# Neural Networks

## Hopfield Nets, Auto Associators, Boltzmann machines

## Spring 2025

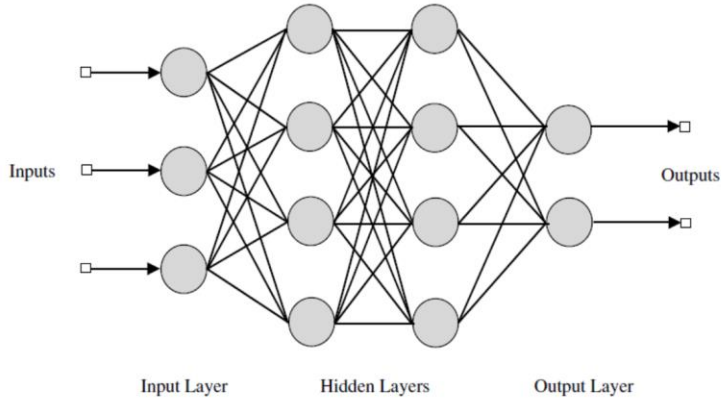
# 2024 Nobel Prize in Physics

## They trained artificial neural networks using physics

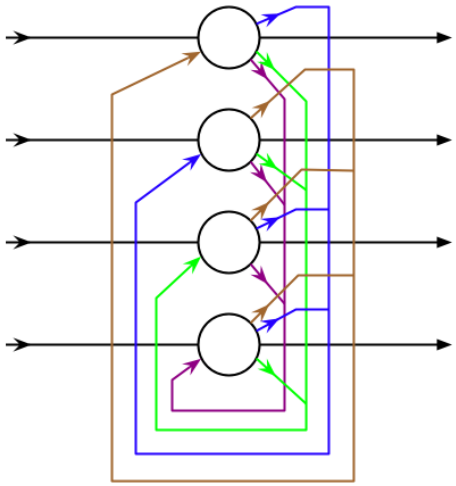
This year's two Nobel Laureates in Physics have used tools from physics to develop methods that are the foundation of today's powerful machine learning. John Hopfield created an associative memory that can store and reconstruct images and other types of patterns in data. Geoffrey Hinton invented a method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

# Story so far

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



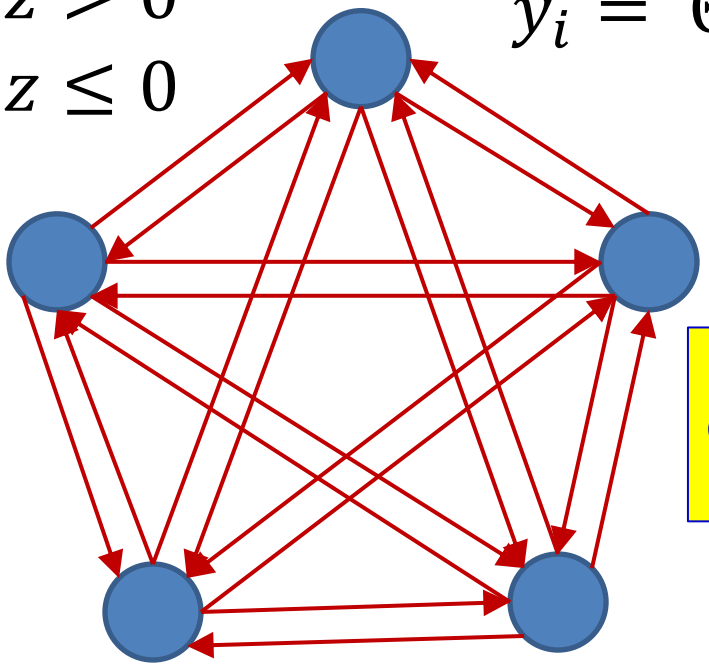

- But what about..



# Consider this loopy network

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

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



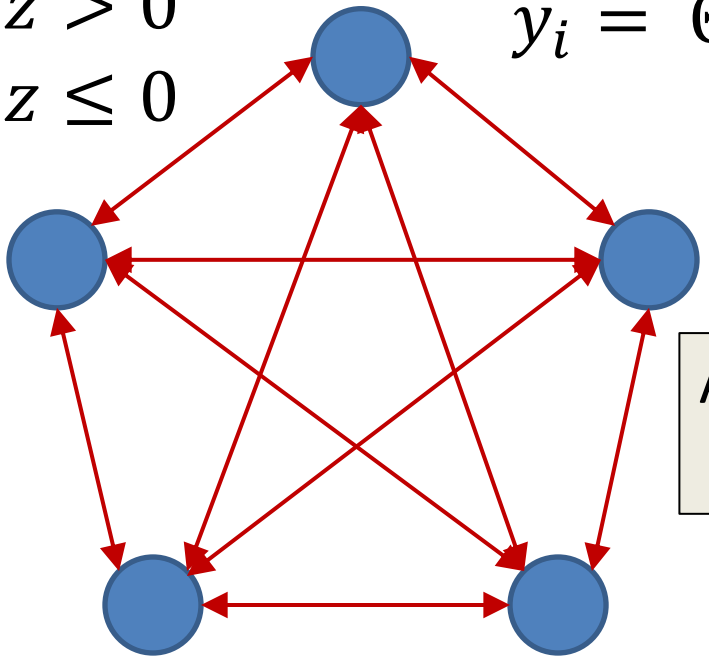
The output of a neuron affects the input to the neuron

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

# Consider this loopy network

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

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



A symmetric network:

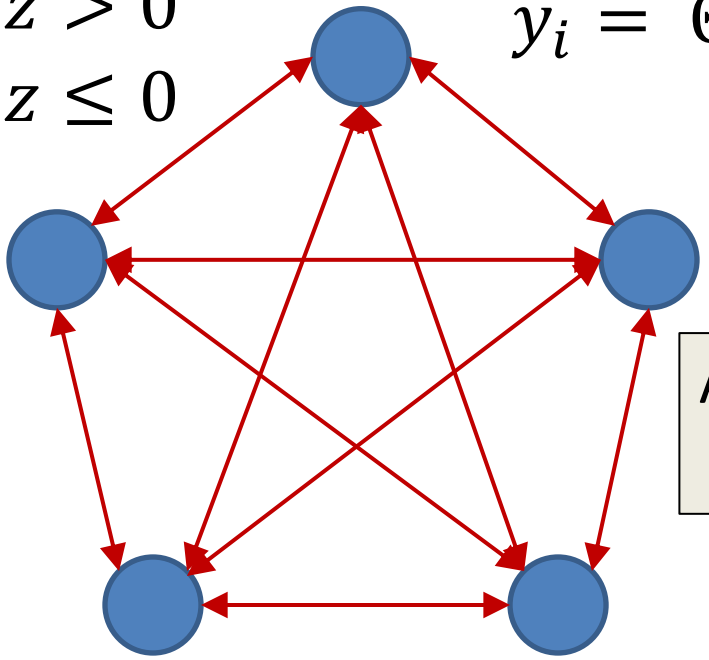
$$w_{ij} = w_{ji}$$

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

# Hopfield Net

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

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

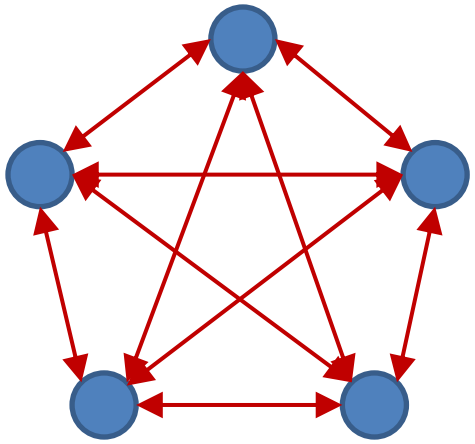


A symmetric network:

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

- 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

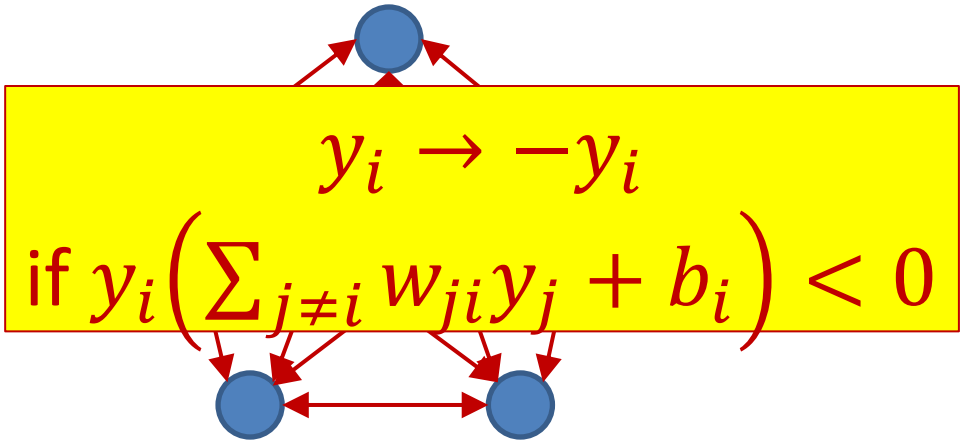


$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

- 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

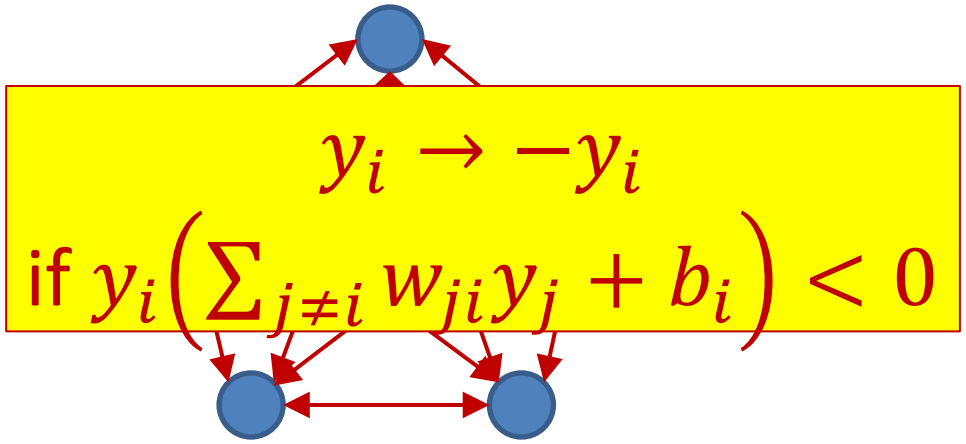



$$y_i = \Theta\left(\sum_{j\neq i} w_{ji}y_j + b_i\right)$$

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

- 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




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

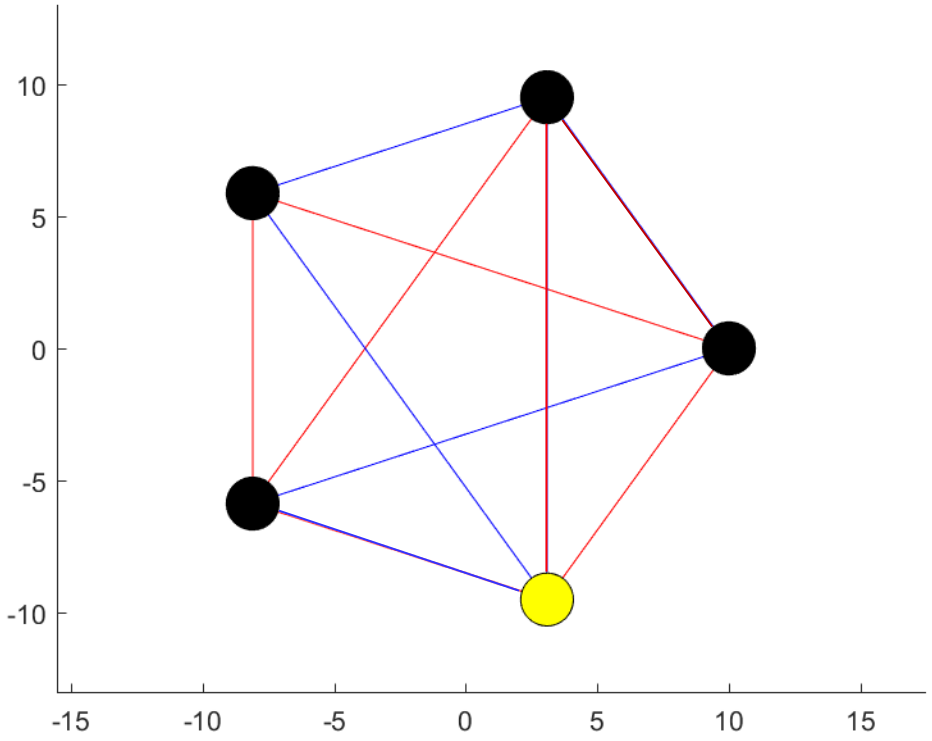
$$\Theta(z) = \begin{cases} +1 \; if \; z > 0 \\ -1 \; if \; z \leq 0 \end{cases}$$

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!

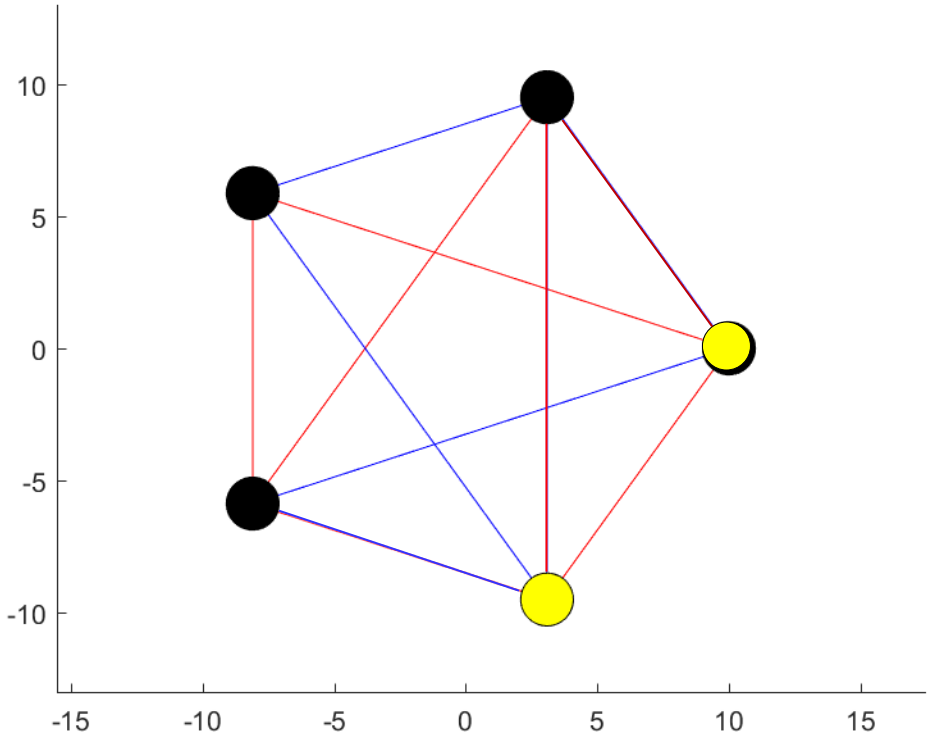
- es a "field" $\sum_{j\neq i} w_{ji}y_j + b_i$
- s own sign, it does not respond
- If the sign of the field opposes its own sign, it "flips" to match the sign of the field

