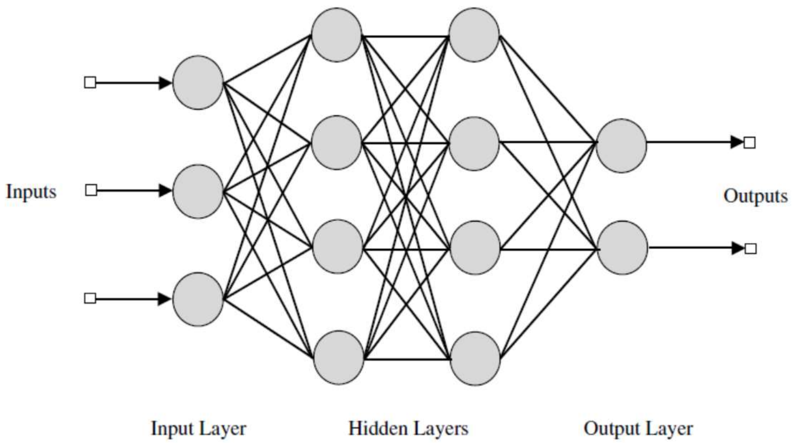
Neural Networks

Hopfield Nets, Auto Associators, Boltzmann machines

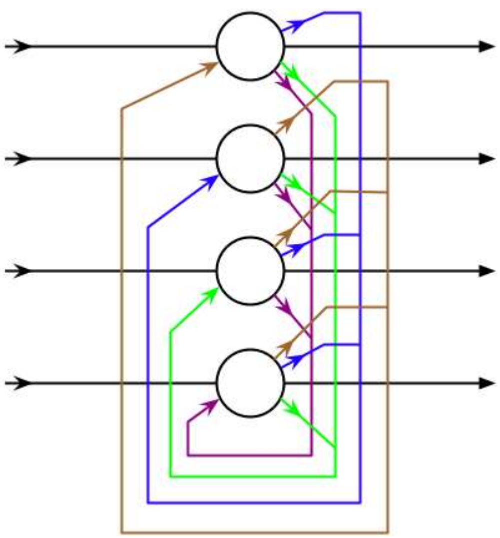
Fall 2023

Story so far

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



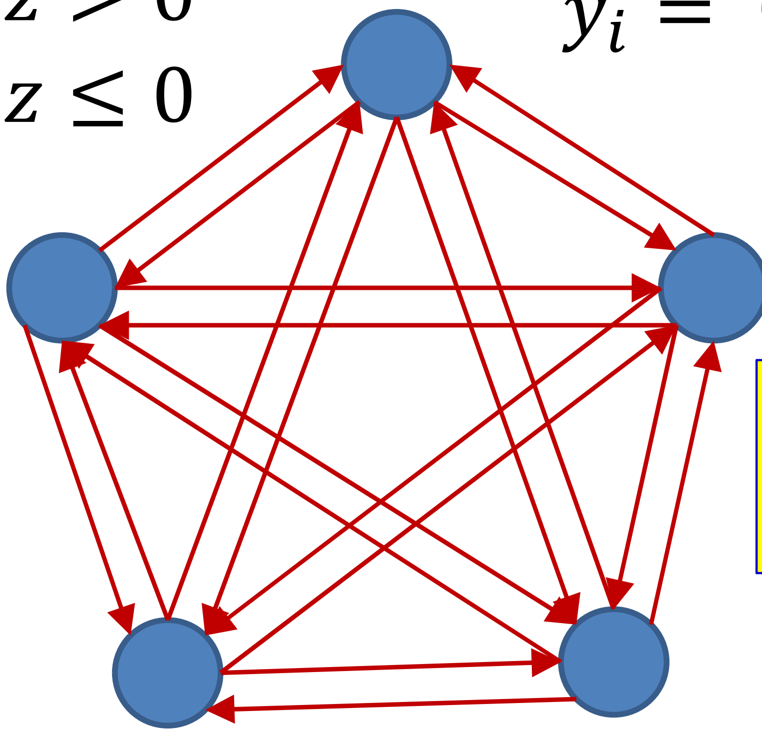
- But what about..



Consider this loopy network

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

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



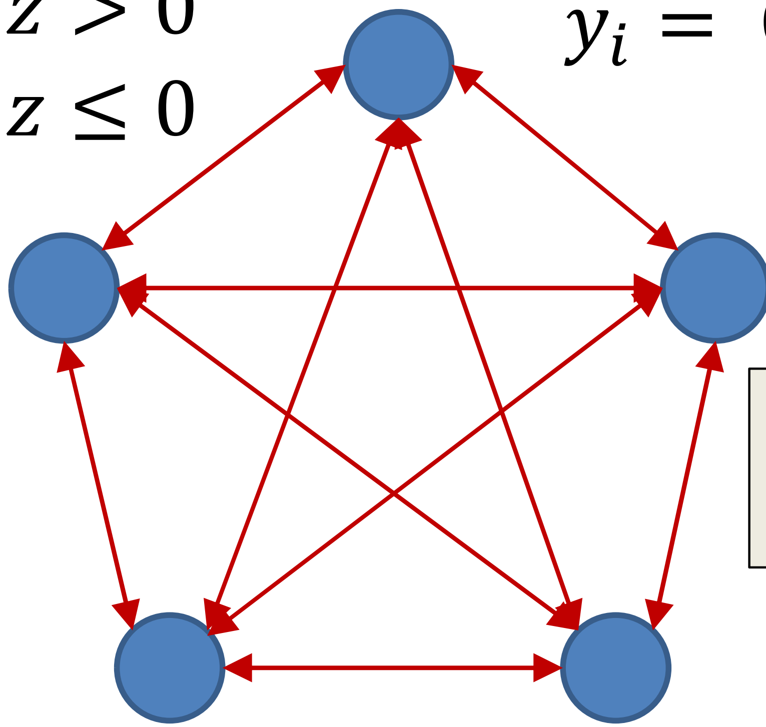
The output of a neuron affects the input to the neuron

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

Consider this loopy network

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

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



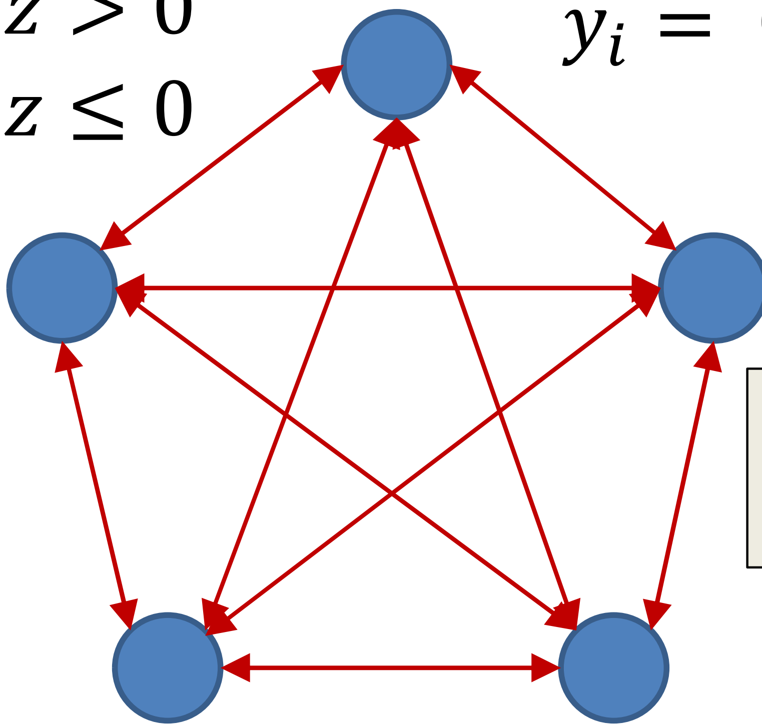
A symmetric network:
$w_{ij} = w_{ji}$

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

Hopfield Net

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

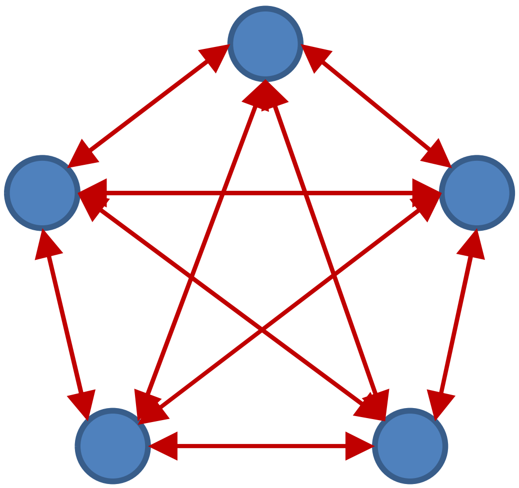
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$



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

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

Loopy network

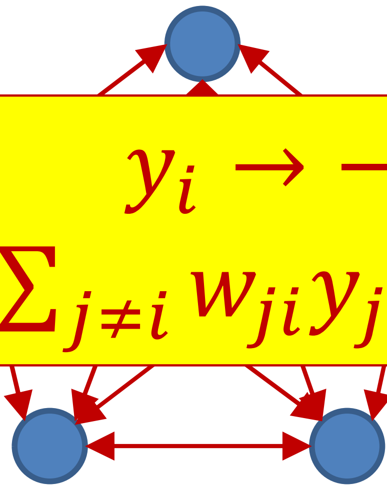


$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

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

Loopy network



$$y_i \rightarrow -y_i$$

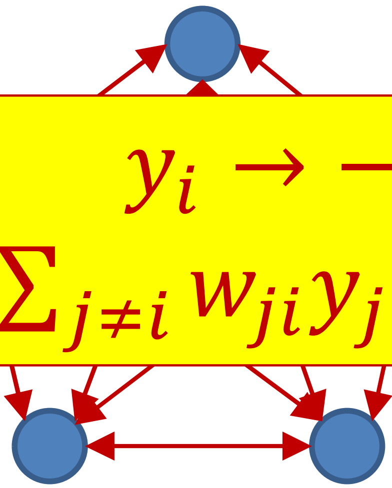
if $y_i\left(\sum_{j \neq i} w_{ji} y_j + b_i\right) < 0$

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

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

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

Loopy network



$$y_i \rightarrow -y_i$$

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

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

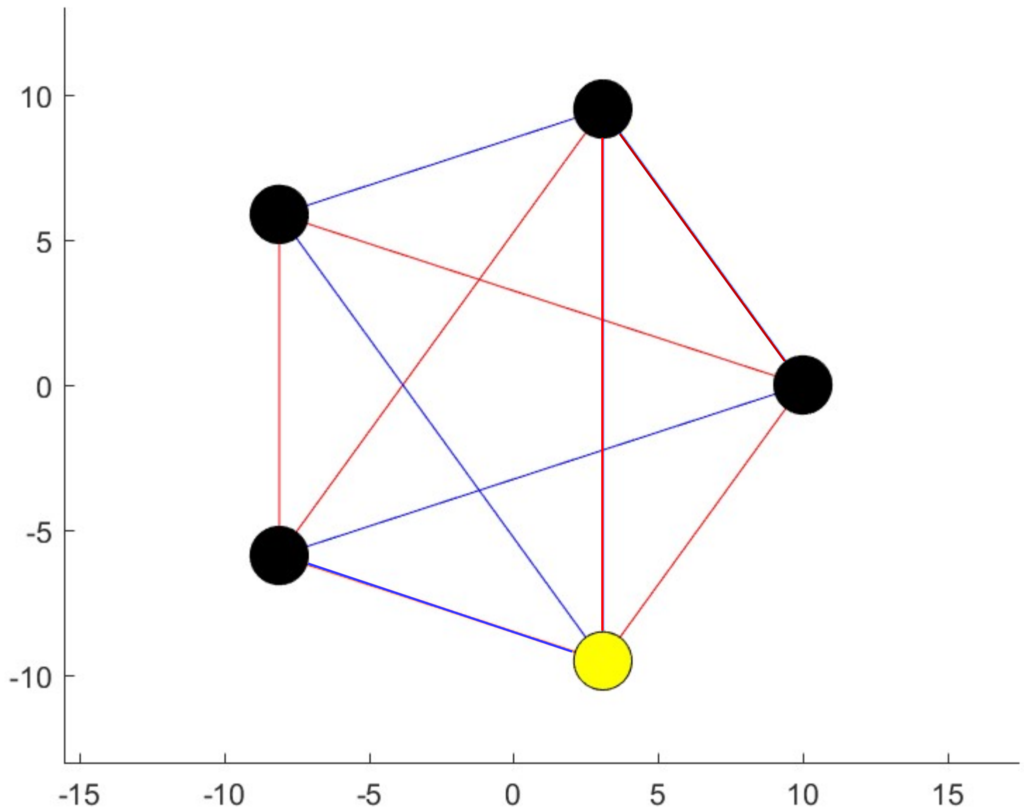
$$\Theta(z) = \begin{cases} +1 \; if \; z > 0 \\ -1 \; if \; z \leq 0 \end{cases}$$

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

But this may cause *other* neurons to flip!

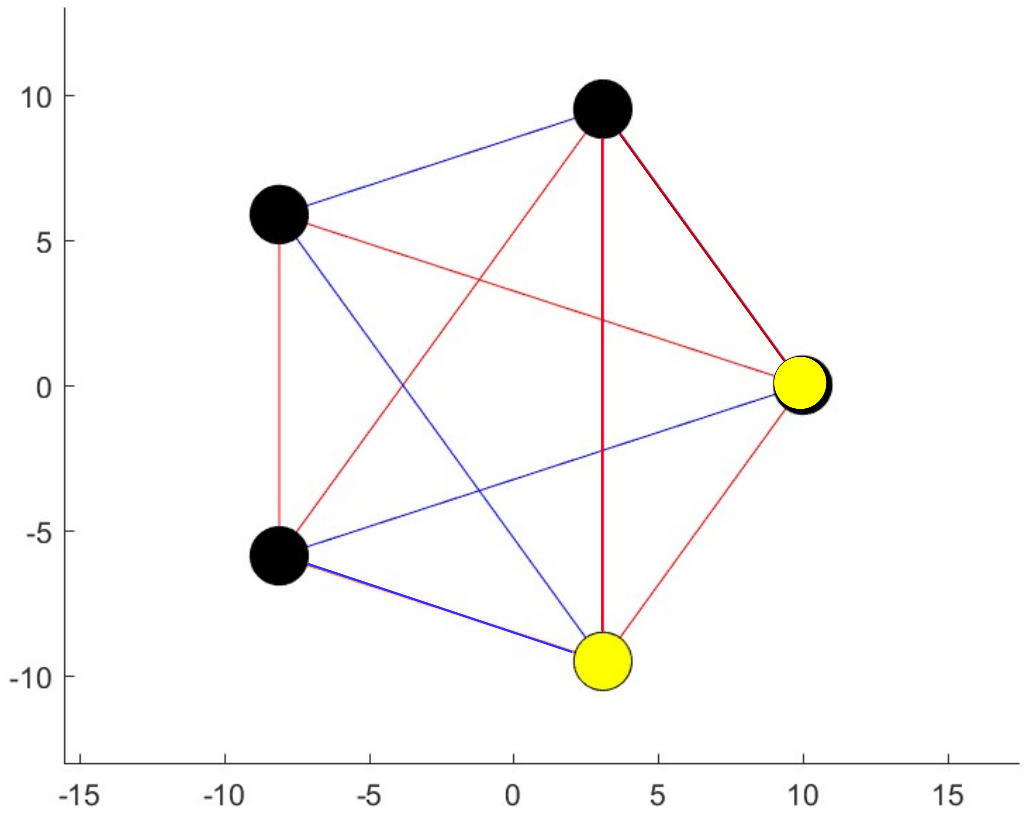
- ...es a "field" $\sum_{j\neq i} w_{ji}y_j + b_i$
- ...s own sign, it does not respond
- If the sign of the field opposes its own sign, it "flips" to match the sign of the field

Example



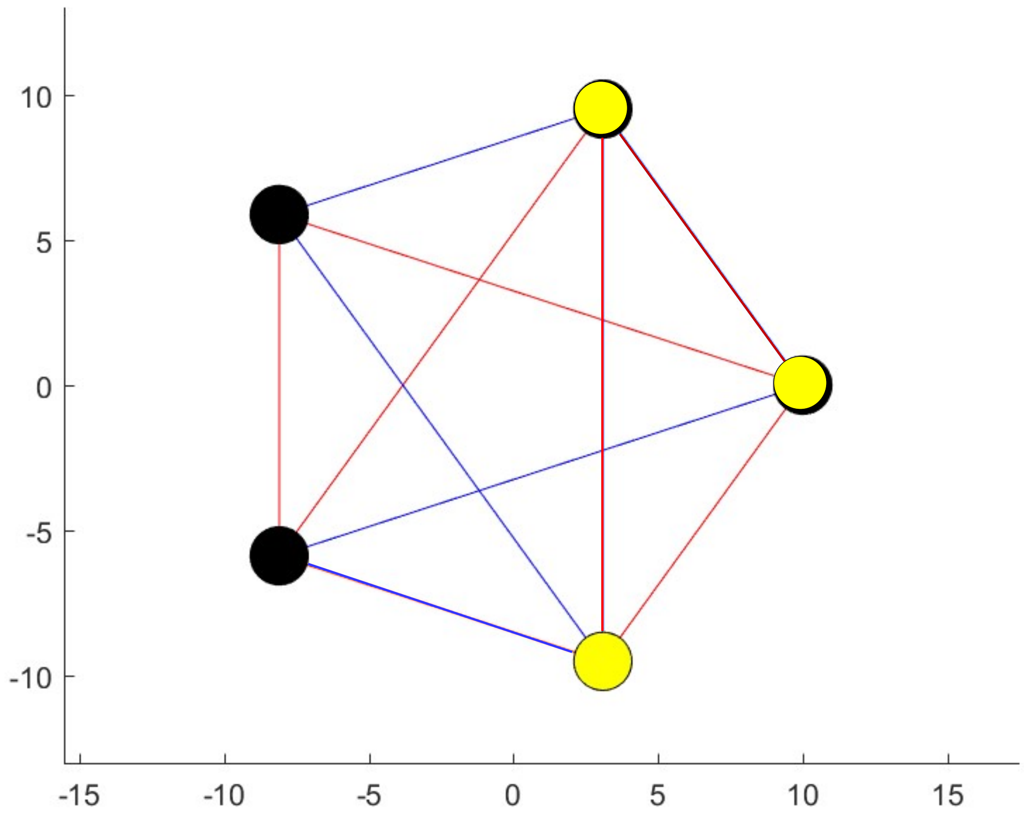
- Red edges are +1, blue edges are -1
- Yellow nodes are -1, black nodes are +1

Example



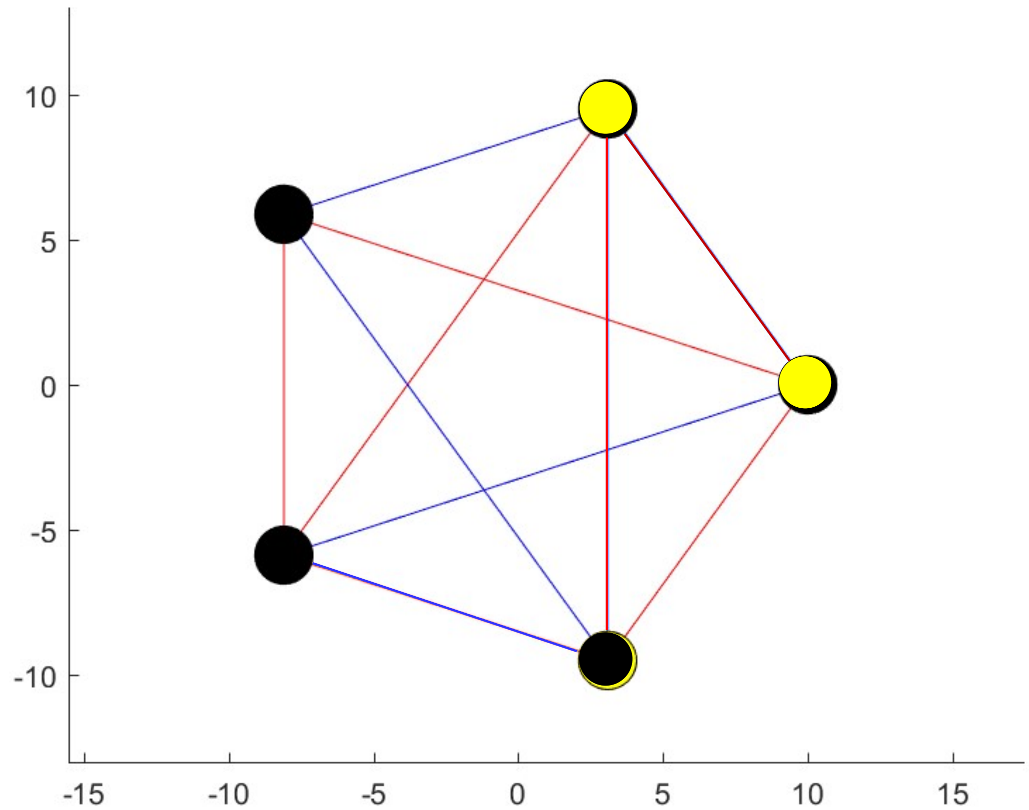
- Red edges are +1, blue edges are -1
- Yellow nodes are -1, black nodes are +1

Example



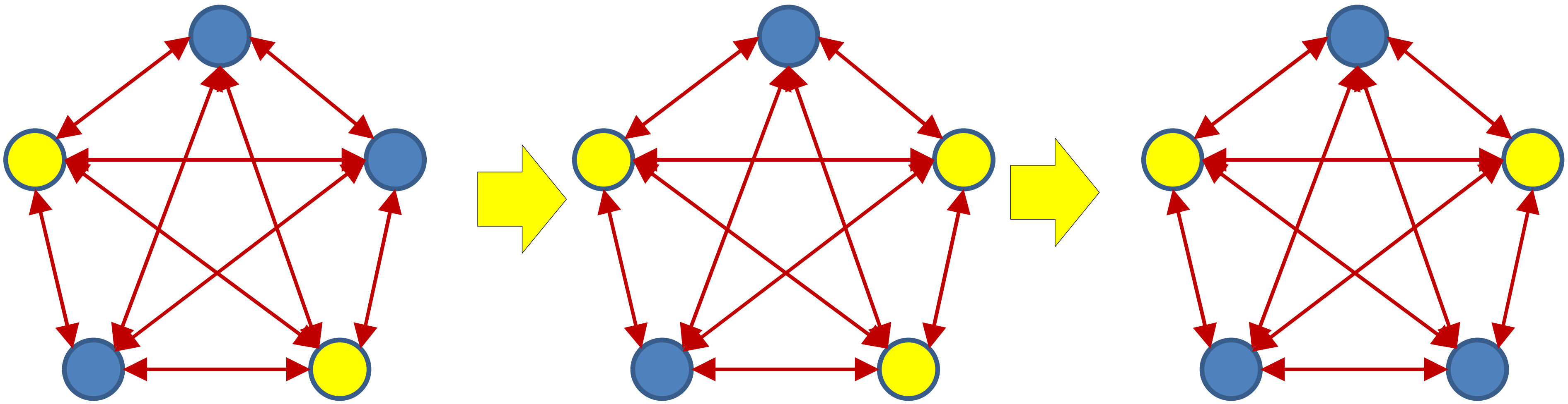
- Red edges are +1, blue edges are -1
- Yellow nodes are -1, black nodes are +1

