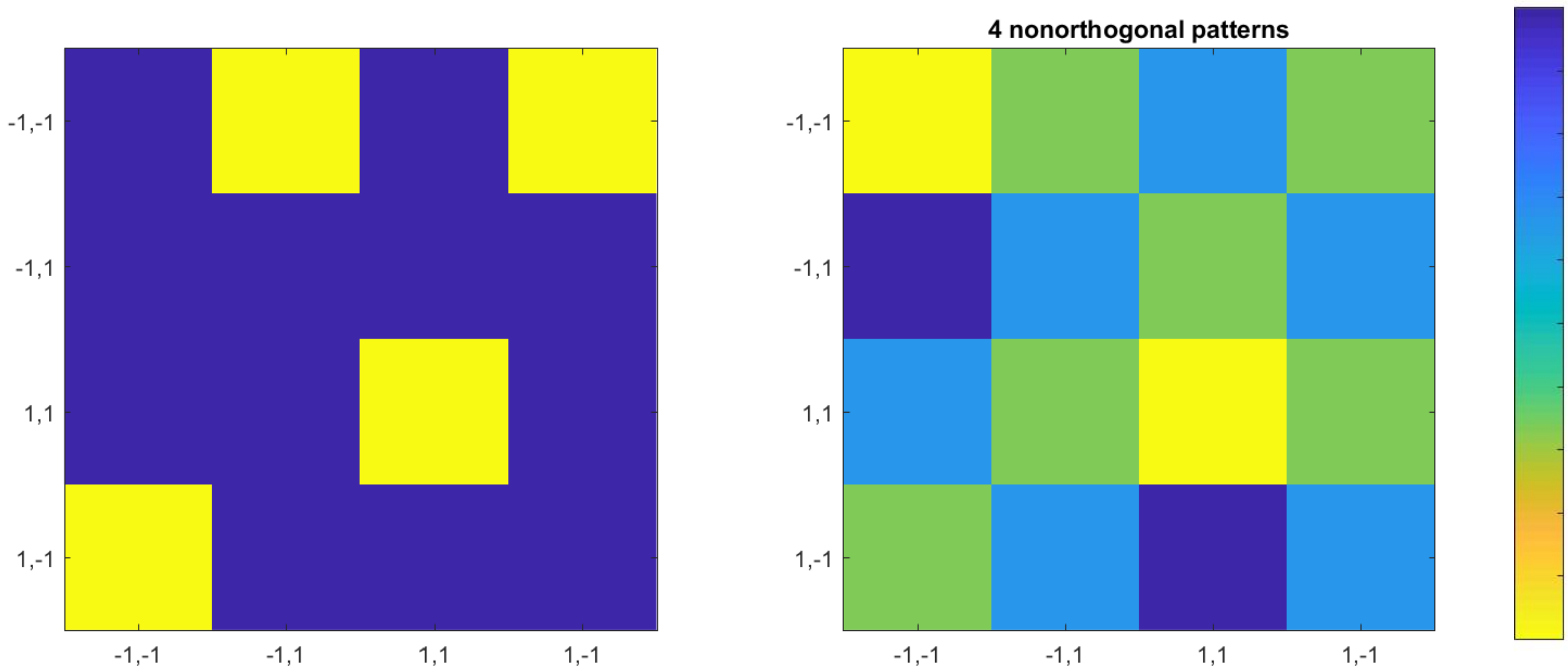
Four orthogonal 4-bit patterns





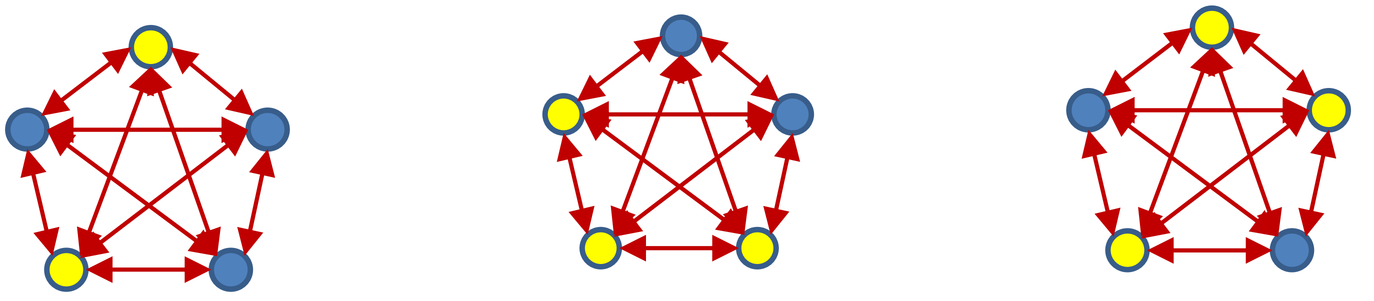
- All patterns are stationary, but none are stable
  - Total wipe out

*Four non*orthogonal 4-bit patterns



- Believe it or not, *all patterns are stored* for K = N!
  - Only "collisions" when the ghost of one pattern occurs next to another
    - [1 1 1 1] and its ghost are strong attractors (why)

How many patterns can we store?

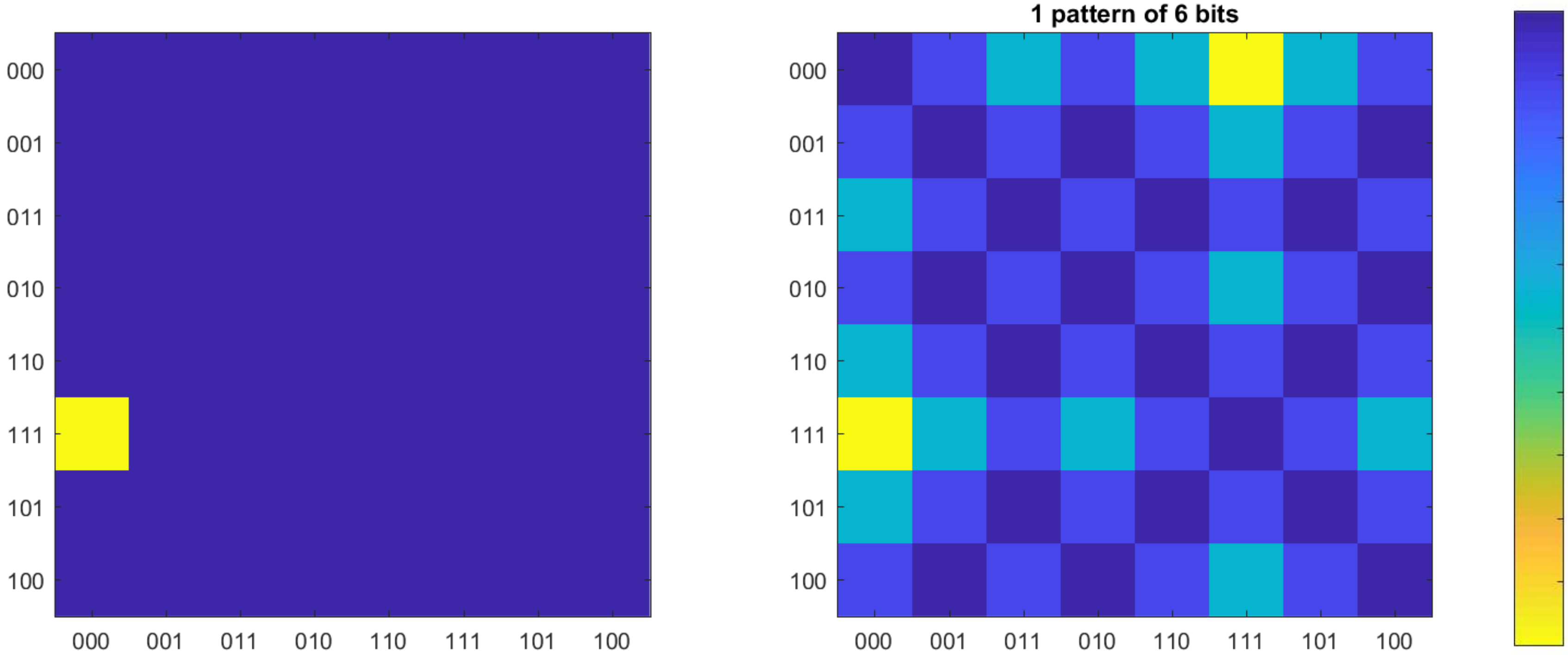


- Hopfield: For a network of N neurons can store up to $0.14N$ patterns

- Apparently a fuzzy statement
  - What does it really mean to say "stores" 0.14N patterns?
    - Stationary? Stable? No other local minima?

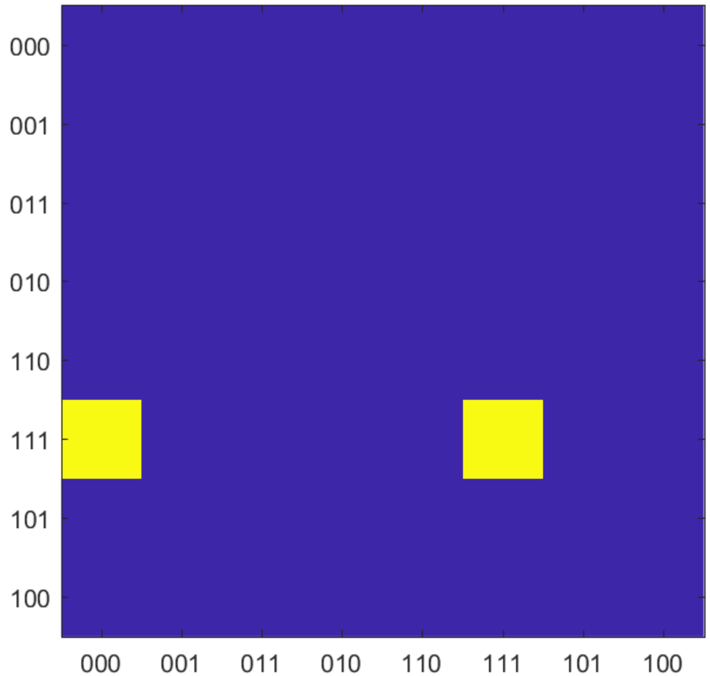
- N=4 may not be a good case (N too small)

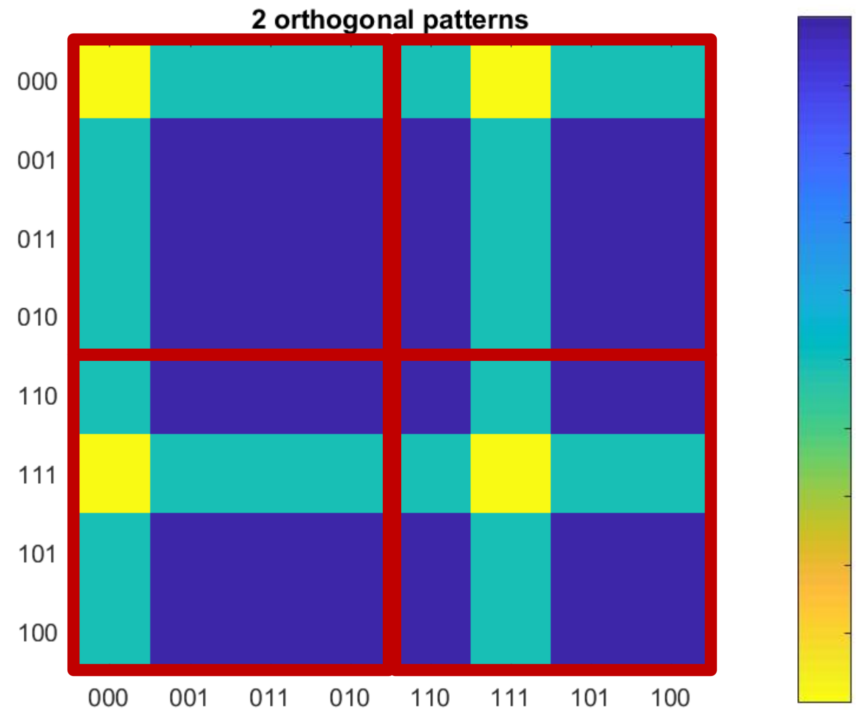
A 6-bit pattern



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

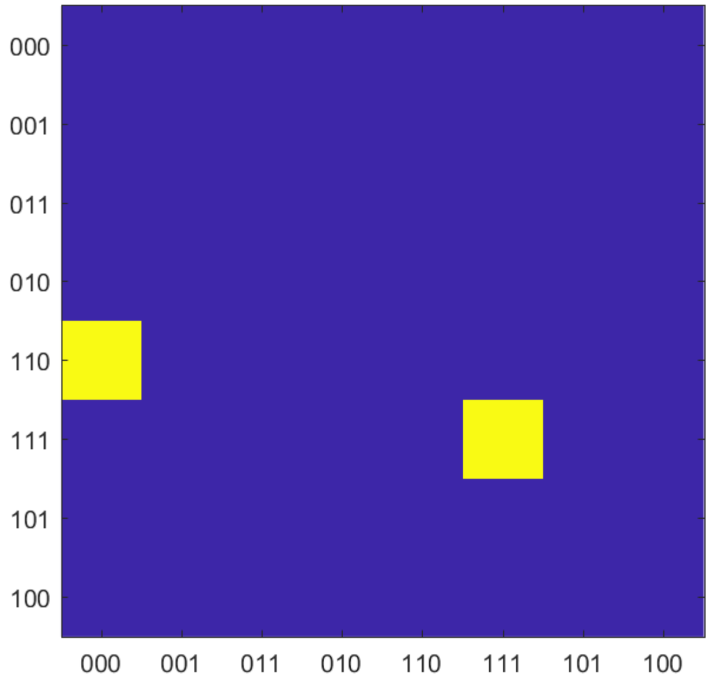
Two orthogonal 6-bit patterns

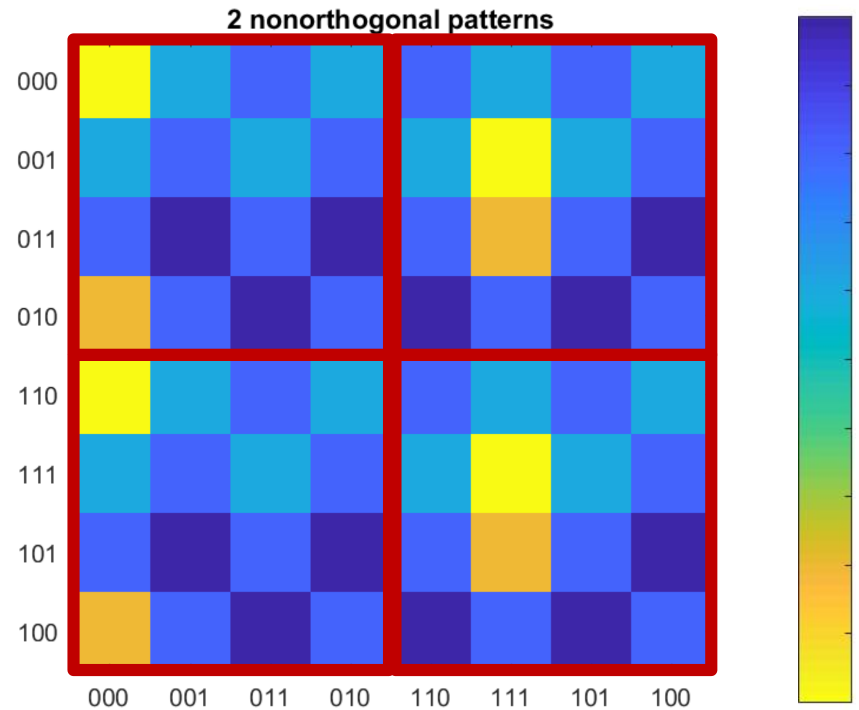




- Perfectly stationary and stable
- Several spurious “fake-memory” local minima..
  – Figure over-states the problem: actually a 3-D Kmap