# Example



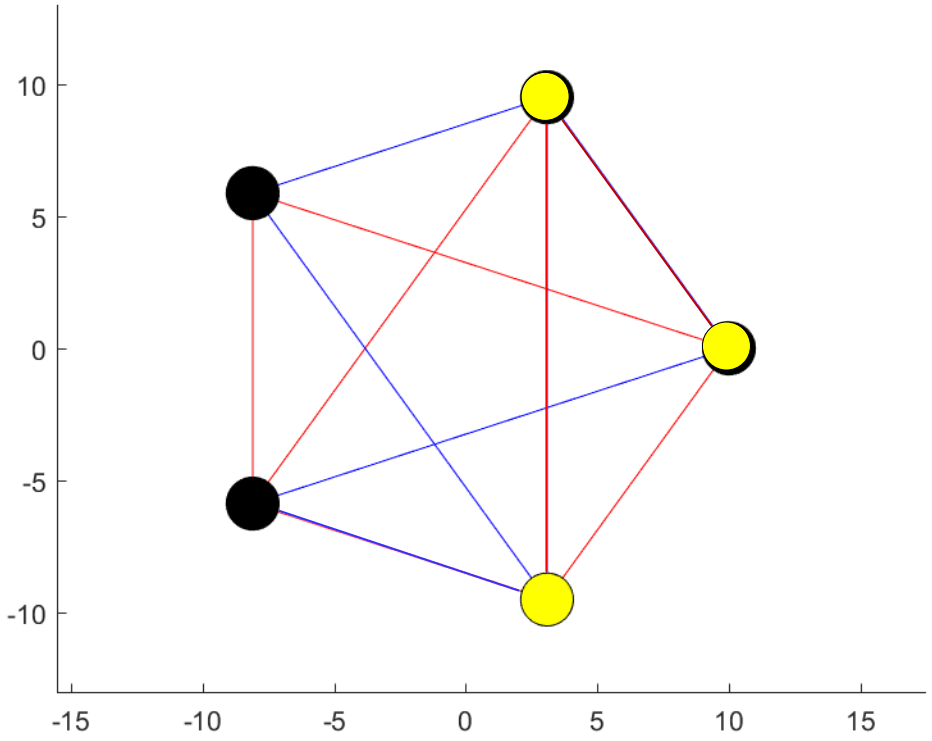

- Red edges are +1,  blue edges are -1
- Yellow nodes are -1, black nodes are +1

# Example



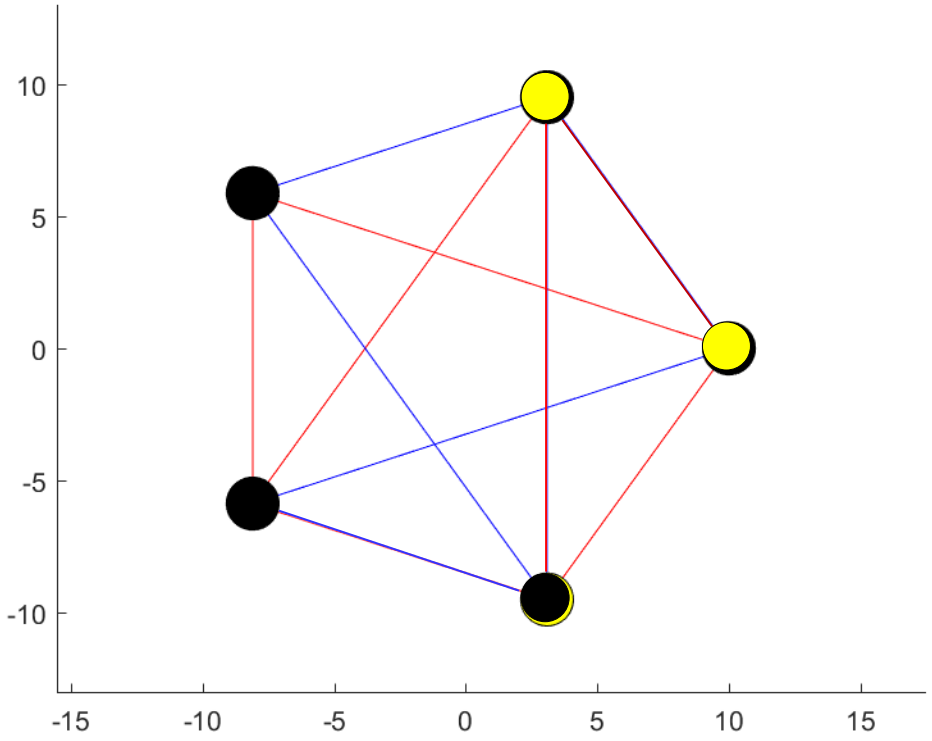

- Red edges are +1,  blue edges are -1
- Yellow nodes are -1, black nodes are +1

# Example



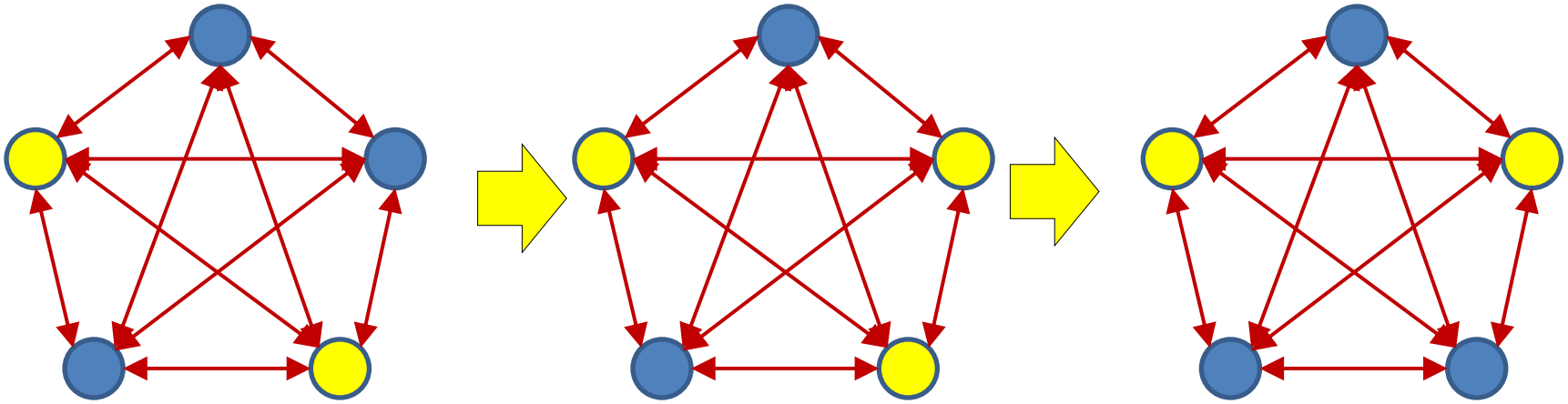

- Red edges are +1,  blue edges are -1
- Yellow nodes are -1, black nodes are +1

# Example




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

# Loopy network

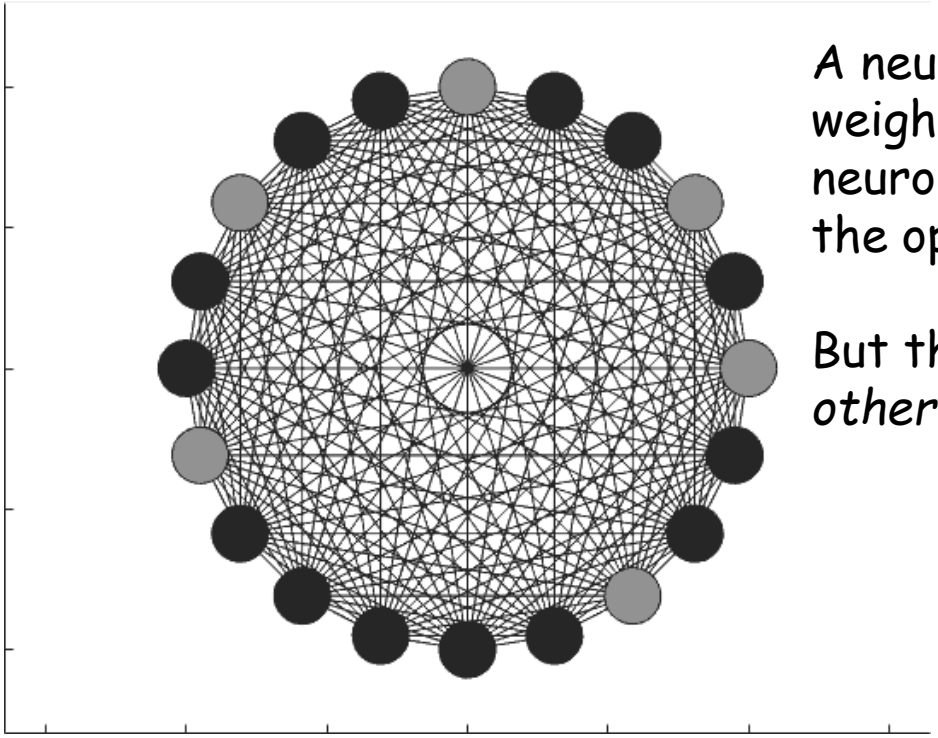


- If the sign of the field at any neuron opposes its own sign, it “flips” to match the field
  - Which will change the field at other nodes
    - Which may then flip
      - Which may cause other neurons including the first one to flip…
        - » And so on…

# 20 evolutions of a loopy net

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

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

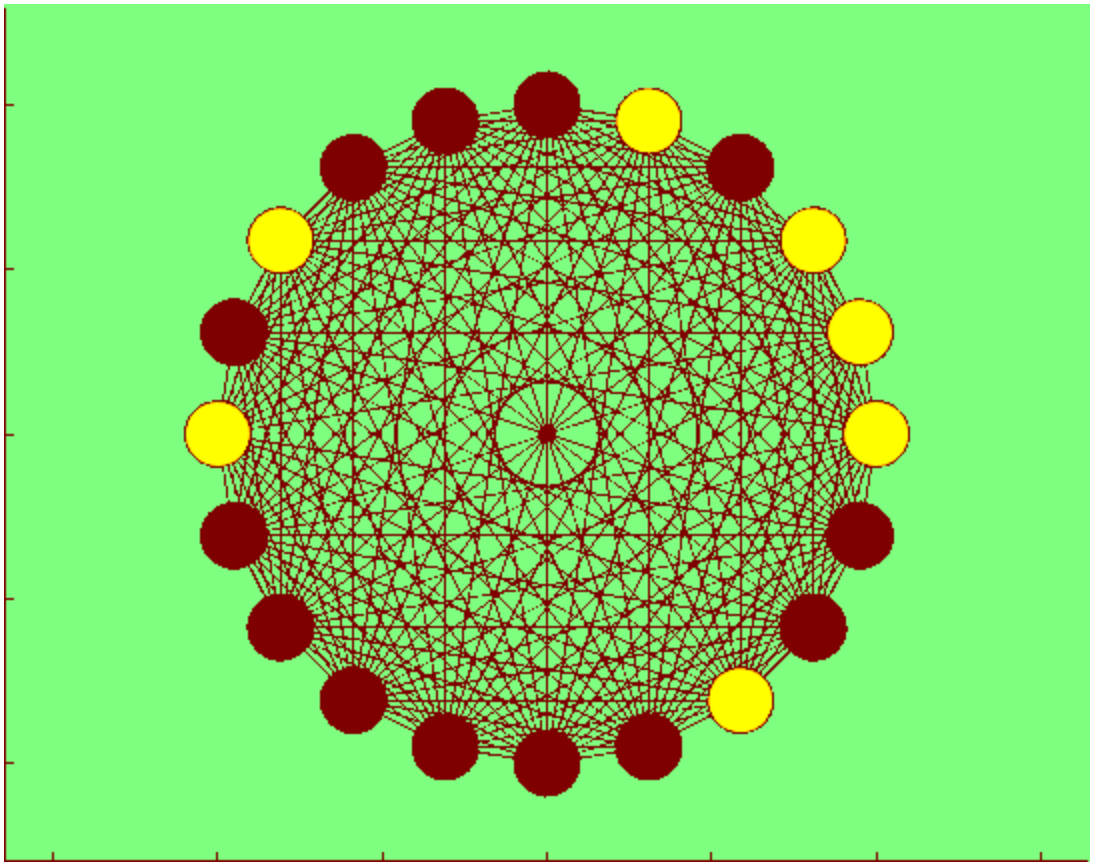


A neuron "flips" if weighted sum of other neuron's outputs is of the opposite sign

But this may cause *other* neurons to flip!

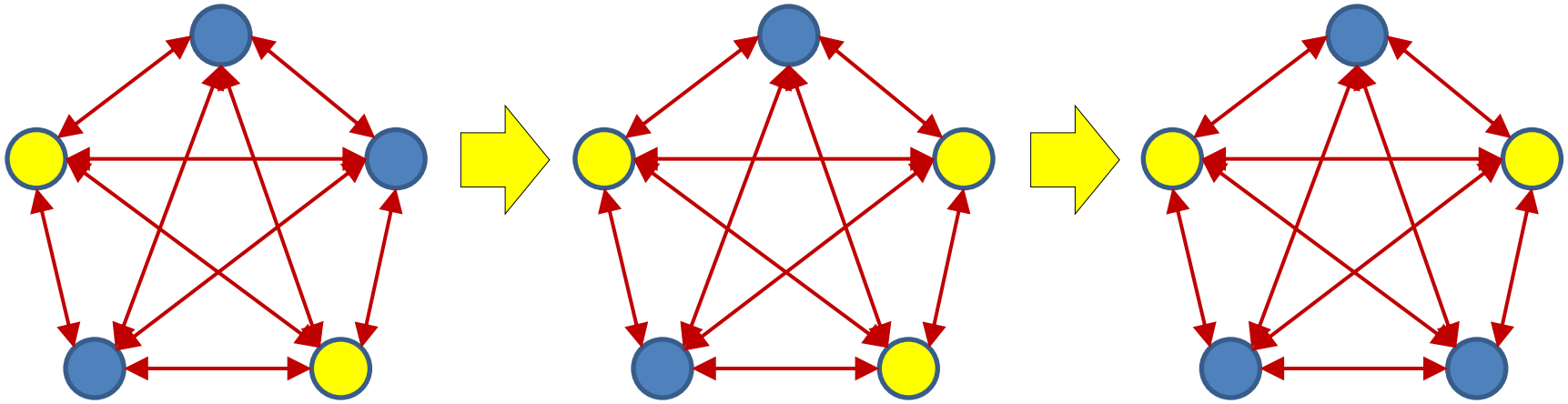
- All neurons which do not "align" with the local field "flip"

# 120 evolutions of a loopy net



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

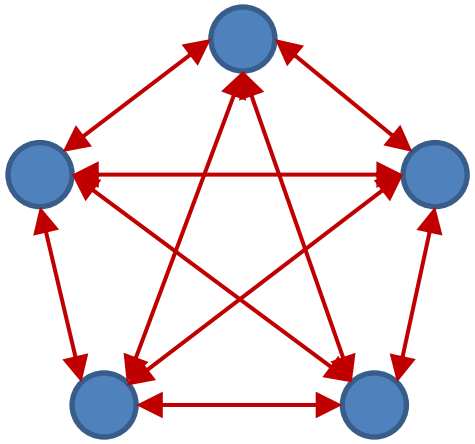
# Loopy network



- 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



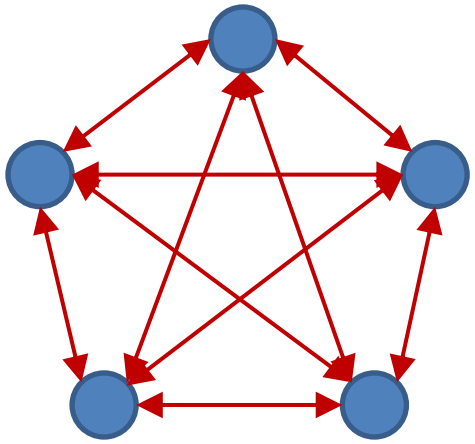
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

- 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 $y_i^+ = -y_i^-$

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

  - 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 $w_{ii} = 0$

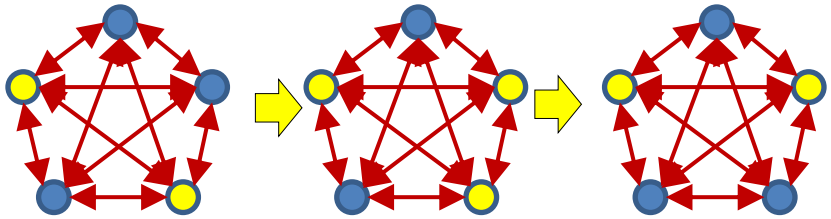
- For any unit $k$ that "flips" because of the local field

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

- This is strictly positive

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

# Hopfield Net



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

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

- $D$ is bounded

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

- The minimum increment of $D$ in a flip is

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

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

# Poll 1

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

- True
- False

Hopfield networks will evolve continuously, forever

- True
- False

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

- True
- False

# Story so far

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

# Poll 1

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

- **True**
- False

**Hopfield networks will evolve continuously, forever**

- True
- **False**

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

- **True**
- False

# 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

# The Energy of a Hopfield Net

- Define the *Energy* of the network as

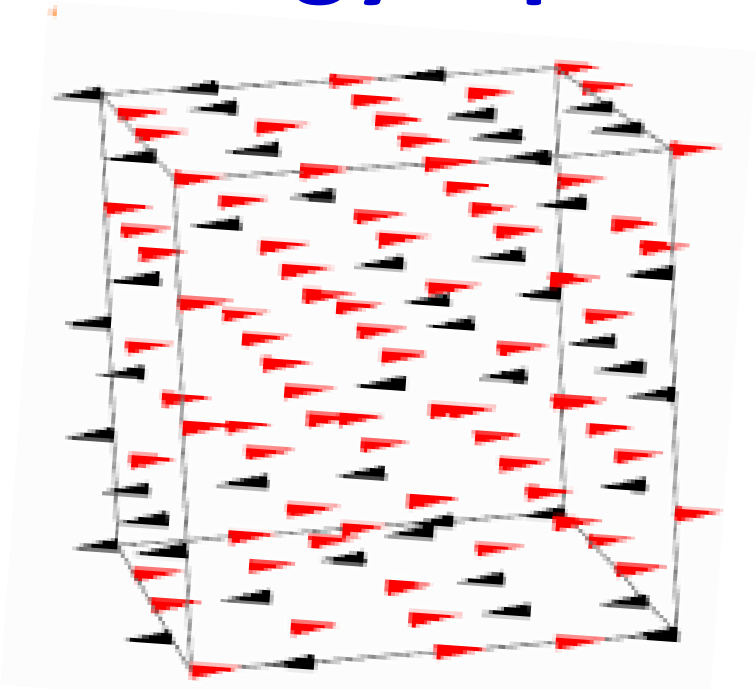
$$E = -\frac{1}{2}\left(\sum_{i,j\neq i} w_{ij}y_i y_j - \sum_i b_i y_i\right)$$

  - Just 0.5 times the negative of $D$

- The evolution of a Hopfield network constantly decreases its energy

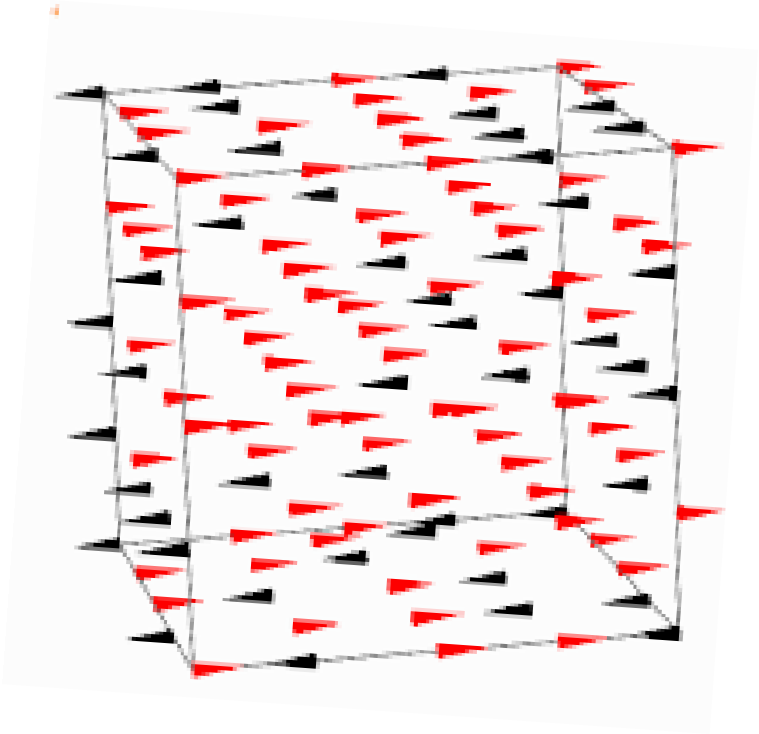
- Where did this "energy" concept suddenly sprout from?