Example



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

Loopy network

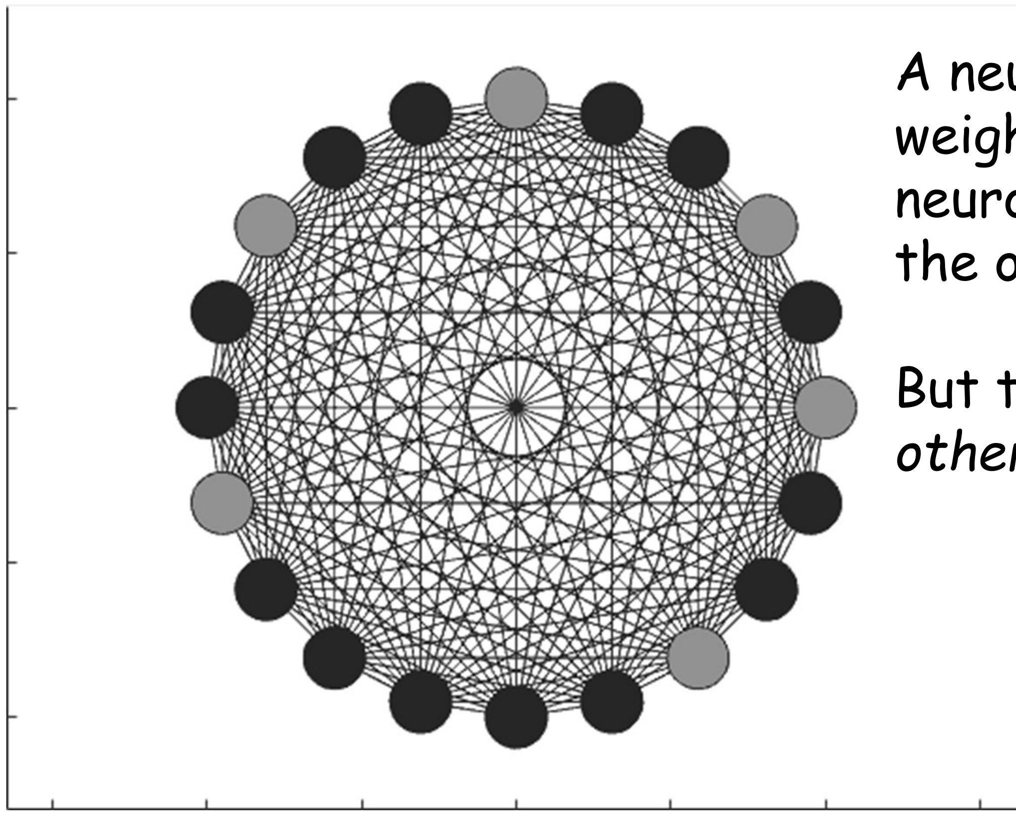


- If the sign of the field at any neuron opposes its own sign, it “flips” to match the field
  - Which will change the field at other nodes
    - Which may then flip
      - Which may cause other neurons including the first one to flip…
        - » And so on…

20 evolutions of a loopy net

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

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

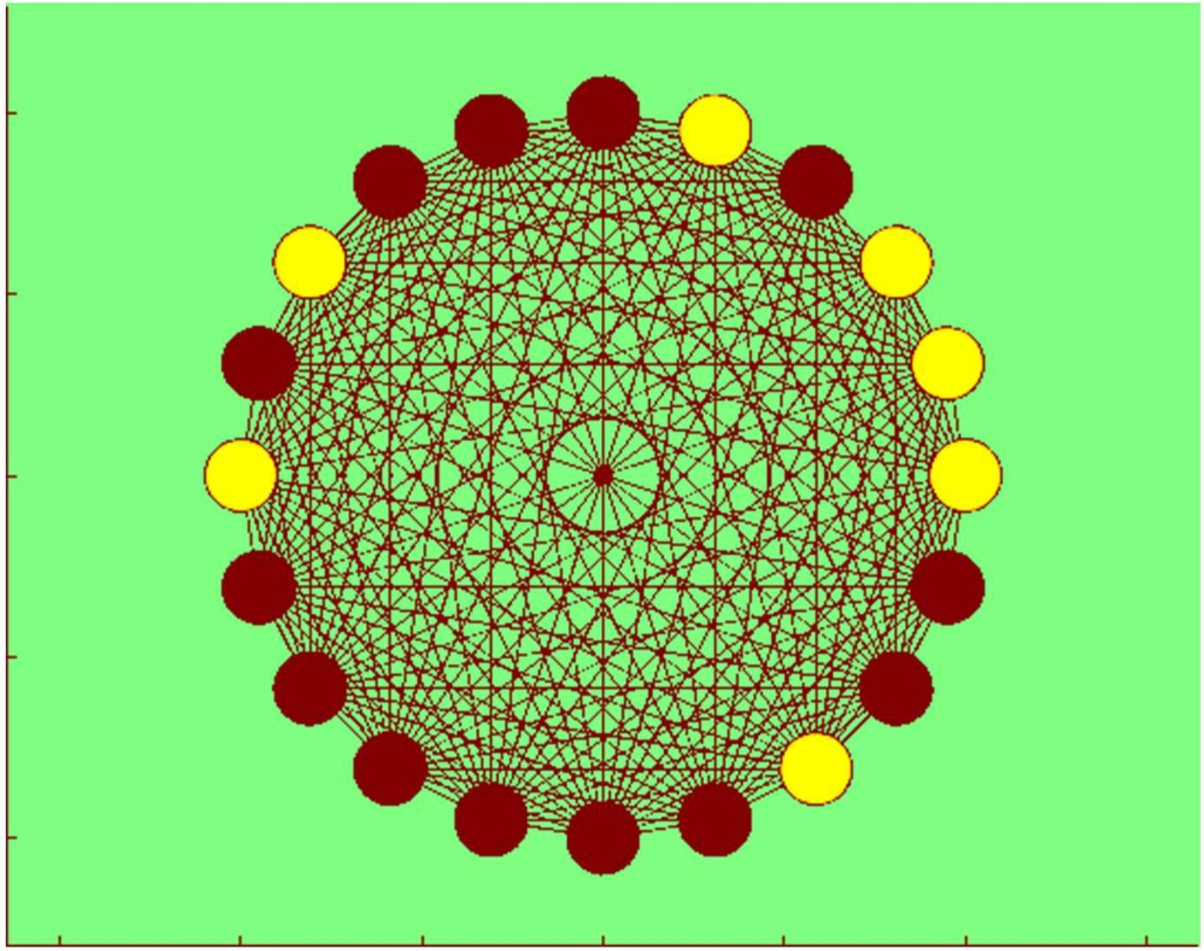


A neuron "flips" if weighted sum of other neuron's outputs is of the opposite sign

But this may cause *other* neurons to flip!

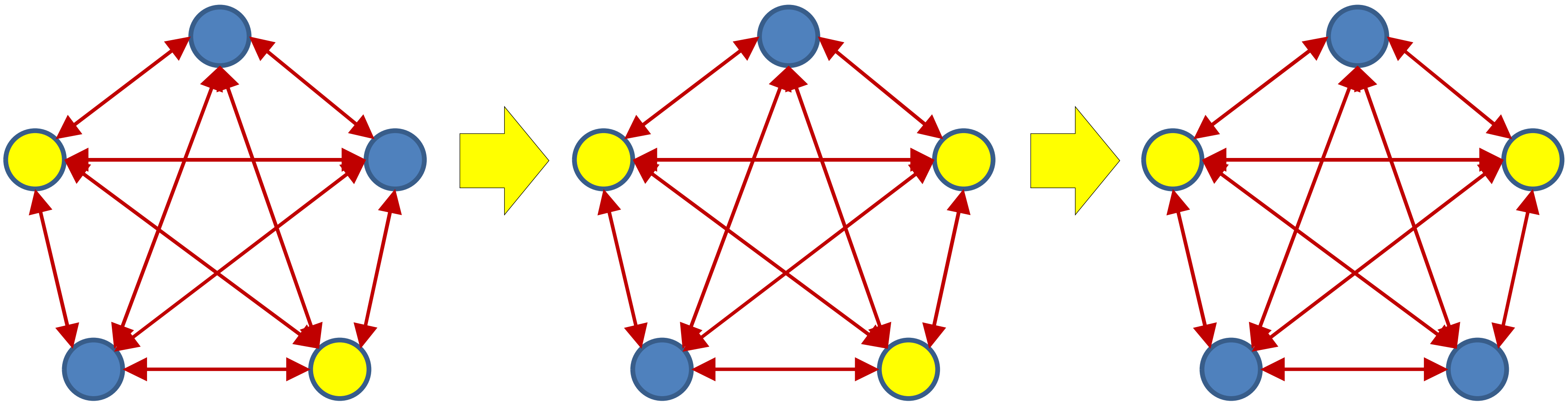
- All neurons which do not "align" with the local field "flip"

120 evolutions of a loopy net



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

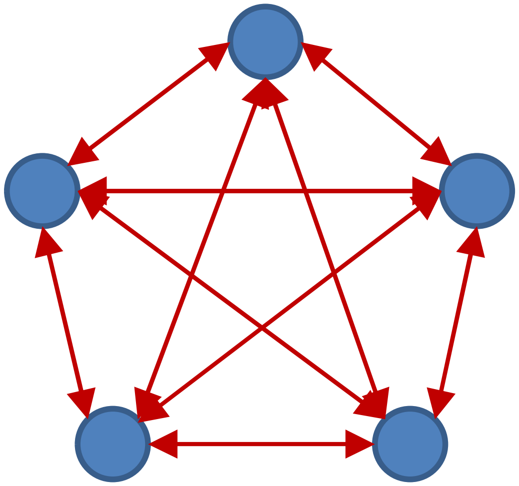
Loopy network



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

- *Will this behavior continue for ever??*

Loopy network



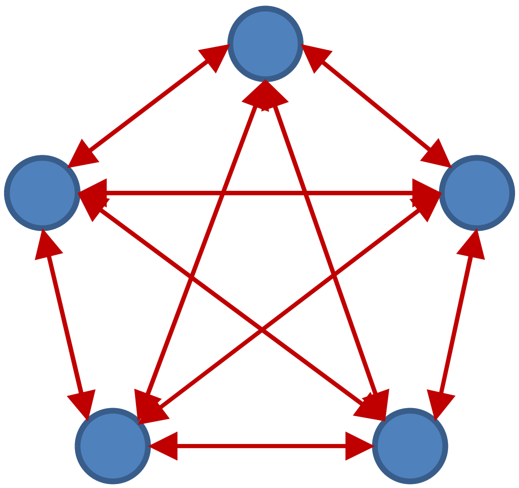
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

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

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

Loopy network



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

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

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

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

  – This term is always positive!

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

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

Globally

- Consider the following sum across *all* nodes

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

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

  - Assume $w_{ii} = 0$

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

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

- This is strictly positive

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

Upon flipping a single unit

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

- Expanding

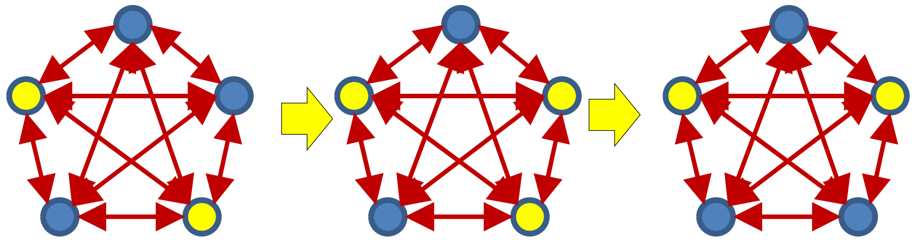
$$\Delta D(y_k) = (y_k^+ - y_k^-)\left(\sum_{j \neq k} w_{jk} y_j + b_k\right)$$

  - All other terms that do not include y_k cancel out

- This is always positive!

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

Hopfield Net



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

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

- D is bounded

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

- The minimum increment of D in a flip is

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

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

The Energy of a Hopfield Net

- Define the *Energy* of the network as

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

  - Just 0.5 times the negative of D
    - The 0.5 is only needed for convention
- The evolution of a Hopfield network constantly decreases its energy

Story so far

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

Poll 1

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

- True
- False

Hopfield networks will evolve continuously, forever

- True
- False

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

- True
- False

Poll 1

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

- **True**
- False

Hopfield networks will evolve continuously, forever

- True
- **False**

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

- **True**
- False

The Energy of a Hopfield Net

- Define the *Energy* of the network as

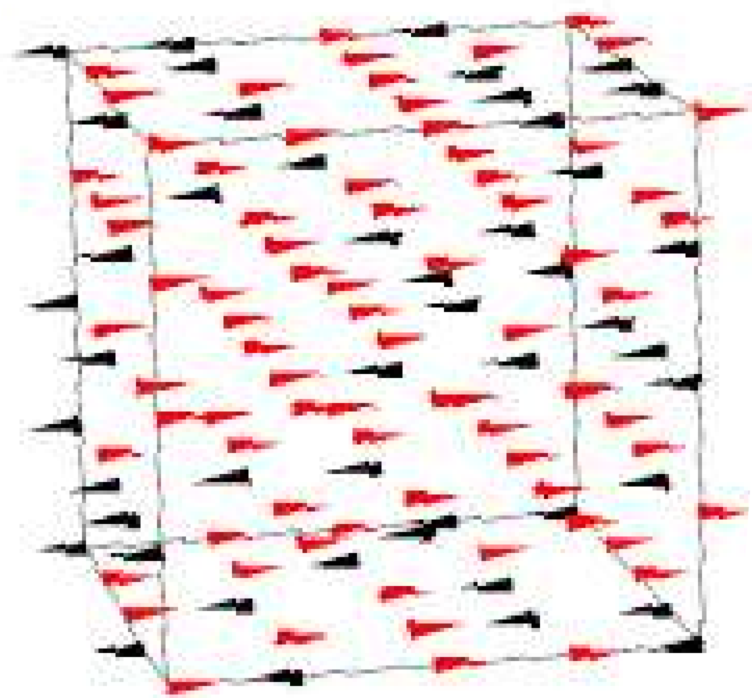
$$E = -\frac{1}{2}\left(\sum_{i,j\neq i} w_{ij}y_iy_j - \sum_i b_iy_i\right)$$

  - Just 0.5 times the negative of D

- The evolution of a Hopfield network constantly decreases its energy

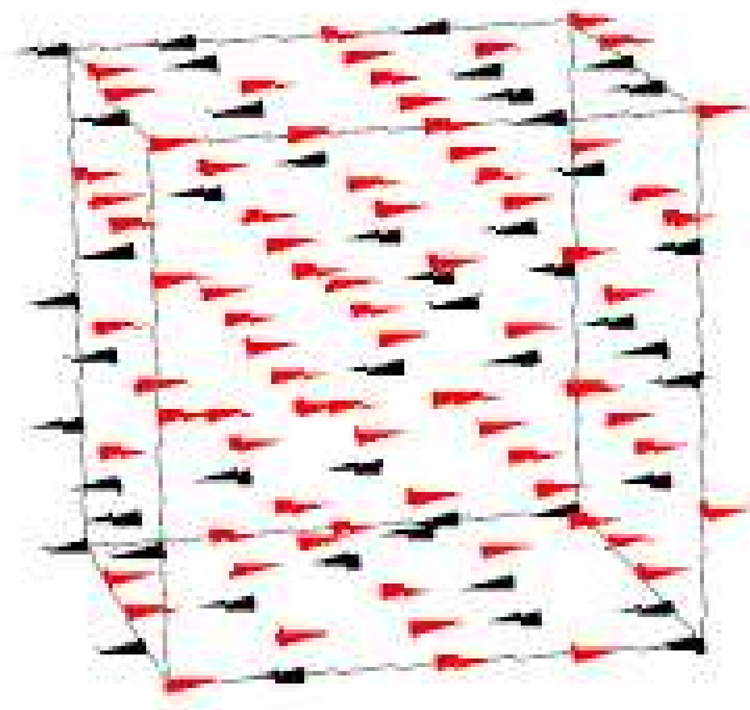
- Where did this "energy" concept suddenly sprout from?

Analogy: Spin Glass



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

Analogy: Spin Glasses



Total field at current dipole:

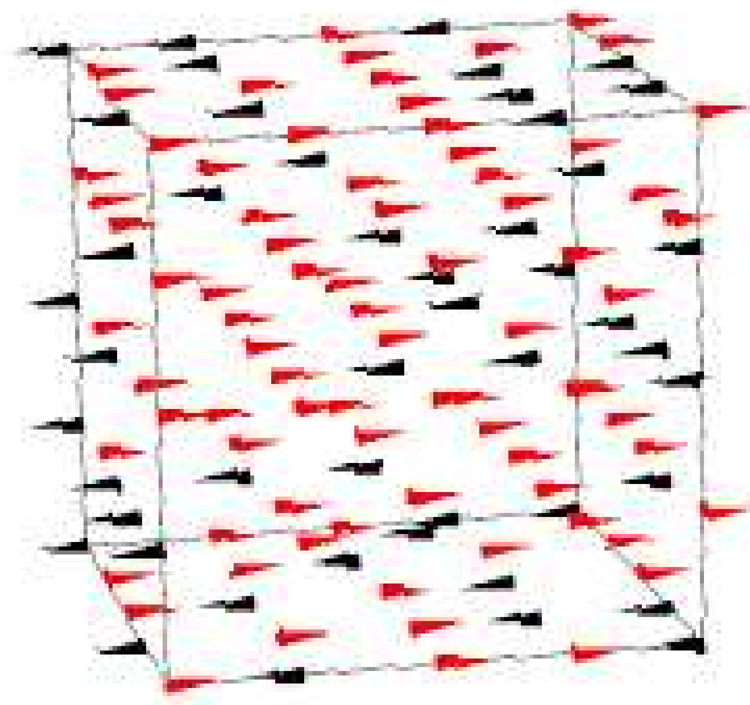
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

intrinsic

external

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

Analogy: Spin Glasses



Total field at current dipole:

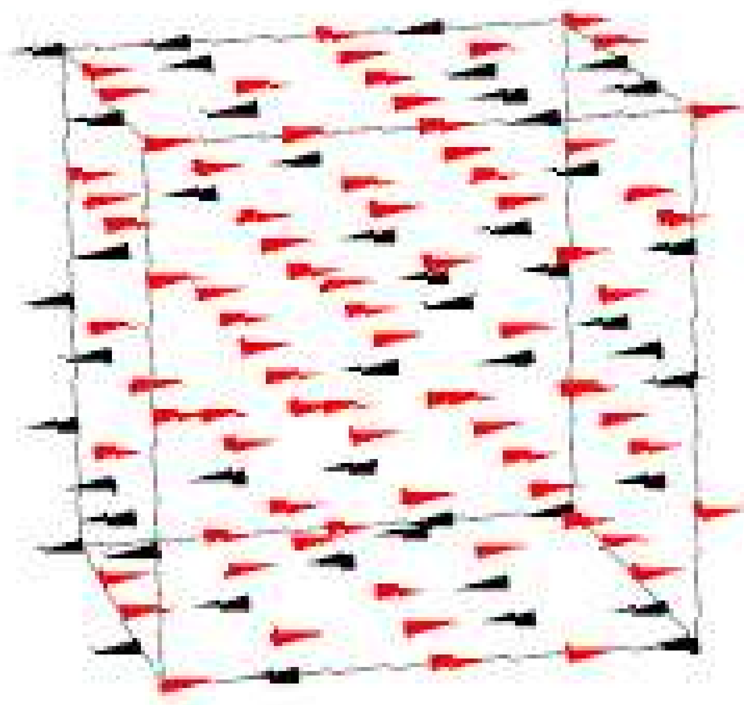
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

Response of current dipole

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

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

Analogy: Spin Glasses



Total field at current dipole:

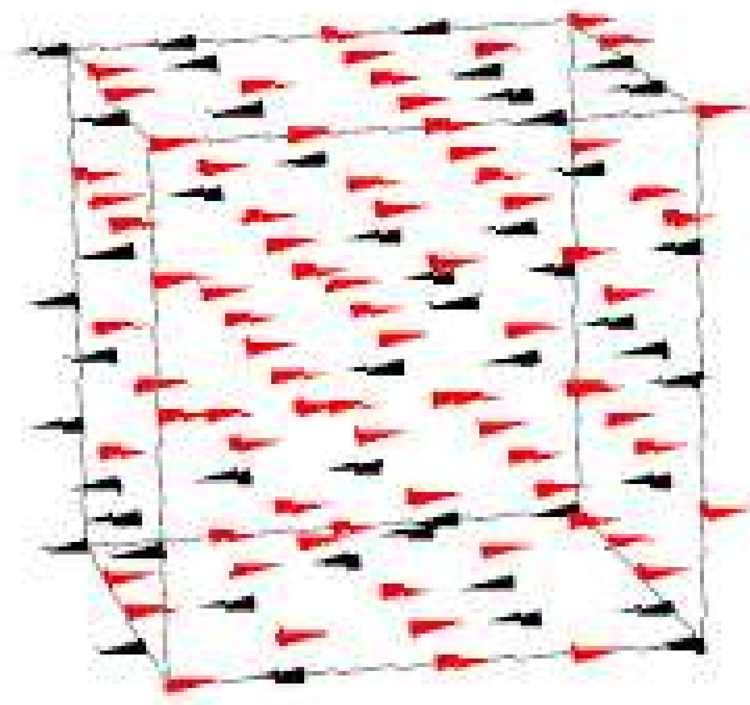
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

Response of current dipole

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

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

Analogy: Spin Glasses



Total field at current dipole:

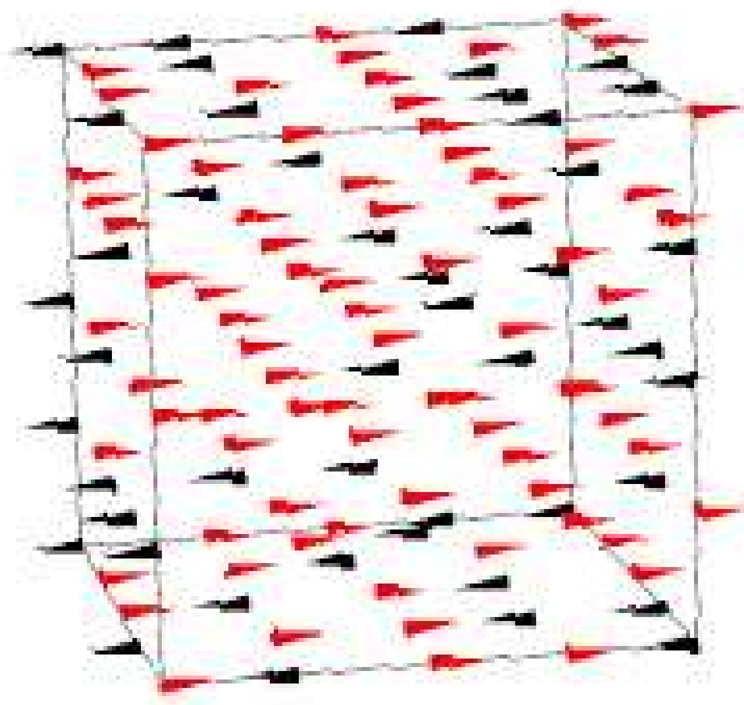
$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$

Response of current dipole

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

- When will it stop???