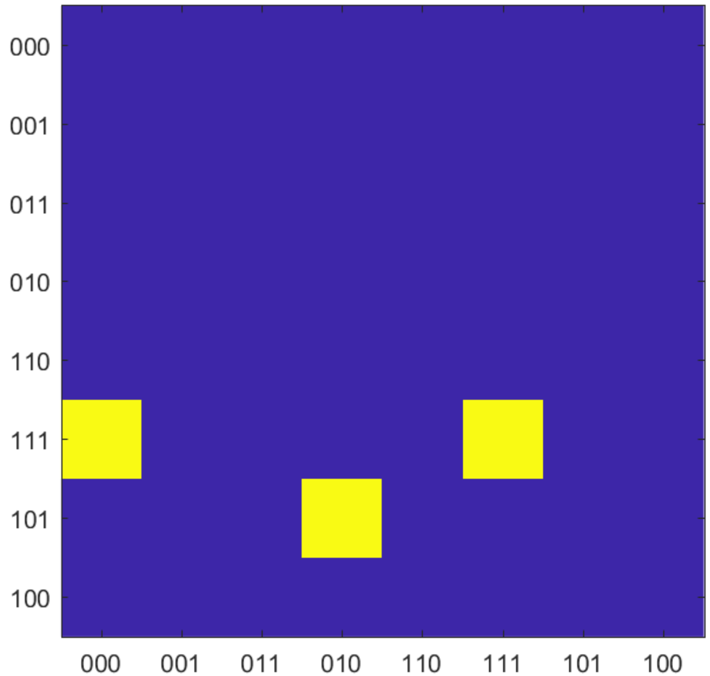
Two non-orthogonal 6-bit patterns

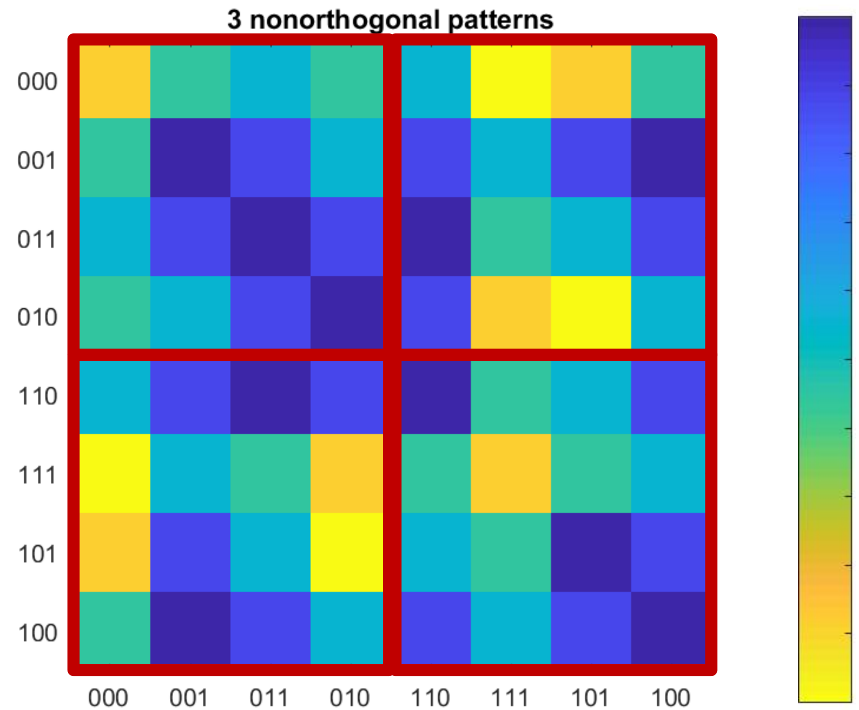




- Perfectly stationary and stable
- Some spurious “fake-memory” local minima..
  - But every stored pattern has “bowl”
  - *Fewer* spurious minima than for the orthogonal case

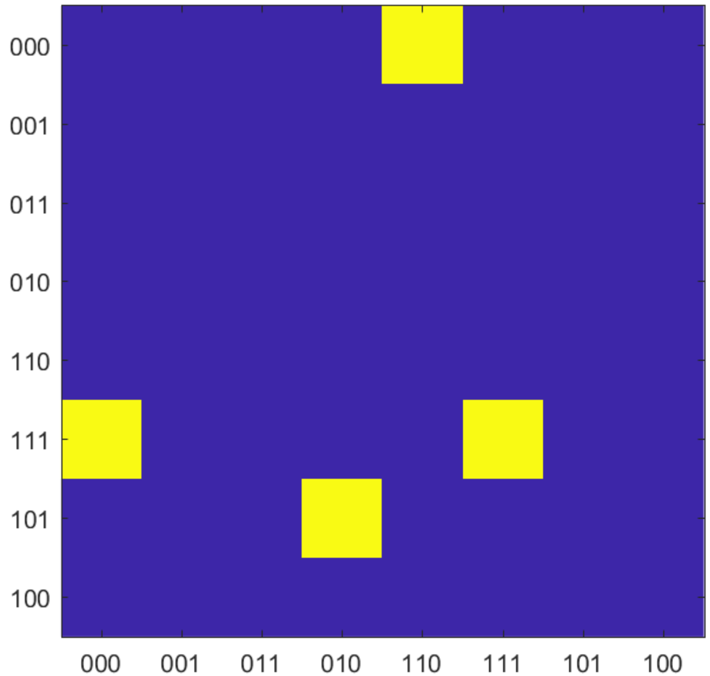
Three *non*-orthogonal 6-bit patterns

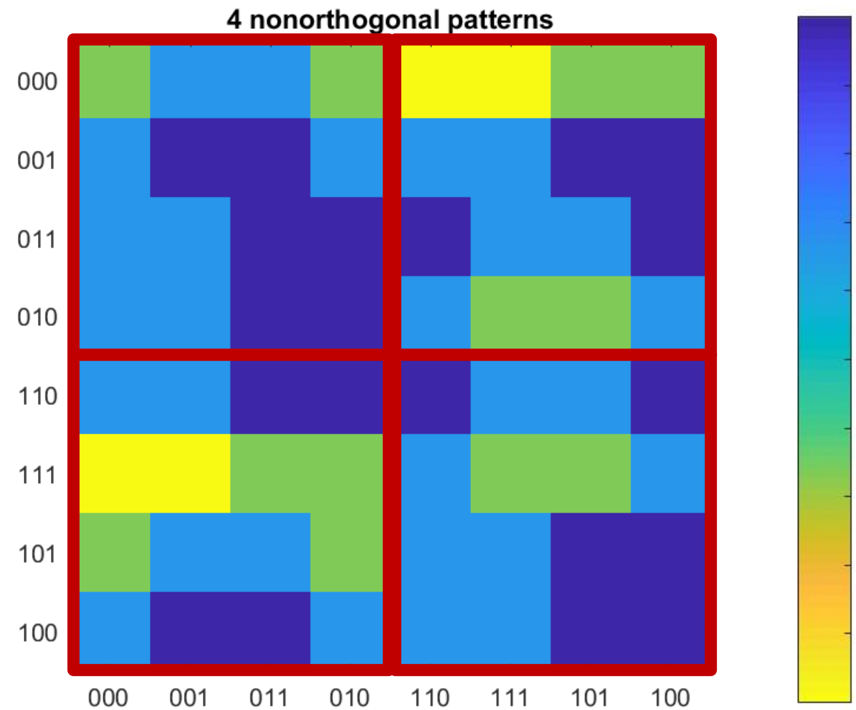




- Note: Cannot have 3 or more orthogonal 6-bit patterns..
- Patterns are perfectly stationary and stable (K > 0.14N)
- Some spurious “fake-memory” local minima..
  - But every stored pattern has “bowl”
  - *Fewer* spurious minima than for the orthogonal 2-pattern case

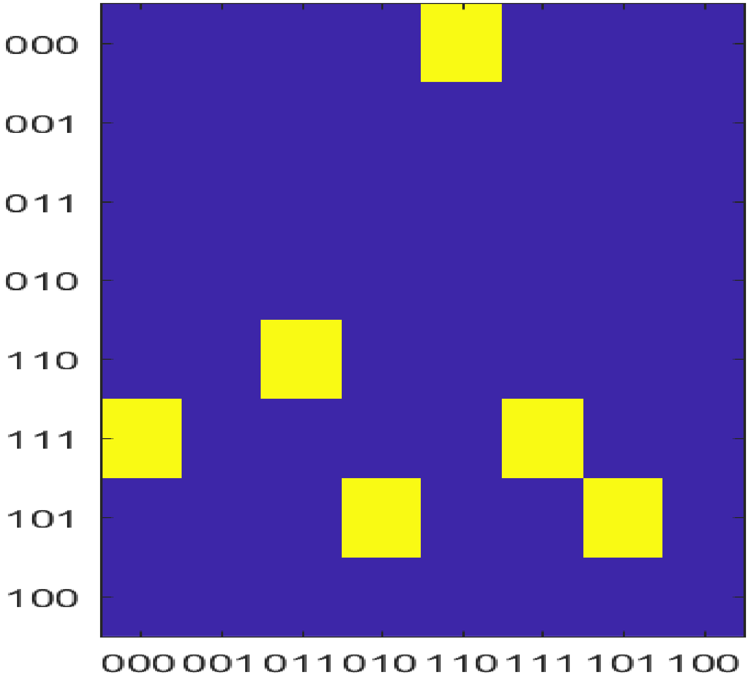
Four *non*-orthogonal 6-bit patterns

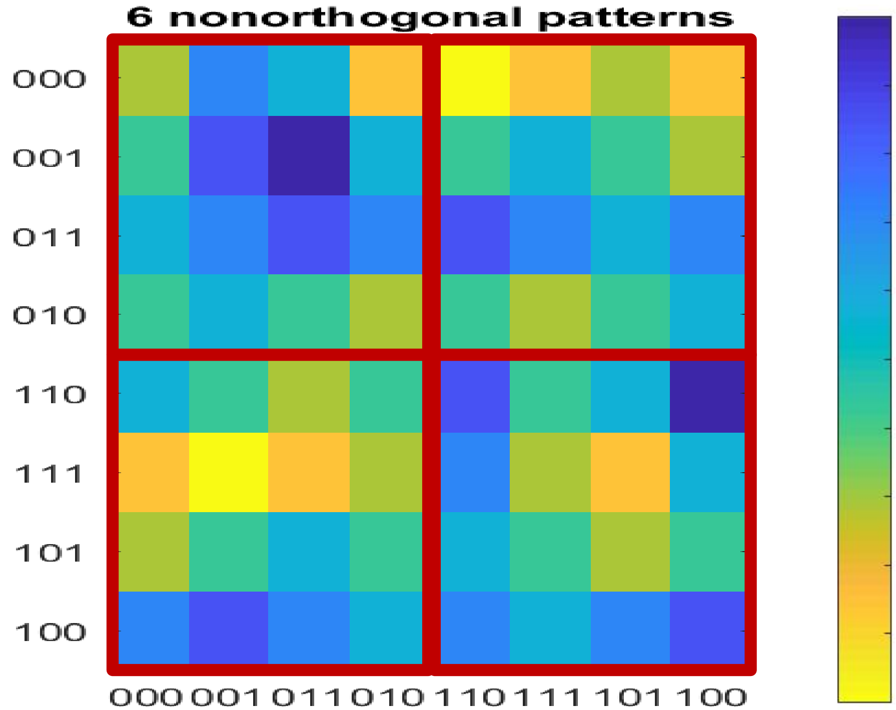




- Patterns are perfectly stationary and stable for K > 0.14N
- *Fewer* spurious minima than for the orthogonal 2-pattern case
  - Most fake-looking memories are in fact ghosts..

Six *non*-orthogonal 6-bit patterns



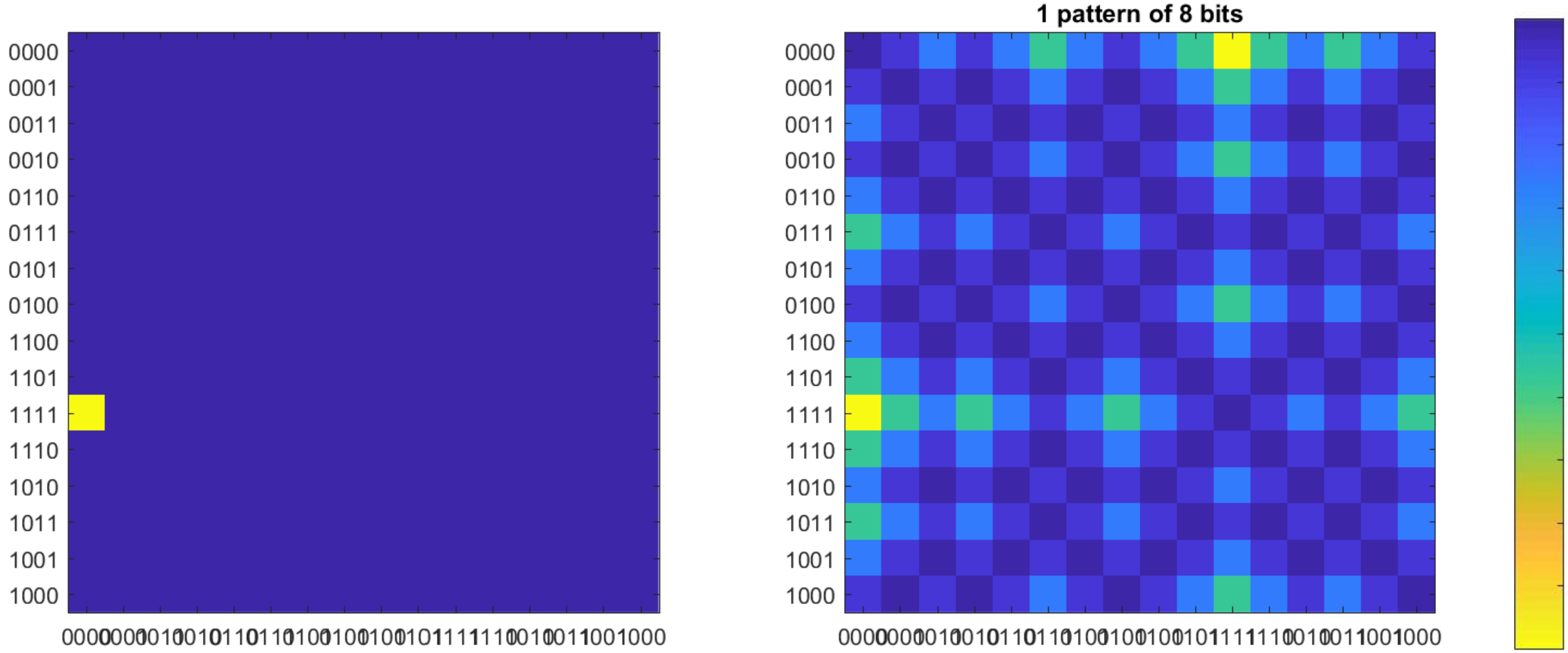


- Breakdown largely due to interference from “ghosts”
- But patterns are stationary, and often stable
  - For K >> 0.14N

More visualization..