# 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:

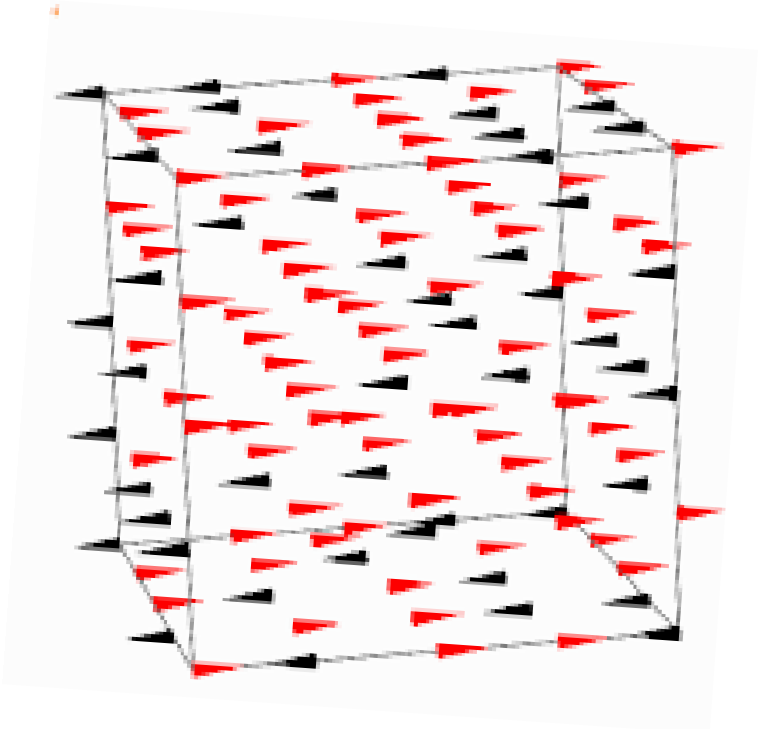
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

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:

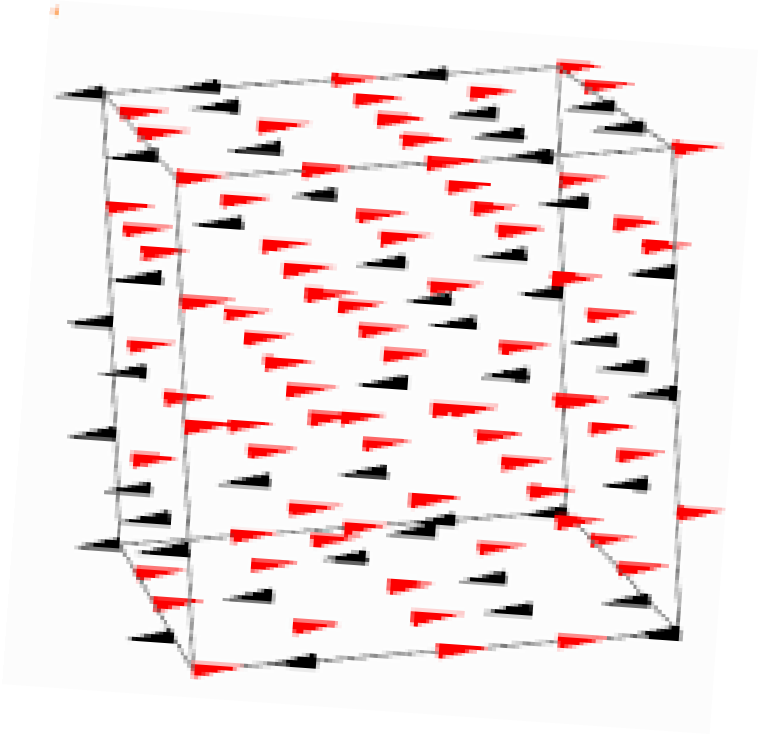
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

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:

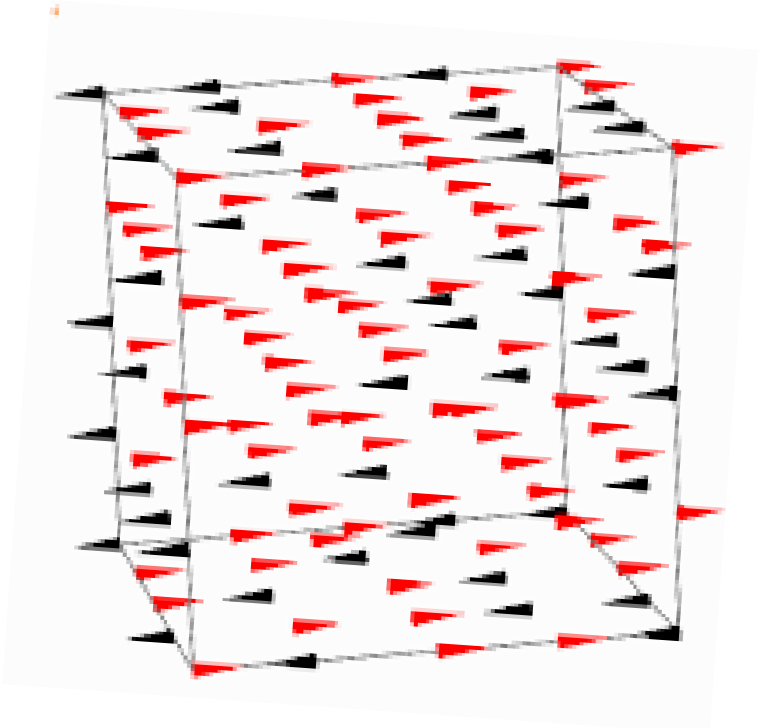
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

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???

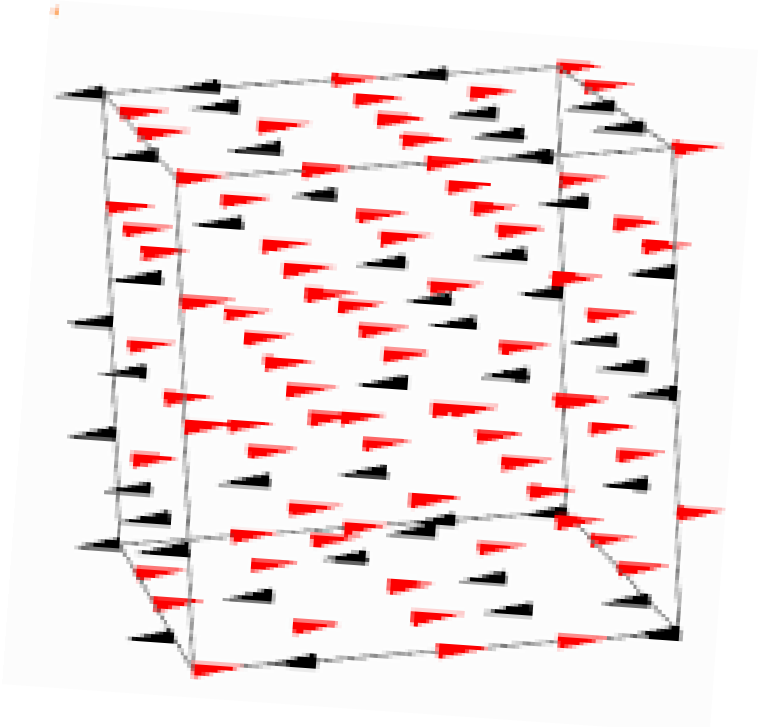
Total field at current dipole:

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

Response of current dipole

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

# 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

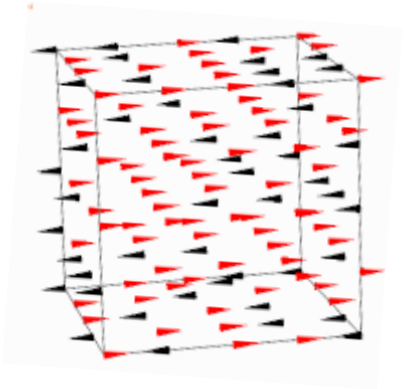
$$x_i = \begin{cases} x_i \; if \; sign\big(x_i \, f(p_i)\big) = 1 \\ -x_i \; otherwise \end{cases}$$

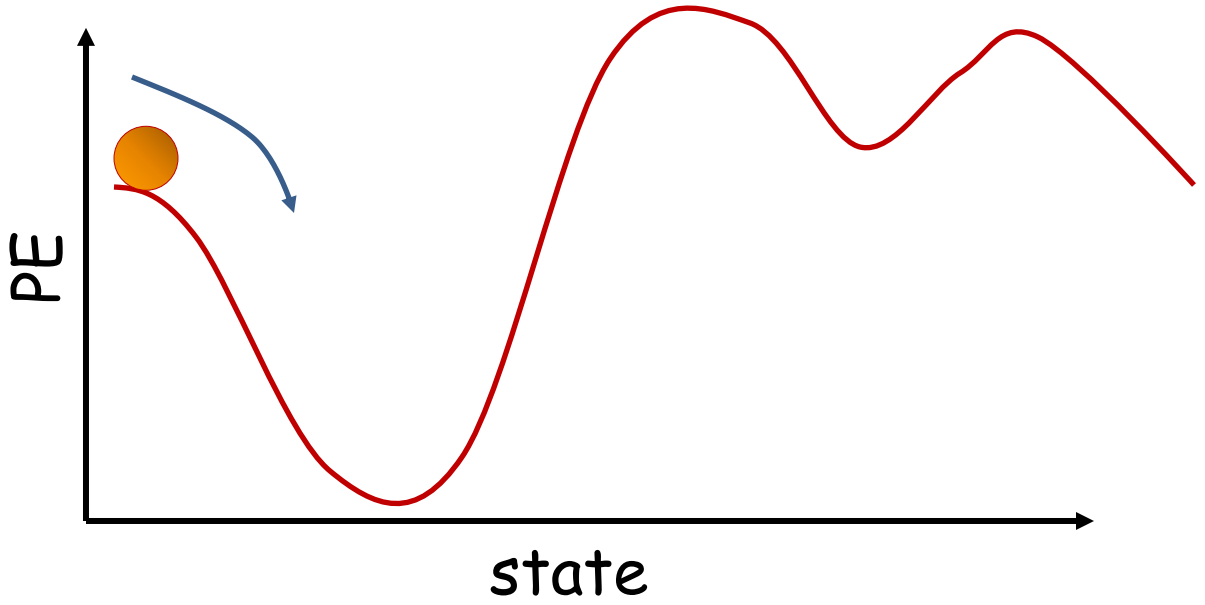
- The "Hamiltonian" (total energy) of the system

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

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

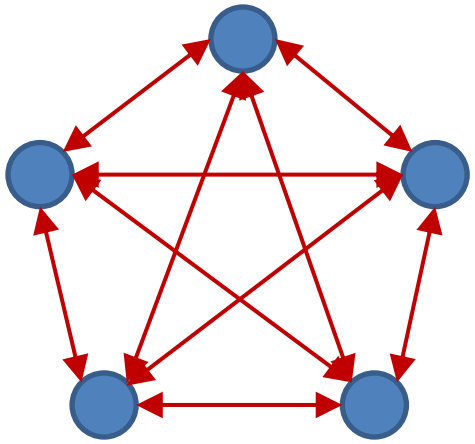
# Spin Glasses






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

# Hopfield Network



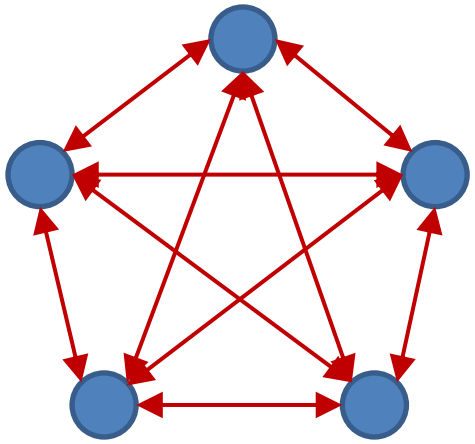
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

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

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

# Hopfield Network



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

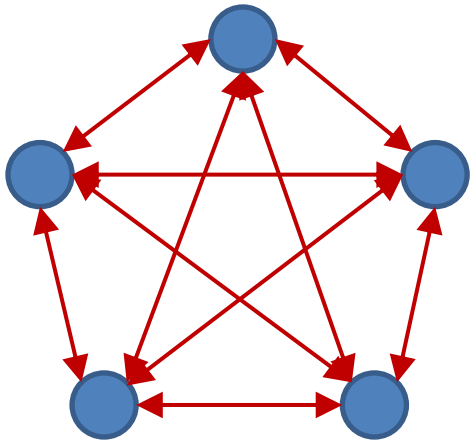
$$\Theta(z) = \begin{cases} +1 \; if \; z > 0 \\ -1 \; if \; z \leq 0 \end{cases}$$

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

We will not always explicitly show the bias

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

# Hopfield Network



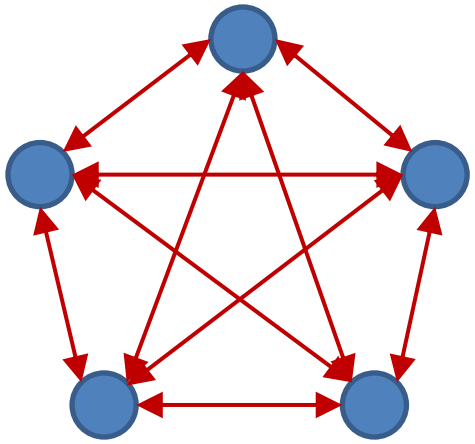
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j\right)$$

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

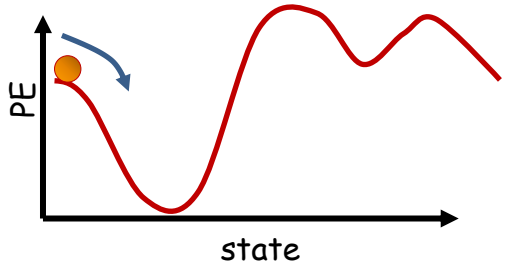
$$E = -\frac{1}{2} \sum_{i,j<i} w_{ij} y_i y_j$$

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

# Evolution

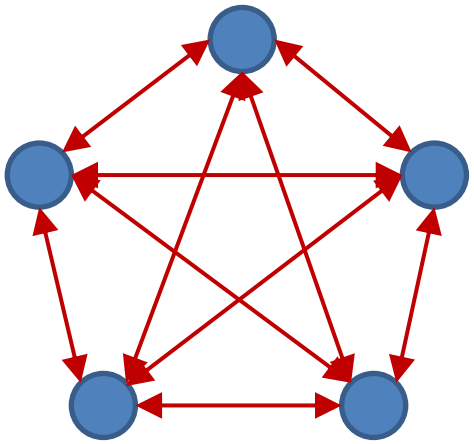


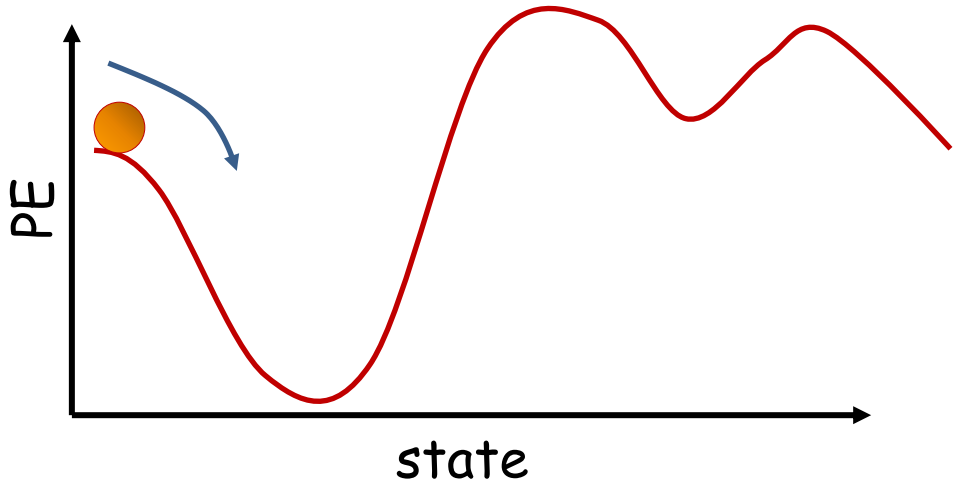
$$E = -\frac{1}{2}\sum_{i,j<i} w_{ij} y_i y_j$$