Analogy: Spin Glasses



Total field at current dipole:

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

Response of current dipole

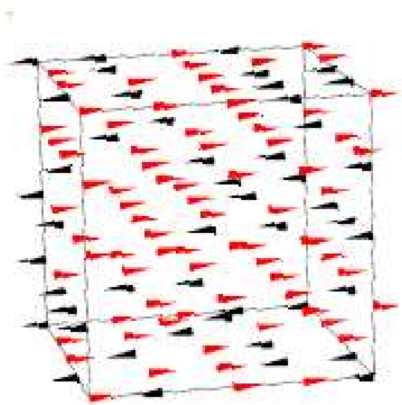
$$x_i = \begin{cases} x_i \; if \; sign\left(x_i \; f(p_i)\right) = 1 \\ -x_i \; otherwise \end{cases}$$

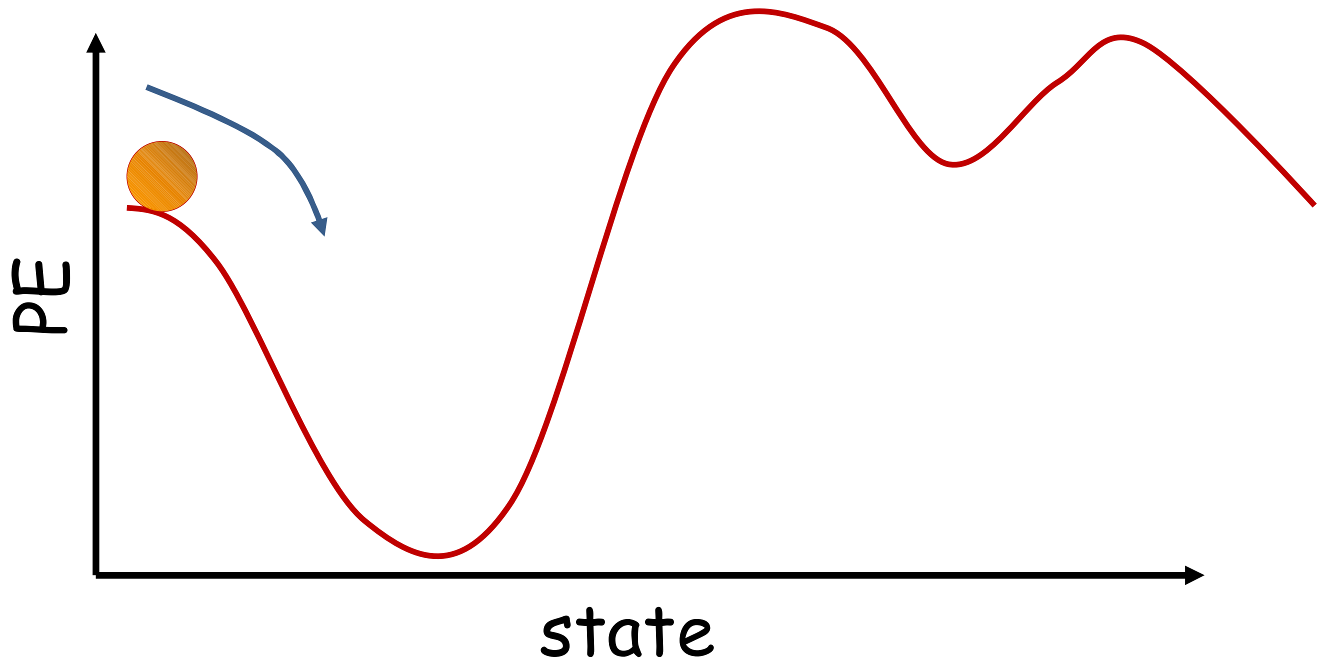
- The "Hamiltonian" (total energy) of the system

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

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

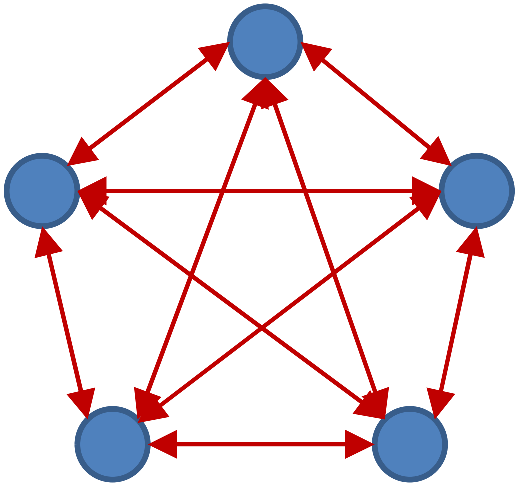
Spin Glasses





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

Hopfield Network



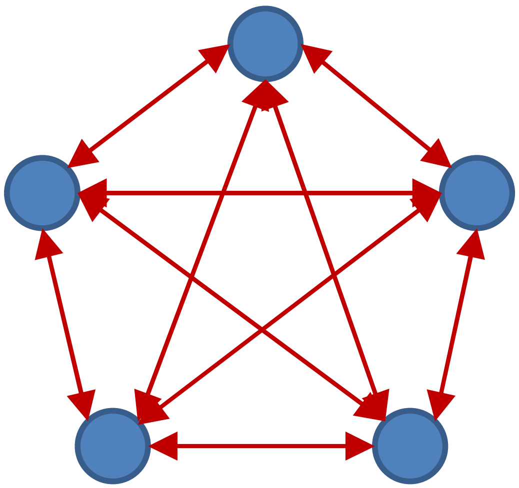
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

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

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

Hopfield Network



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

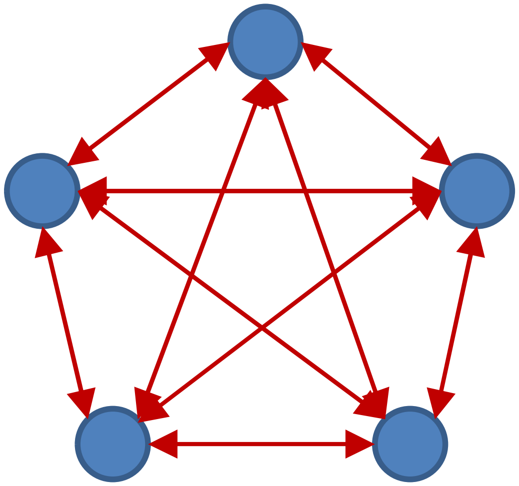
$$\Theta(z) = \begin{cases} +1 \ if \ z > 0 \\ -1 \ if \ z \leq 0 \end{cases}$$

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

We will not always explicitly show the bias

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

Hopfield Network



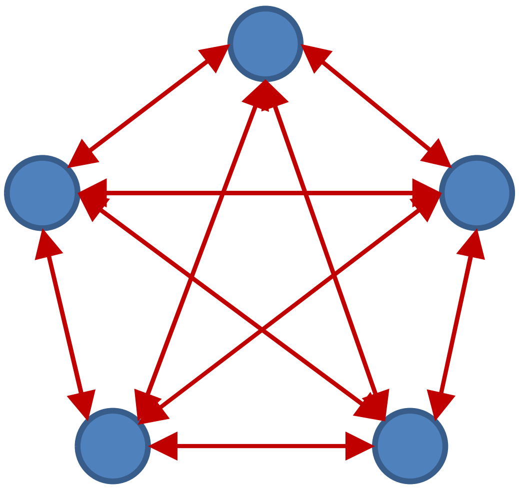
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j\right)$$

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

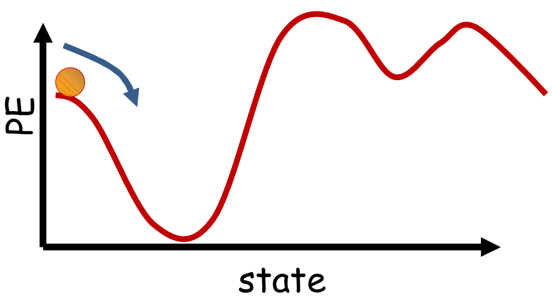
$$E = -\frac{1}{2} \sum_{i,j<i} w_{ij} y_i y_j$$

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

Evolution

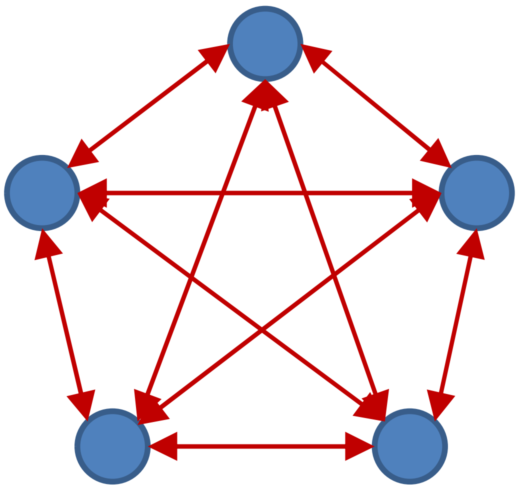


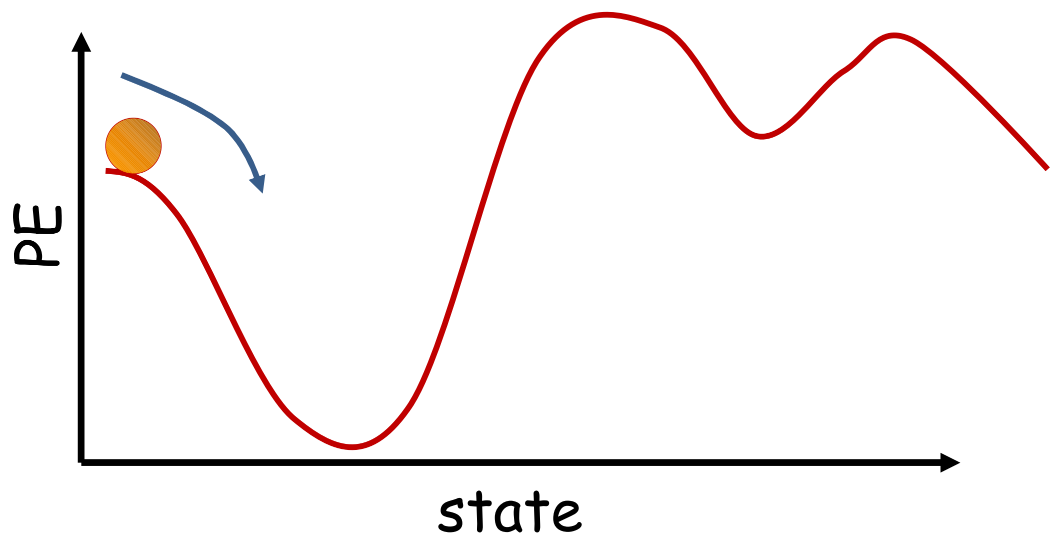
$$E = -\frac{1}{2}\sum_{i,j<i} w_{ij} y_i y_j$$



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

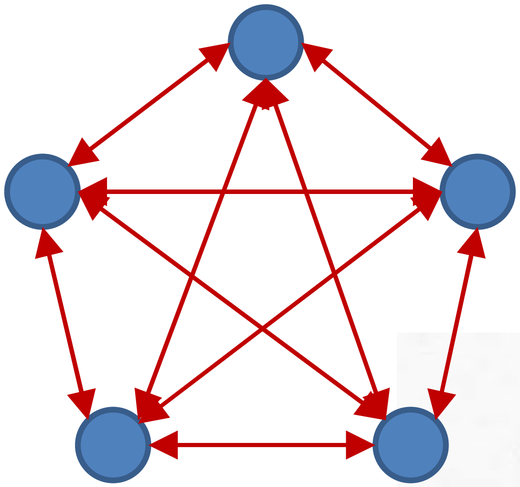
Content-addressable memory





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

Evolution



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

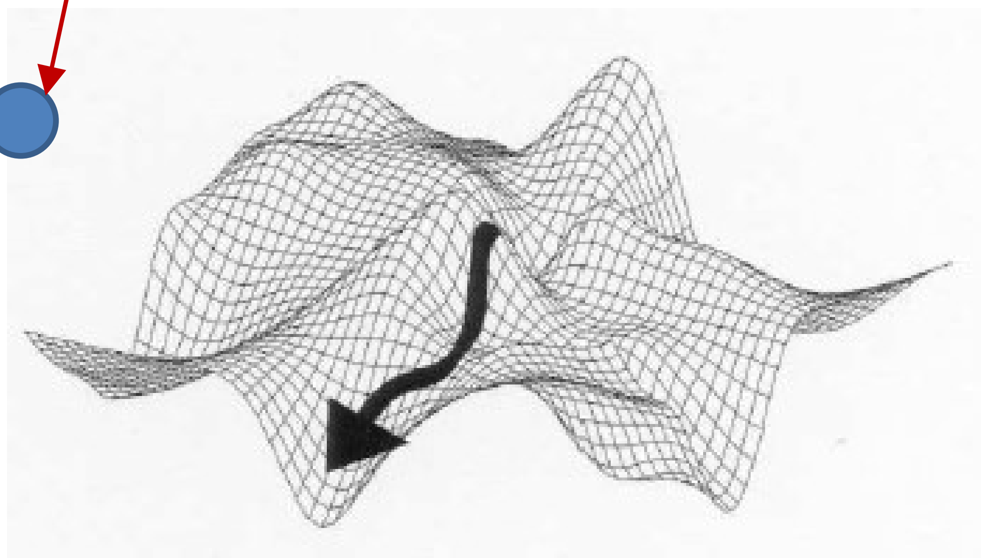
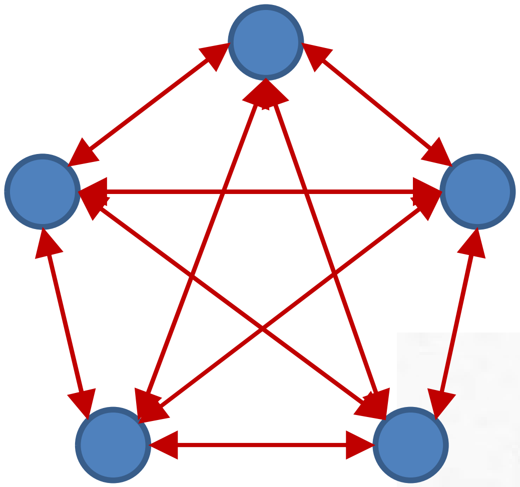


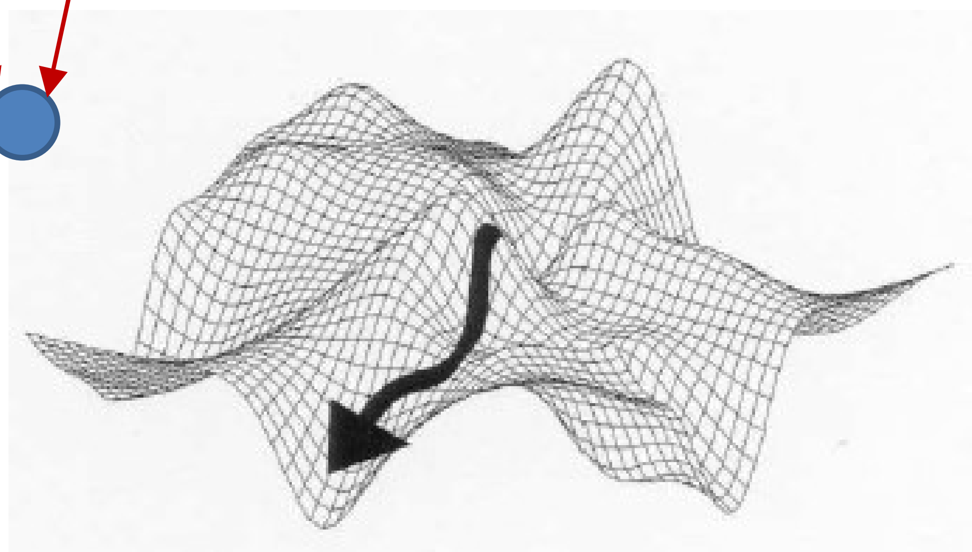
Image pilfered from unknown source

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

Evolution

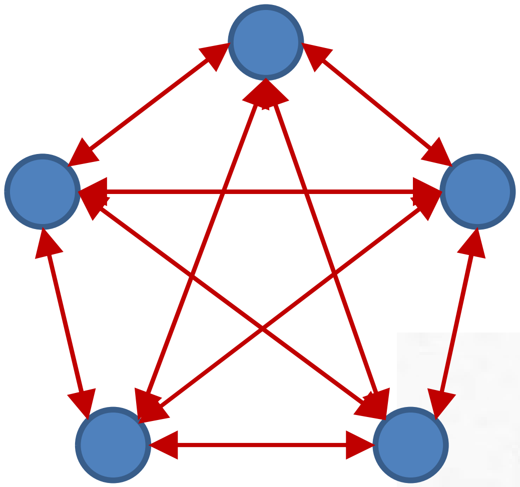


$$E = -\frac{1}{2}\sum_{i,j<i} w_{ij} y_i y_j$$



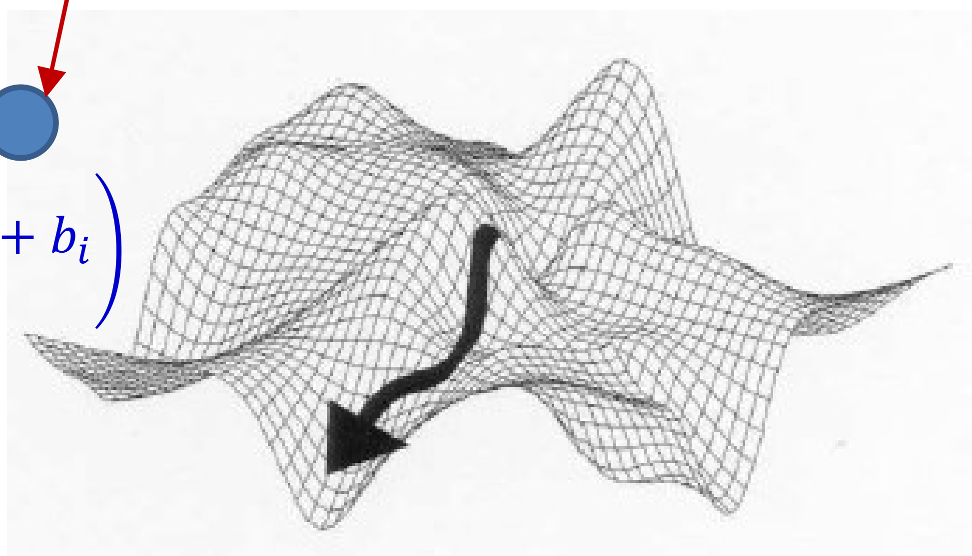
- The network will evolve until it arrives at a local minimum in the energy contour
- We proved that *every* change in the network will result in *decrease* in energy
  - So path to energy minimum is monotonic

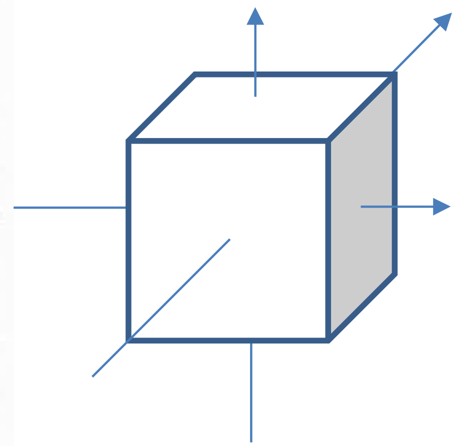
Evolution



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

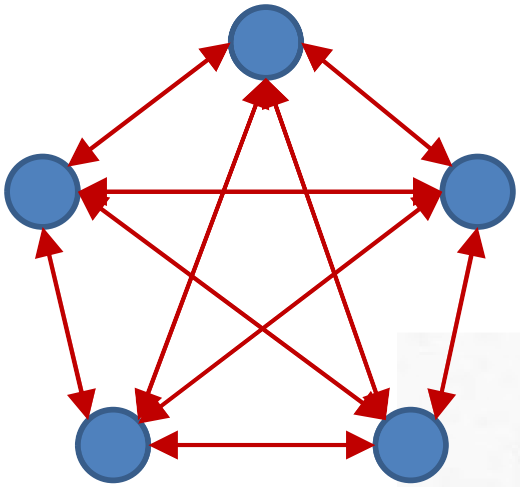
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$





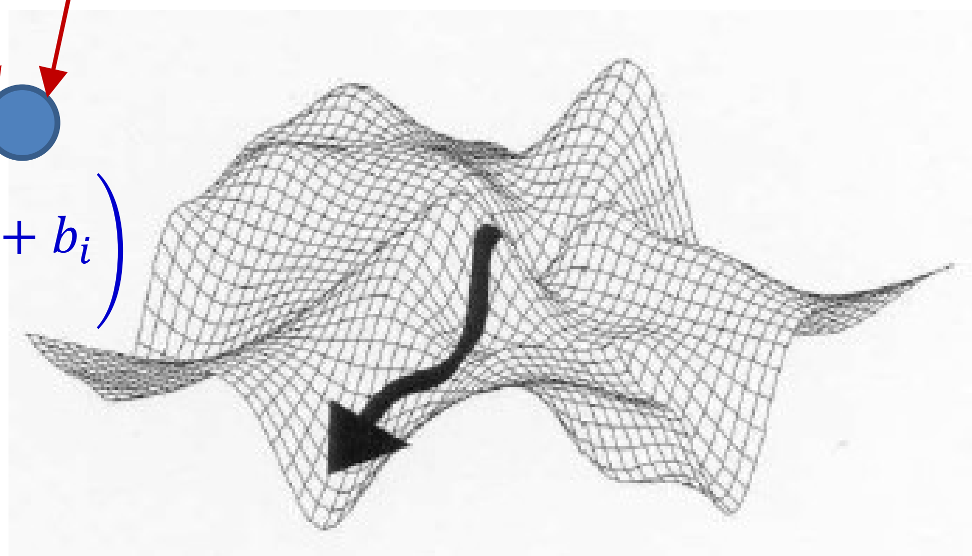
- For threshold activations the energy contour is only defined on a lattice
  - Corners of a unit cube on $[-1,1]^N$