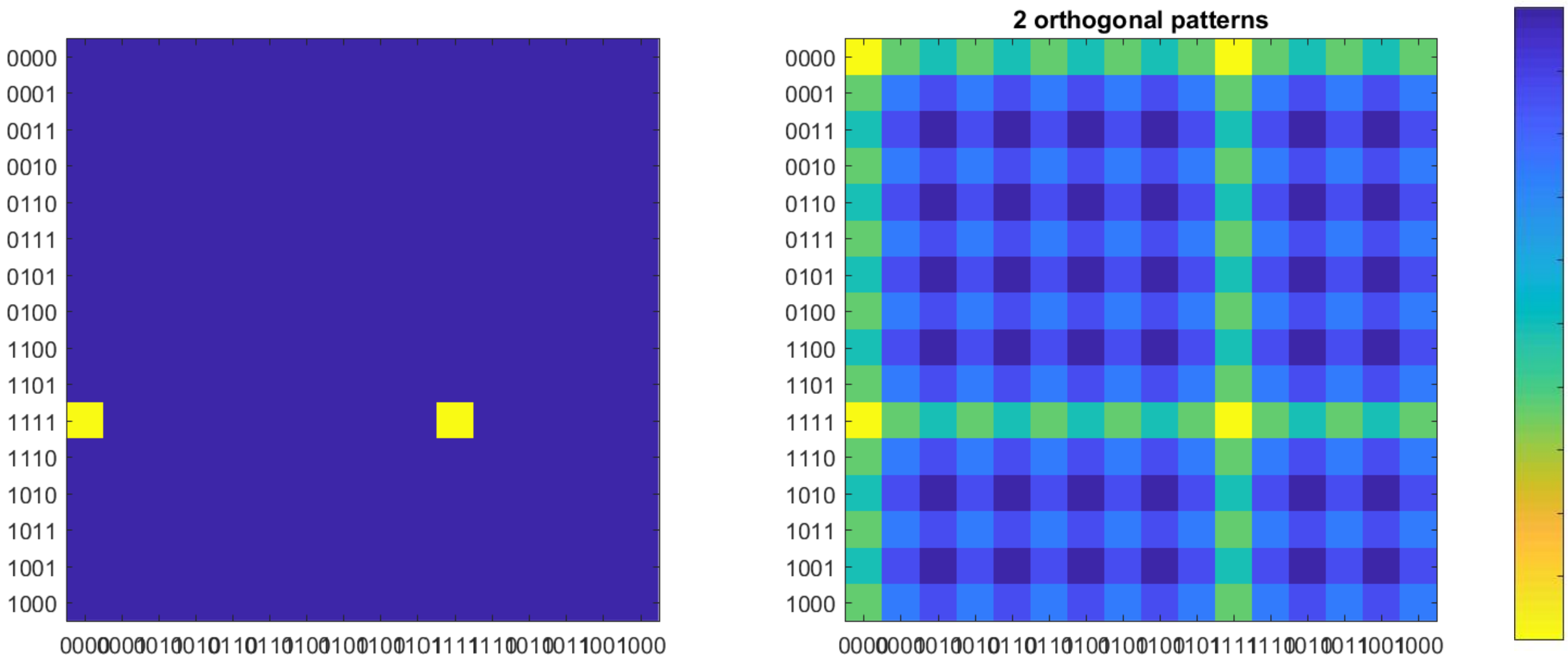
- Lets inspect a few 8-bit patterns
  - Keeping in mind that the Karnaugh map is now a 4-dimensional tesseract

One 8-bit pattern



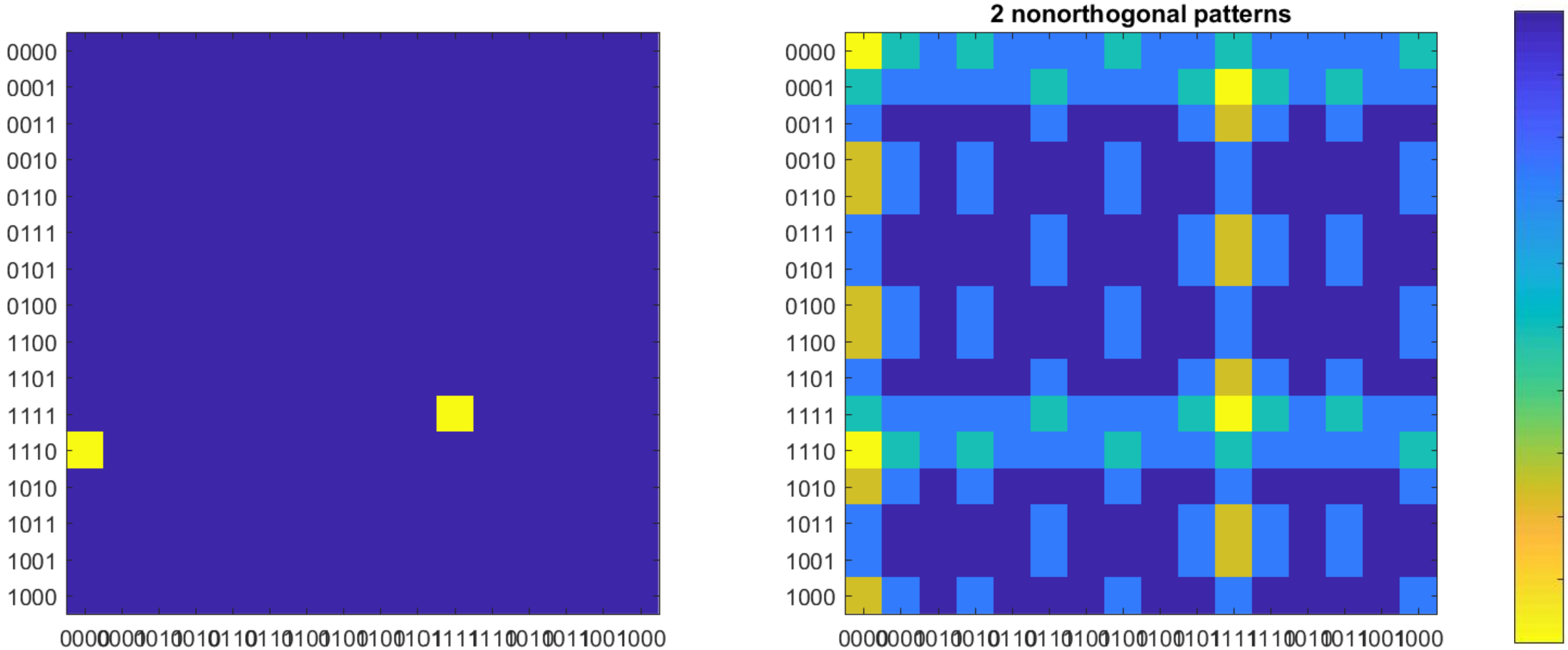
- Its actually cleanly stored, but there are a few spurious minima

Two orthogonal 8-bit patterns



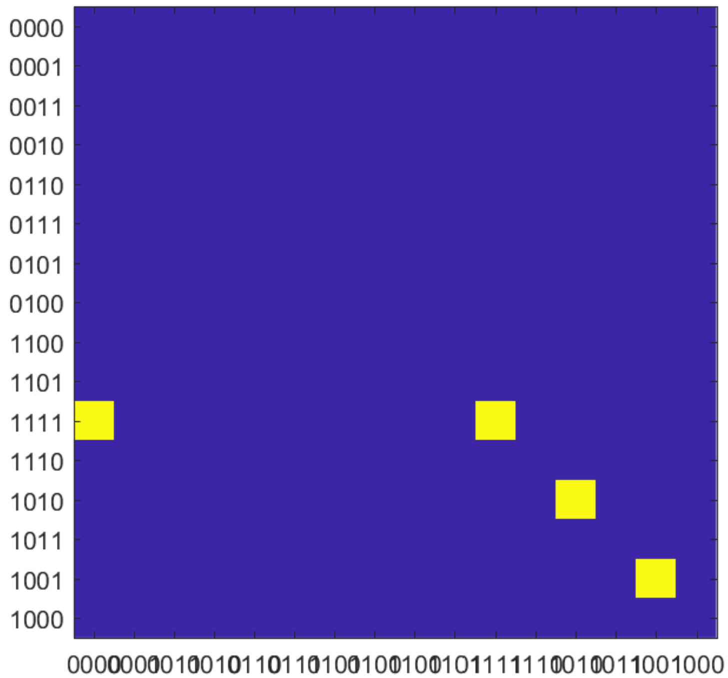
- Both have regions of attraction
- Some spurious minima