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

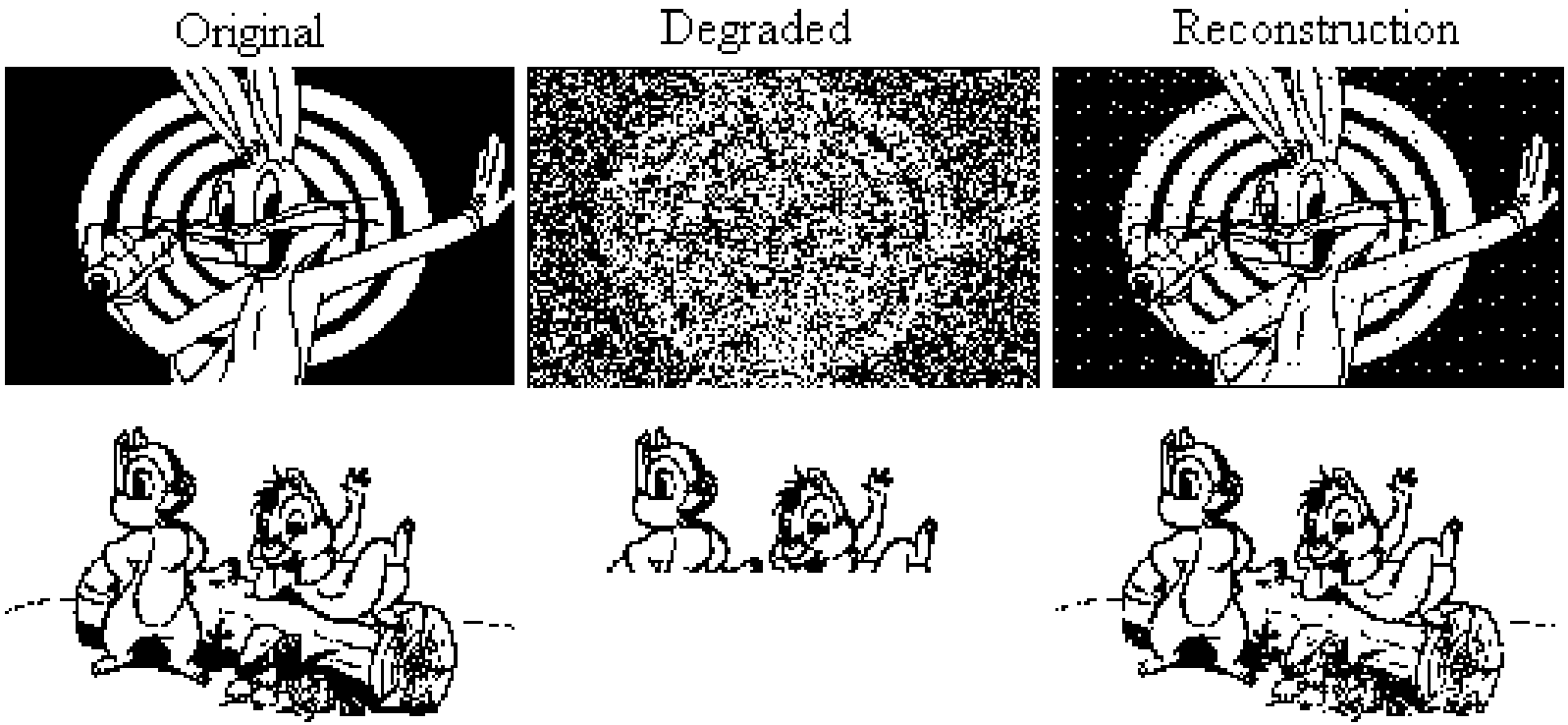
# Content-addressable memory






- 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***

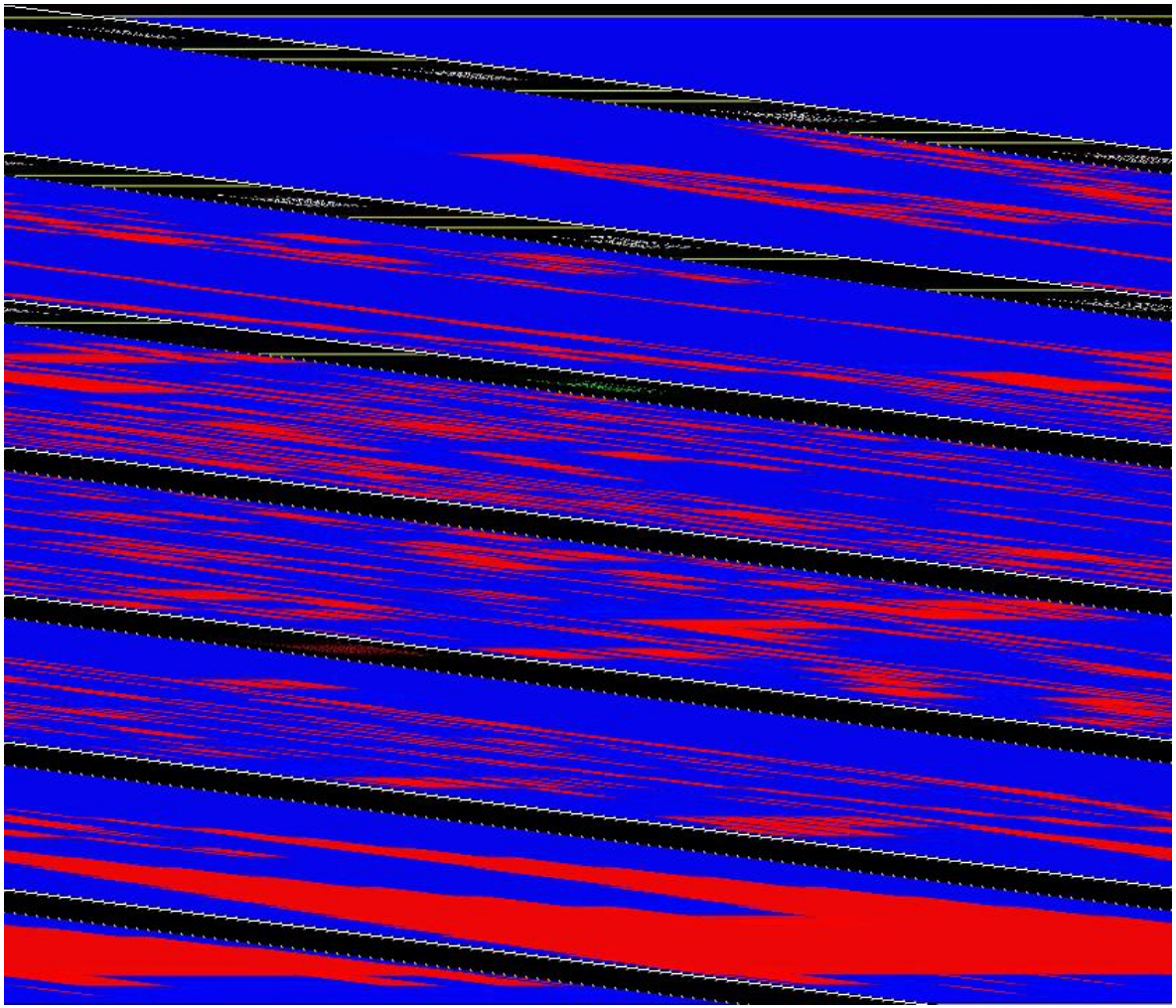
# Examples: Content addressable memory




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

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

# Hopfield net examples



# Computational algorithm

1. Initialize network with initial pattern

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

2. Iterate until convergence

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

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

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

does not change significantly any more

# Computational algorithm

1. Initialize network with initial pattern

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

2. Iterate until convergence

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

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

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

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

does not change significantly any more

# Story so far

- A Hopfield network is a loopy binary network with symmetric connections
  - Neurons try to align themselves to the local field caused by other neurons
- Given an initial configuration, the patterns of neurons in the net will evolve until the "energy" of the network achieves a local minimum
  - The evolution will be monotonic in total energy
  - The dynamics of a Hopfield network mimic those of a spin glass
  - The network is symmetric: if a pattern $Y$ is a local minimum, so is $-Y$
- The network acts as a *content-addressable* memory
  - If you initialize the network with a somewhat damaged version of a local-minimum pattern, it will evolve into that pattern
  - Effectively "recalling" the correct pattern, from a damaged/incomplete version

# Poll 2

Mark all that are correct about Hopfield nets

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

# Poll 2

**Mark all that are correct about Hopfield nets**

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

# Issues

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

# Issues

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

- How many patterns can we store?

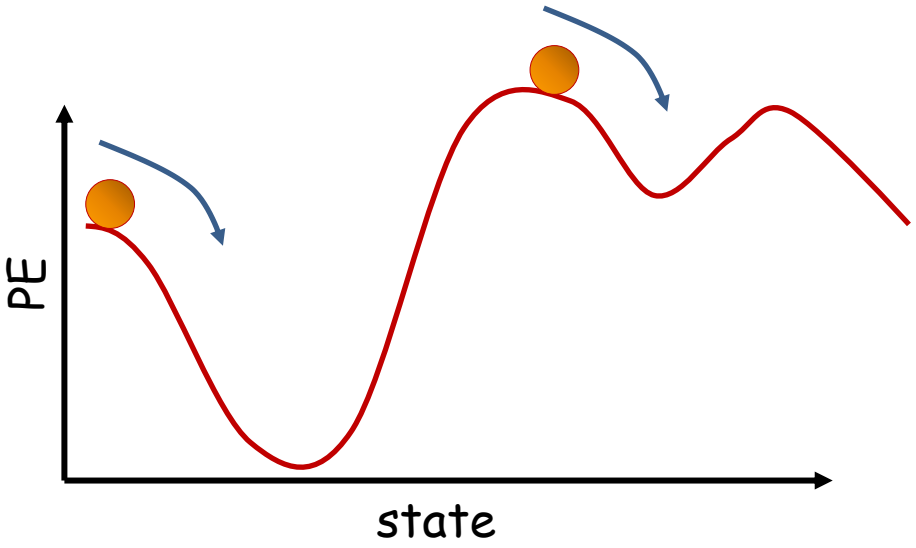
- How to “retrieve” patterns better..

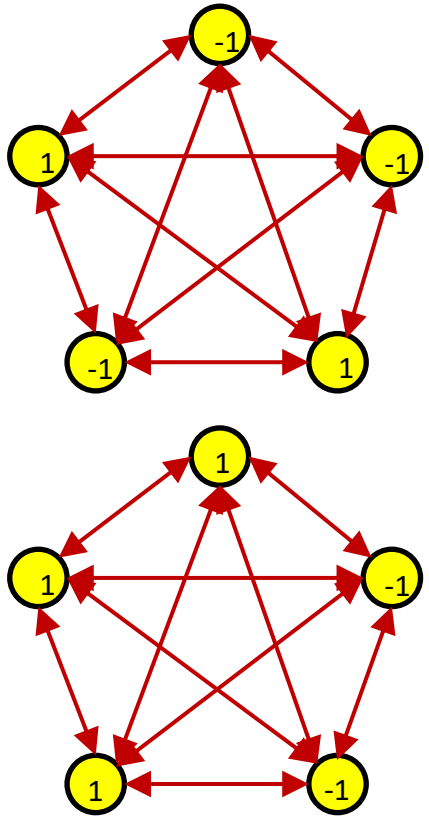
# How do we remember a *specific* pattern?

- How do we teach a network to “remember” this image

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

# Memorized patterns are stable Energy states

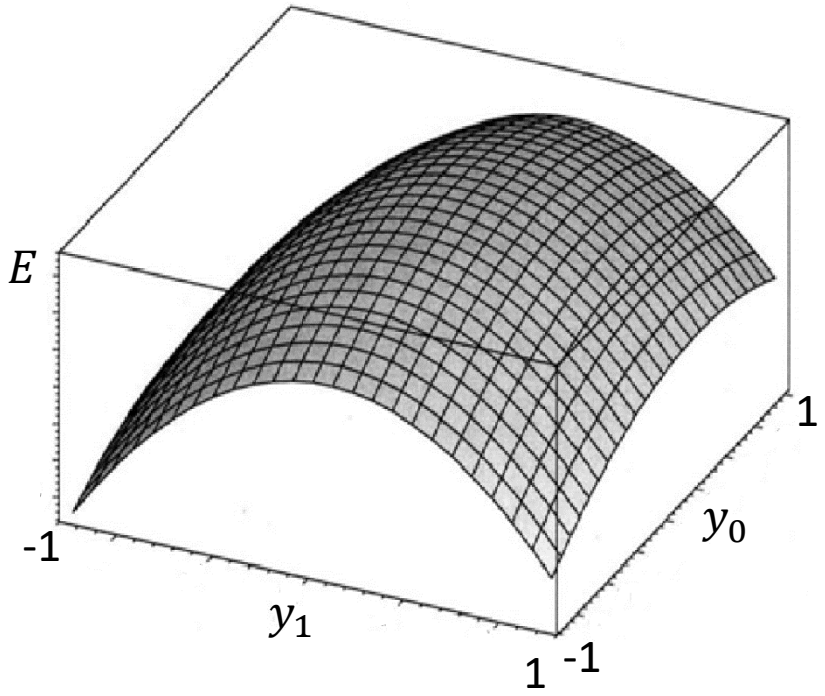



- The energy contour is a function of weights $W$
- Memories are local minima in energy surface
- There can be multiple of them
  - How? The Energy function is quadratic, how does it have multiple minima?

# The Energy function

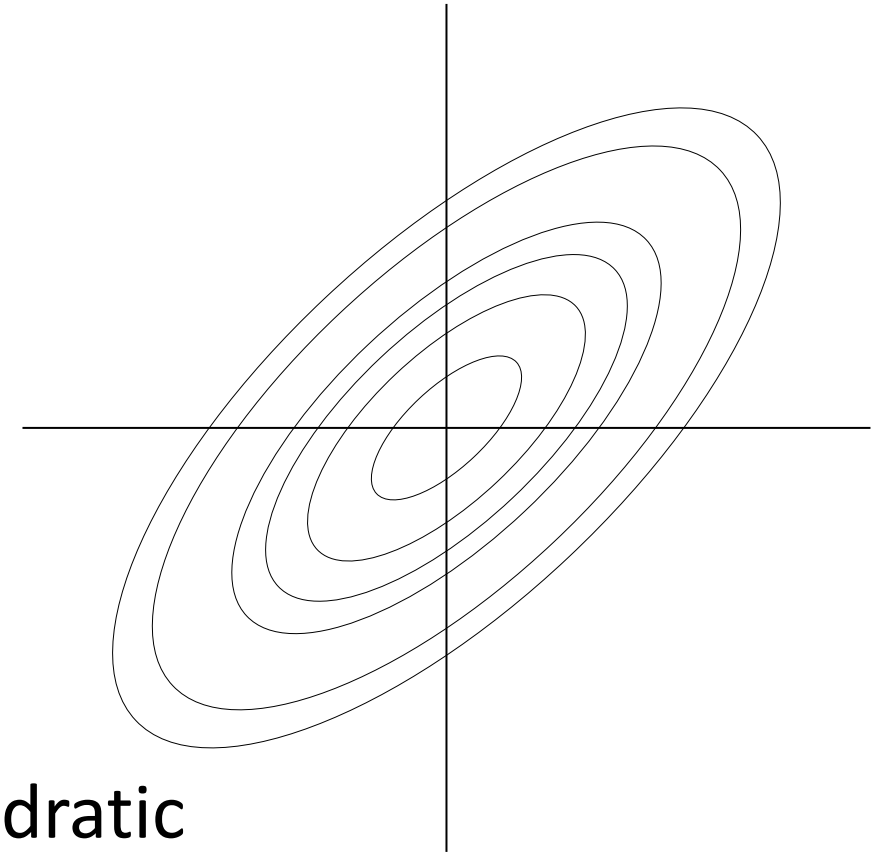
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$




- $E$ is a concave quadratic

# The Energy function

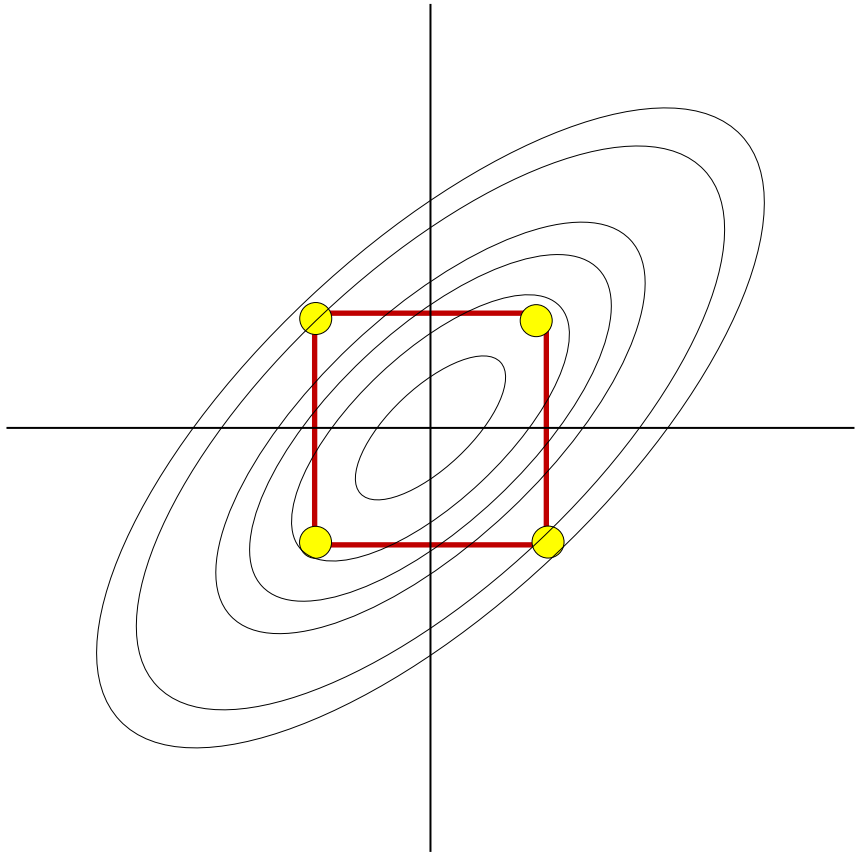
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$



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

# The energy function

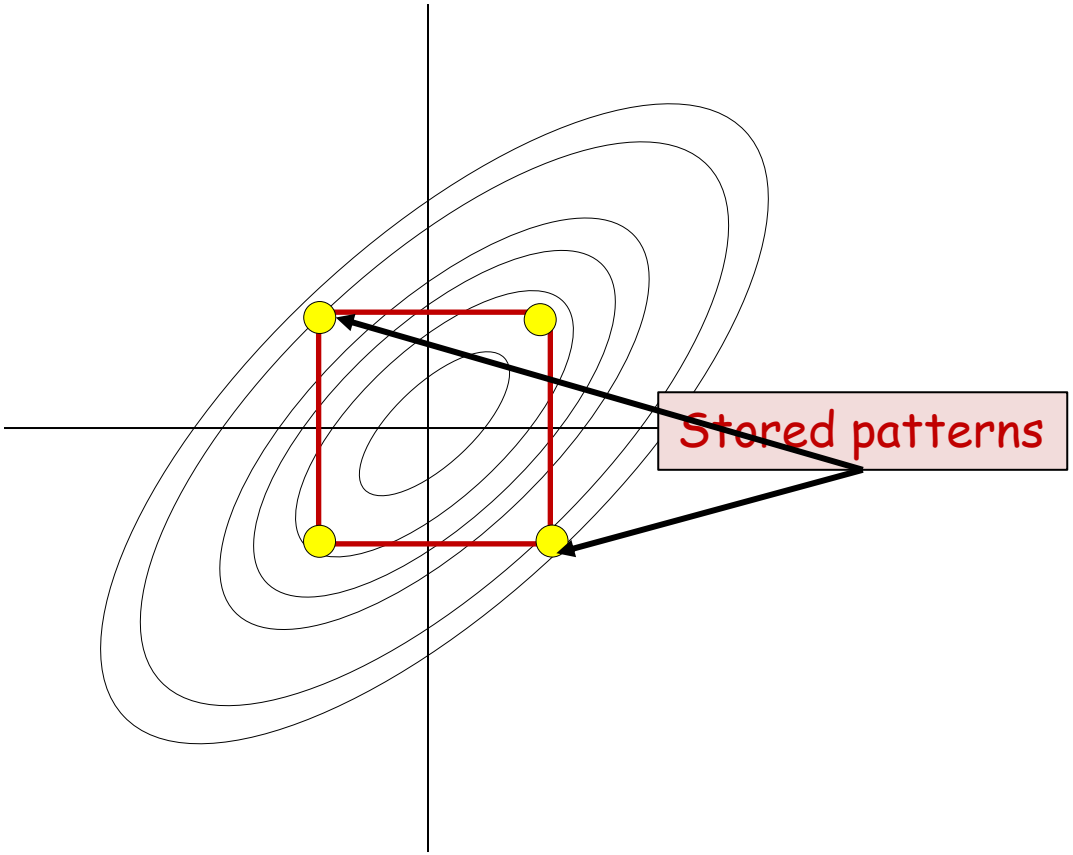
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$