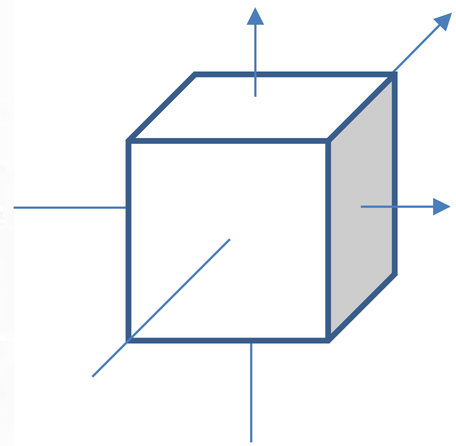
Evolution



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

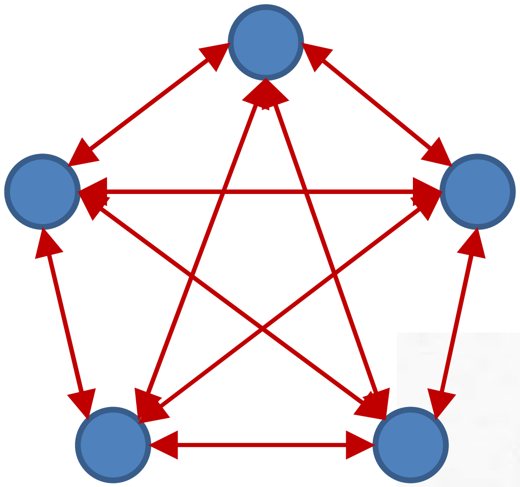
$$E = -\frac{1}{2}\sum_{i,j<i} w_{ij} y_i y_j$$



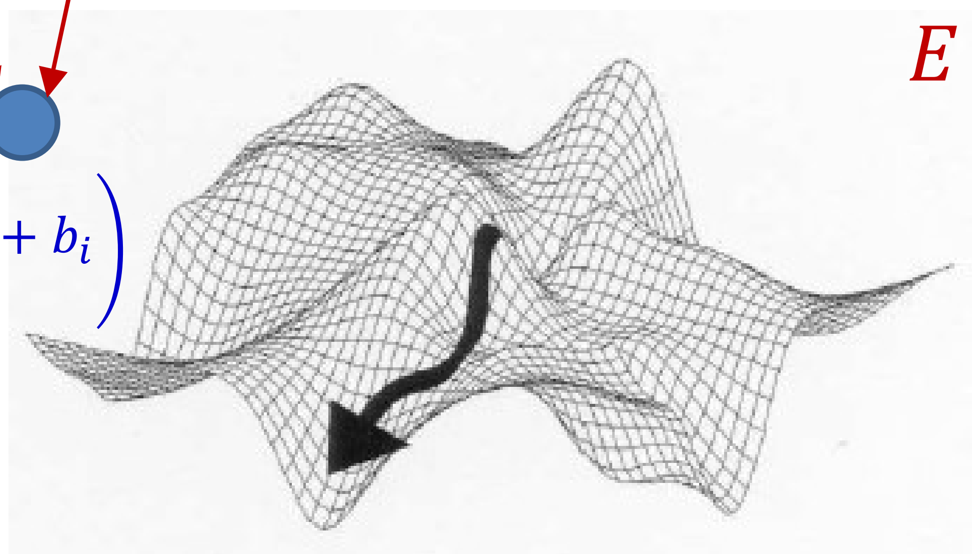


- For threshold activations the energy contour is only defined on a lattice
  - Corners of a unit cube on $[-1,1]^N$
- For tanh activations it will be a continuous function

Evolution



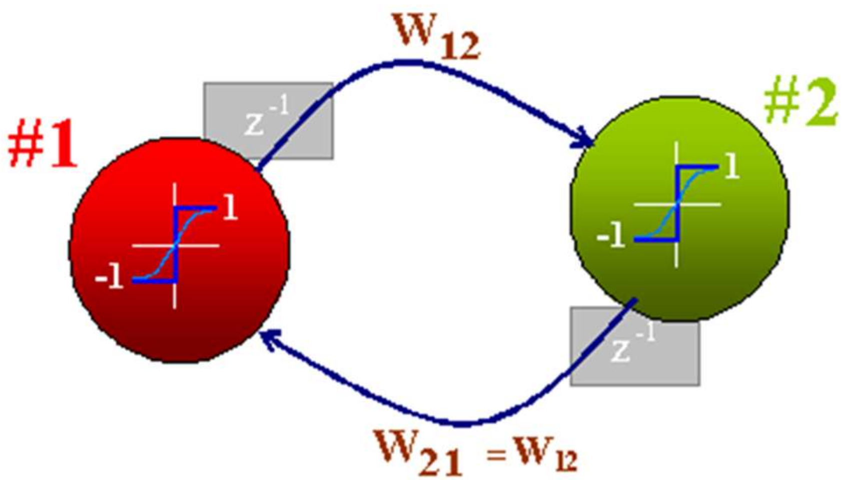
$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

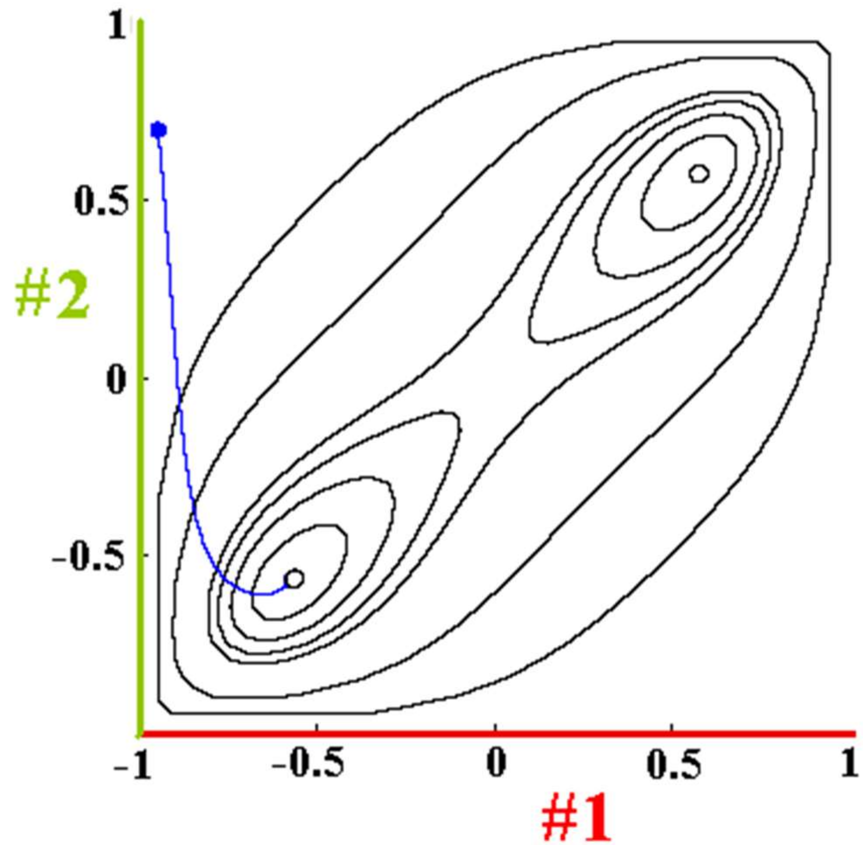


$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$

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

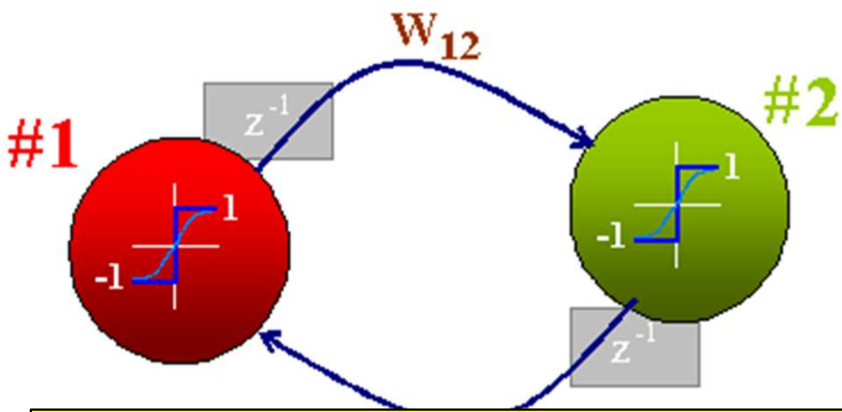
"Energy"contour for a 2-neuron net

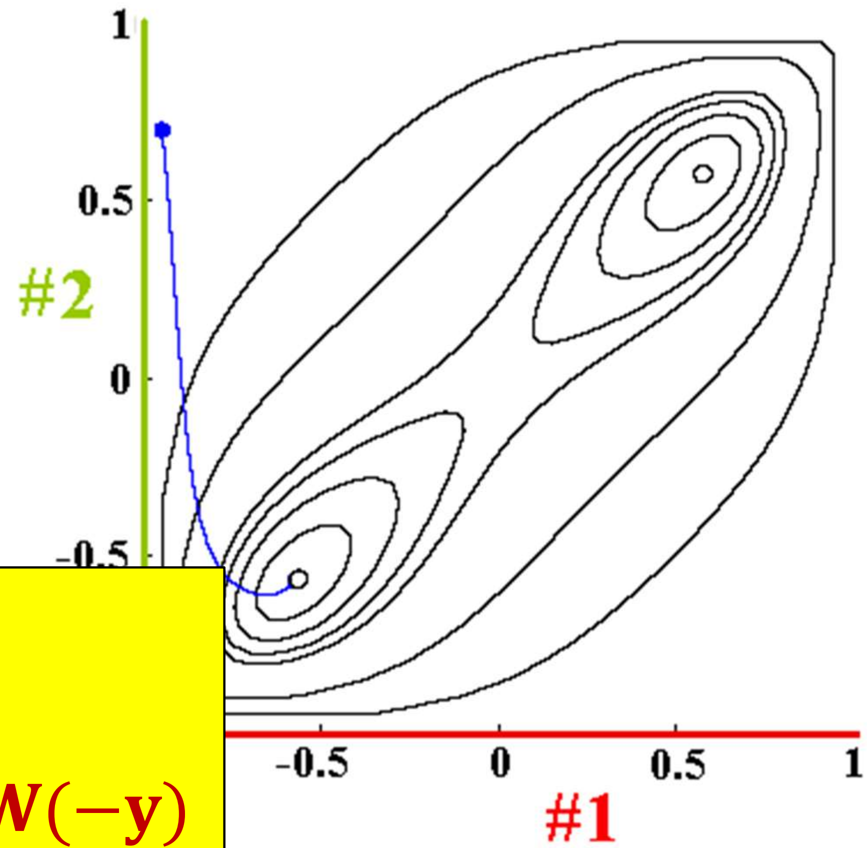




- Two stable states (tanh activation)
  - Symmetric, not at corners
  - Blue arc shows a typical trajectory for tanh activation

“Energy”contour for a 2-neuron net





Why symmetric?

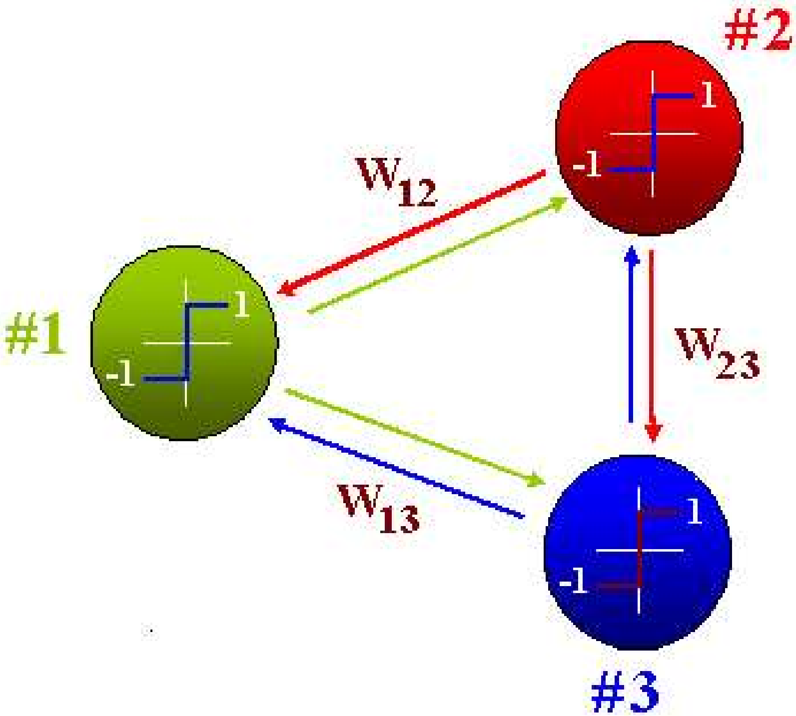
Because $-\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} = -\frac{1}{2}(-\mathbf{y})^T\mathbf{W}(-\mathbf{y})$

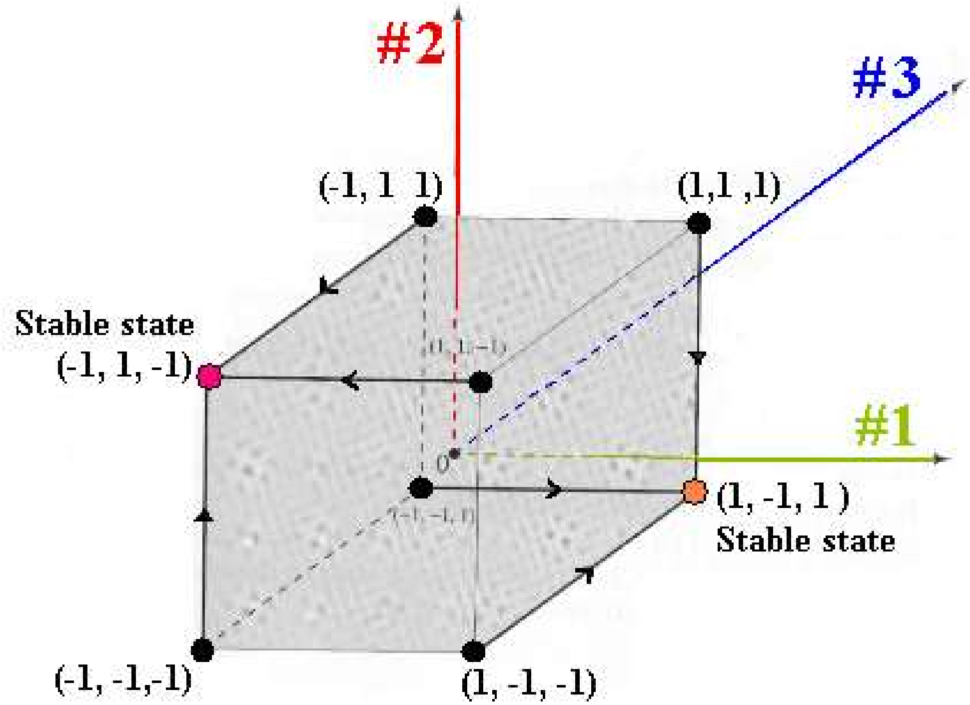
If $\hat{\mathbf{y}}$ is a local minimum, so is $-\hat{\mathbf{y}}$

•

– Blue arc shows a typical trajectory for sigmoid activation

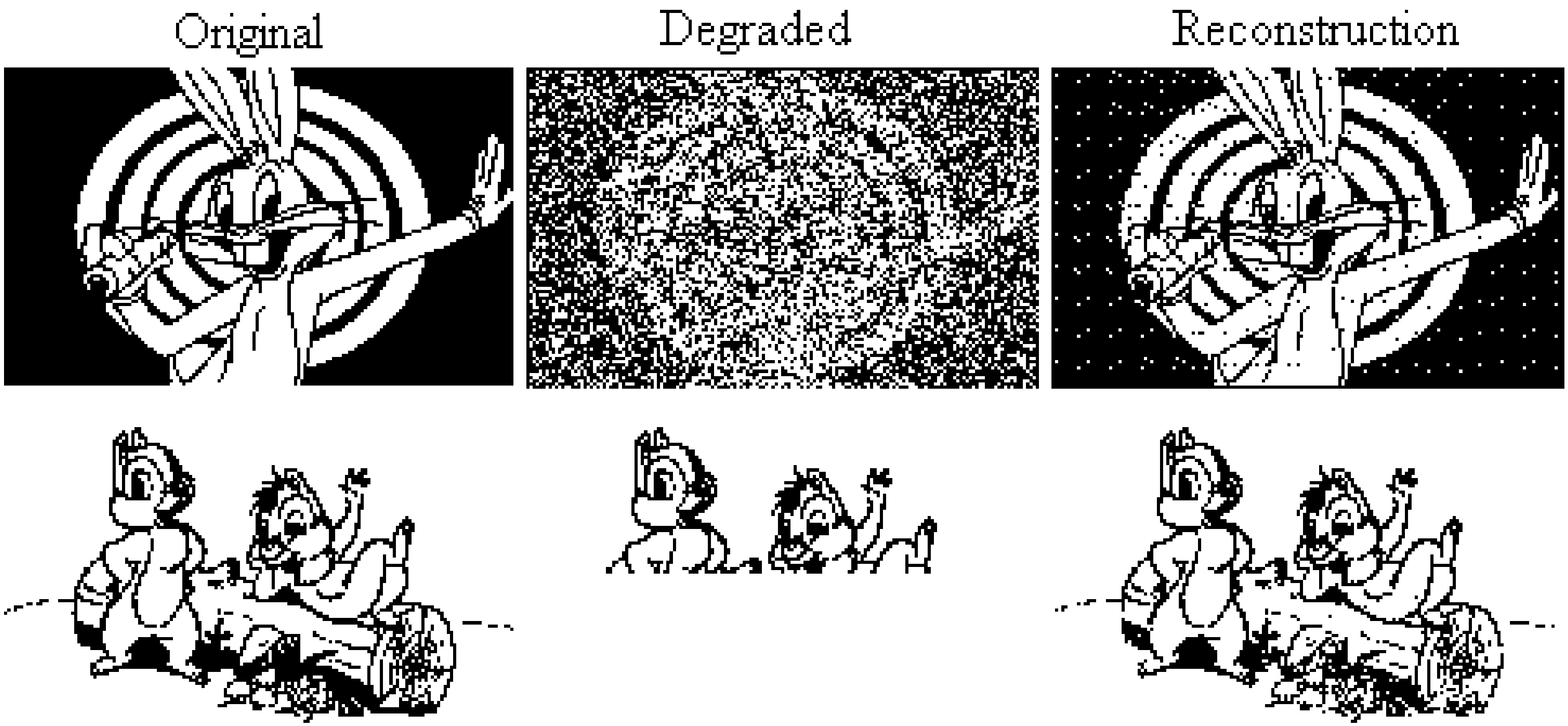
3-neuron net





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

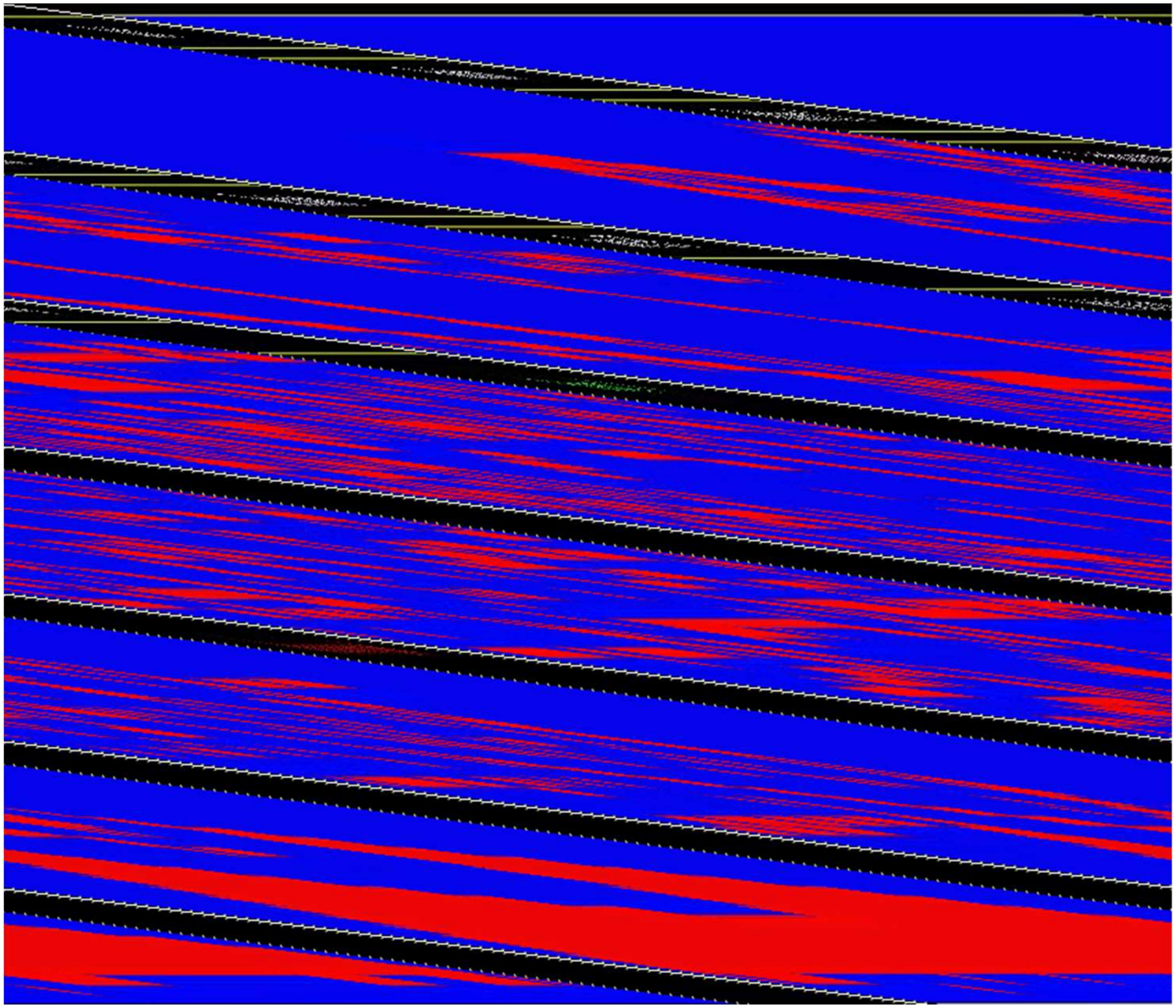
Examples: Content addressable memory



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

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

Hopfield net examples



Computational algorithm

1. Initialize network with initial pattern

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

2. Iterate until convergence

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

- 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

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

2. Iterate until convergence

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

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

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

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

does not change significantly any more

Story so far

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

Poll 2

Mark all that are correct about Hopfield nets

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

Poll 2

Mark all that are correct about Hopfield nets

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

Issues

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

- How many patterns can we store?