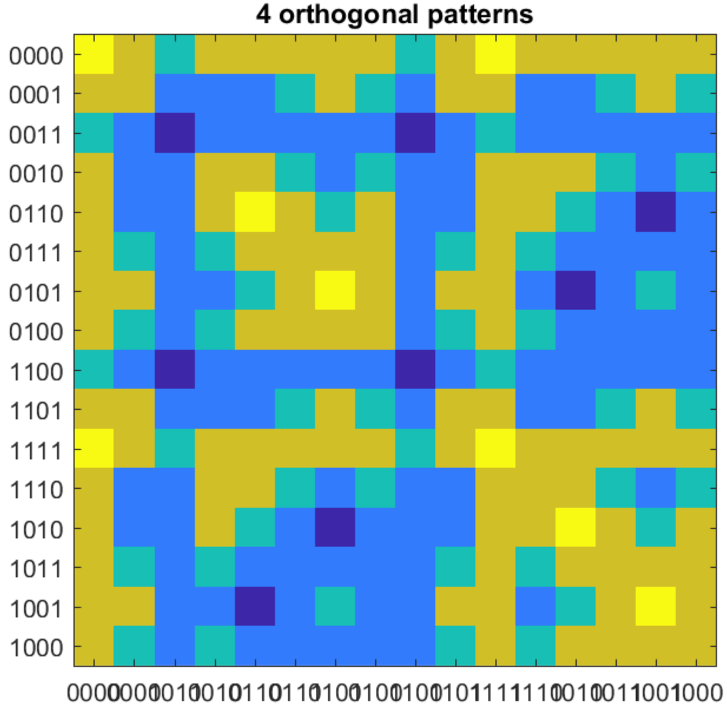
Two non-orthogonal 8-bit patterns



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

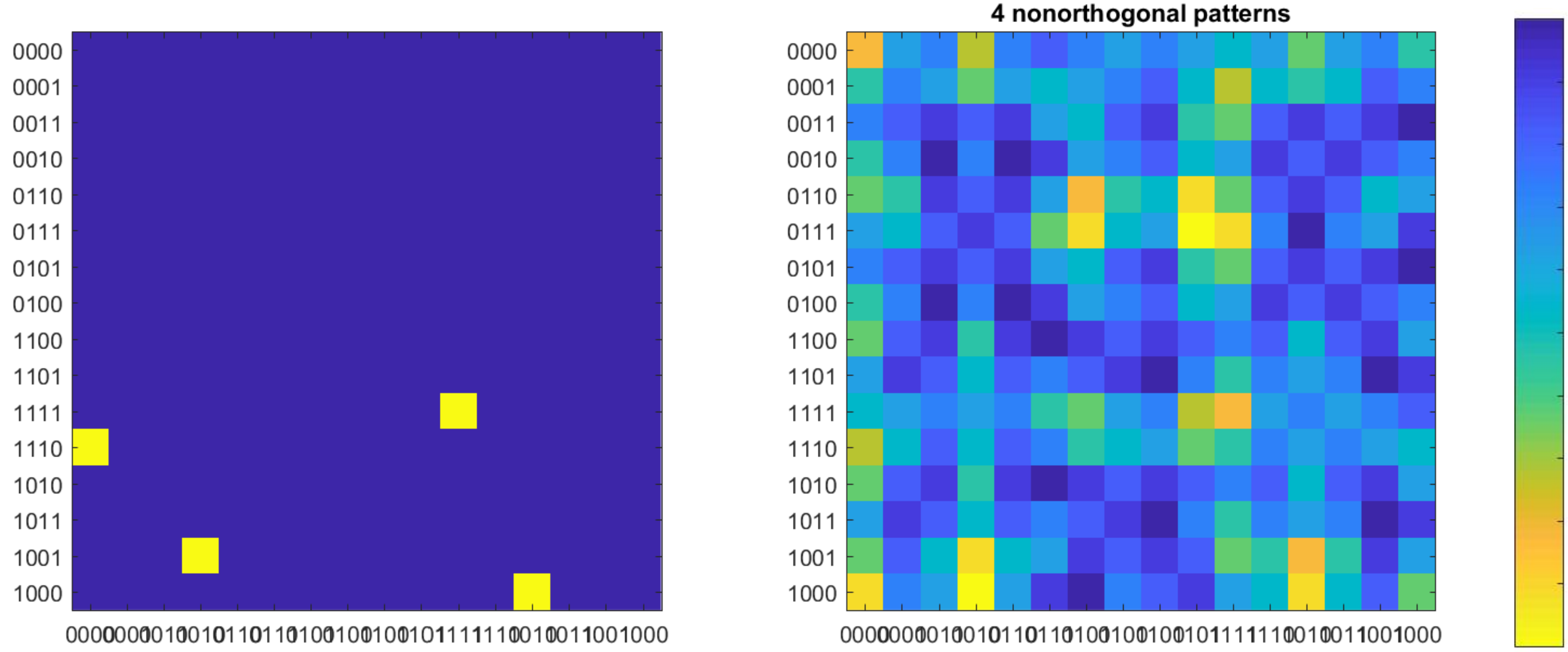
Four orthogonal 8-bit patterns





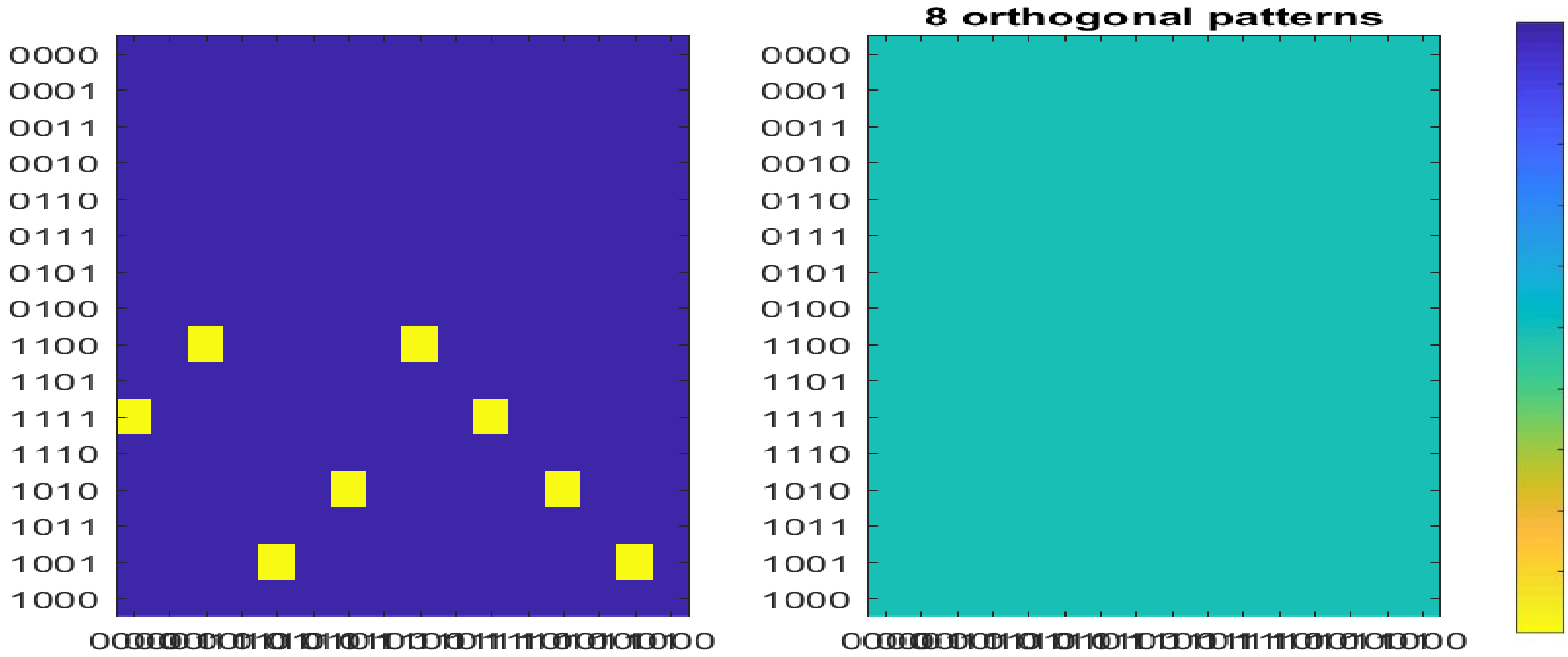
- Successfully stored

Four non-orthogonal 8-bit patterns



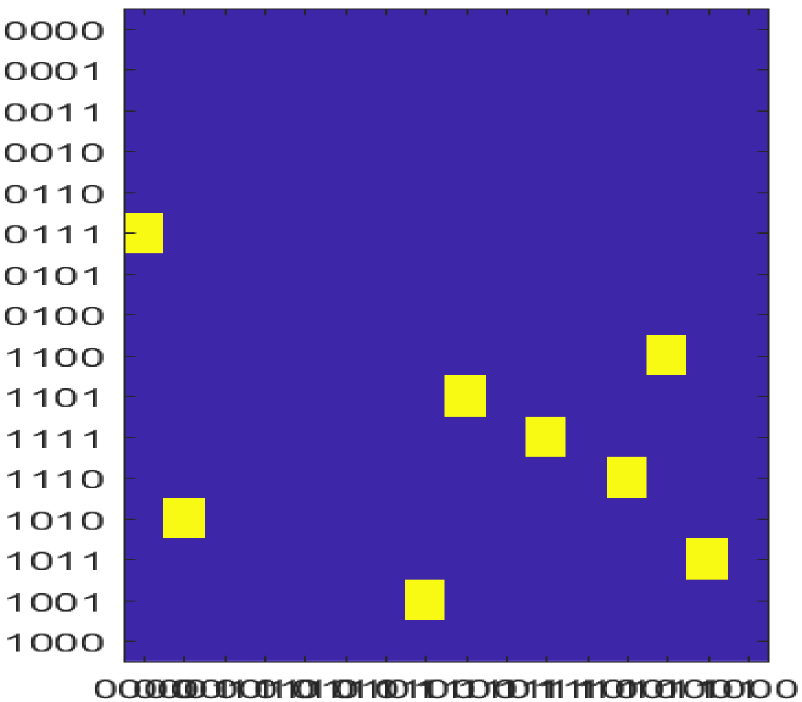
- Stored with interference from ghosts..

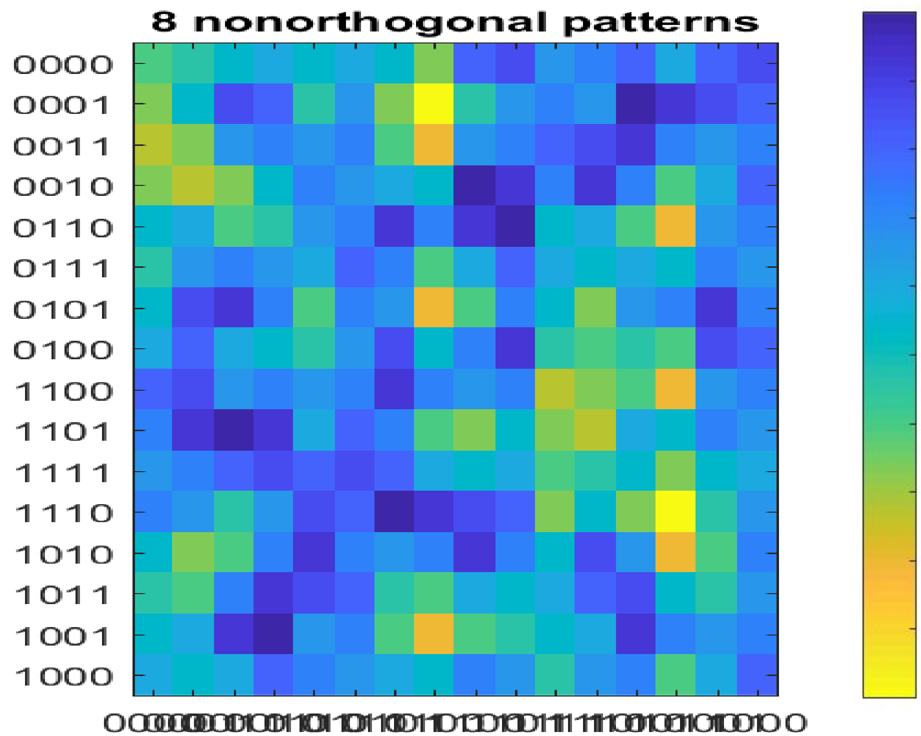
Eight orthogonal 8-bit patterns



- Wipeout

Eight non-orthogonal 8-bit patterns





- Nothing stored
  - Neither stationary nor stable

Making sense of the behavior

- 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
    - So what was Hopfield talking about?
- 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?

Bold Claim

- I can *always* store (upto) N orthogonal patterns such that they are stationary!
  - Although not necessarily stable

- Why?

“Training” the network

- How do we make the network store *a specific* pattern or set of patterns?
  - Hebbian learning
  - Geometric approach
  - Optimization

- Secondary question
  - How many patterns can we store?

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

Energy landscape
only differs by
an additive constant

Gradients and location
of minima remain same

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

Energy landscape only differs by an additive constant

Gradients and location of minima remain same

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

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

Energy landscape
only differs by
an additive constant

Gradients and location
of minima remain same

Both have the
same Eigen vectors

behavior with

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

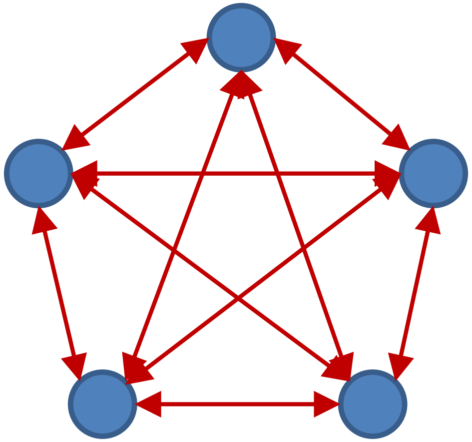
NOTE: This
is a positive
semidefinite matrix

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

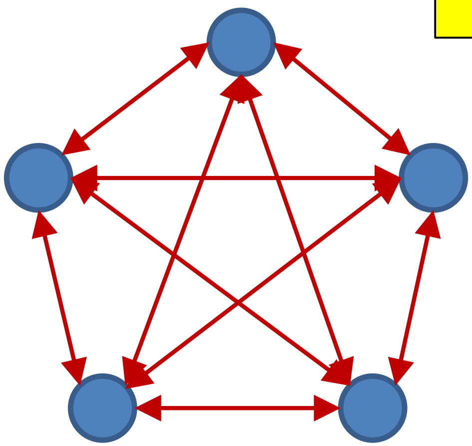
Consider the energy function



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

- 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

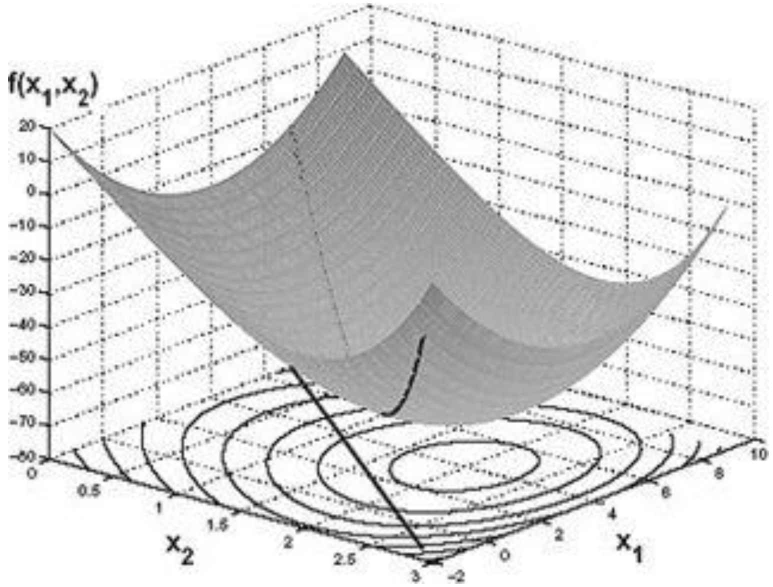
E is convex

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

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

The energy function

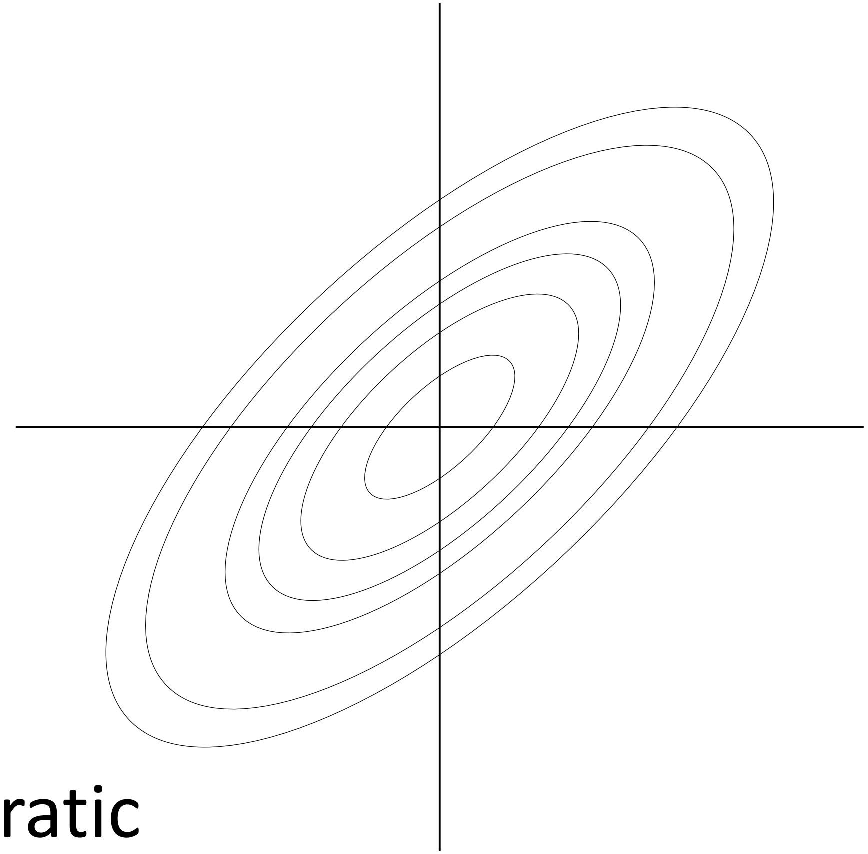
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y}$$



- E is a convex quadratic

The energy function

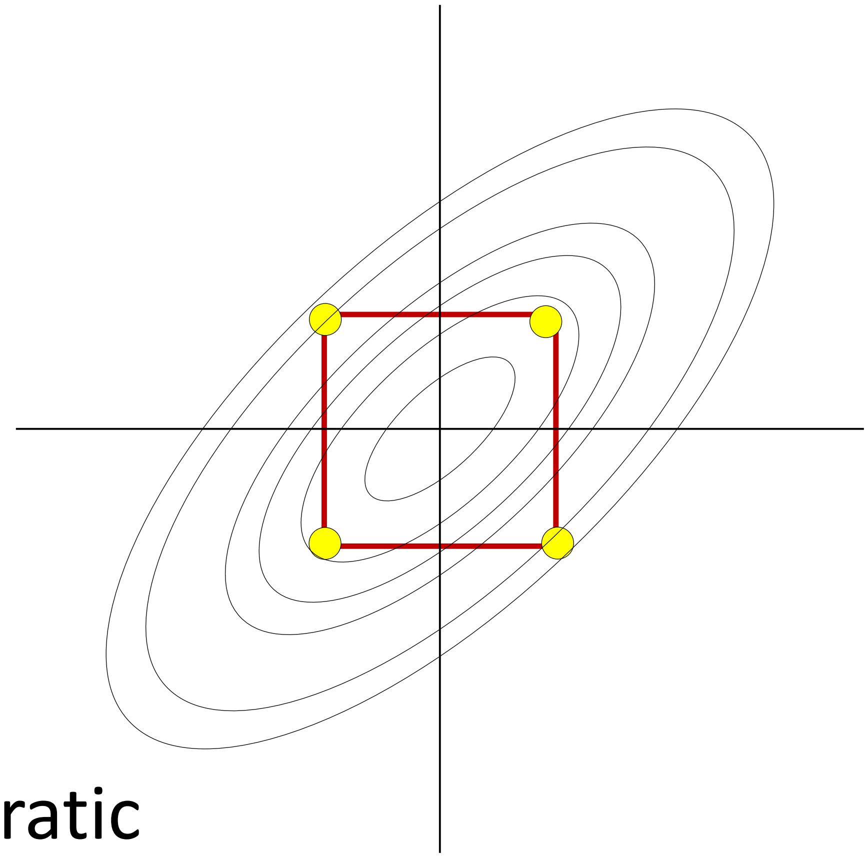
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y}$$



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

The energy function

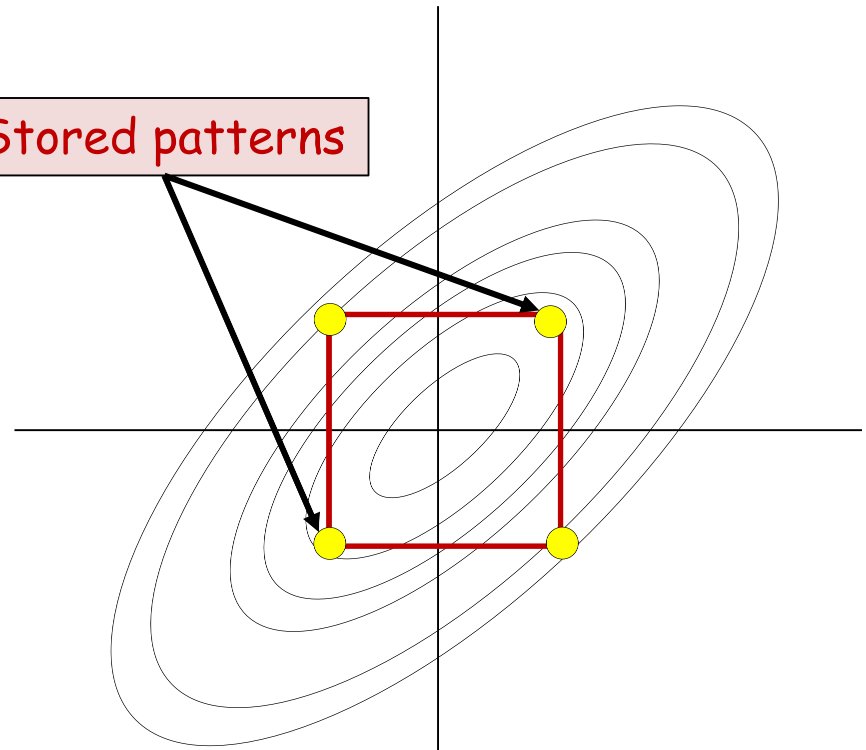
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y}$$