- $E$ 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 $\pm 1$
  - I.e. $y$ lies on the corners of the unit hypercube

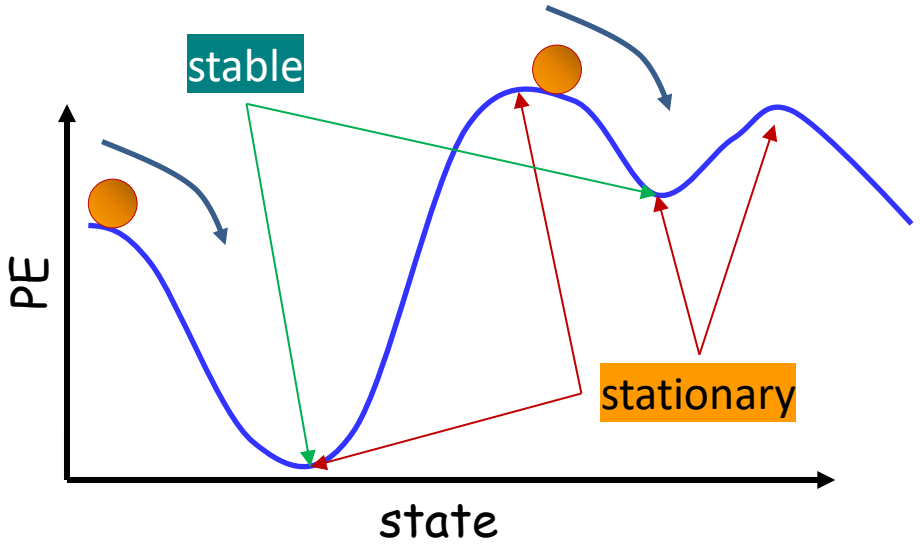
# The energy function

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



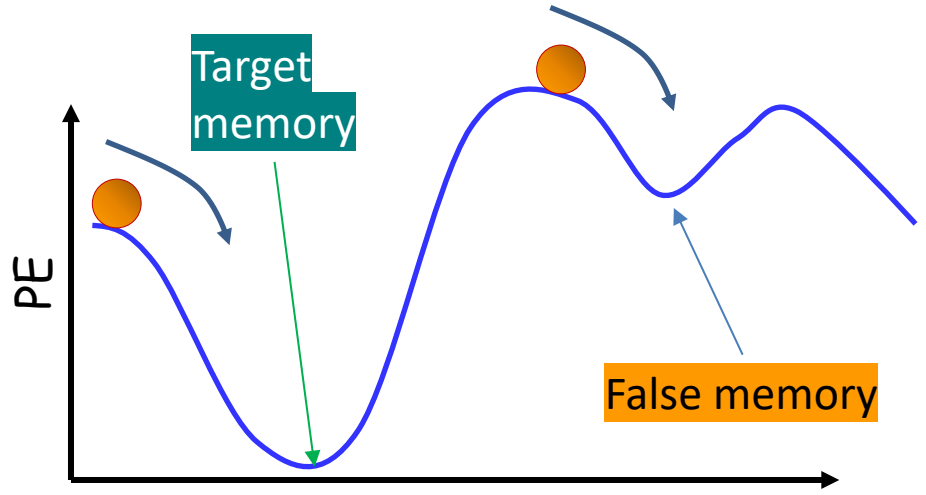

- The stored values of $\mathbf{y}$ are the ones where all adjacent corners are lower on the quadratic
- We can have multiple of them

# Requirements for memory



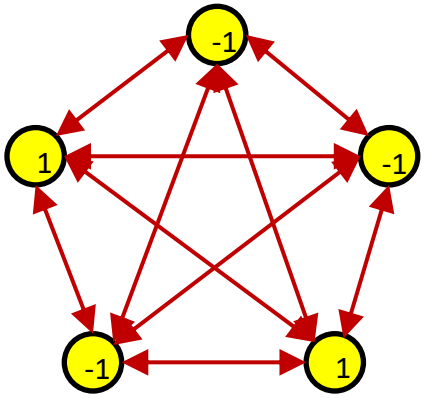

- **Stationarity**: A system in that state should not change spontaneously
  - Wherever the gradient of the energy contour is 0
- **Stability**: If we perturb the system slightly it must return to the memory state
  - Local minima in energy

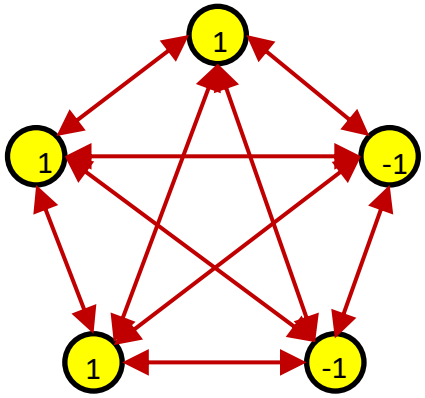
# The problem of 'creating' memories




- We create a memory by choosing the weights $W$ such that the energy contour has local minima at the target patterns and nowhere else

# Storing a pattern

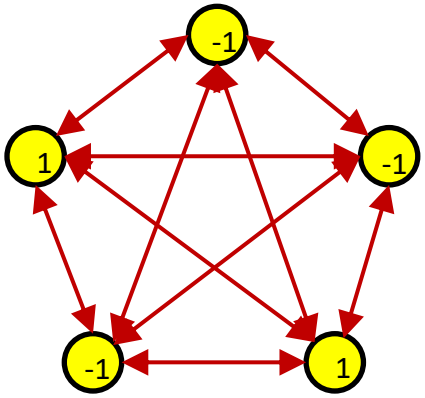



$$E = -\sum_{i}\sum_{j<i} w_{ji} y_j y_i$$

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

# Storing specific patterns

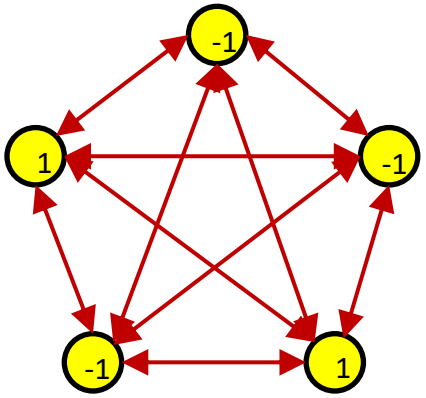



- Storing 1 pattern: We want

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

- This is a stationary pattern

# Storing specific patterns




HEBBIAN LEARNING:

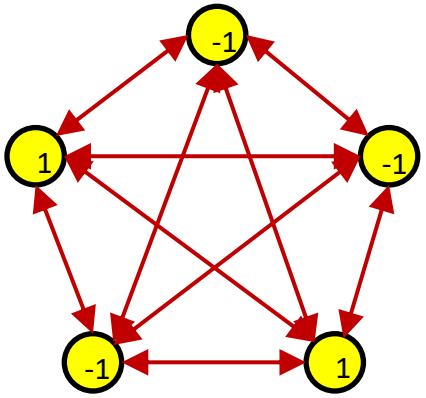
$$w_{ji} = y_j y_i$$

- 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



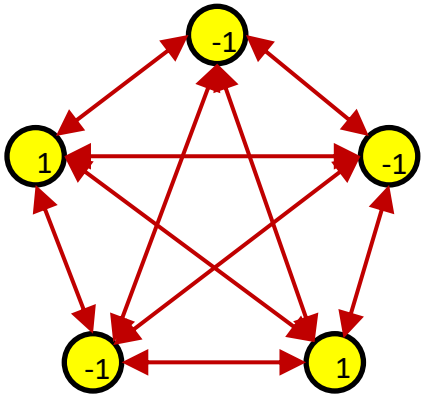

HEBBIAN LEARNING:

$$w_{ji} = y_j y_i$$

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

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

# Storing specific patterns




HEBBIAN LEARNING:

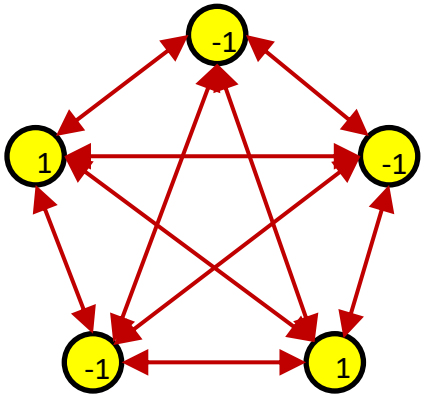
$$w_{ji} = y_j y_i$$

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:

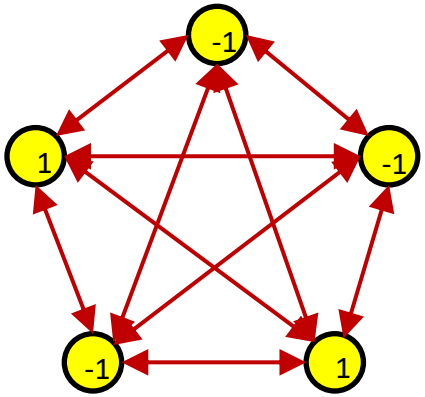
$$w_{ji} = y_j y_i$$

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

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

- This is the lowest possible energy value for the network for binary weights

# Storing specific patterns




HEBBIAN LEARNING:

$$w_{ji} = y_j y_i$$

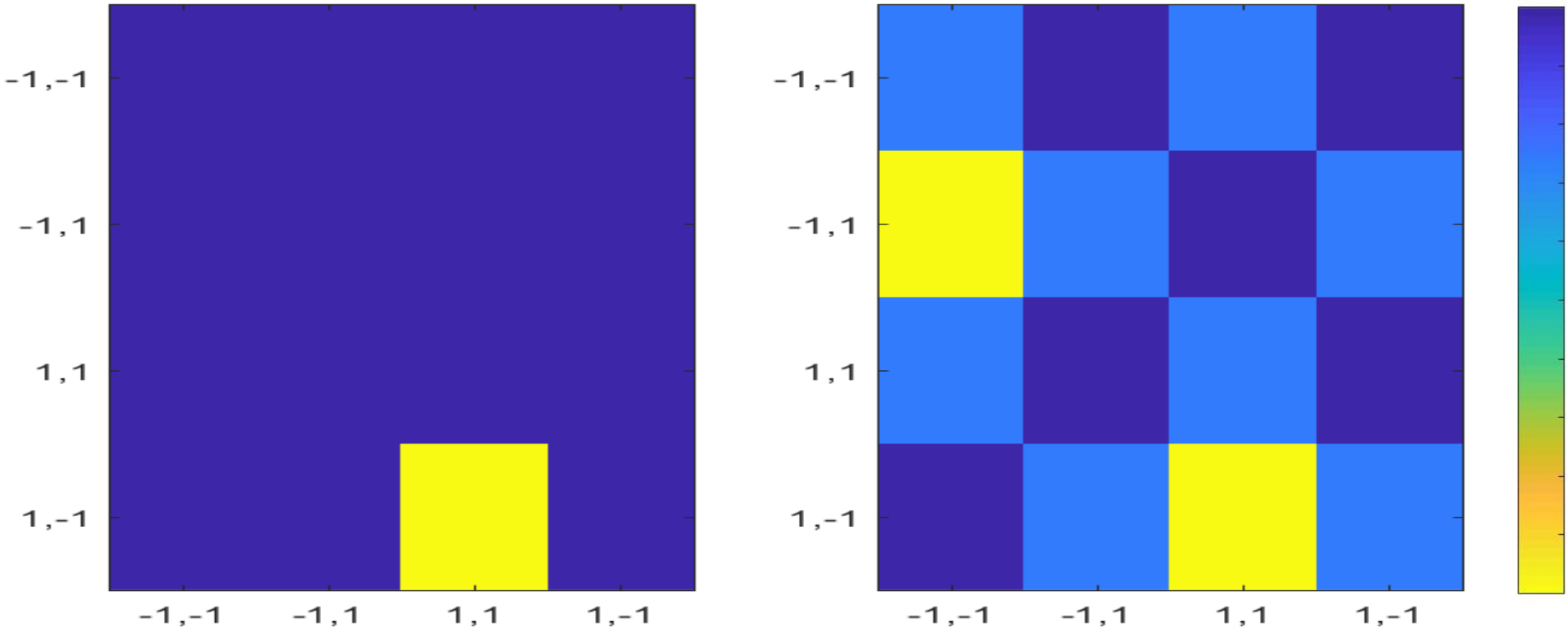
The pattern is *STABLE*

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

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

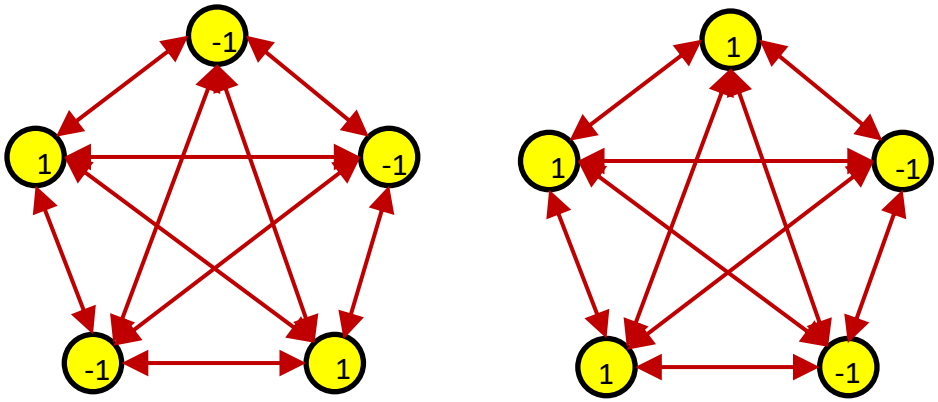
- This is the lowest possible energy value for the network for binary weights

# 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
  - In the absence of a bias, if P is a memory, -P is also a memory because $P^T W P = (-P)^T W (-P)$

# Storing multiple patterns

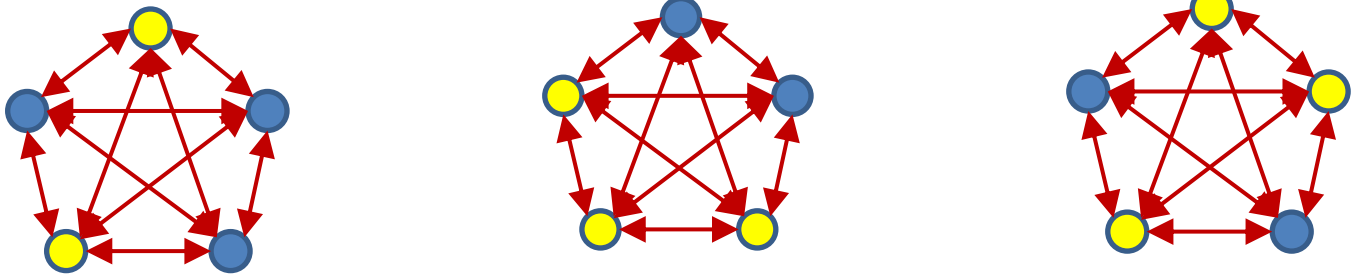



- To store *more* than one pattern

$$w_{ji} = \frac{1}{N} \sum_{\mathbf{y}_p \in \{\mathbf{y}_p\}} y_i^p y_j^p$$

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

# How many patterns can we store?

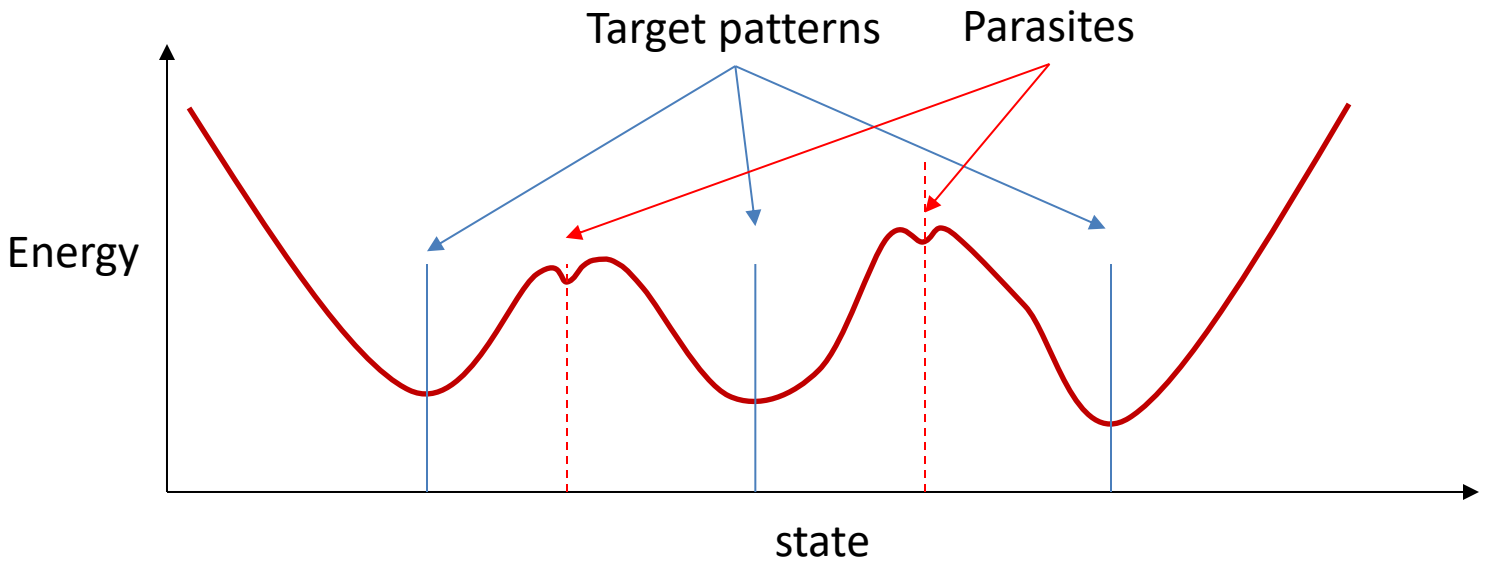


- **Hopfield**: For a network of $N$ neurons can store up to ~$0.15N$ random patterns through Hebbian learning
  - Provided they are "far" enough
- Where did this number come from?
  - Proof on slides