- How to “retrieve” patterns better..

Issues

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

- How many patterns can we store?

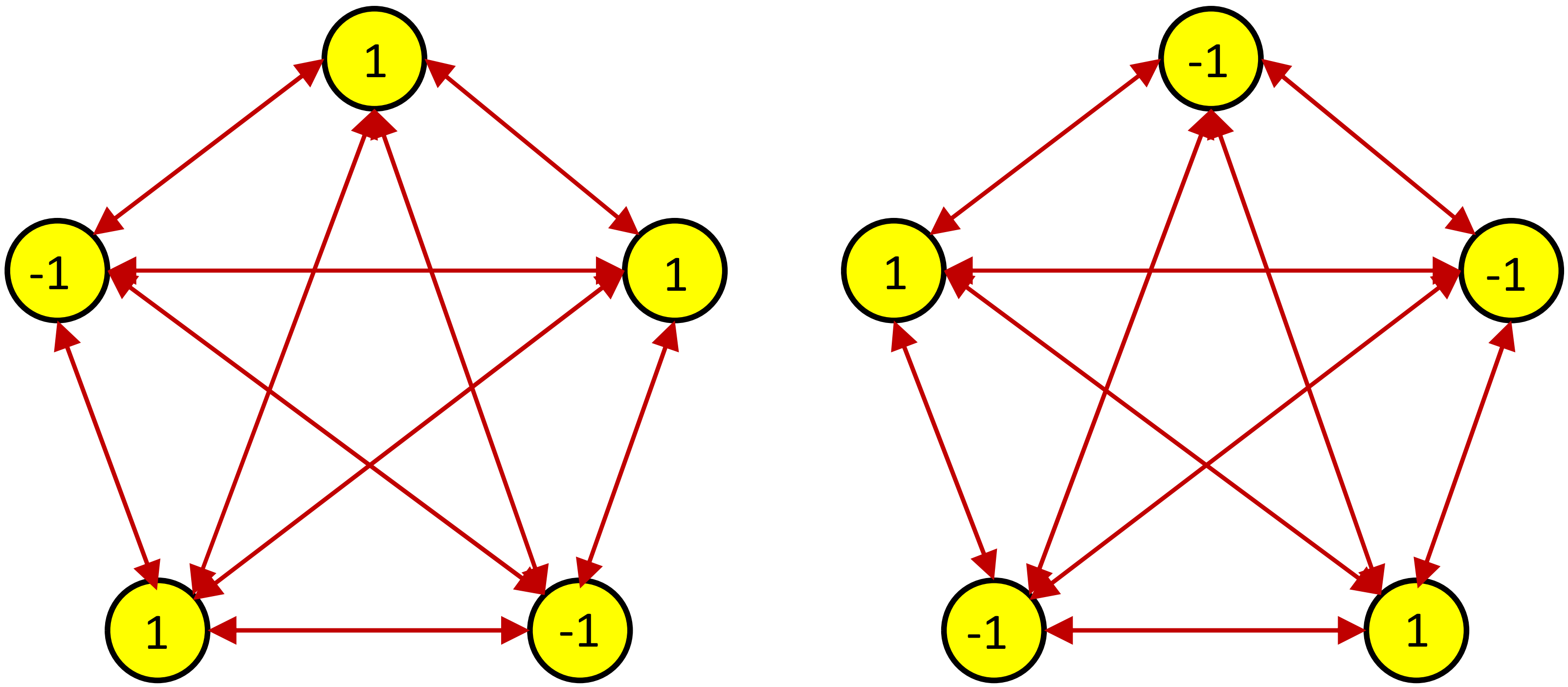
- How to “retrieve” patterns better..

How do we remember a *specific* pattern?

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

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

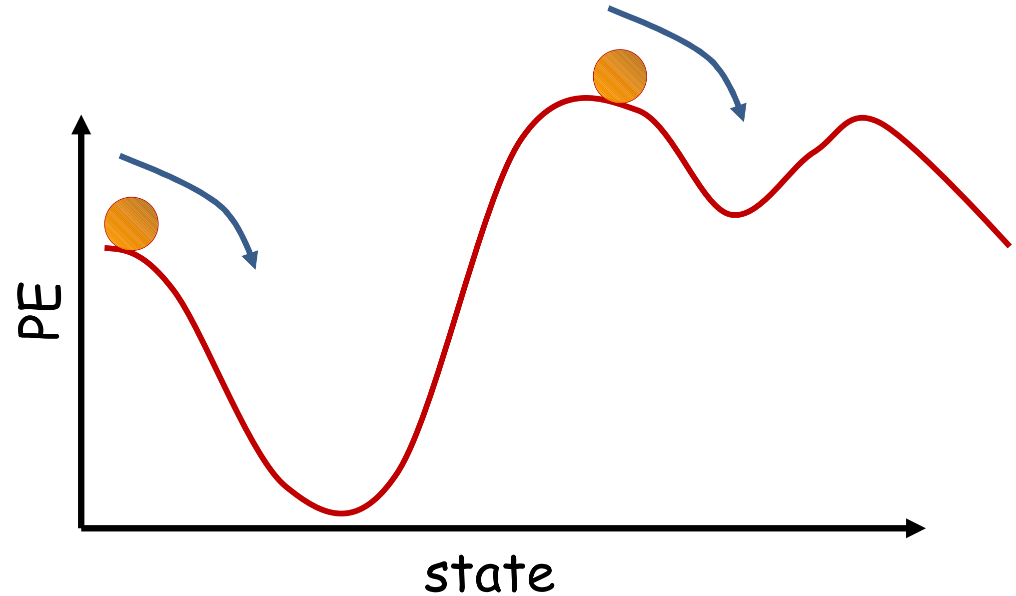
Storing patterns: Training a network

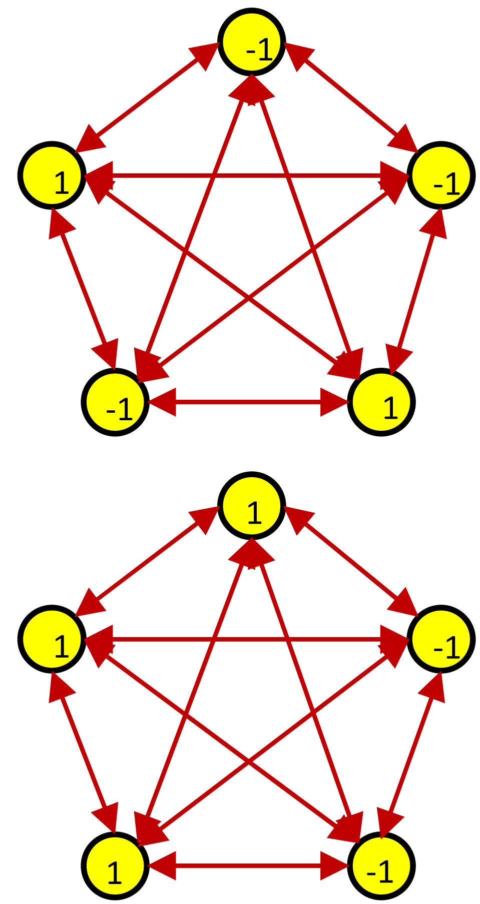


- A network that stores pattern P also naturally stores $-P$
  - Symmetry $E(P) = E(-P)$ since E is a function of $y_i y_j$

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

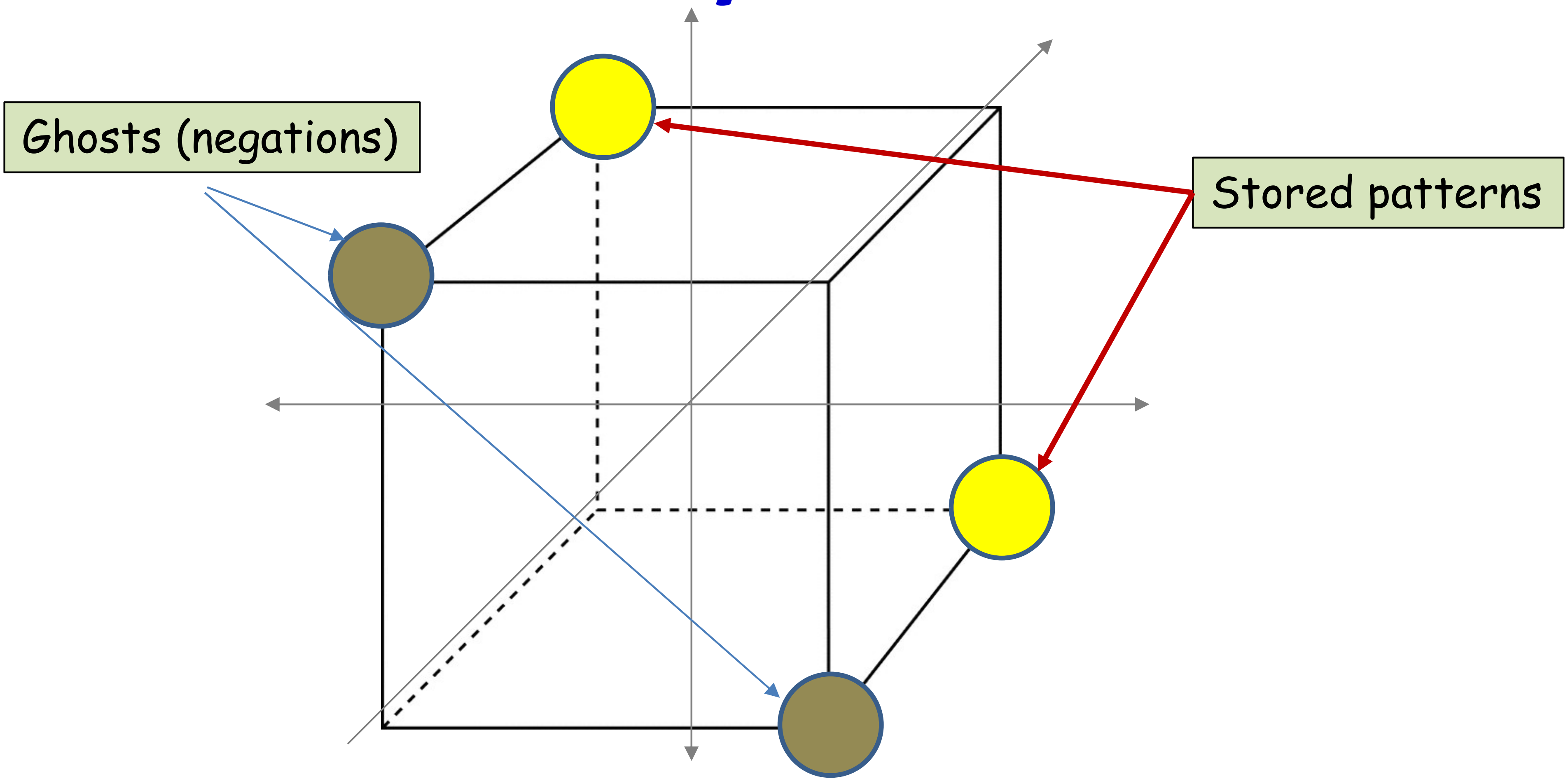
A network can store *multiple* patterns





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

Patterns you can store

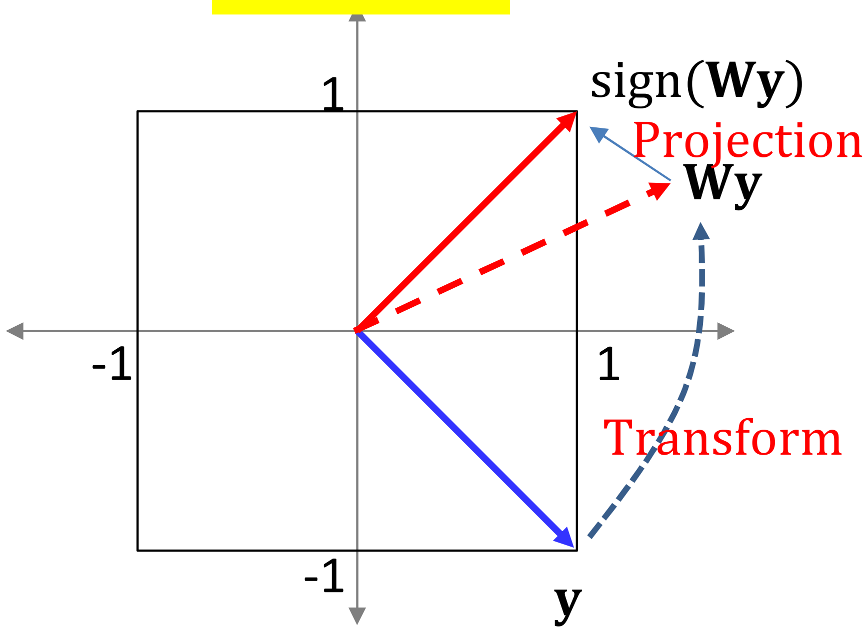


- All patterns are on the corners of a hypercube
  - If a pattern is stored, it's "ghost" is stored as well
  - Intuitively, patterns must ideally be maximally far apart

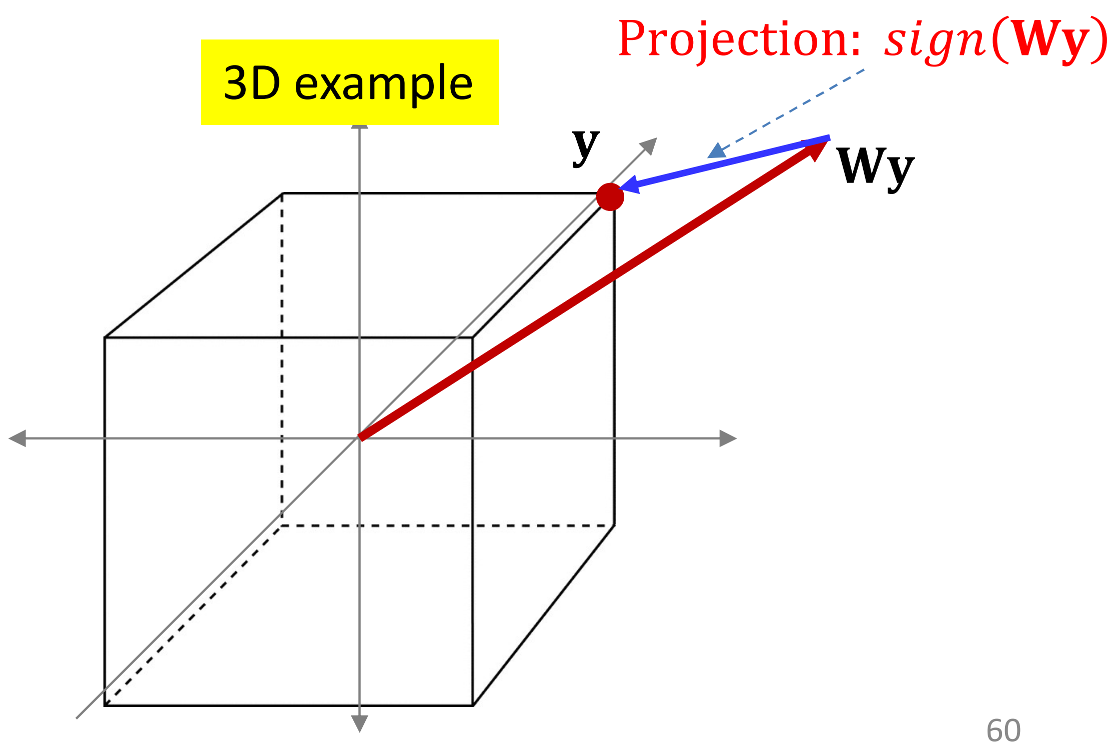
Evolution of the network

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

2D example



3D example



Storing patterns

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

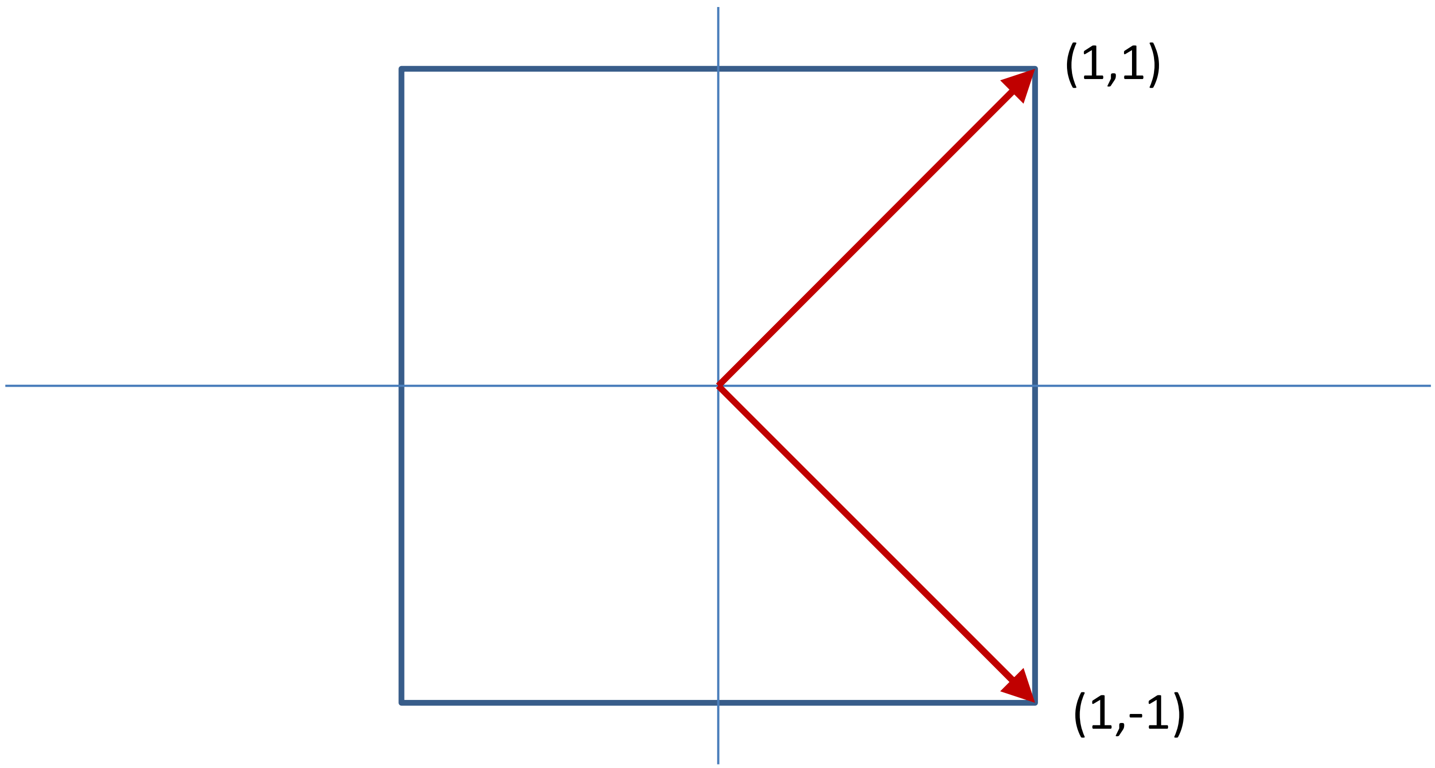
- How many such $\mathbf{y}_p$ can we have?

Random fact that should interest you

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

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

Only N patterns?



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

random fact that should interest you

- The Eigenvectors of any symmetric matrix $\mathbf{W}$ are orthogonal

- The Eigen*values* may be positive or negative

Storing more than one pattern

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

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

Storing patterns

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

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

- 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

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

- ***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

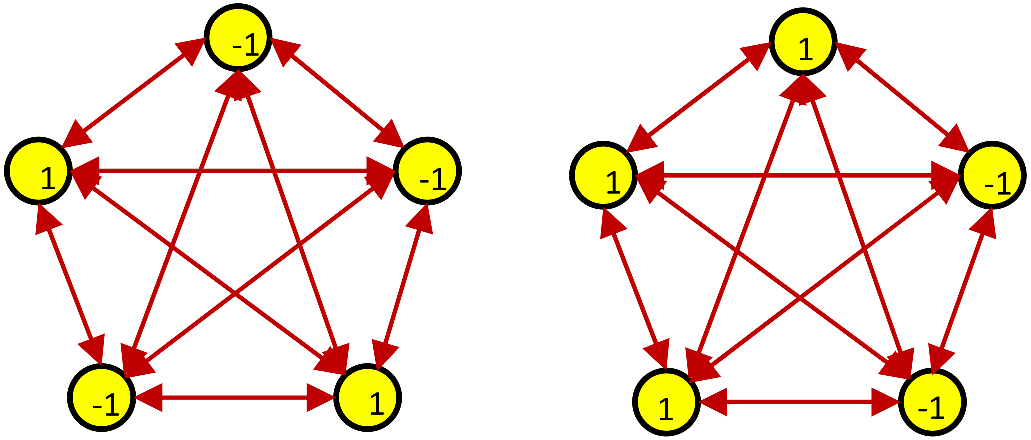
- With an network of N units (i.e. N-bit patterns)
- The maximum number of stable patterns is actually *exponential* in N
  - McElice and Posner, 84'
  - E.g. when we had the H se patterns, *all* patterns are stable

How do we find this network?

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

Can we do something about this?