- 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

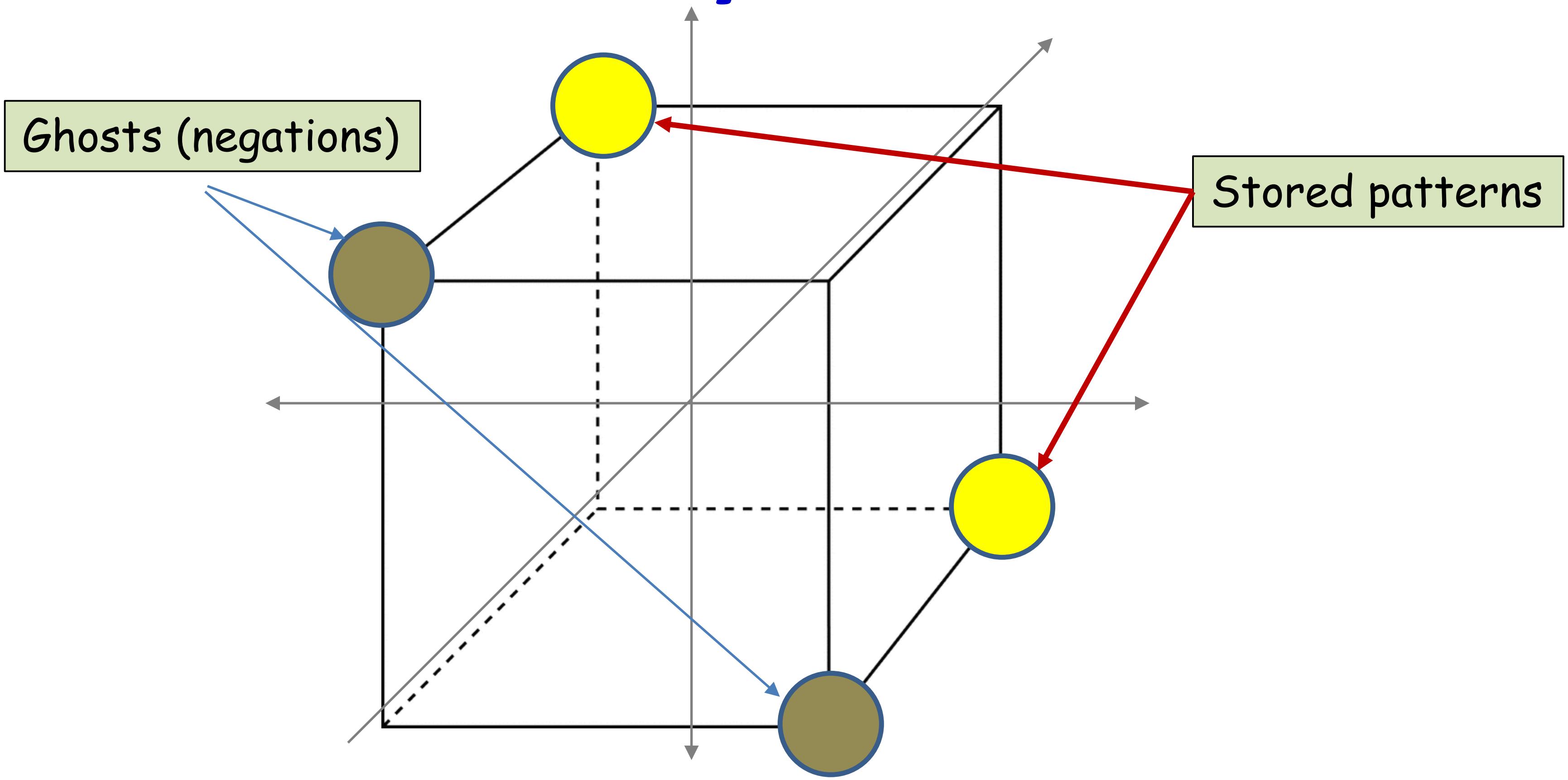
The energy function

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



- 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

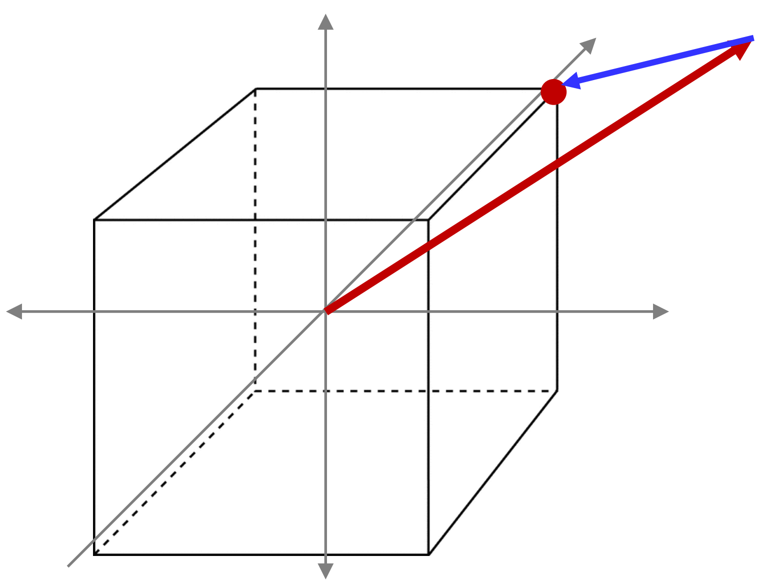


- Ideally must be maximally separated on the hypercube
  - The number of patterns we can store depends on the actual distance between the patterns

Storing patterns

- A pattern $\mathbf{y}_P$ is stored if:
  - $sign(\mathbf{W}\mathbf{y}_p) = \mathbf{y}_p$ for all target patterns

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



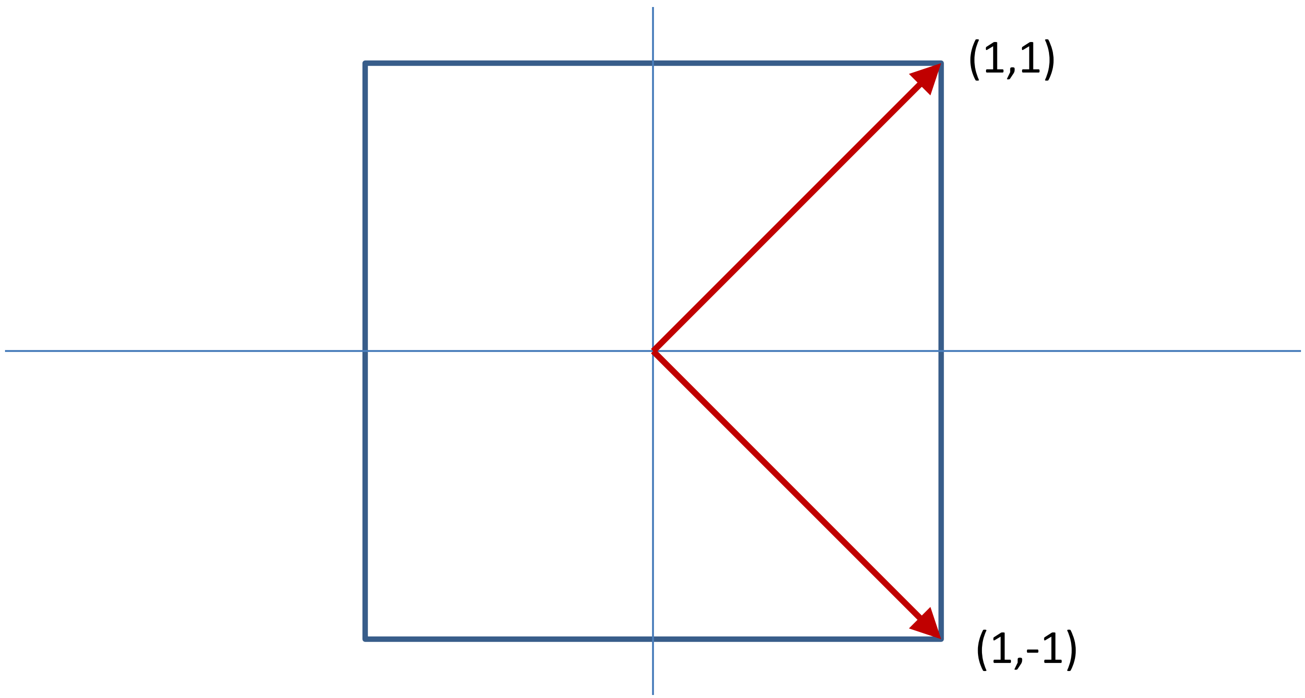
Storing patterns

- 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?

Only N patterns?



- Patterns that differ in $N/2$ bits are orthogonal
- You can have no more than N orthogonal vectors in an N-dimensional space

Another 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 $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_P$
  - 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

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

Storing K orthogonal patterns

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

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

  - $\lambda_1, \ldots, \lambda_K$ 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 $Y = [\mathbf{y}_1\ \mathbf{y}_2 \ldots \mathbf{y}_K\ \mathbf{r}_{K+1}\ \mathbf{r}_{K+2} \ldots \mathbf{r}_N]$

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

  - $\mathbf{r}_{K+1}\ \mathbf{r}_{K+2} \ldots \mathbf{r}_N$ are orthogonal to $\mathbf{y}_1\ \mathbf{y}_2 \ldots \mathbf{y}_K$
  - $\lambda_1 = \lambda_2 = \lambda_K = 1$
  - $\lambda_{K+1}, \ldots, \lambda_N = 0$
- All patterns orthogonal to $\mathbf{y}_1\ \mathbf{y}_2 \ldots \mathbf{y}_K$ are also stationary
  - Although not stable

Storing N orthogonal patterns

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

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

  - $\lambda_1 = \lambda_2 = \lambda_N = 1$
- The Eigen vectors of 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 $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N$ *span the space*
- Any pattern $\mathbf{y}$ can be written as

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

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

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

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

Storing *K* orthogonal patterns

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

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

  - $\mathbf{r}_{\boldsymbol{K}+1}\ \mathbf{r}_{\boldsymbol{K}+2} \dots \mathbf{r}_N$ are orthogonal to $\mathbf{y}_1\ \mathbf{y}_2 \dots \mathbf{y}_K$
  - $\lambda_1 = \lambda_2 = \lambda_K = 1$
  - $\lambda_{K+1}, \dots, \lambda_N = 0$
- All patterns orthogonal to $\mathbf{y}_1\ \mathbf{y}_2 \dots \mathbf{y}_K$ are stationary
- 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
  - 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: Lanczos iterations
    - Can write $\mathbf{Y}_P = \mathbf{Y}_{ortho}\mathbf{B}, \rightarrow \mathbf{W} = \mathbf{Y}_{ortho}\mathbf{B}\Lambda\mathbf{B}^T\mathbf{Y}_{ortho}^T$

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 Hebbian net with N orthogonal base patterns, *all* patterns are stable

- 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
- 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**

How do we find this network?

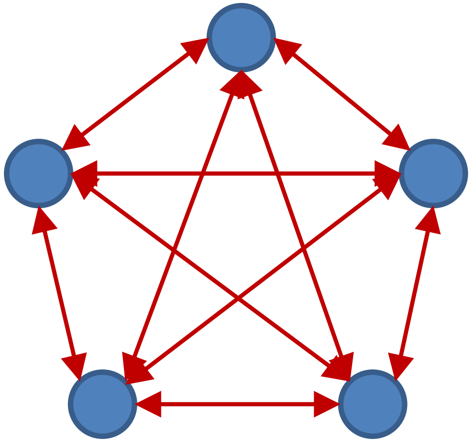
Can we do something about this?

A different tack

- How do we make the network store *a specific* pattern or set of patterns?
  - Hebbian learning
  - Geometric approach
  - Optimization

- Secondary question
  - How many patterns can we store?

Consider the energy function

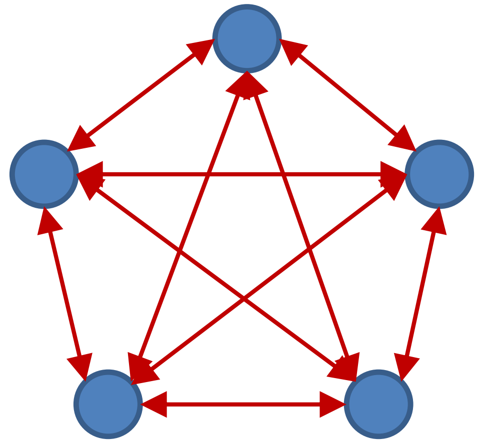


$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y}$$

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

Alternate Approach to Estimating the Network

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



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

Optimizing W (and b)

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

The bias can be captured by another fixed-value component

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

Optimizing W

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

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

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

Optimizing W

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

- Simple gradient descent:

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

Optimizing W

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

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

Optimizing W

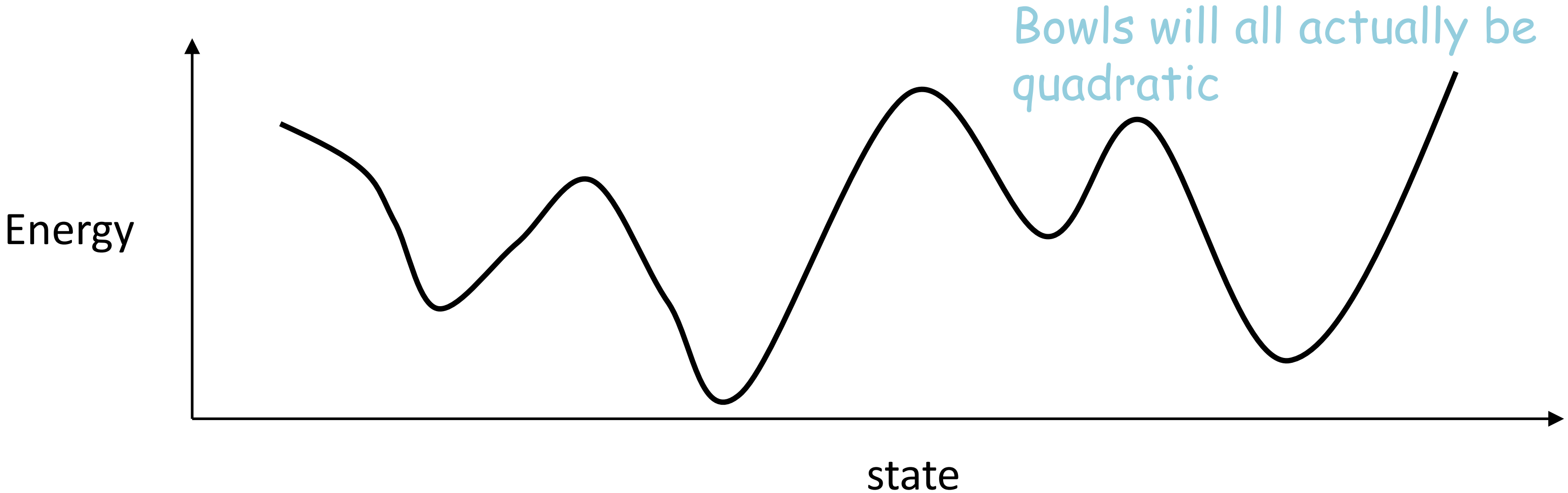
$$\mathbf{W} = \mathbf{W} + \eta\left(\sum_{\mathbf{y}\in\mathbf{Y}_P} \mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y}\notin\mathbf{Y}_P} \mathbf{y}\mathbf{y}^T\right)$$

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

The training again..