# Observations

- Can store up to 0.14N *random* (uncorrelated) patterns with moderate recall error (0.4%) using Hebbian learning
  - Many "parasitic" patterns
    - Undesired patterns that also become stable or attractors
- In reality, the net has 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

# Poll 3

Mark all that are true

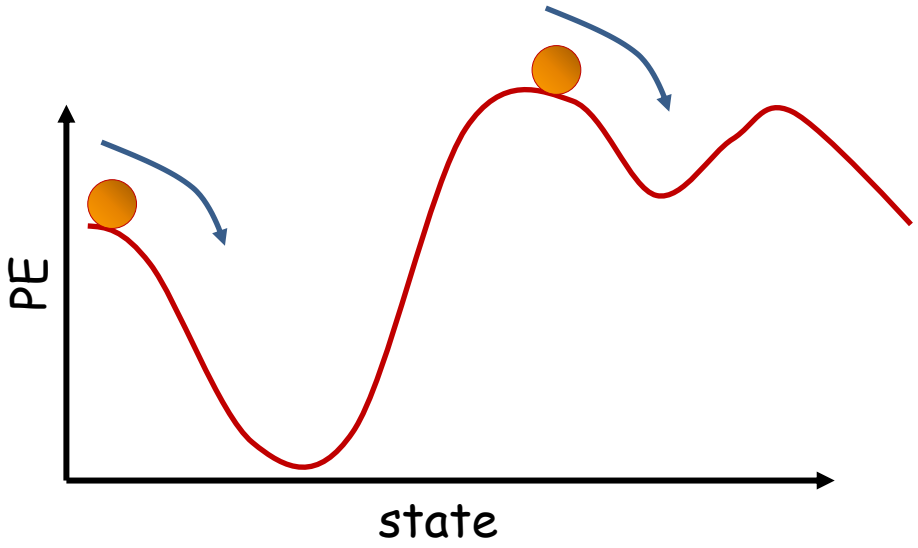
- We can try to “assign” memories to a Hopfield network through Hebbian learning of the weights matrix
- All patterns learned through Hebbian learning will be “remembered”
- The N-bit Hopfield network has the capacity to remember up to 0.14N patterns

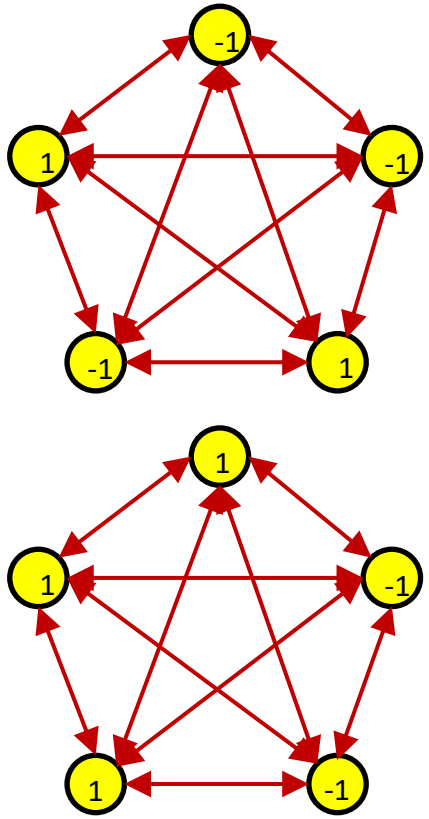
# Poll 3

**Mark all that are true**

- **We can try to "assign" memories to a Hopfield network through Hebbian learning of the weights matrix**
- All patterns learned through Hebbian learning will be "remembered"
- The N-bit Hopfield network has the capacity to remember up to 0.14N patterns

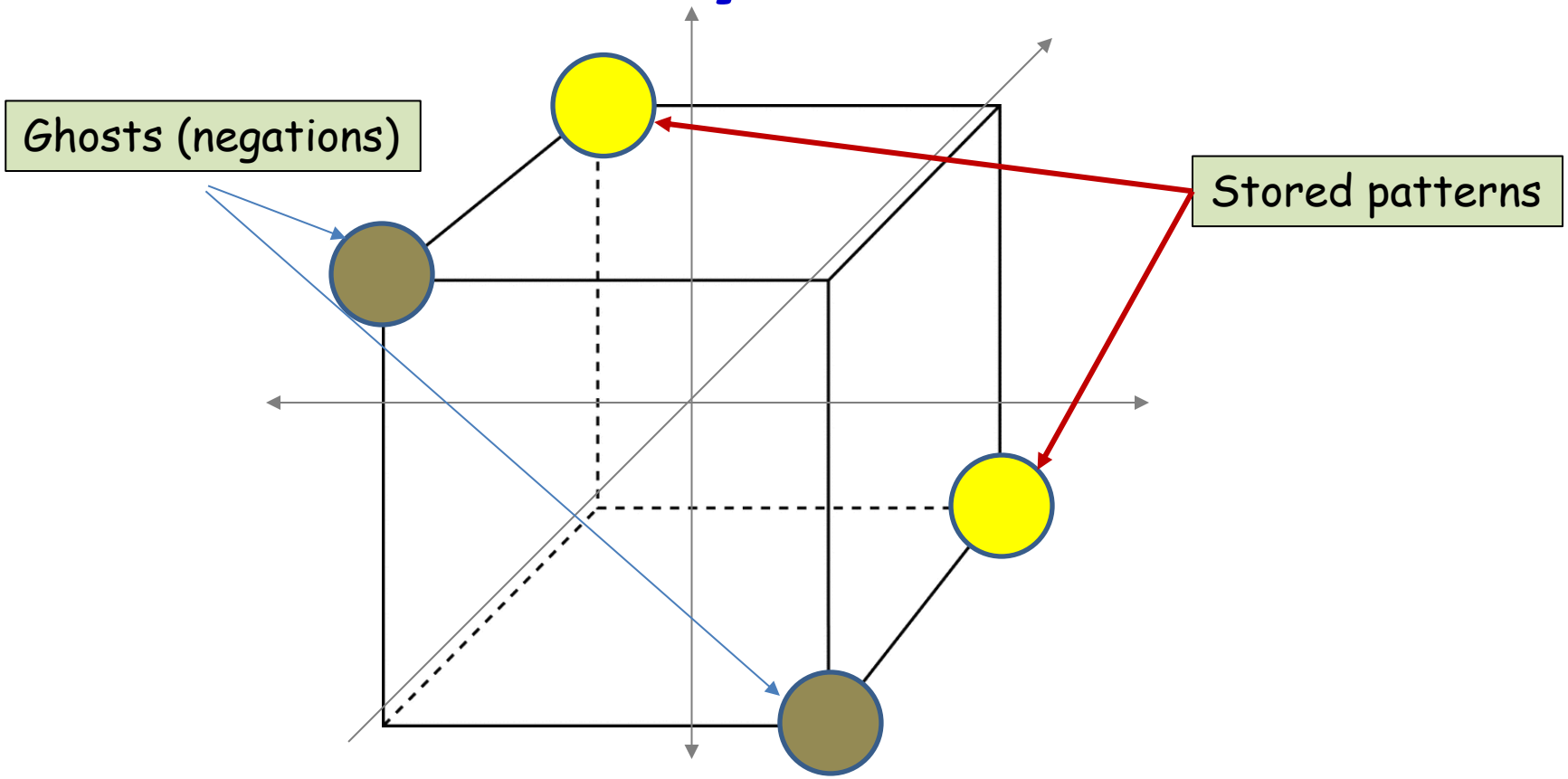
# A network can store *multiple* patterns




- 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 $-P$
- **How many patterns can we store intentionally?**

# Patterns you can store

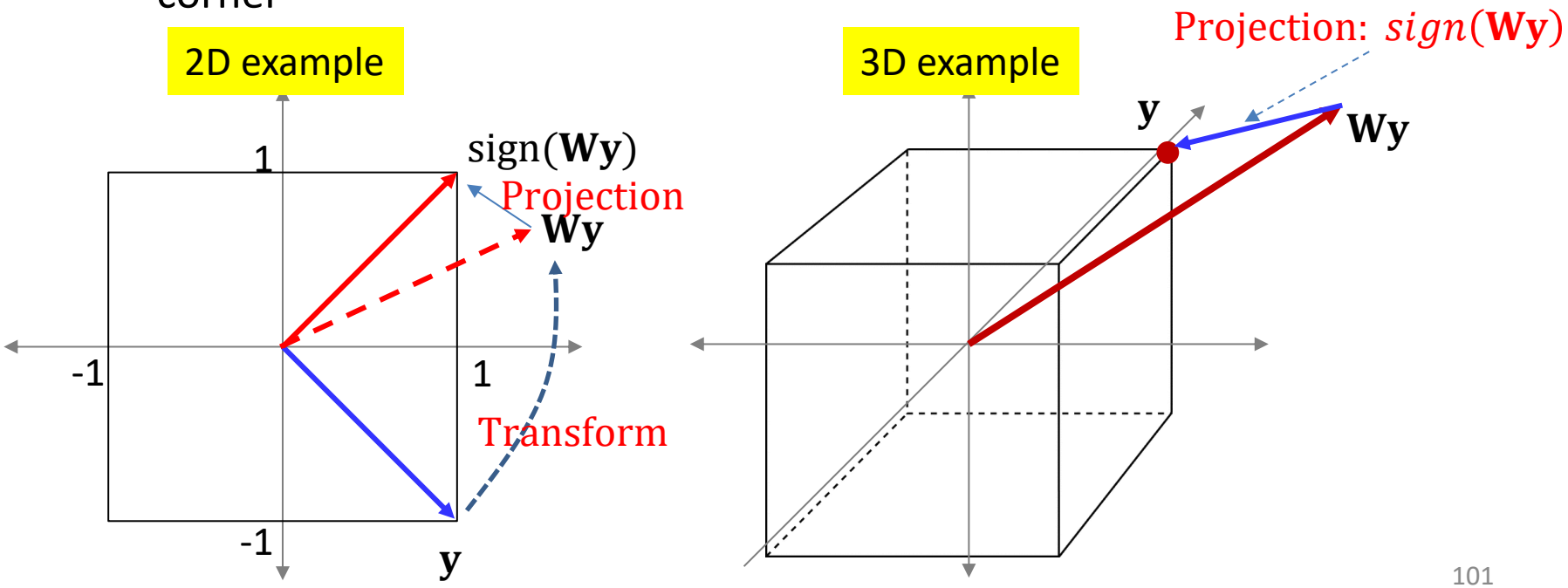



- All patterns are on the corners of a hypercube
  - If a pattern is stored, it's "ghost" is stored as well
  - Intuitively, patterns must ideally be maximally far apart

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



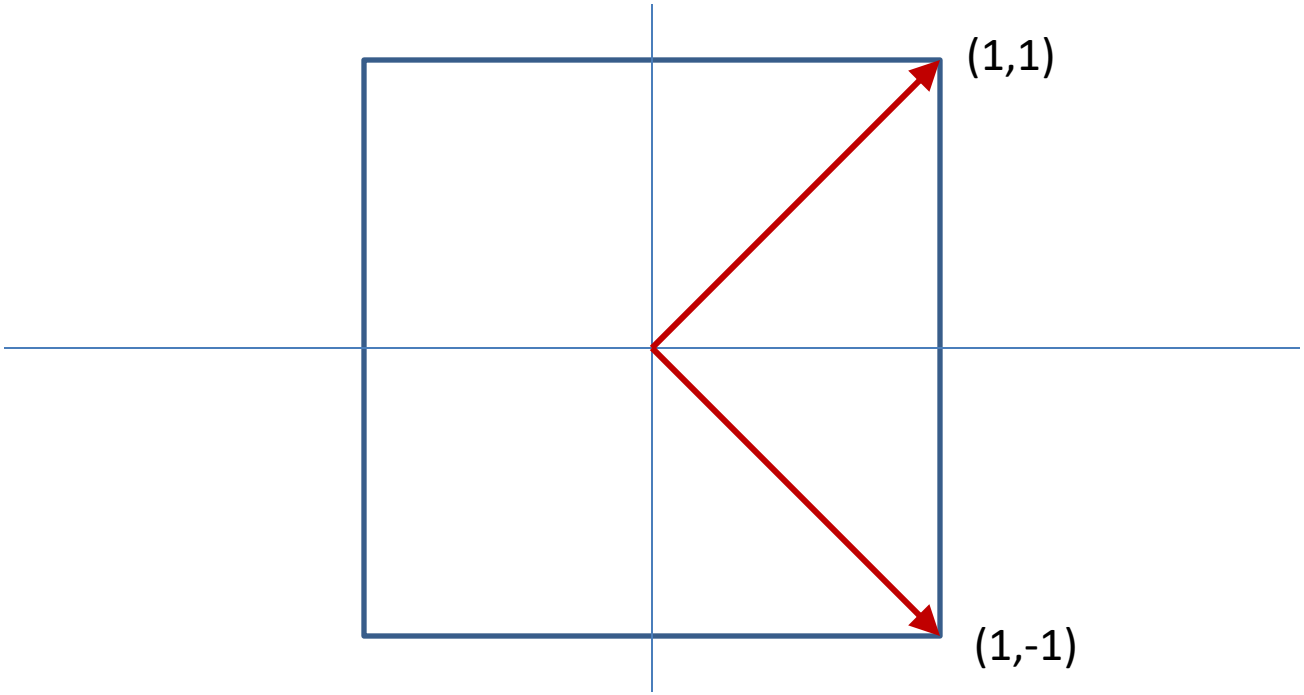
# Storing patterns

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




- 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 patterns

- Any (binary) eigen vector with a real eigen value is stored

$$\mathbf{y}_p \leftarrow sign(\mathbf{W}\mathbf{y}_p) = sign(\lambda \mathbf{y}_p) = \pm \mathbf{y}_p$$

- A square matrix $\mathbf{W}$ can have up to $N$ eigen vectors
  - So, we can "intentionally" store up to $N$ patterns
- Problem?

# Storing $N$ orthogonal patterns

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

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

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

$$= a_1\lambda_1\mathbf{y}_1 + a_2\lambda_2\mathbf{y}_2 + \cdots + a_N\lambda_N\mathbf{y}_N$$

- Many of these will have the form

$$\text{sign}(\mathbf{W}\mathbf{y}) = \mathbf{y}$$

- ***Spurious memories***
- *The fewer memories we store, and the more distant they are, the more likely we are to eliminate spurious memories*

# 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

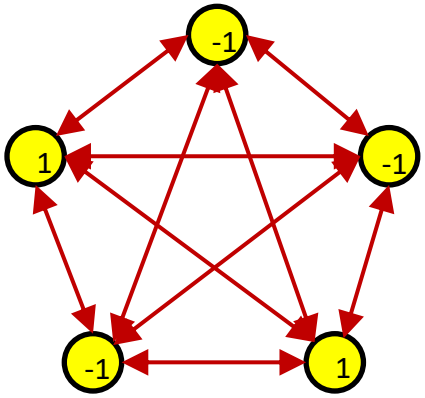
- With a network of $N$ units (i.e. $N$-bit patterns)
- The maximum number of stable patterns is actually *exponential* in $N$
  - McElice and Posner, 84'
  - E.g. when we had the H… …ase patterns, *all* patterns are stable

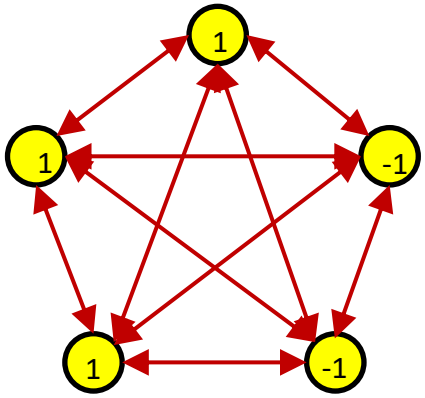
How do we find this network?

- 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?