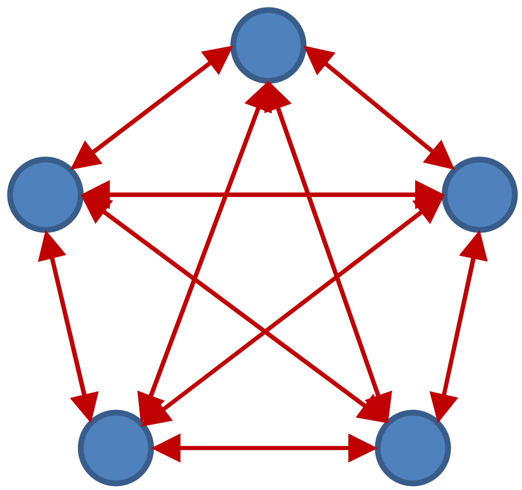
Storing a pattern



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

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

Consider the energy function

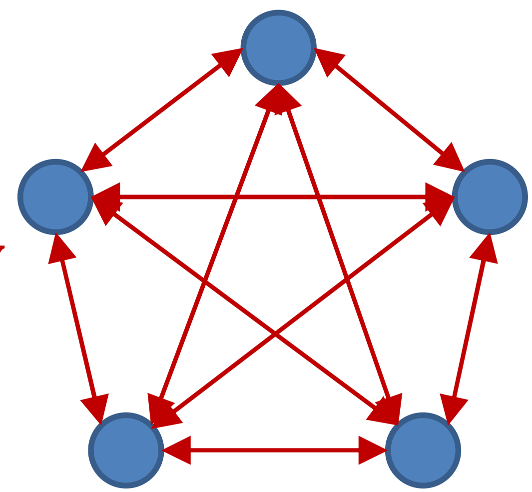


$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y}$$

- 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)

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

The bias can be captured by another fixed-value component

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

Optimizing W

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

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

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

Optimizing W

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

- Simple gradient descent:

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

Optimizing W

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

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

Optimizing W

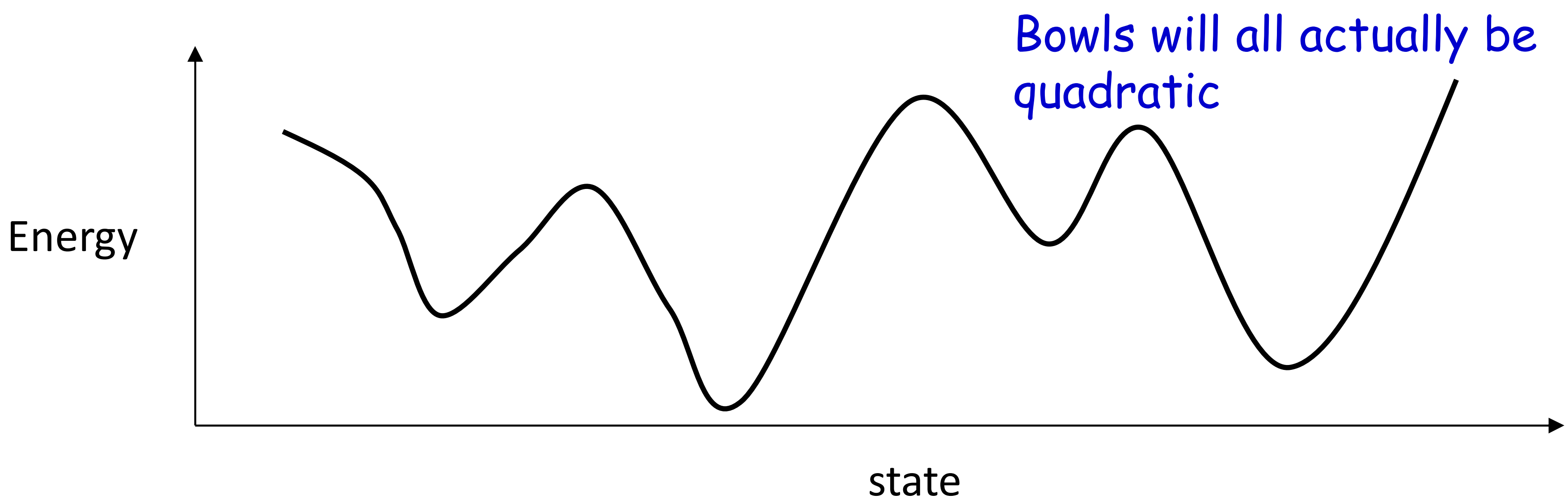
$$\mathbf{W} = \mathbf{W} + \eta \left(\sum_{\mathbf{y} \in \mathbf{Y}_P} \mathbf{y}\mathbf{y}^T - \sum_{\mathbf{y} \notin \mathbf{Y}_P} \mathbf{y}\mathbf{y}^T \right)$$

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

The training again..

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

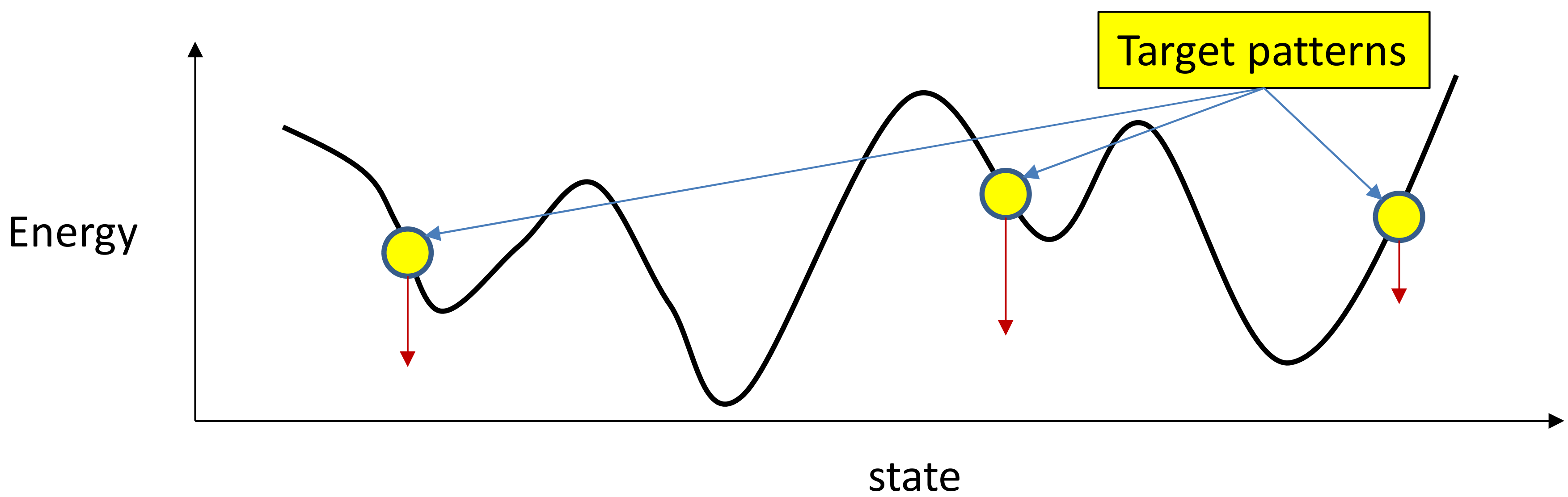
- Note the energy contour of a Hopfield network for any weight $\mathbf{W}$



The training again

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

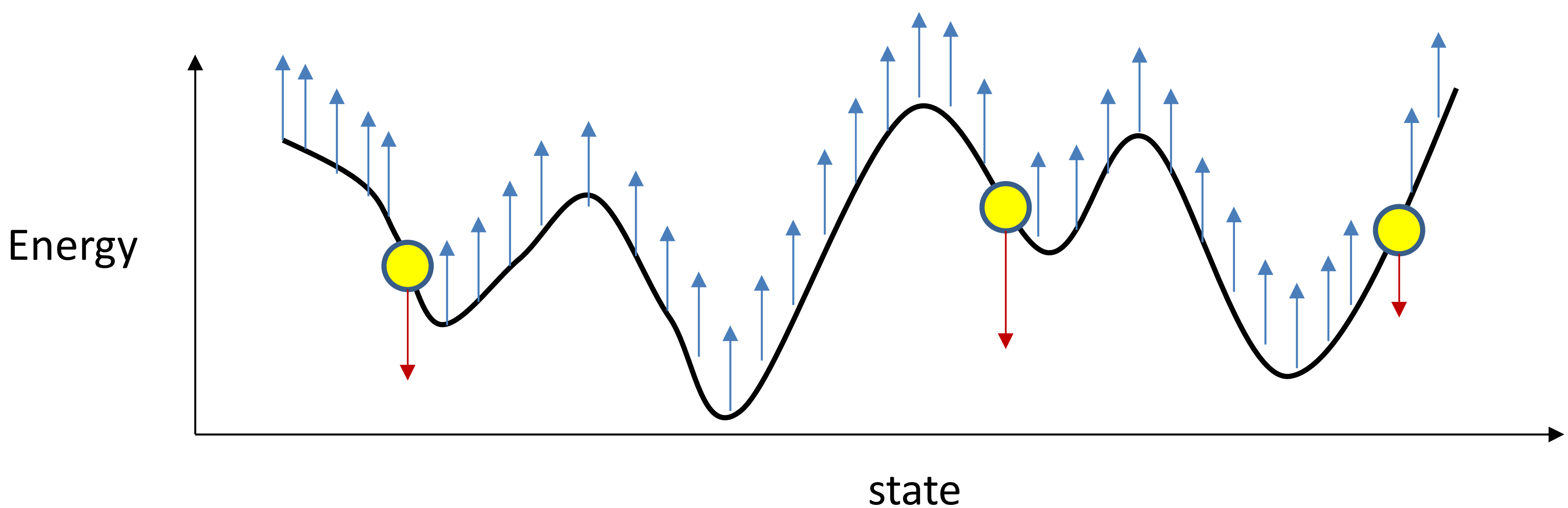
- The first term tries to *minimize* the energy at target patterns
  - Make them local minima
  - Emphasize more "important" memories by repeating them more frequently



The negative class

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

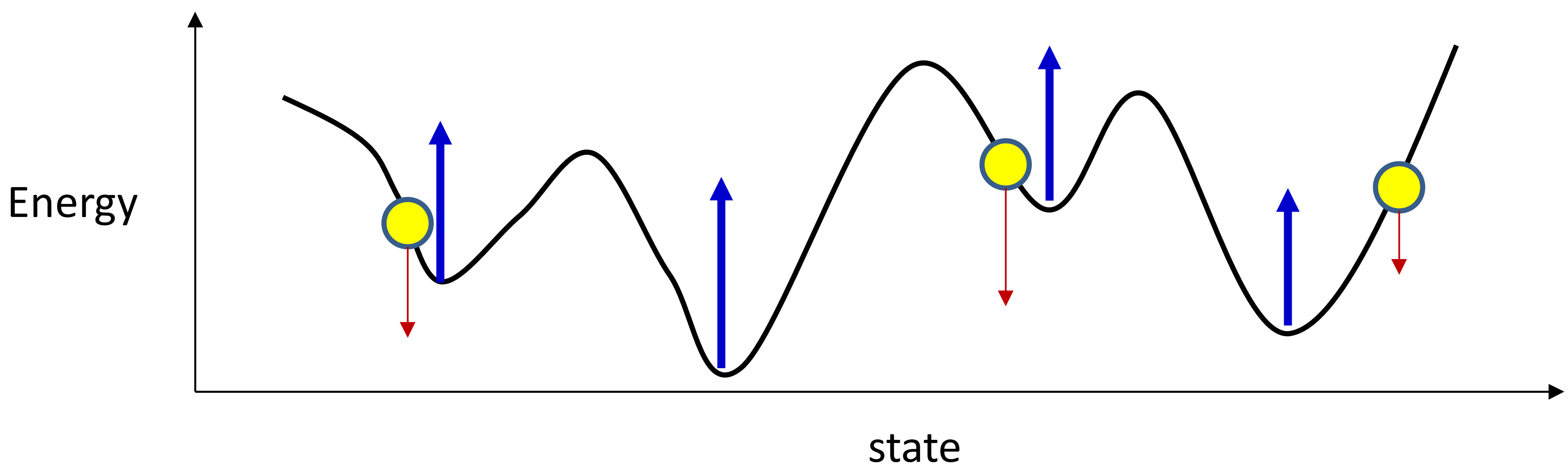
- The second term tries to “raise” all non-target patterns
  - Do we need to raise *everything*?



Option 1: Focus on the valleys

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

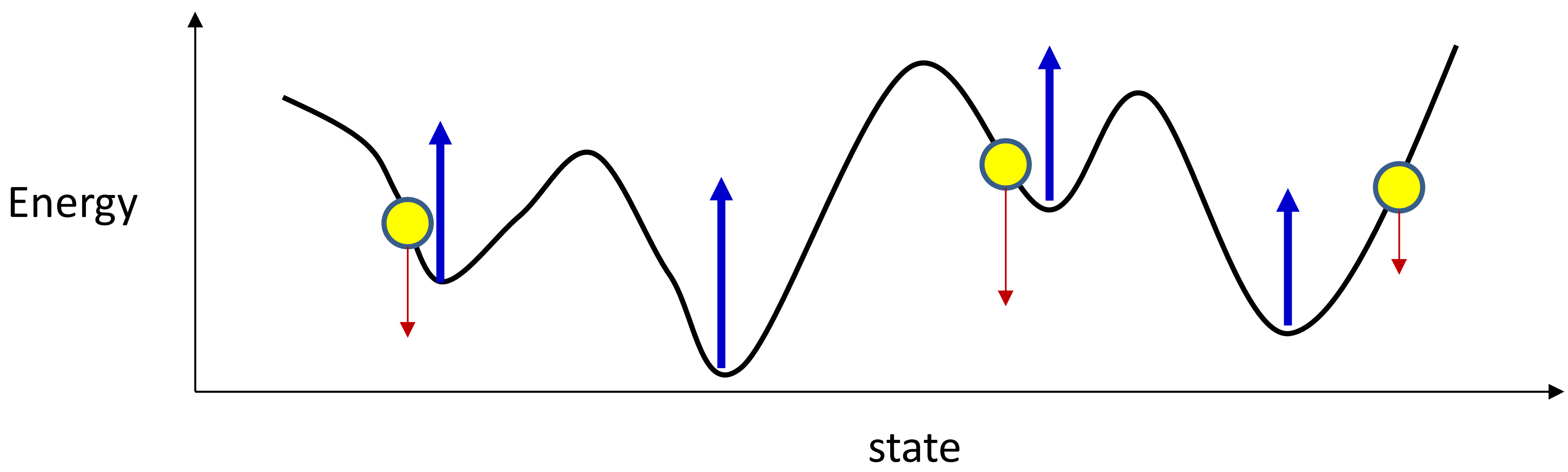
- Focus on raising the valleys
  - If you raise *every* valley, eventually they'll all move up above the target patterns, and many will even vanish



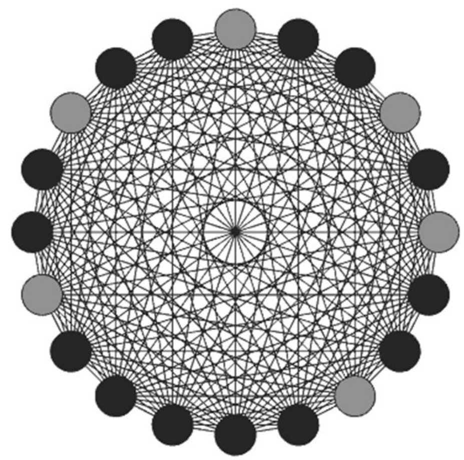
Identifying the valleys..

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

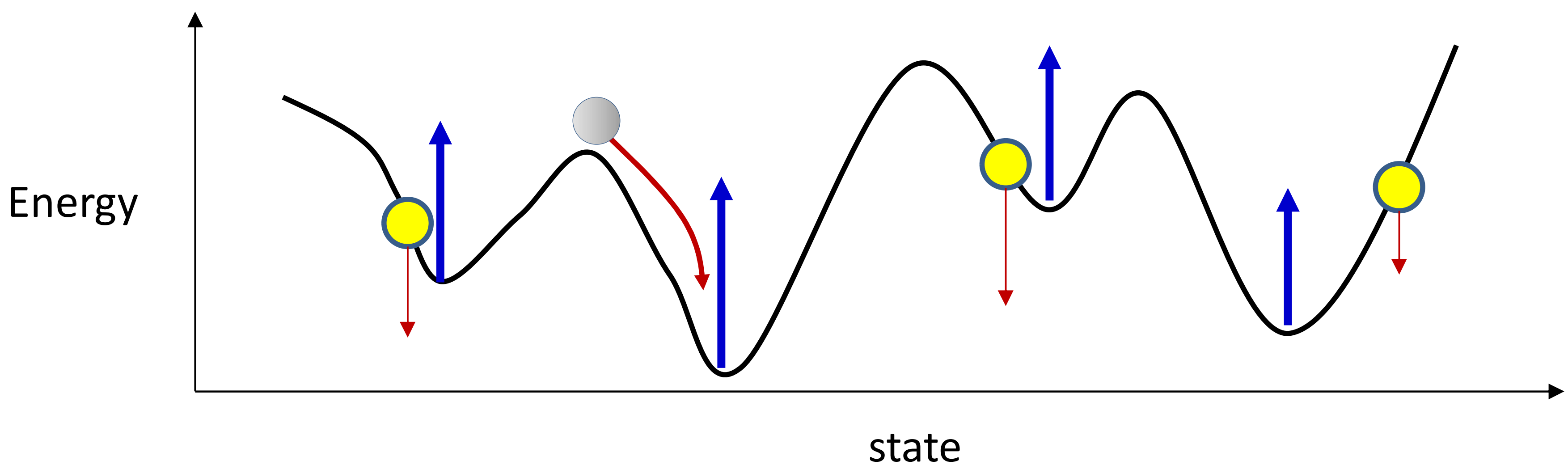
- Problem: How do you identify the valleys for the current $\mathbf{W}$?



Identifying the valleys..



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



Training the Hopfield network

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

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

Training the Hopfield network: SGD version

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

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

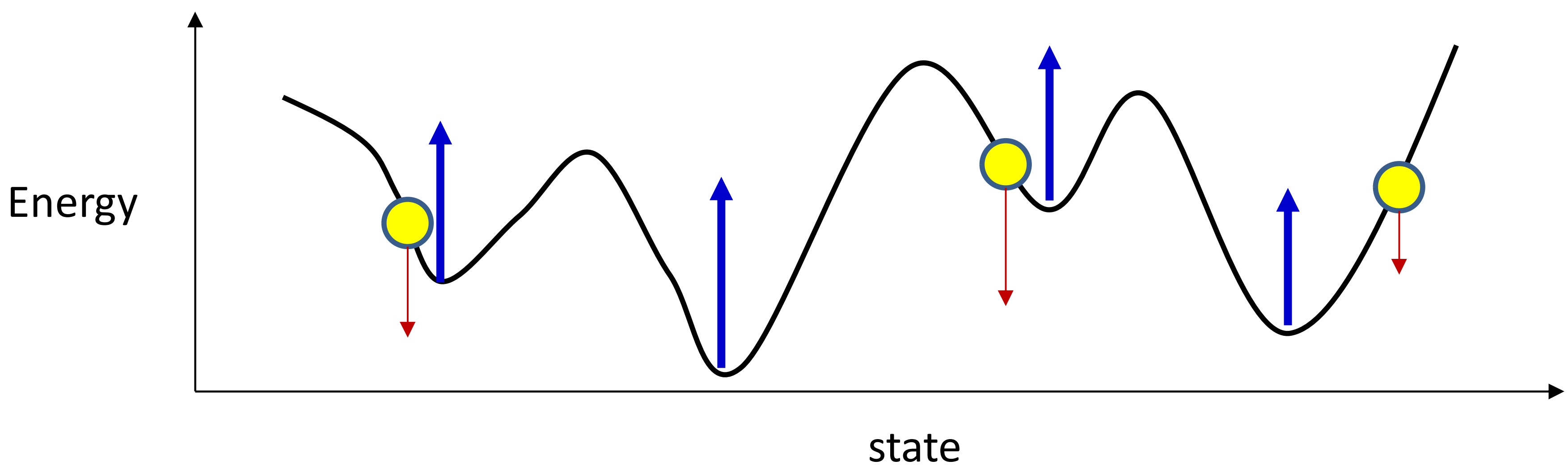
Training the Hopfield network

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

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

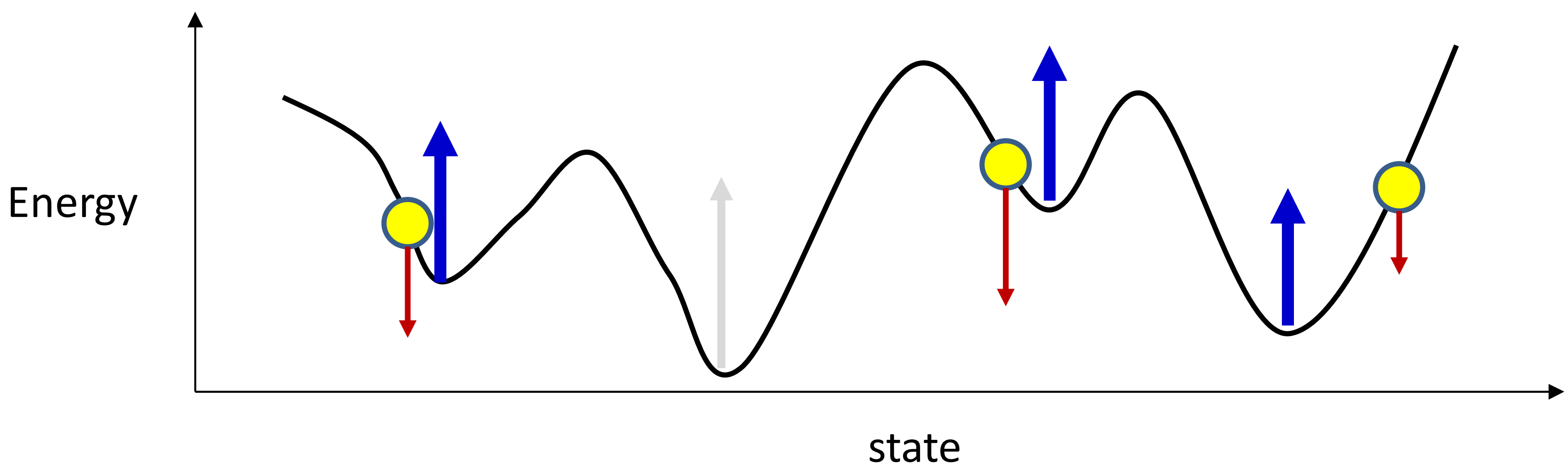
Which valleys?

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

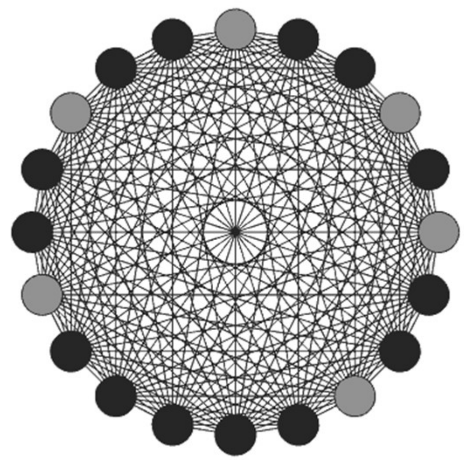


Which valleys?