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

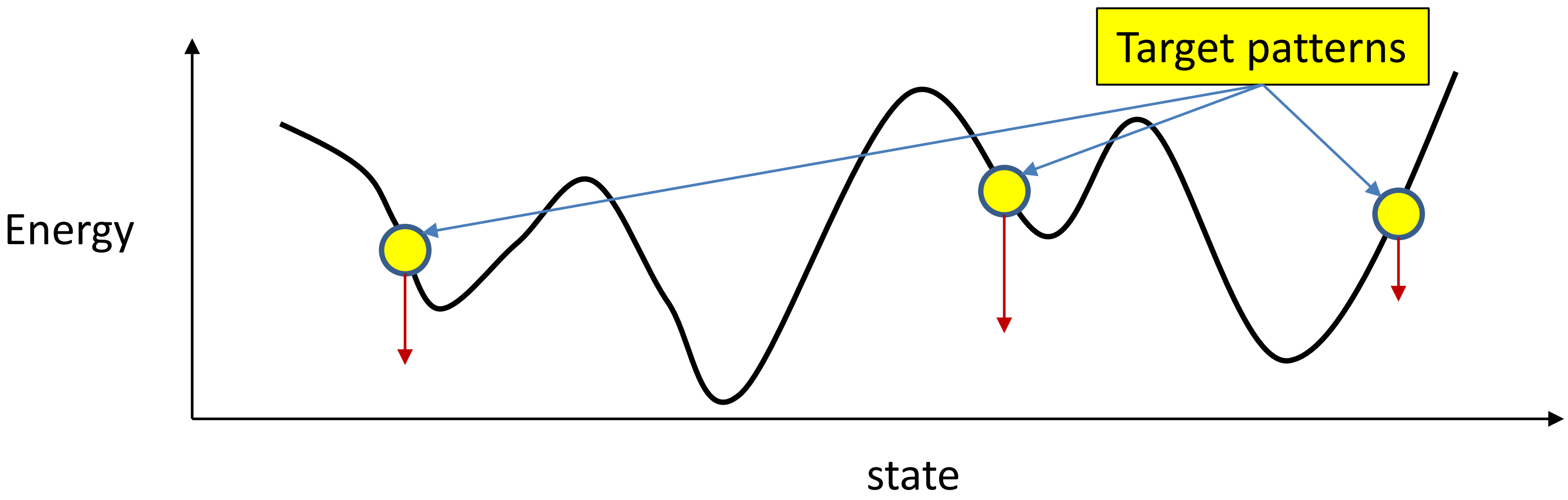
- Note the energy contour of a Hopfield network for any weight $\mathbf{W}$



The training again

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

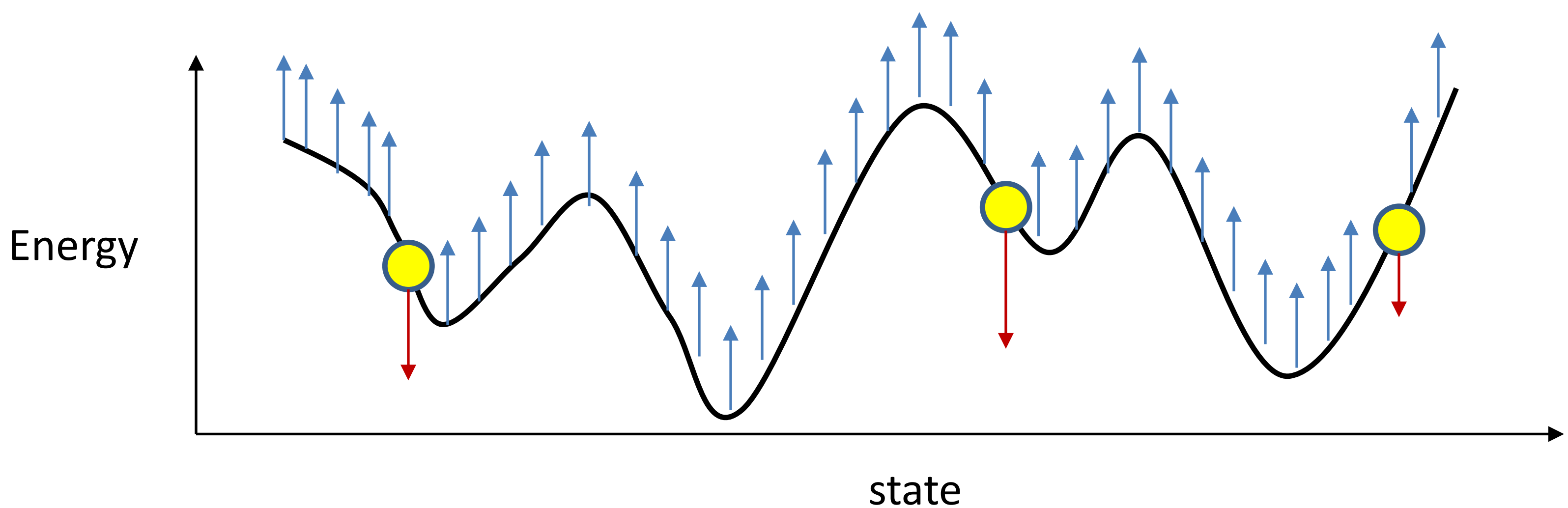
- The first term tries to *minimize* the energy at target patterns
  - Make them local minima
  - Emphasize more “important” memories by repeating them more frequently



The negative class

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

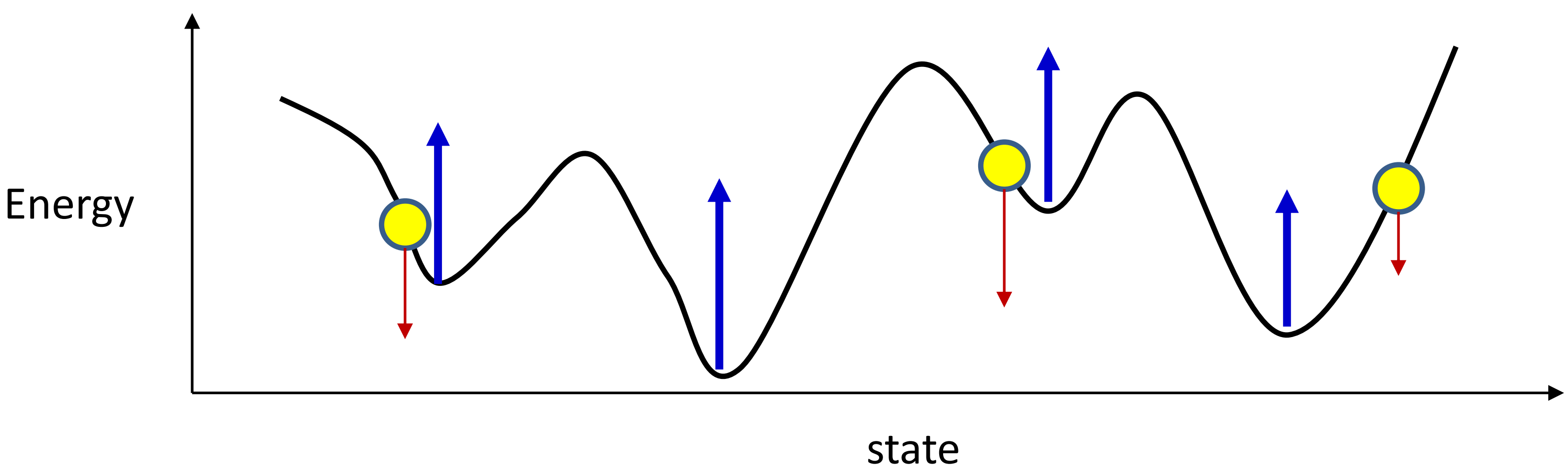
- The second term tries to “raise” all non-target patterns
  - Do we need to raise *everything*?



Option 1: Focus on the valleys

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

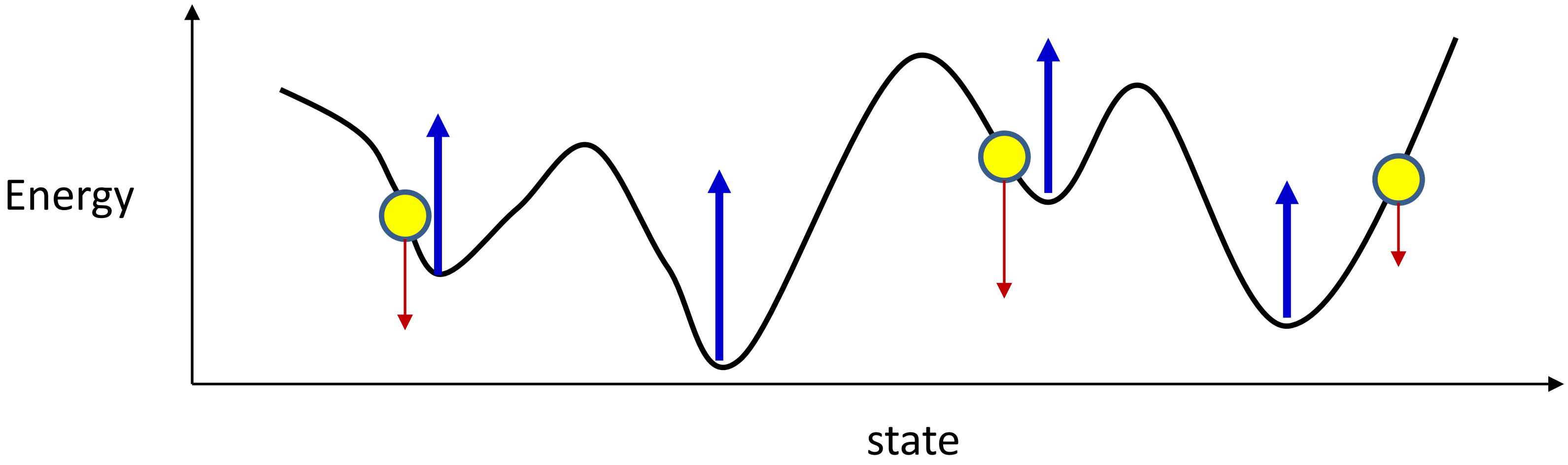
- Focus on raising the valleys
  - If you raise *every* valley, eventually they'll all move up above the target patterns, and many will even vanish



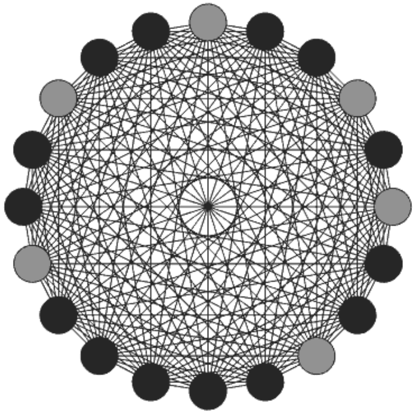
Identifying the valleys..

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

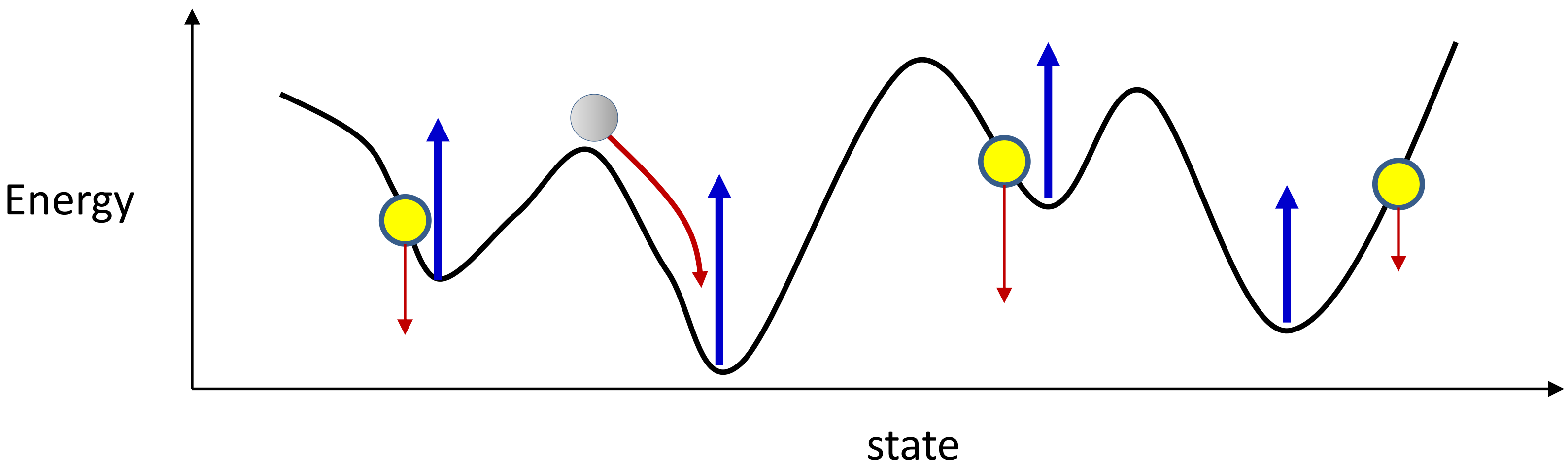
- Problem: How do you identify the valleys for the current $\mathbf{W}$?



Identifying the valleys..



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



Training the Hopfield network

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

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

Training the Hopfield network: SGD version

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

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

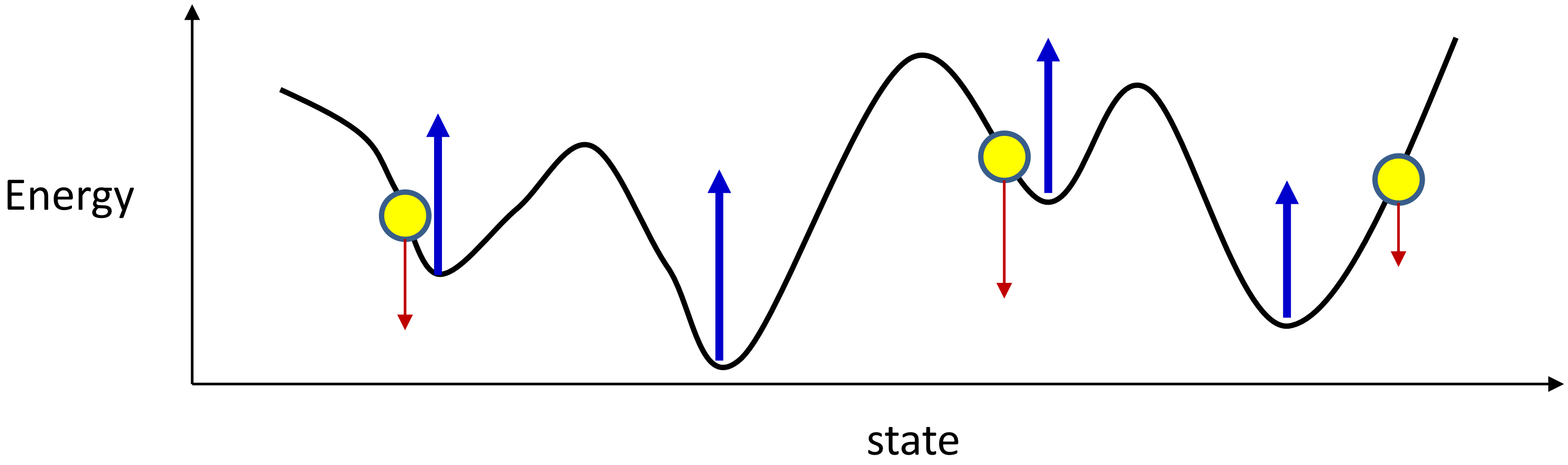
Training the Hopfield network

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

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

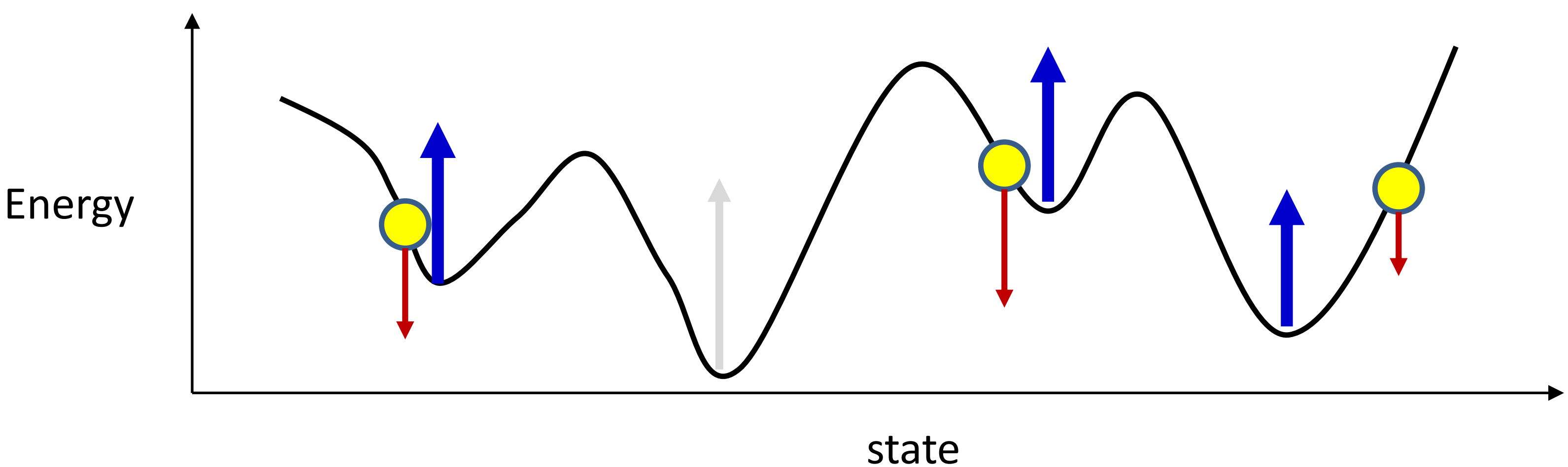
Which valleys?