# Storing a pattern

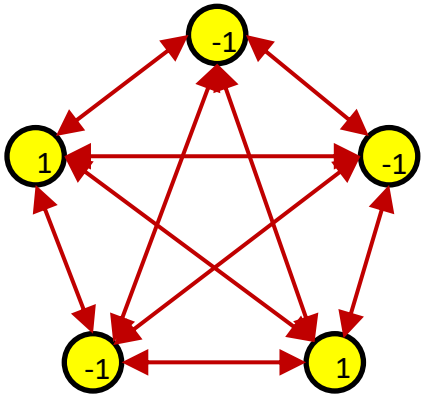


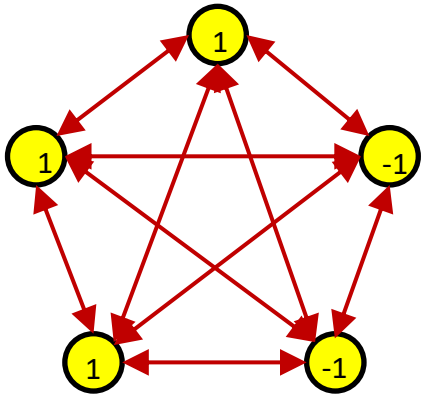




$$E = -\sum_{i}\sum_{j<i} w_{ji} y_j y_i$$

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

# Storing a pattern

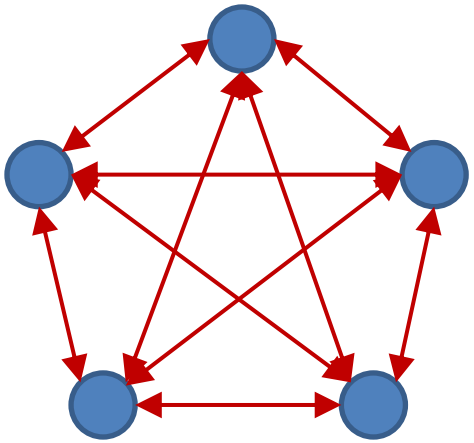



$$E = -\sum_{i}\sum_{j<i} w_{ji} y_j y_i$$

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

# Consider the energy function

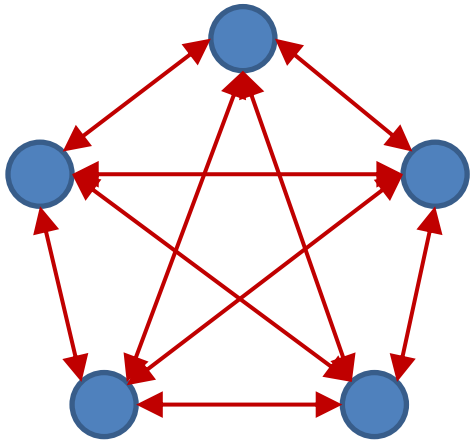


$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y}$$

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

# Estimating the Network

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



- Estimate $\mathbf{W}$ (and $\mathbf{b}$) such that
  - $E$ is minimized for $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_P$
  - $E$ 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)

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

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

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

- Simple gradient descent:

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

# Optimizing W

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

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

# Optimizing W

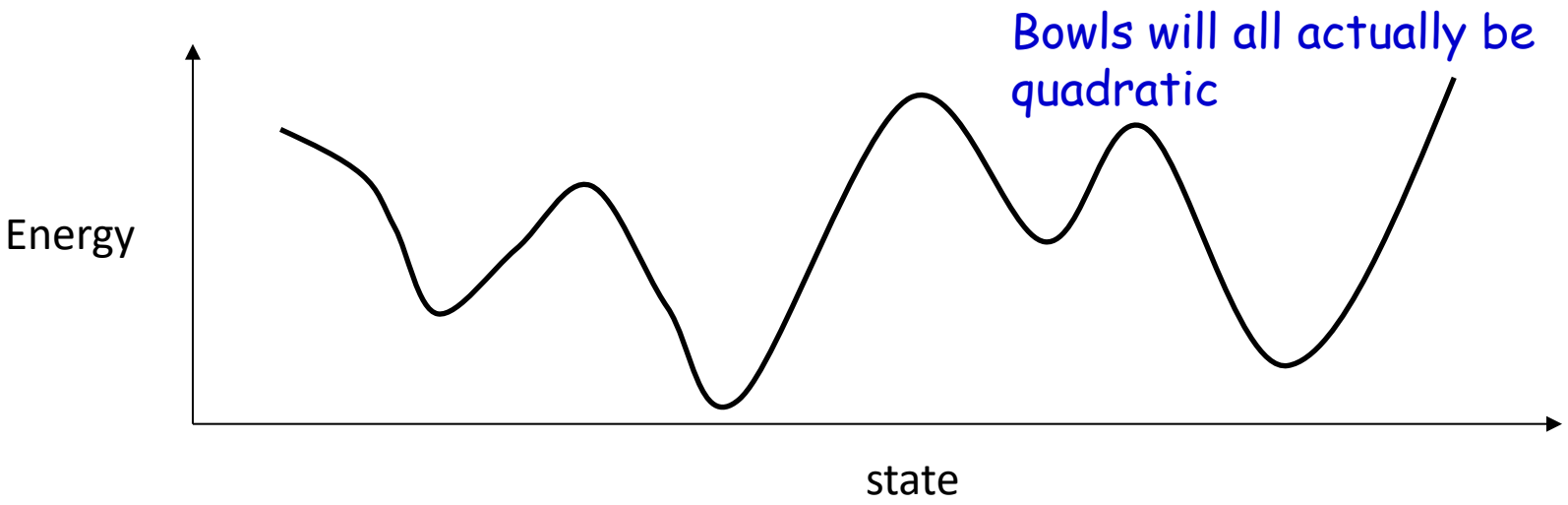
$$\mathbf{W} = \mathbf{W} + \eta \left( \sum_{\mathbf{y} \in \mathbf{Y}_P} \mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y} \notin \mathbf{Y}_P} \mathbf{y}\mathbf{y}^T \right)$$

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

# The training again..

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

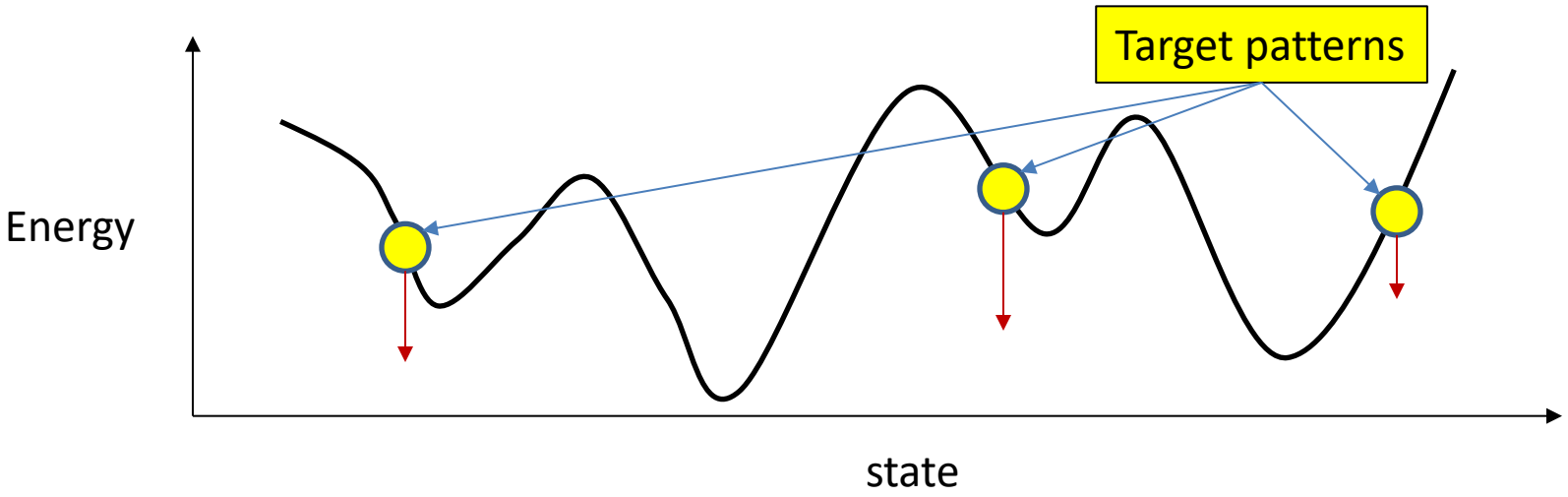
- Note the energy contour of a Hopfield network for any weight $\mathbf{W}$



# The training again

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

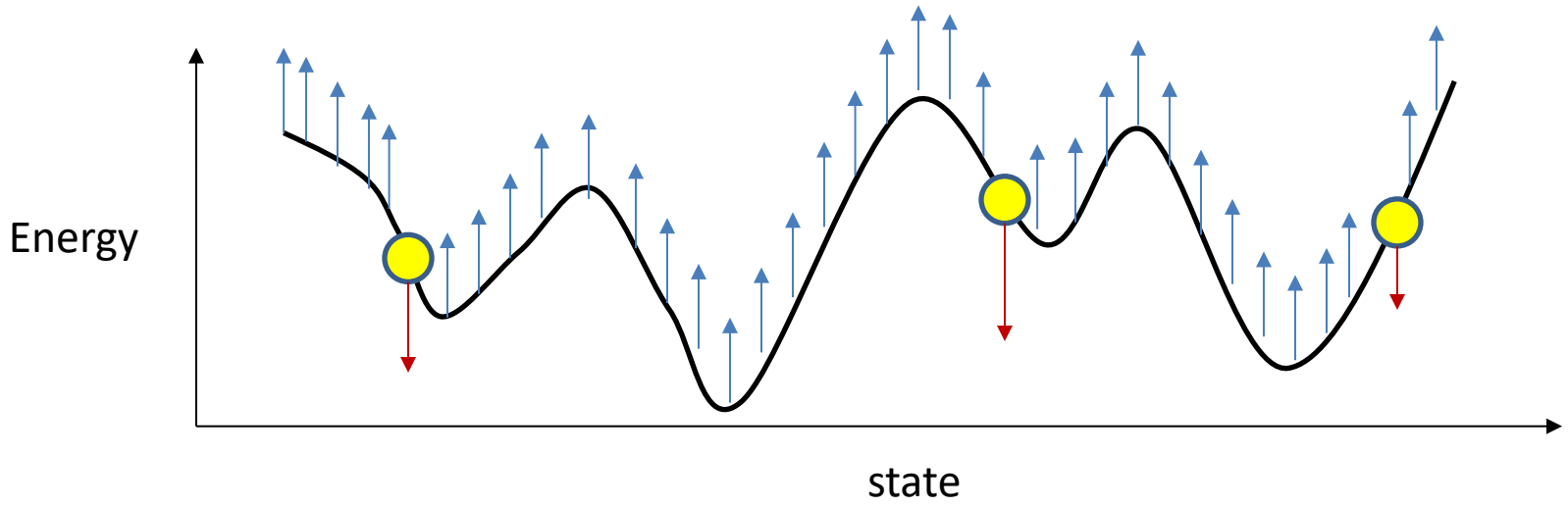
- The first term tries to *minimize* the energy at target patterns
  - Make them local minima
  - Emphasize more "important" memories by repeating them more frequently



# The negative class

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

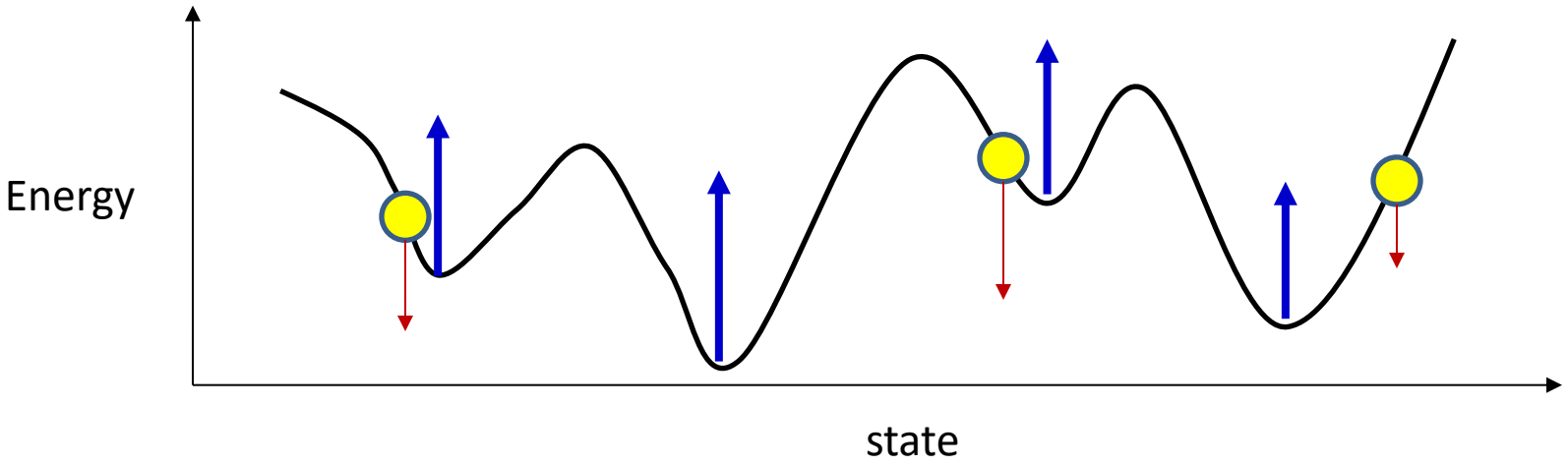
- The second term tries to "raise" all non-target patterns
  - Do we need to raise *everything*?



# Option 1: Focus on the valleys

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

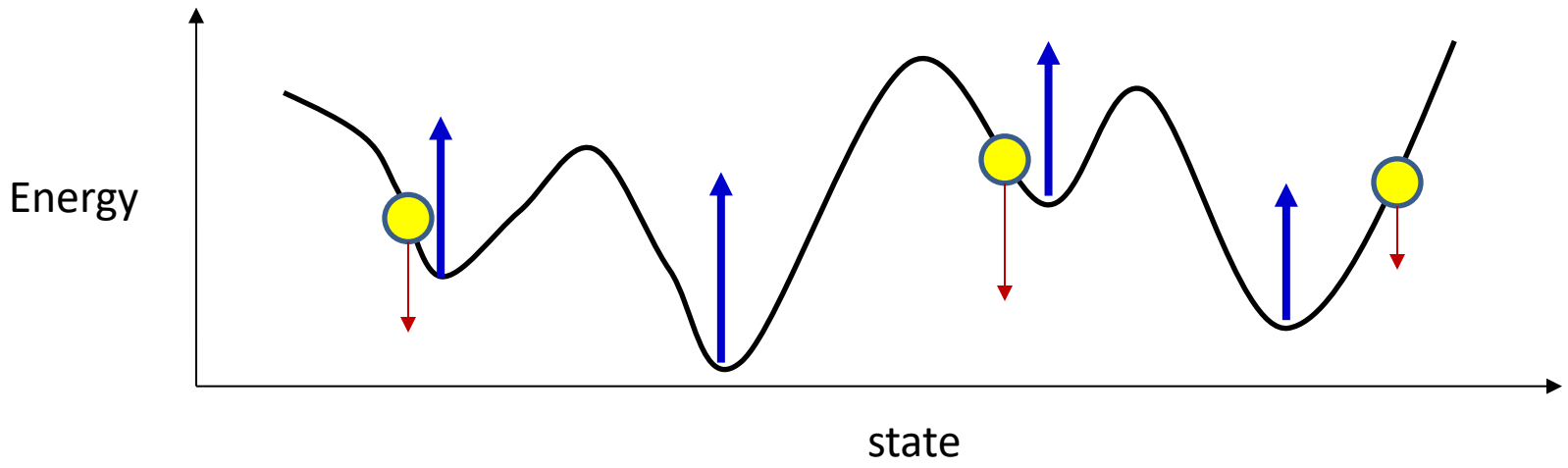
- Focus on raising the valleys
  - If you raise *every* valley, eventually they'll all move up above the target patterns, and many will even vanish



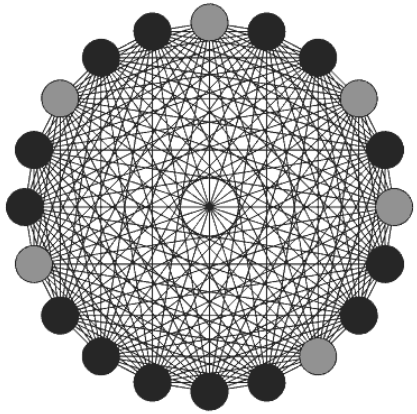
# Identifying the valleys..

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

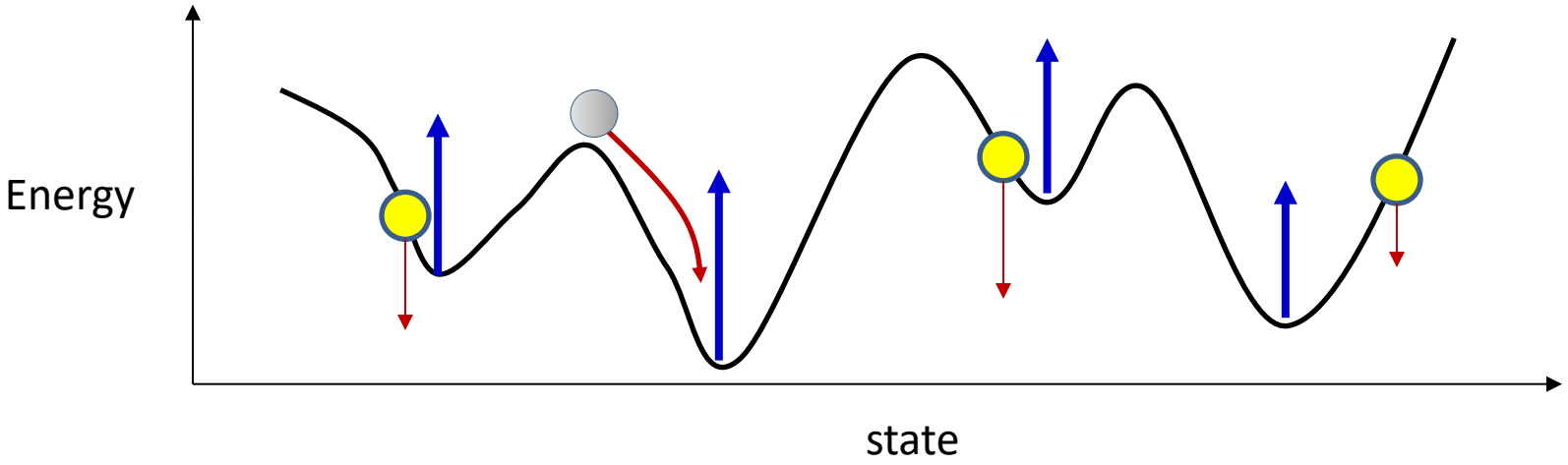
- Problem: How do you identify the valleys for the current $\mathbf{W}$?