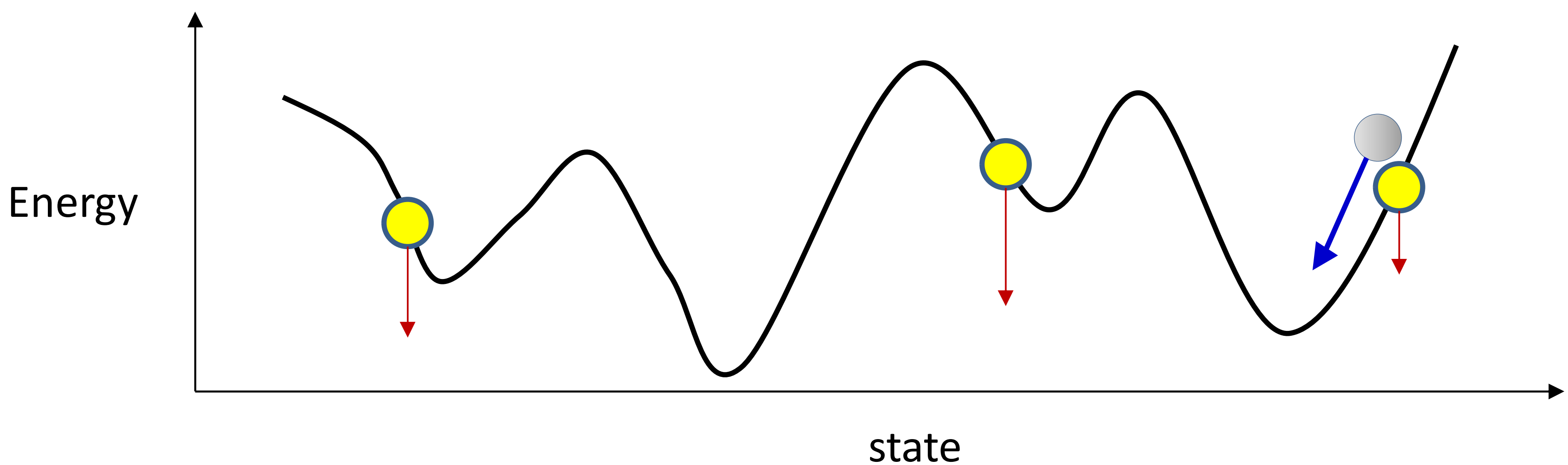
- Should we *randomly* sample valleys?
  - Are all valleys equally important?
- Major requirement: memories must be stable
  - They *must* be broad valleys
- Spurious valleys in the neighborhood of memories are more important to eliminate



Identifying the valleys..



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



Training the Hopfield network

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

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

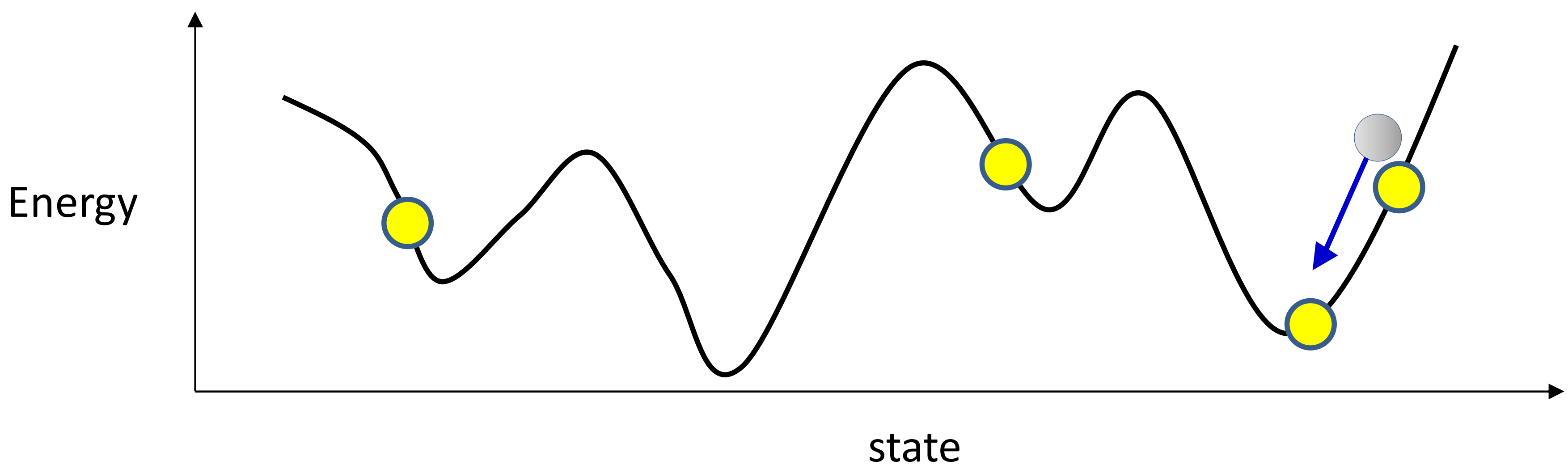
Training the Hopfield network: SGD version

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

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

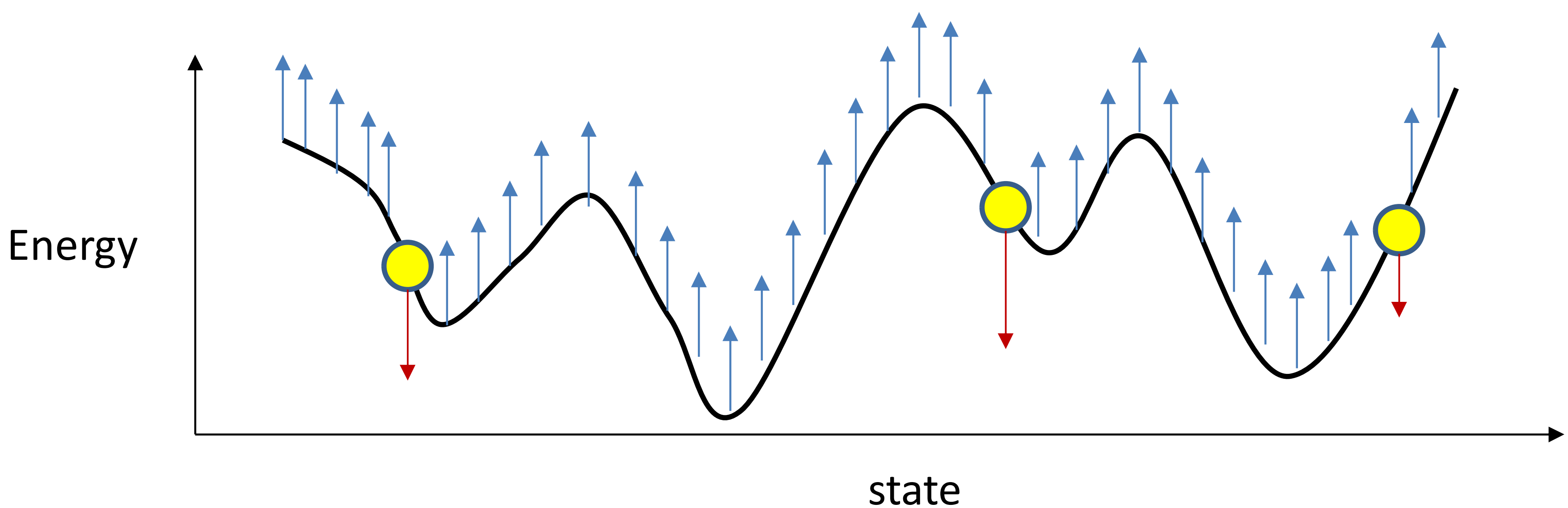
A possible problem

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



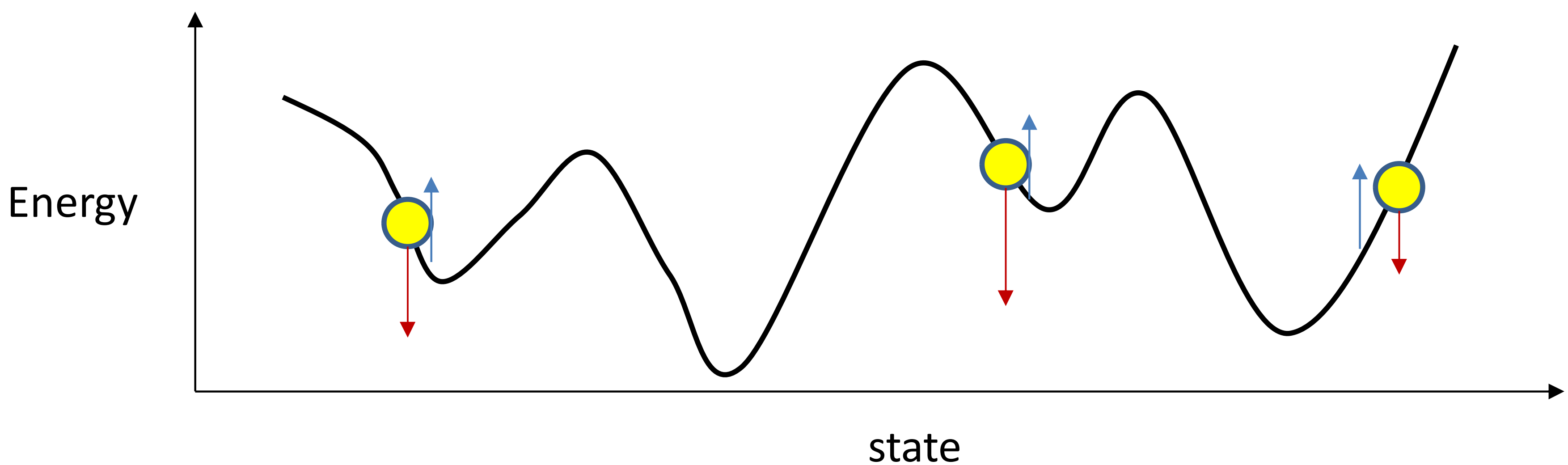
A related issue

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



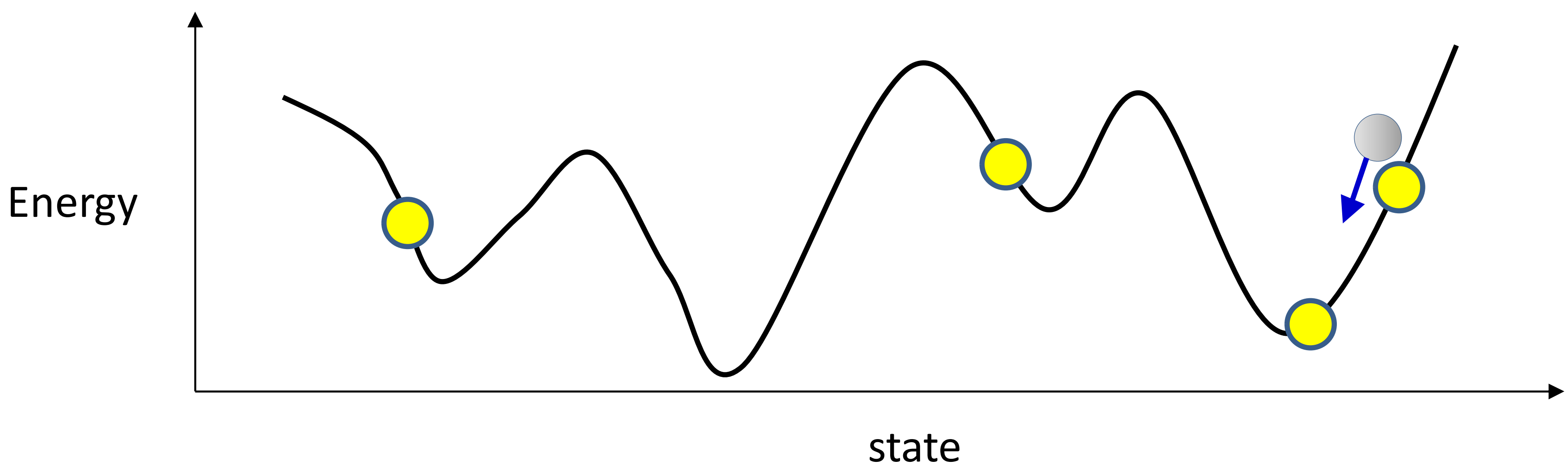
A related issue

- 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

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

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

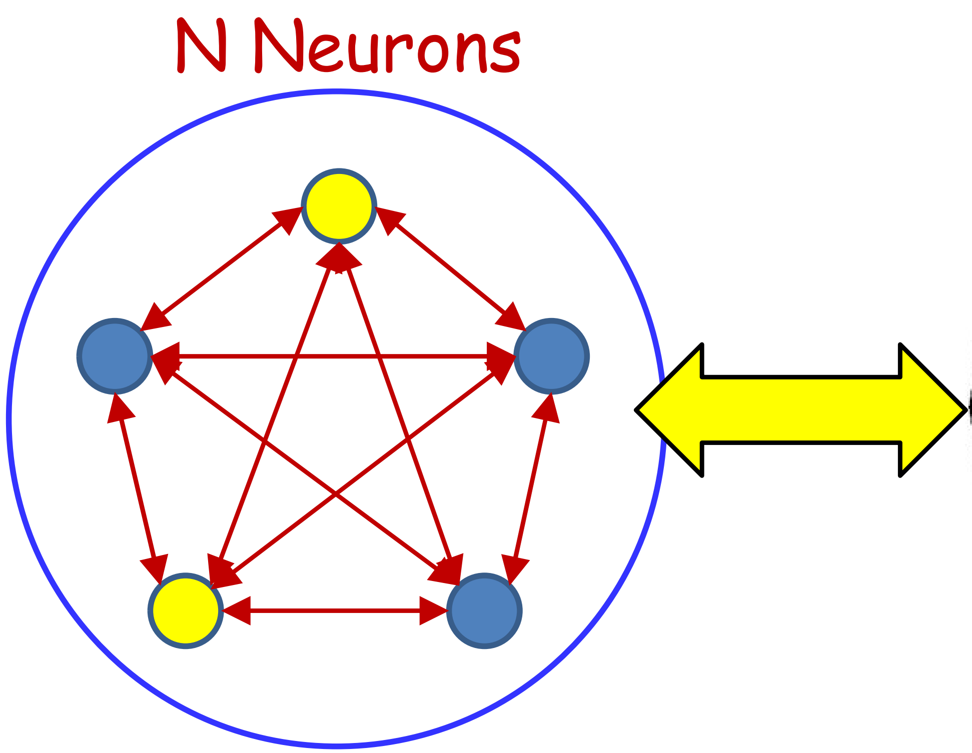
Story so far

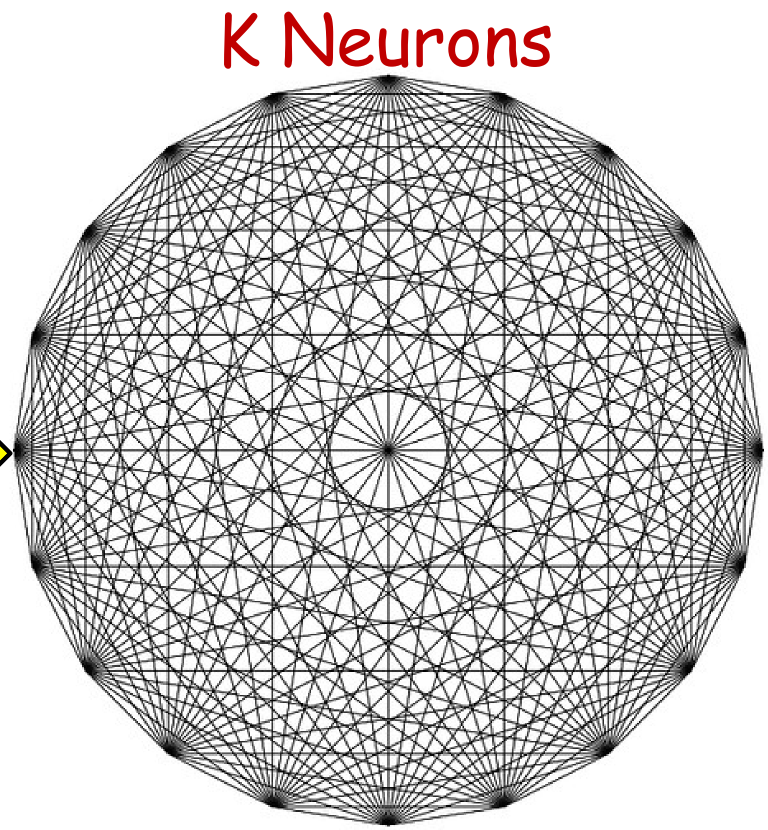
- Hopfield nets with N neurons can store up to N random patterns
  - But comes with many parasitic memories
- Networks that store $O(N)$ memories can be trained through optimization
  - By minimizing the energy of the target patterns, while increasing the energy of the neighboring patterns

Storing more than N patterns

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

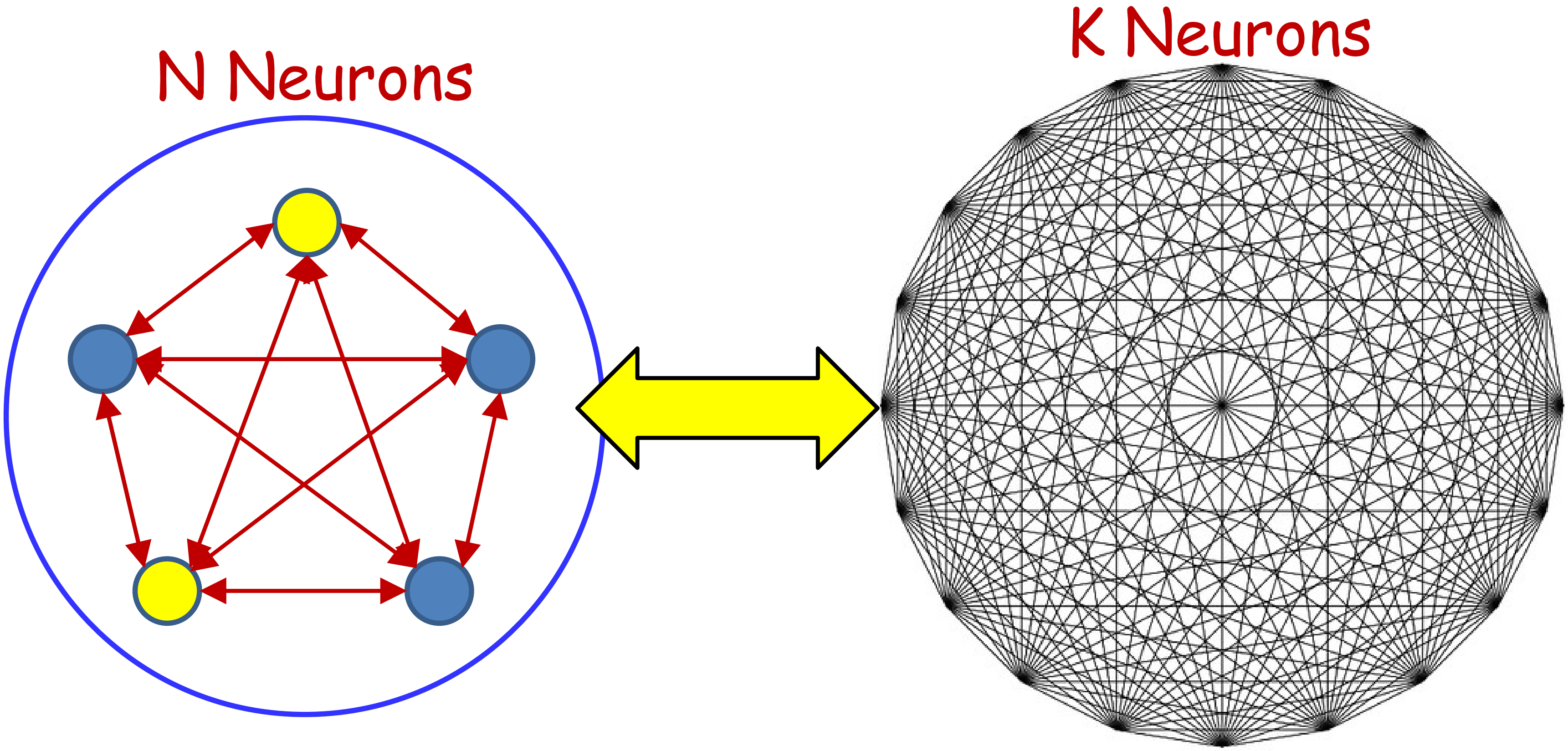
Expanding the network





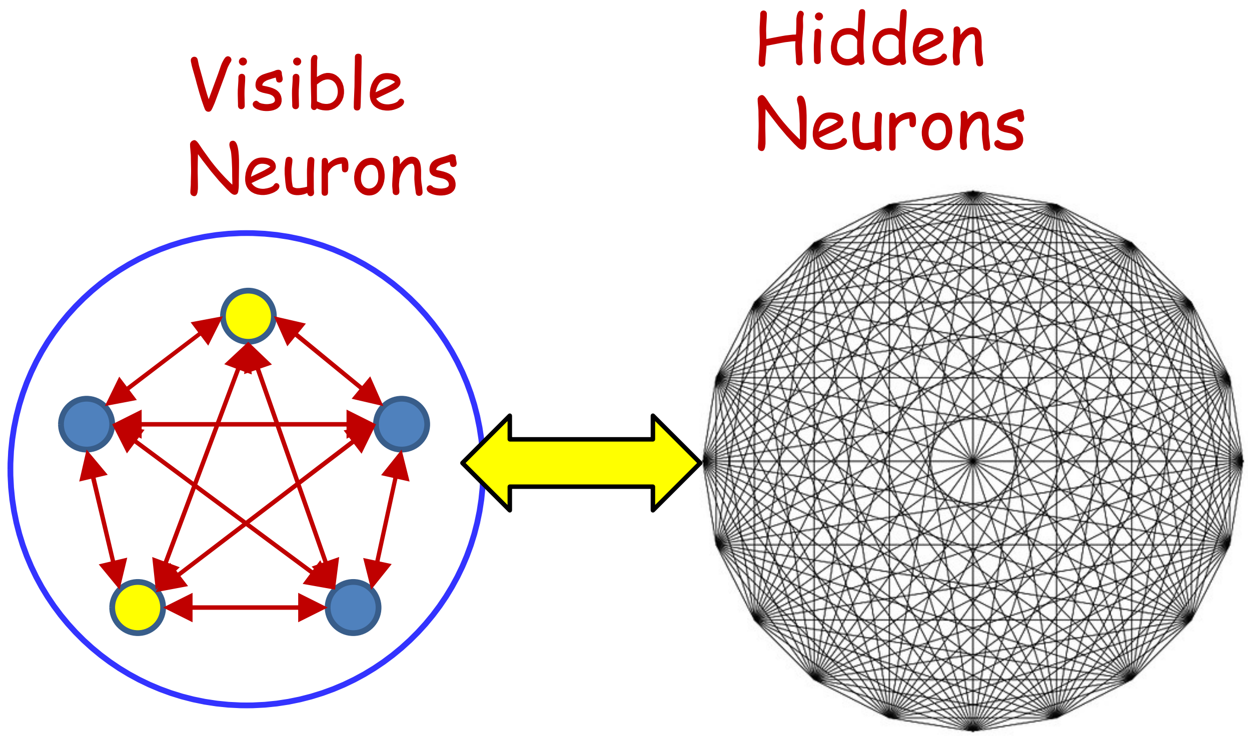
- Add a large number of neurons whose actual values you don't care about!