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

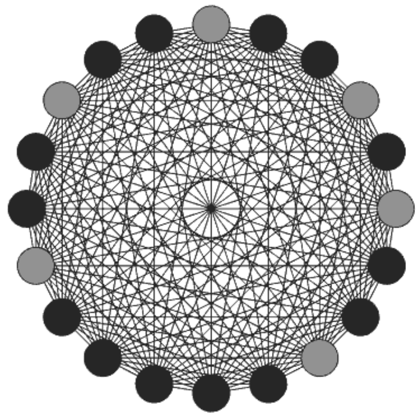


Which valleys?

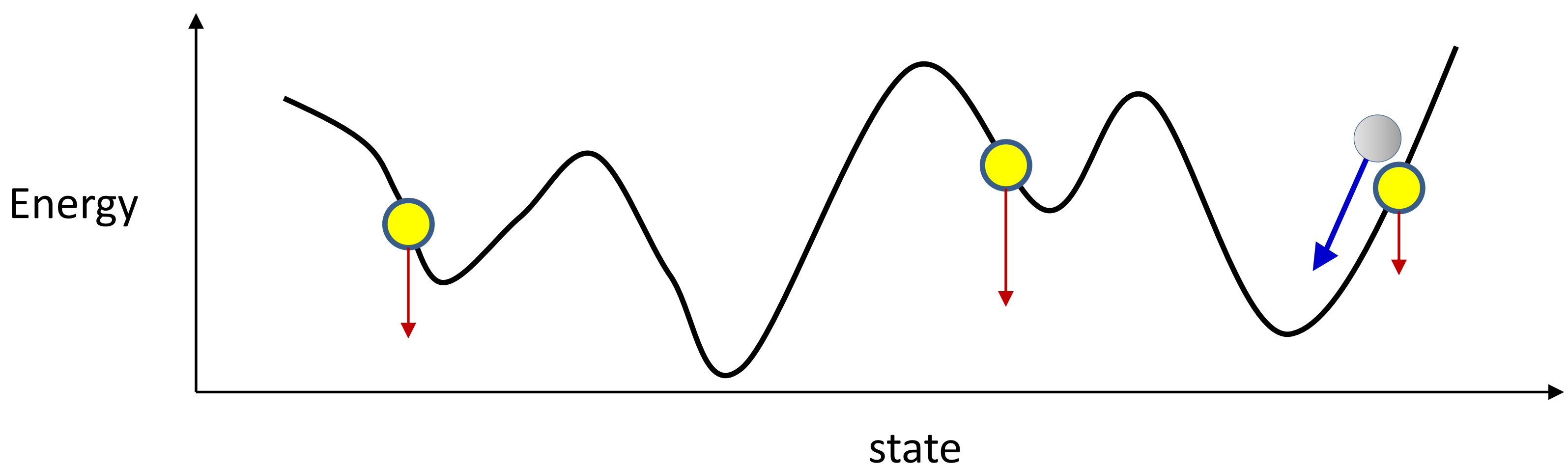
- Should we *randomly* sample valleys?
  - Are all valleys equally important?
- Major requirement: memories must be stable
  - They *must* be broad valleys
- Spurious valleys in the neighborhood of memories are more important to eliminate



Identifying the valleys..



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



Training the Hopfield network

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

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

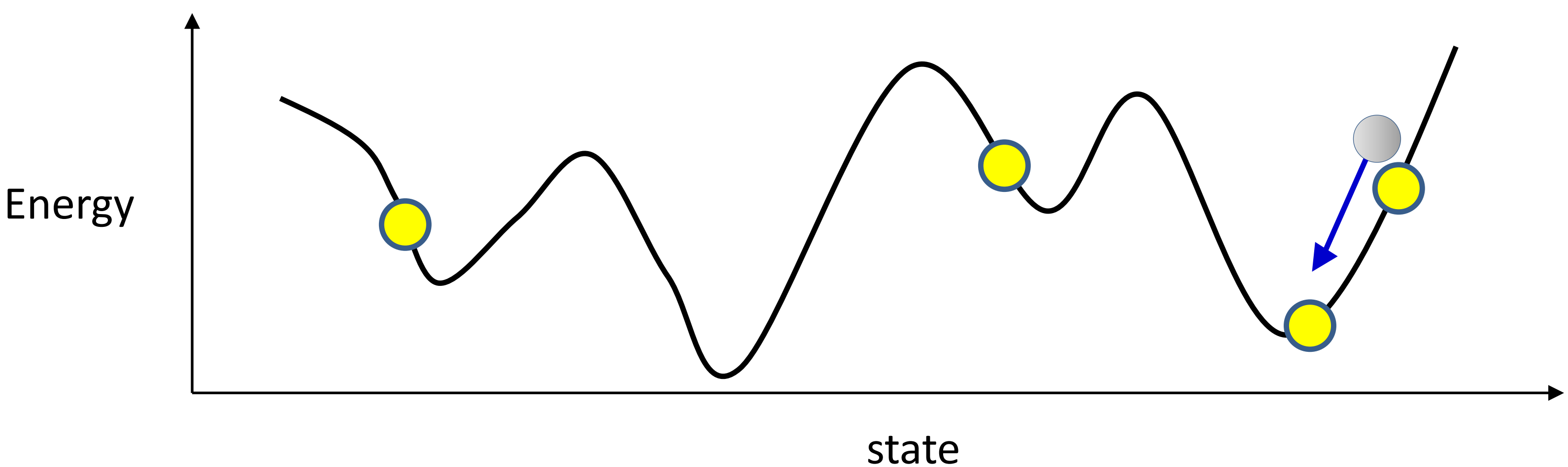
Training the Hopfield network: SGD version

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

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

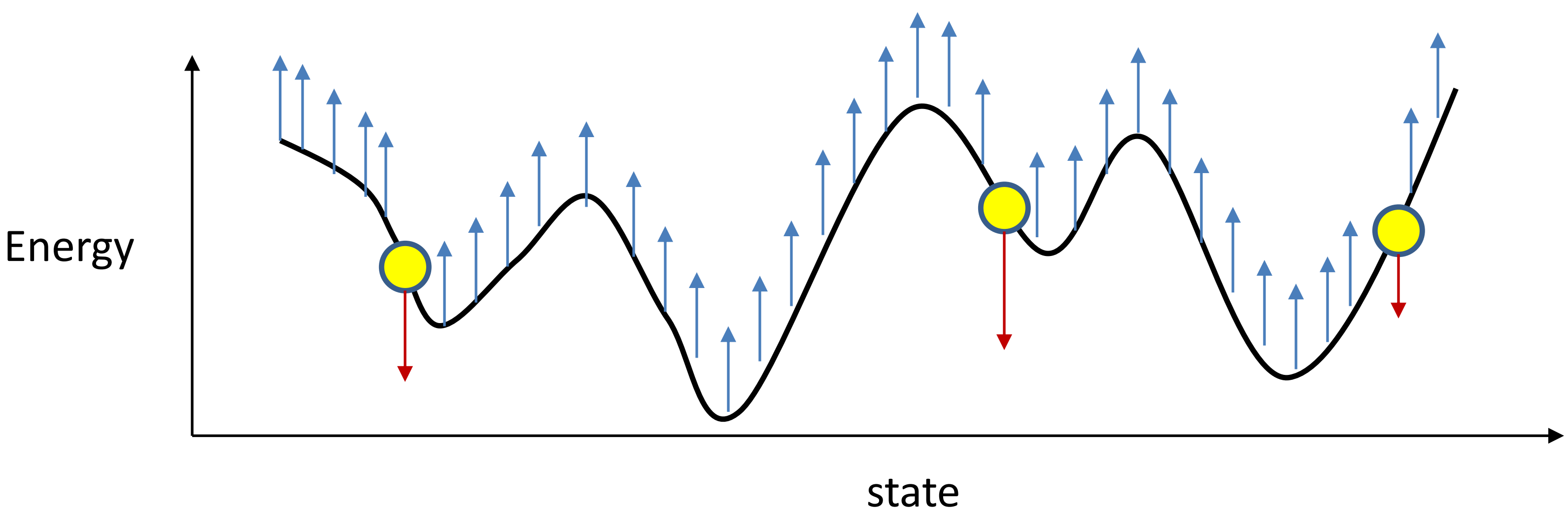
A possible problem

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



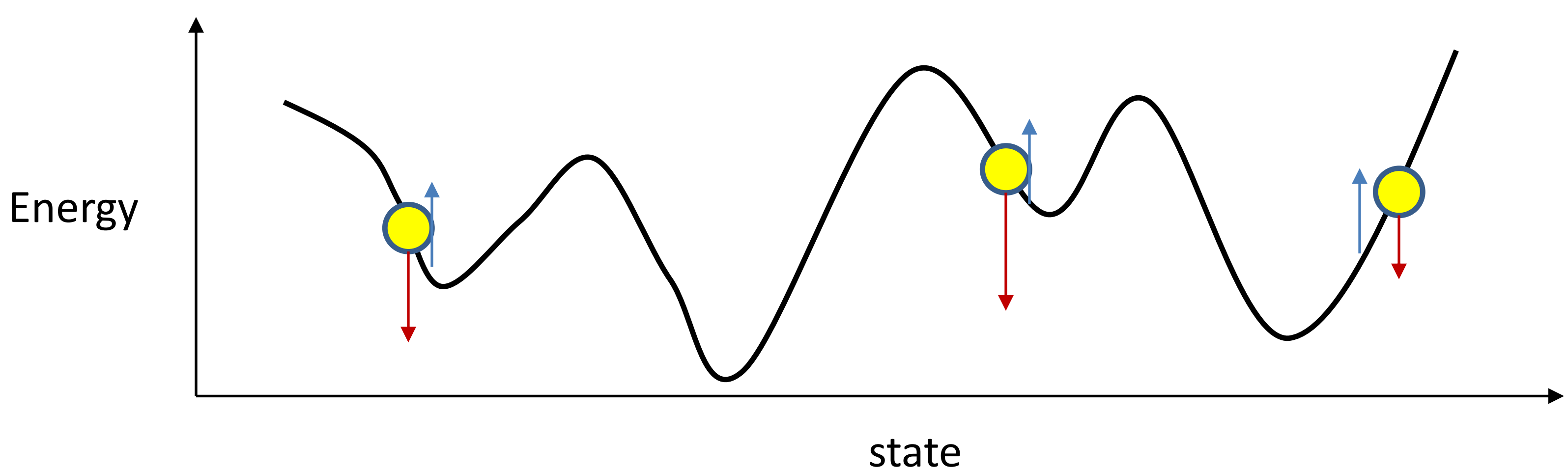
A related issue

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



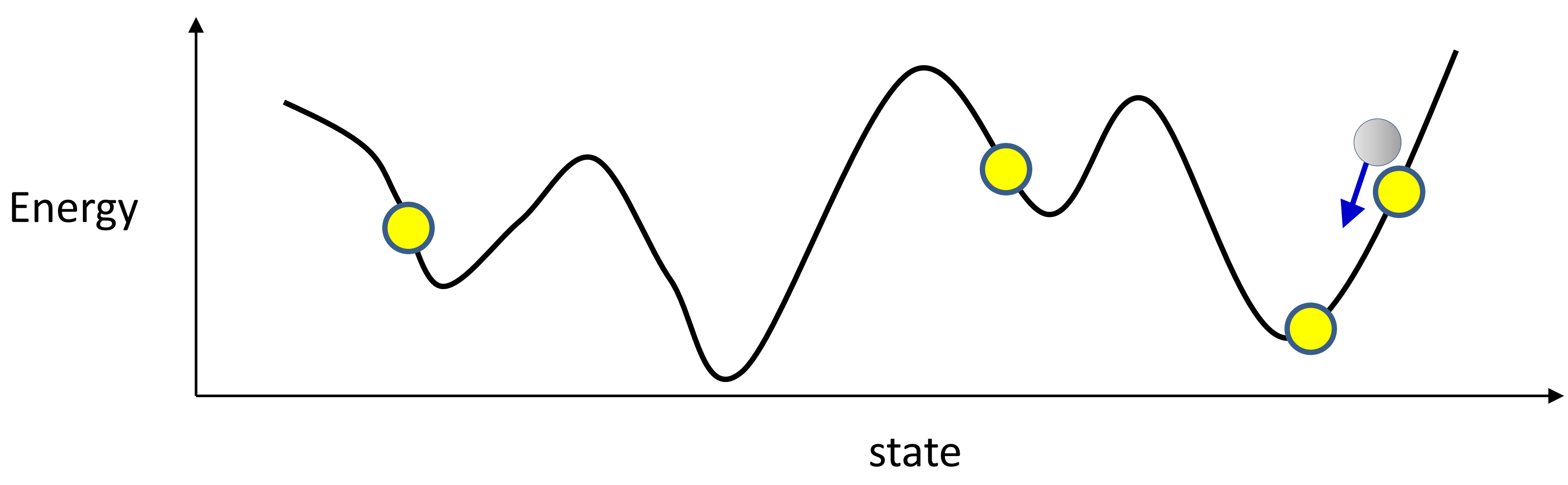
A related issue

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



Raising the neighborhood

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



Training the Hopfield network: SGD version

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

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

A probabilistic interpretation

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

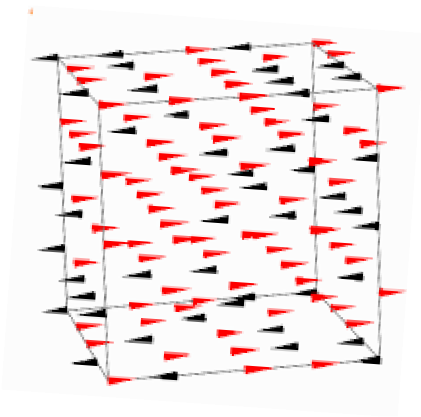
- For continuous $\mathbf{y}$, the *energy* of a pattern is a perfect analog to the *negative log likelihood* of a Gaussian density
- For *binary* y it is the analog of the negative log likelihood of a *Boltzmann distribution*
  - **Minimizing energy maximizes log likelihood**

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

The Boltzmann Distribution

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

$$P(\mathbf{y}) = Cexp\left(\frac{-E(\mathbf{y})}{kT}\right)$$

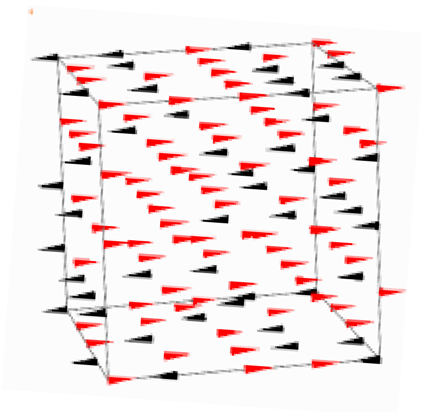


$$C = \frac{1}{\sum_{\mathbf{y}} P(\mathbf{y})}$$

- k is the Boltzmann constant
- T is the temperature of the system
- The energy terms are like the loglikelihood of a Boltzmann distribution at $T = 1$
  – Derivation of this probability is in fact quite trivial..