# Identifying the valleys..



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



# Training the Hopfield network

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

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

# Training the Hopfield network: SGD version

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

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

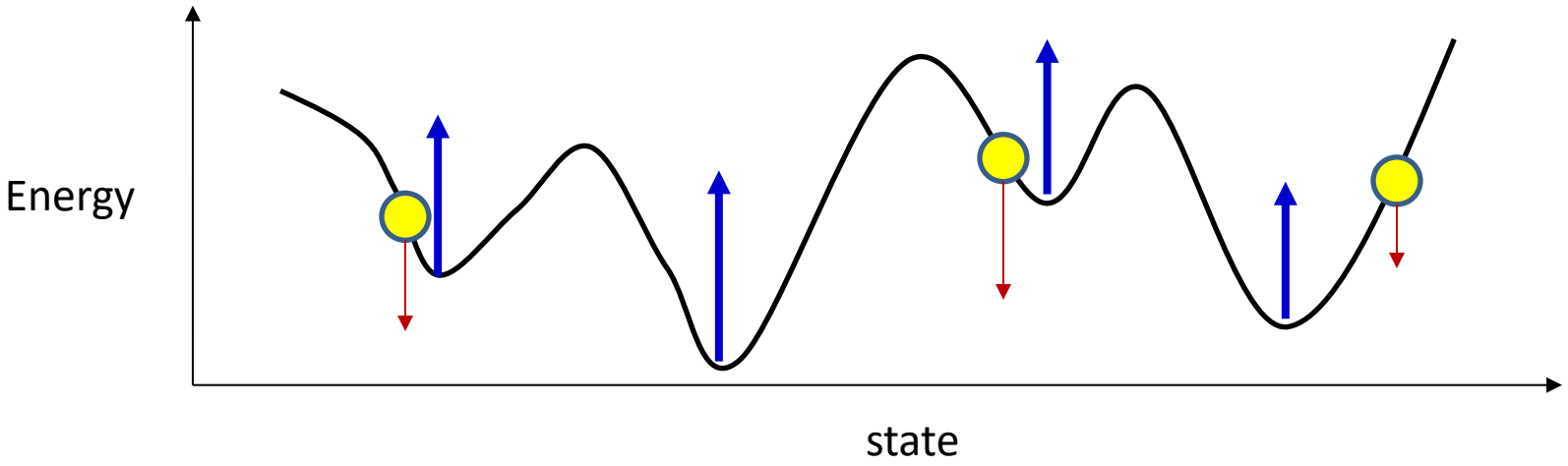
# Training the Hopfield network

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

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

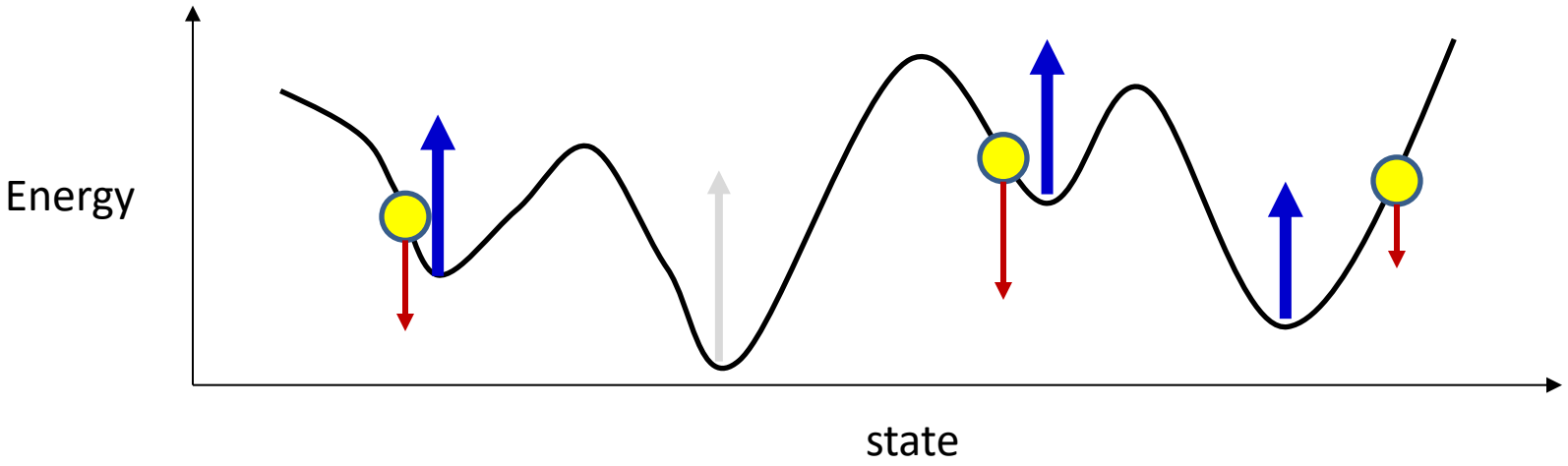
# Which valleys?

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

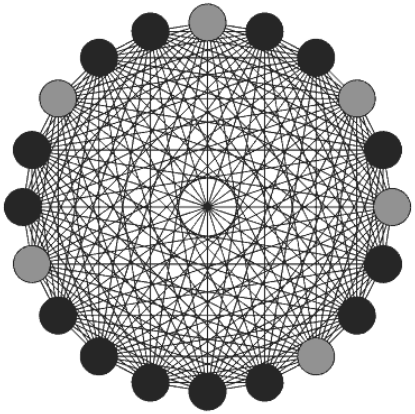


# Which valleys?

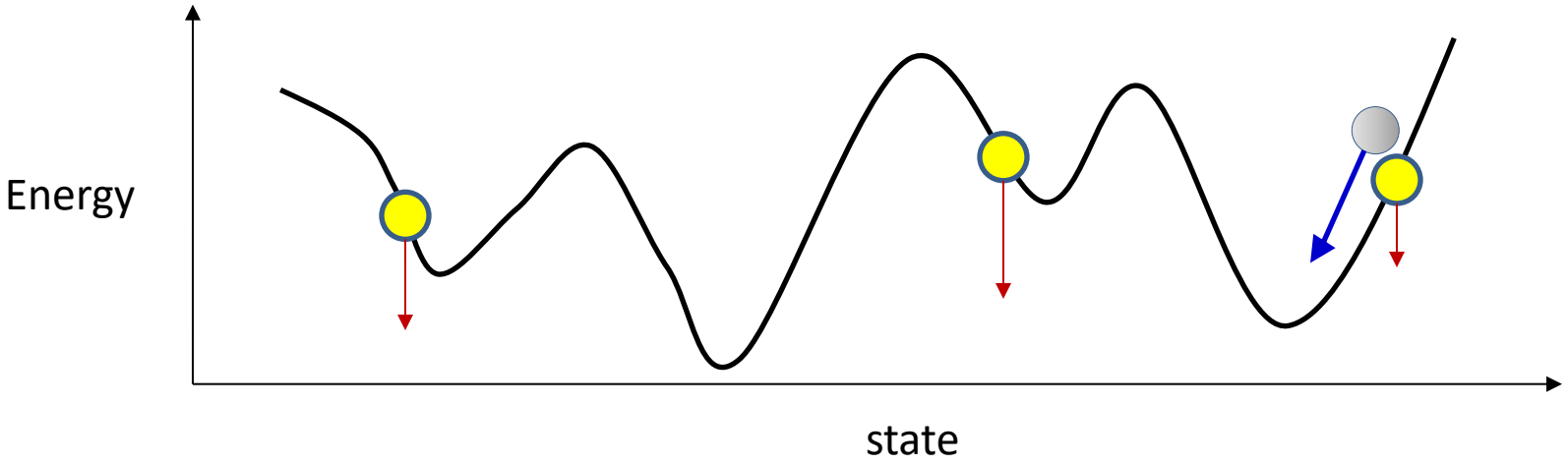
- Should we *randomly* sample valleys?
  - Are all valleys equally important?
- Major requirement: memories must be stable
  - They *must* be broad valleys
- Spurious valleys in the neighborhood of memories are more important to eliminate



# Identifying the valleys..



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



# Training the Hopfield network

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

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

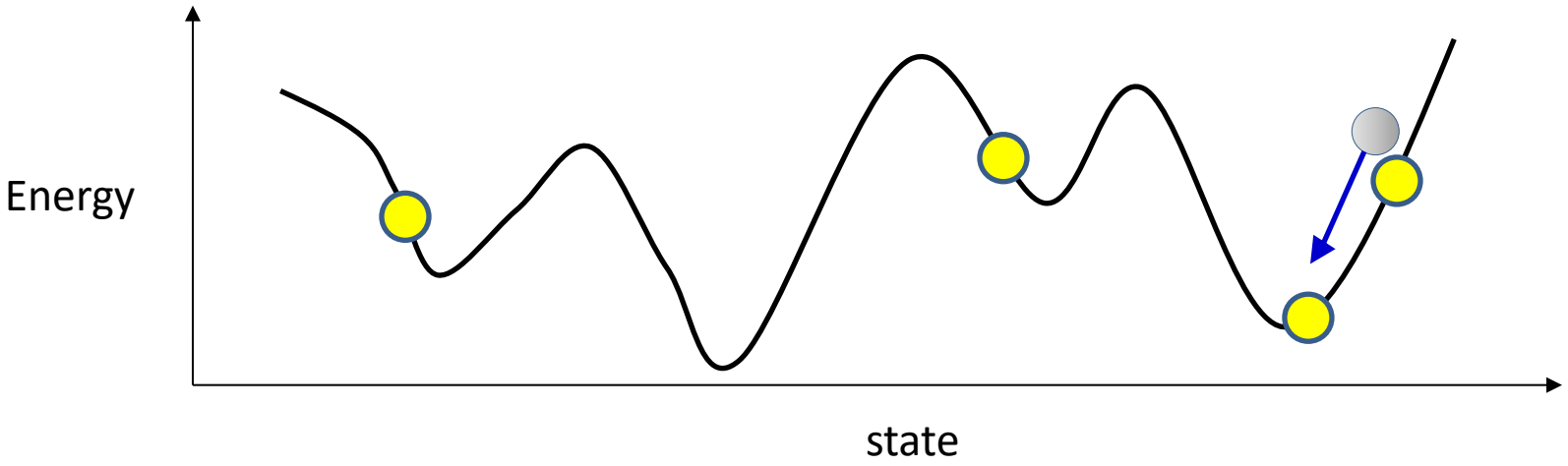
# Training the Hopfield network: SGD version

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

- 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
    - $\mathbf{W} = \mathbf{W} + \eta(\mathbf{y}_p \mathbf{y}_p^T - \mathbf{y}_v \mathbf{y}_v^T)$

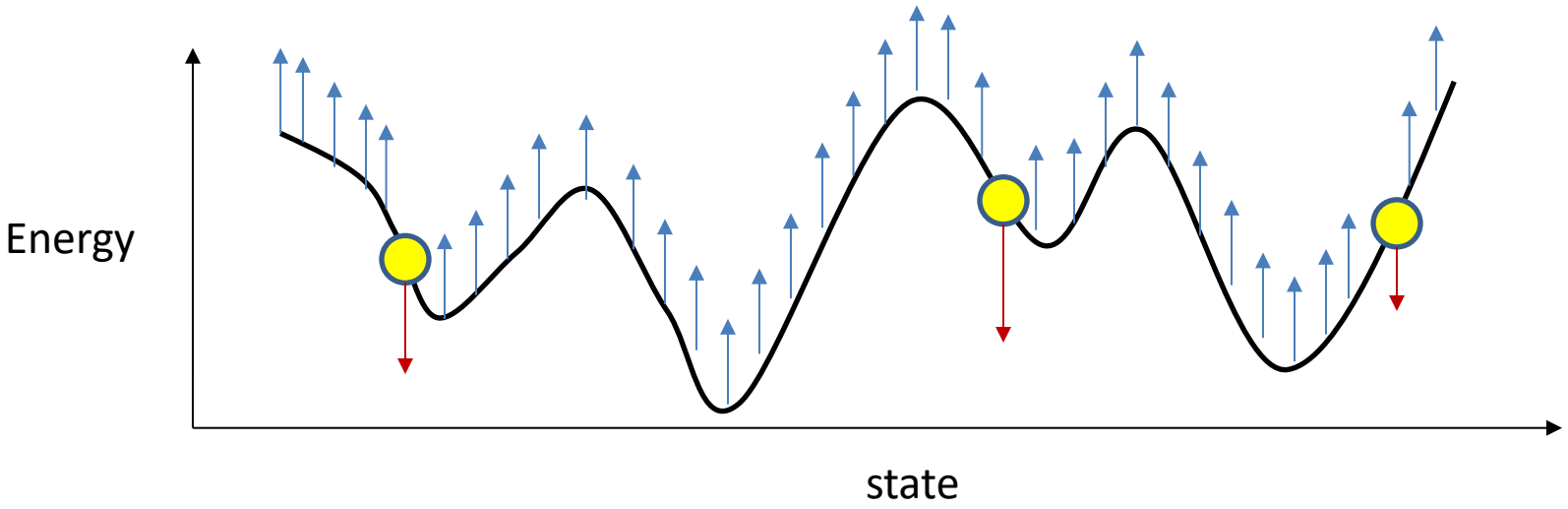
# A possible problem

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



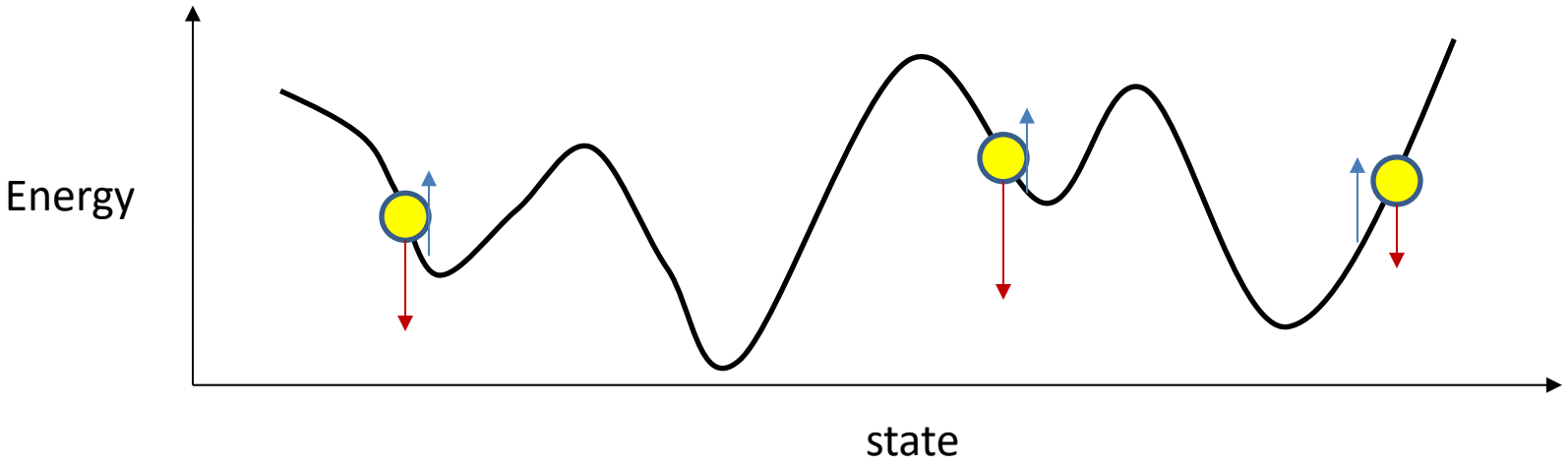
# A related issue

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



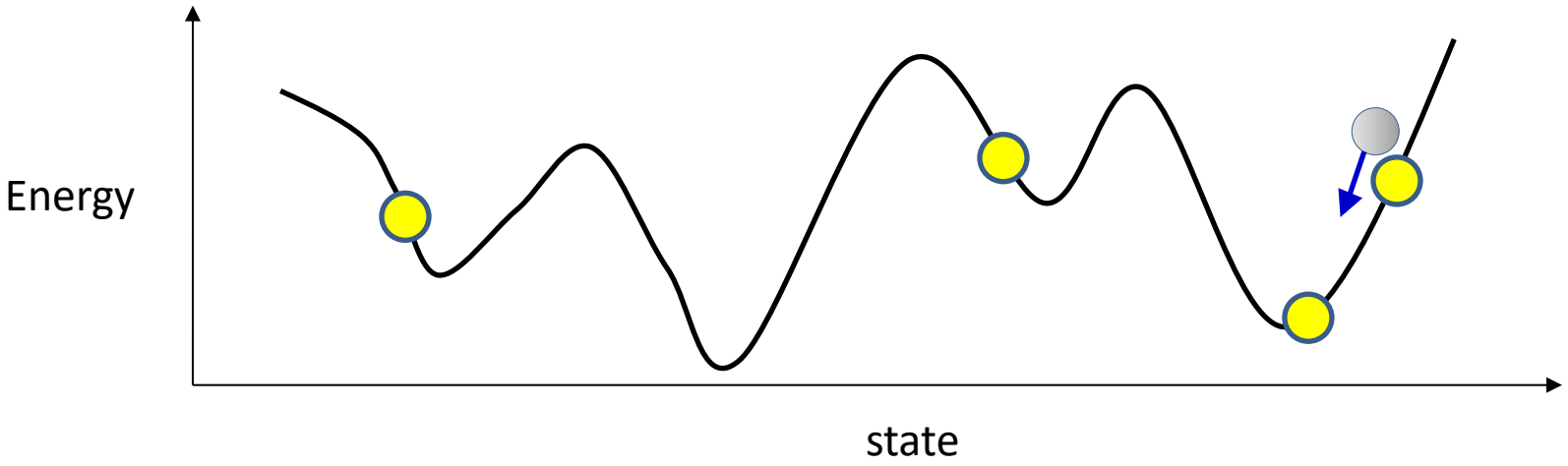
# A related issue

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



# Raising the neighborhood

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



# Training the Hopfield network: SGD version

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

- 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
    - $\mathbf{W} = \mathbf{W} + \eta(\mathbf{y}_p \mathbf{y}_p^T - \mathbf{y}_d \mathbf{y}_d^T)$

# Poll 4

- Mark all statements that are true
  - Hopfield networks can be trained using gradient descent
  - Each gradient descent step is identical to Hebbian learn where we learn target patterns and unlearn non-target ones
  - It is necessary to update parameters for *all* non-target patterns
  - Each update takes many steps of computation for each training instance because the network iterations must converge to local minima

# Poll 4

- Mark all statements that are true
  - **Hopfield networks can be trained using gradient descent**
  - **Each gradient descent step is identical to Hebbian learn where we learn target patterns and unlearn non-target ones**
  - It is necessary to update parameters for *all* non-target patterns
  - Each update takes many steps of computation for each training instance because the network iterations must converge to local minima

# Story so far

- Hopfield nets with $N$ neurons can store up to $N$ random patterns
  - But comes with many parasitic memories

- Networks that store $O(N)$ memories can be trained through optimization
  - By minimizing the energy of the target patterns, while increasing the energy of the neighboring patterns

# Storing more than N patterns

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

- Next class…