Continuing the Boltzmann analogy

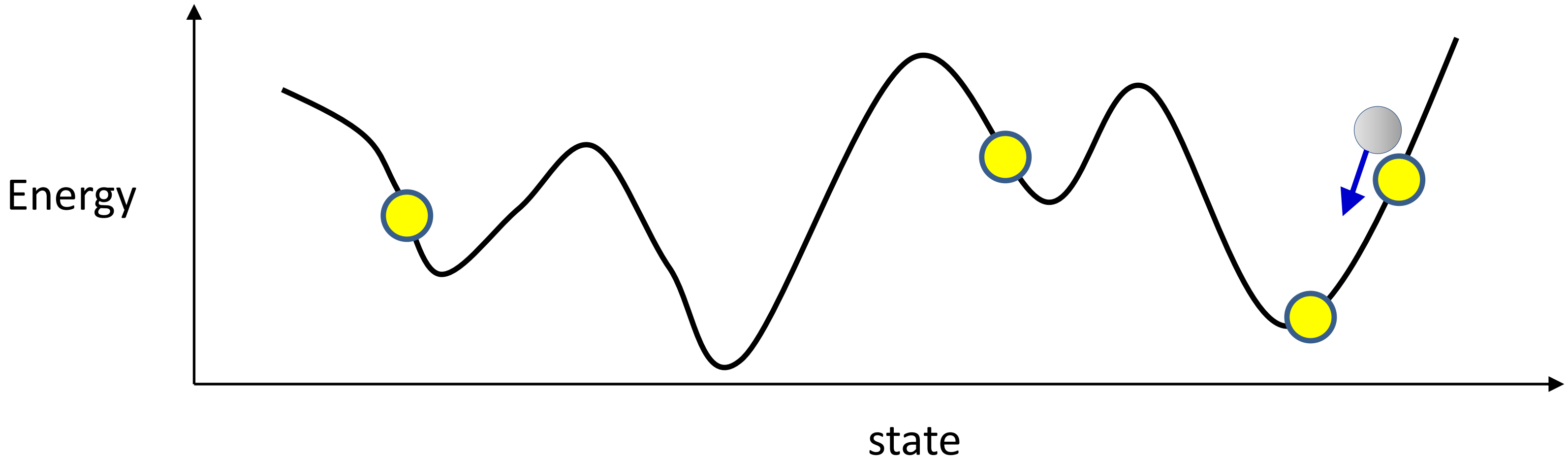
$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y} \qquad P(\mathbf{y}) = Cexp\left(\frac{-E(\mathbf{y})}{kT}\right)$$



$$C = \frac{1}{\sum_{\mathbf{y}} P(\mathbf{y})}$$

- The system *probabilistically* selects states with lower energy
  - With infinitesimally slow cooling, at $T = 0$, it arrives at the global minimal state

Spin glasses and Hopfield nets



- Selecting a next state is akin to drawing a sample from the Boltzmann distribution at $T = 1$, in a universe where $k = 1$

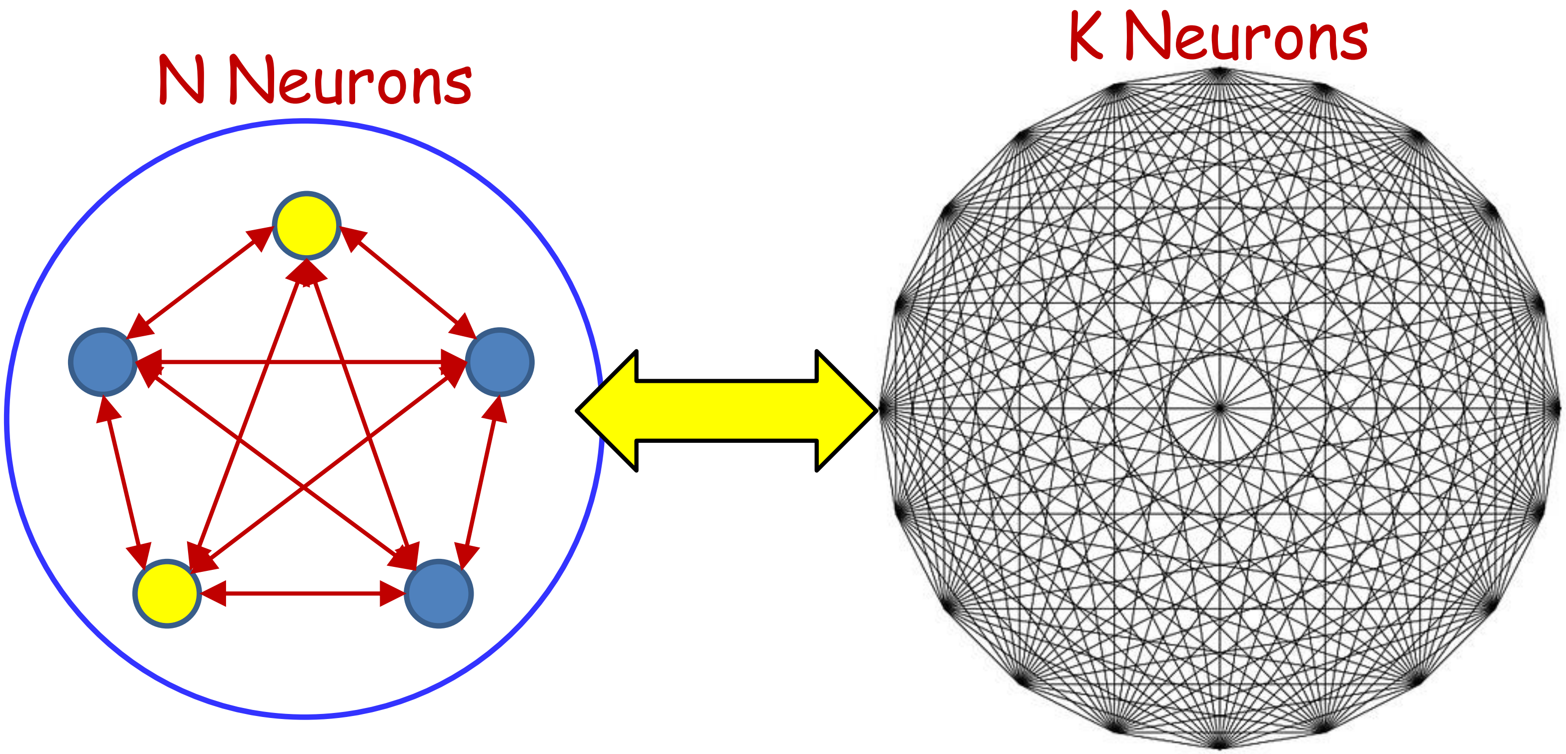
Lookahead..

- The Boltzmann analogy
- Adding capacity to a Hopfield network

Storing more than N patterns

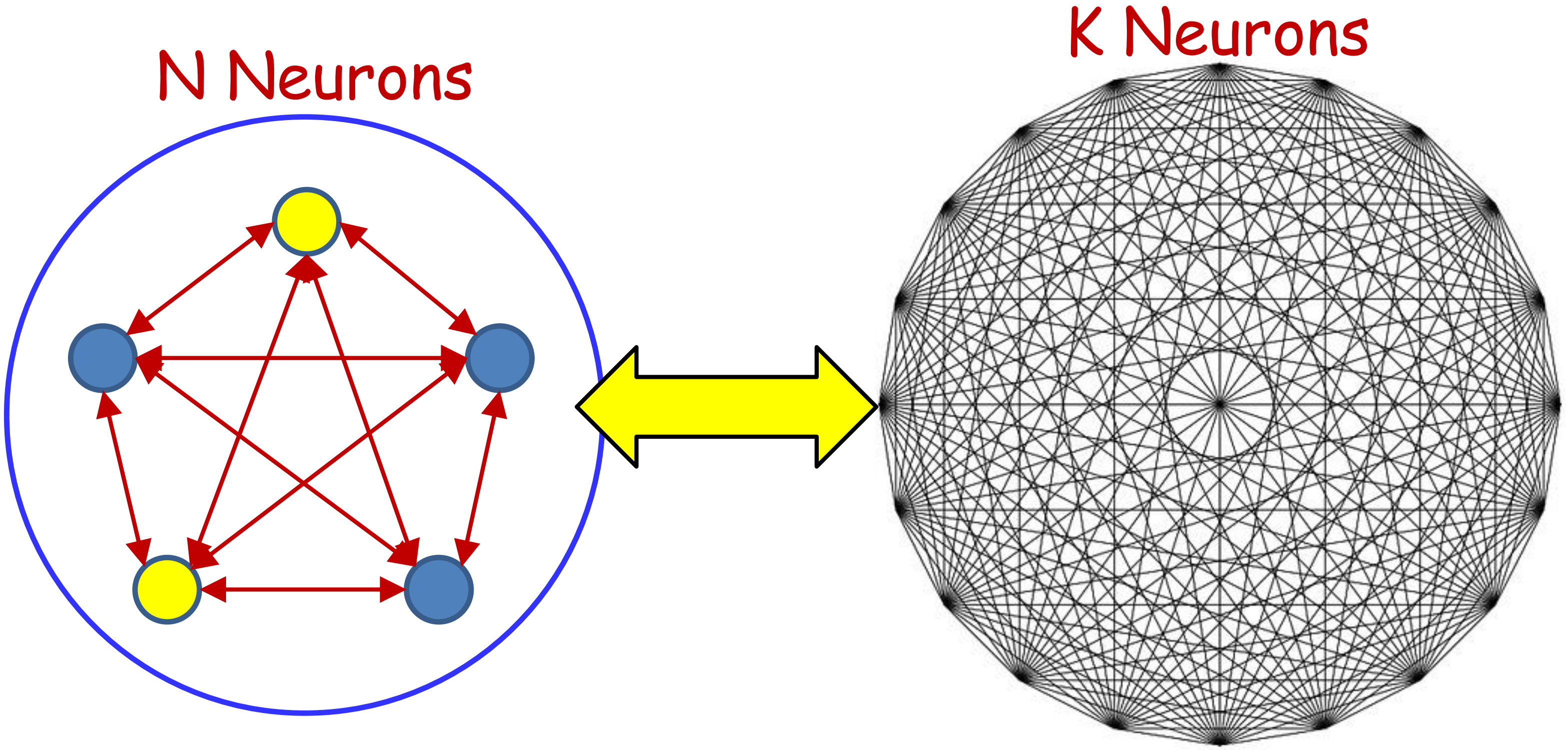
- How do we increase the capacity of the network
  - Store more patterns

Expanding the network



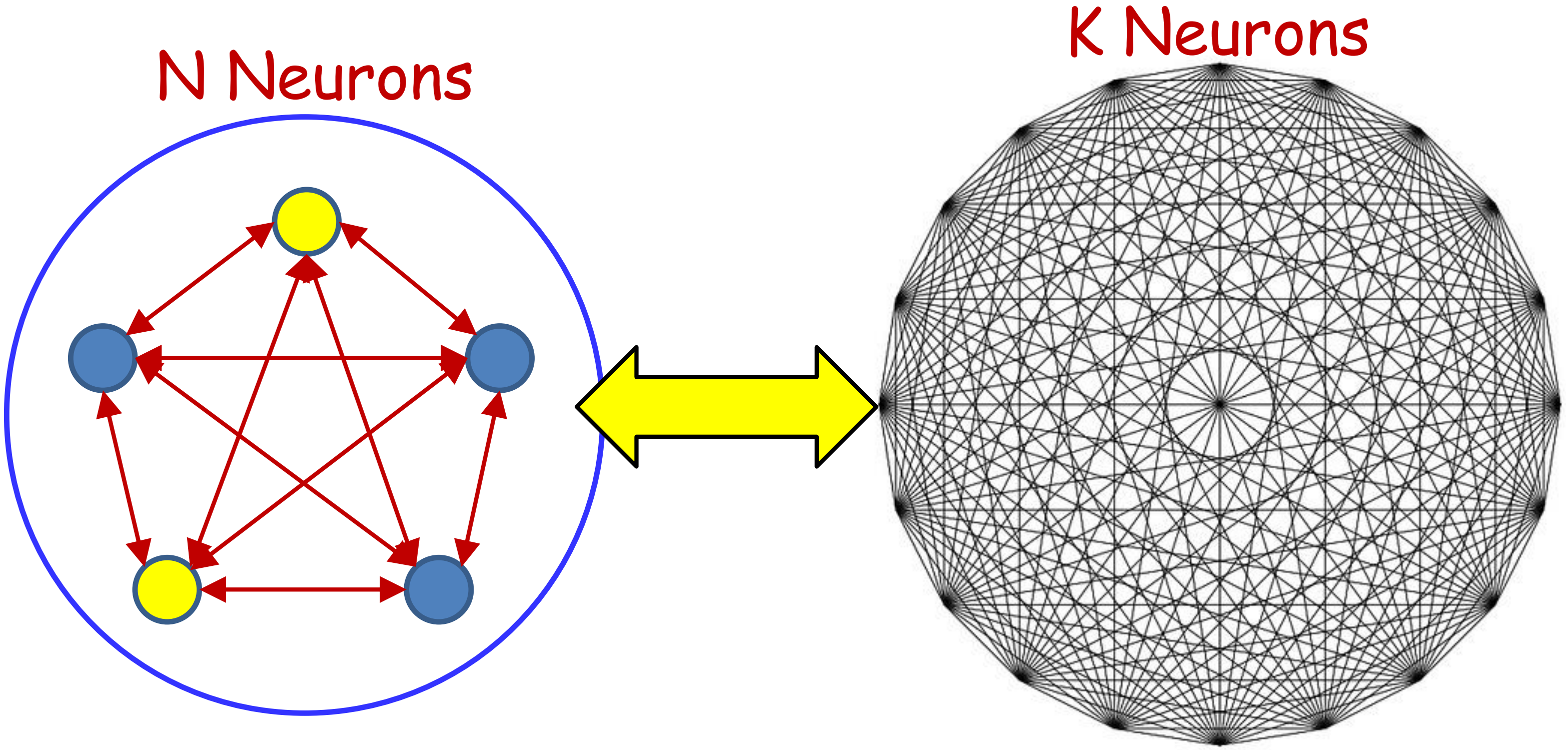
- Add a large number of neurons whose actual values you don't care about!

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…
- Friday please…