Expanded Network



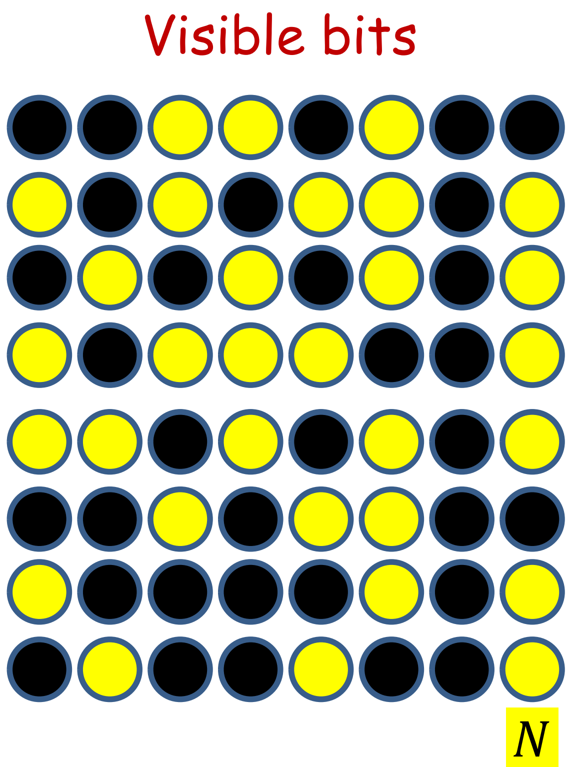
- New capacity: $\sim(N + K)$ patterns
  - Although we only care about the pattern of the first N neurons
  - We're interested in *N-bit* patterns

Terminology



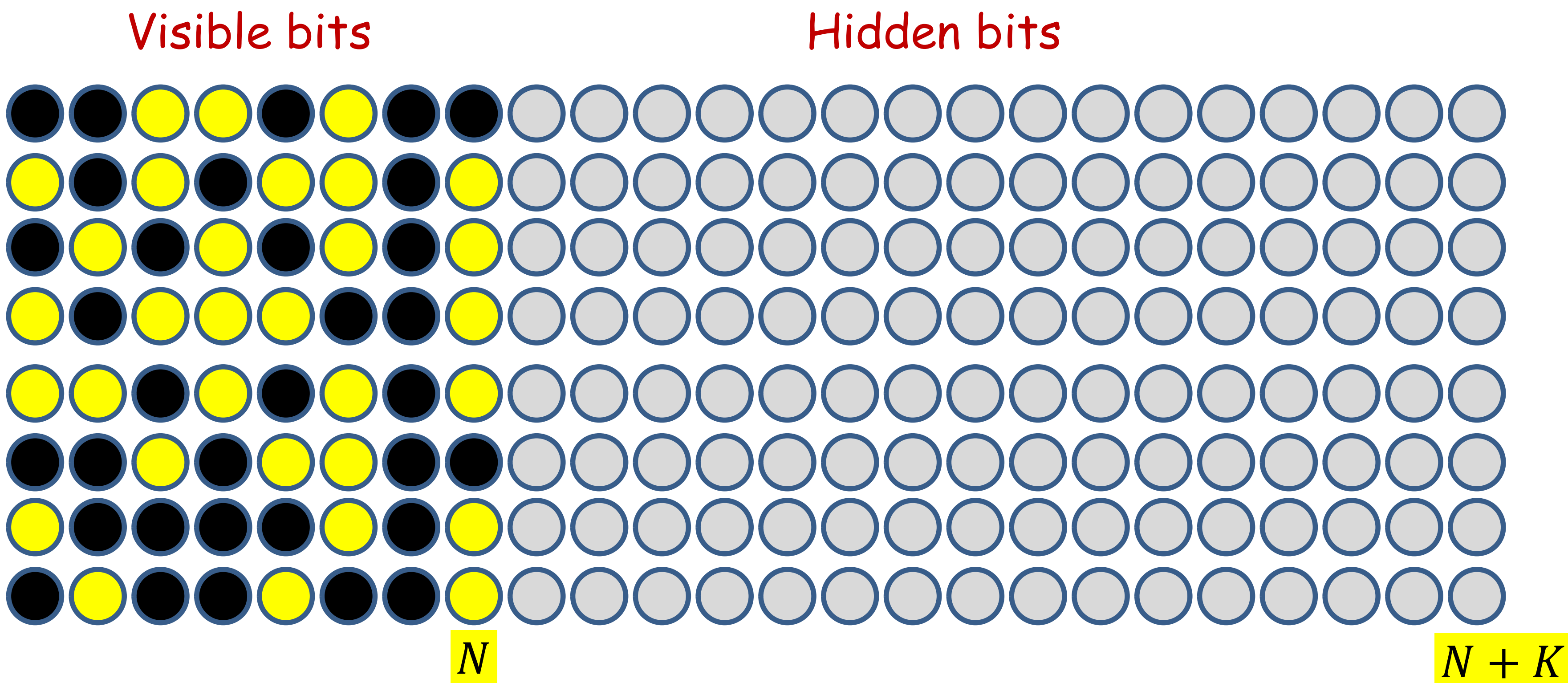
- Terminology:
  - The neurons that store the actual patterns of interest: *Visible neurons*
  - The neurons that only serve to increase the capacity but whose actual values are not important: *Hidden neurons*
  - These can be set to anything in order to store a visible pattern

Increasing the capacity: bits view



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

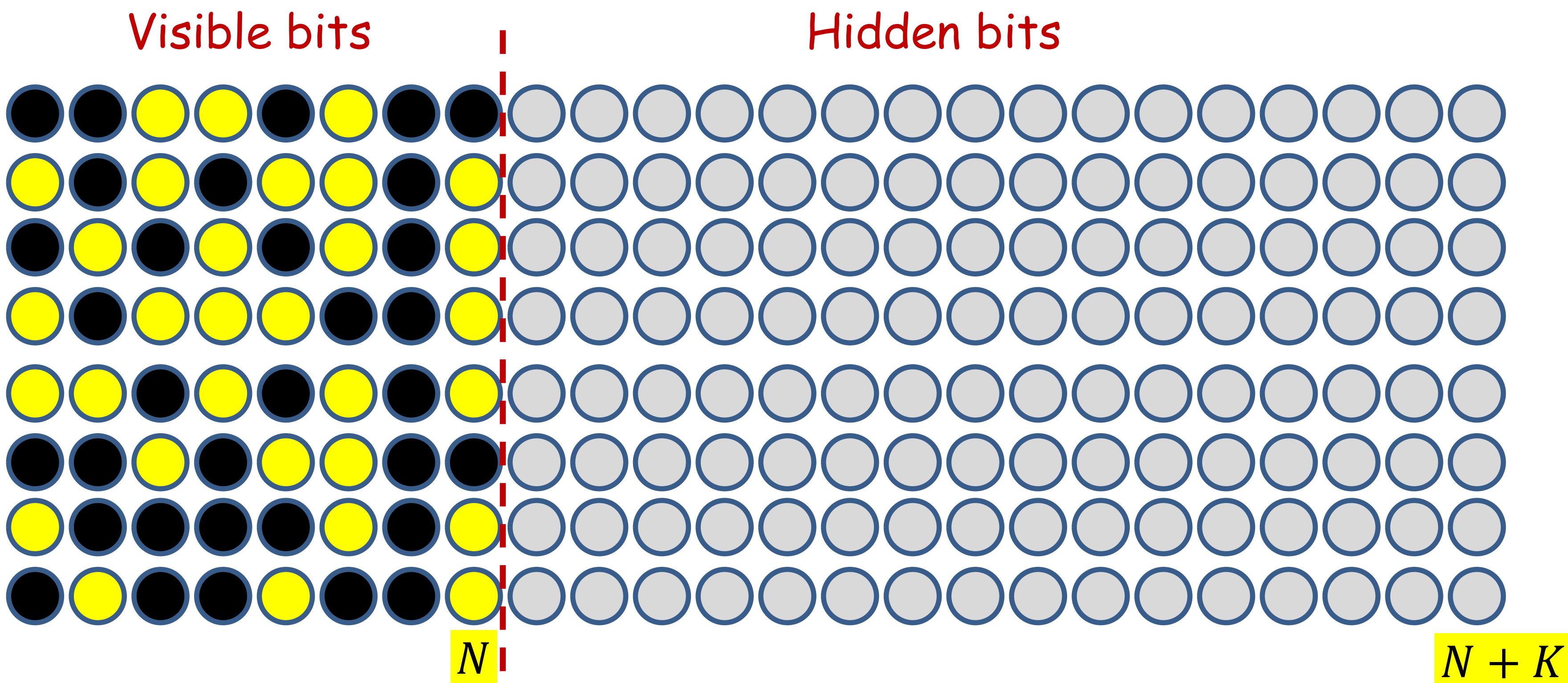
Increasing the capacity: bits view



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

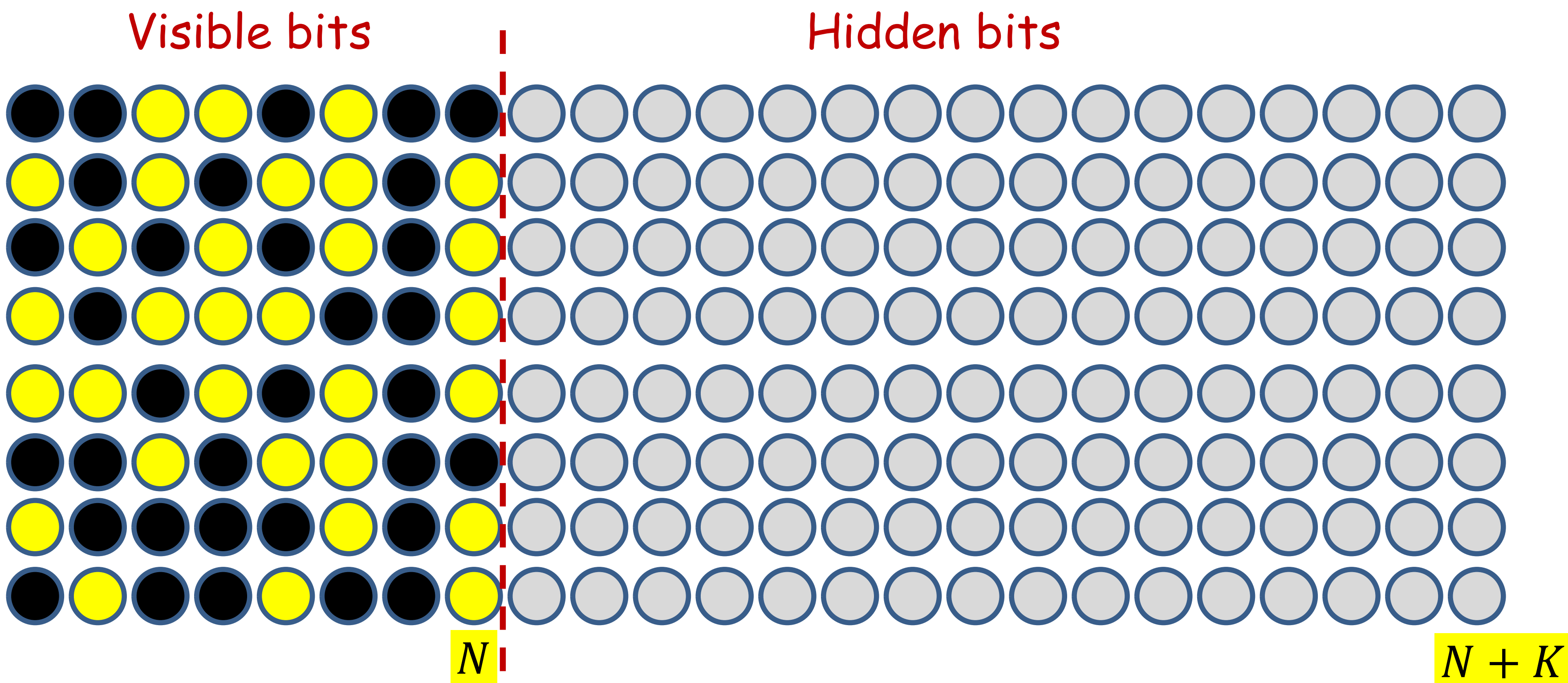
- So, let's *pad* the patterns with *K* "don't care" bits
  - The new width of the patterns is N+K
  - Now we can store N+K patterns!

Issues: Storage



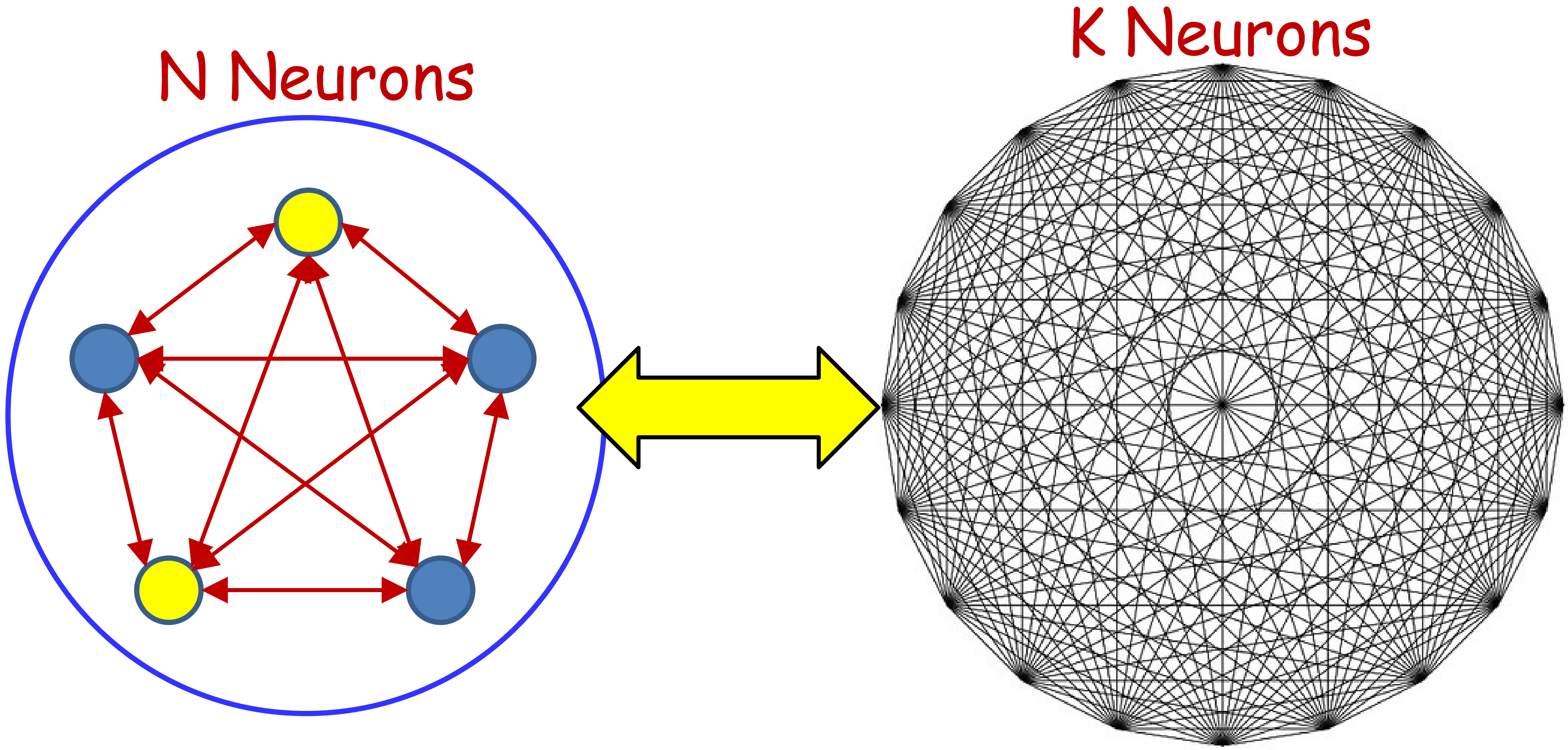
- What patterns do we fill in the don't care bits?
  - Simple option: Randomly
    - Flip a coin for each bit
  - We could even compose *multiple* extended patterns for a base pattern to increase the probability that it will be recalled properly
    - Recalling any of the extended patterns from a base pattern will recall the base pattern
- How do we store the patterns?
  - Standard optimization method should work

Issues: Recall



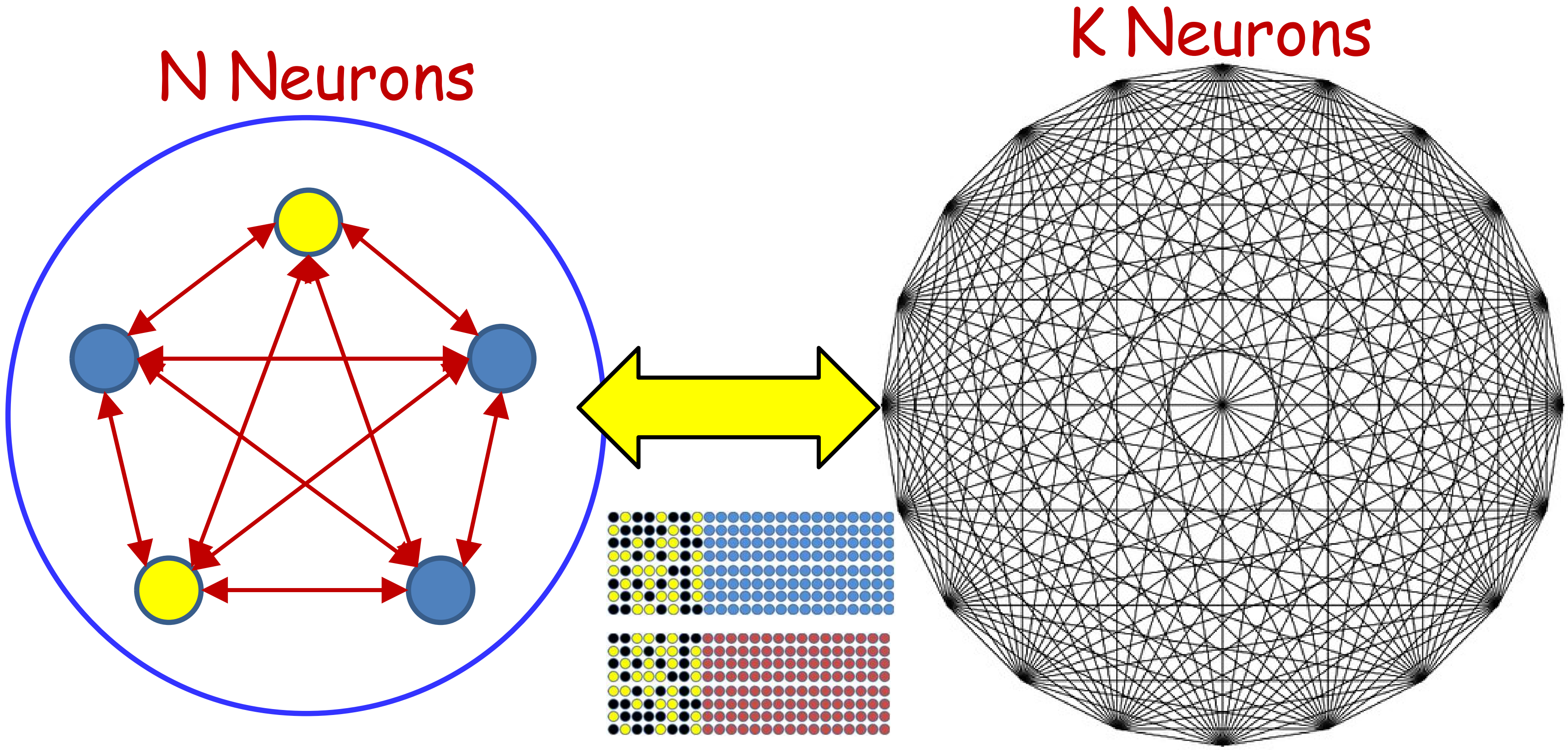
- How do we retrieve a memory?
- Can do so using usual "evolution" mechanism
- But this is not taking advantage of a key feature of the extended patterns:
  - Making errors in the don't care bits doesn't matter

Robustness of recall



- The value taken by the K hidden neurons during recall doesn't really matter
  - Even if it doesn't match what we actually tried to store
- Can we take advantage of this somehow?

Robustness of recall



- Also, we can have multiple extended patterns with the same pattern over visible bits
  - Can we exploit this somehow?

Taking advantage of don't care bits

- Simple random setting of don't care bits, and using the usual training and recall strategies for Hopfield nets should work
- However, it doesn't sufficiently exploit the redundancy of the don't care bits
  - Possible to set the don't care bits such that the overall pattern (and hence the "visible" bits portion of the pattern) is more memorable
  - Also, may have multiple don't-care patterns for a target pattern
    - Multiple valleys, in which the visible bits remain the same, but don't care bits vary
- To exploit it properly, it helps to view the Hopfield net differently: as a probabilistic machine

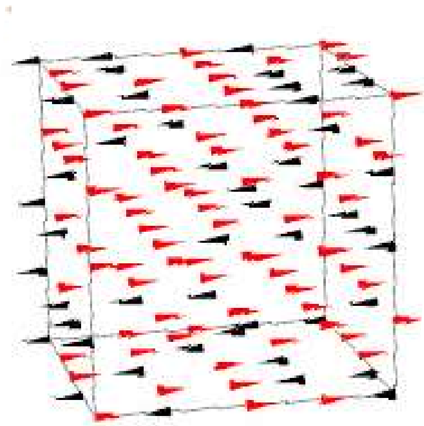
A probabilistic interpretation of Hopfield Nets

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

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

The Boltzmann Distribution

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

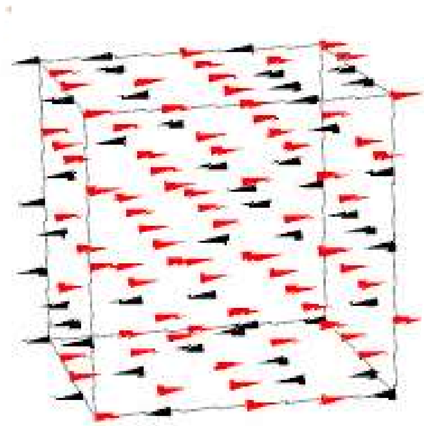


$$C = \frac{1}{\sum_{\mathbf{y}} exp\left(\frac{-E(\mathbf{y})}{kT}\right)}$$

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

Continuing the Boltzmann analogy

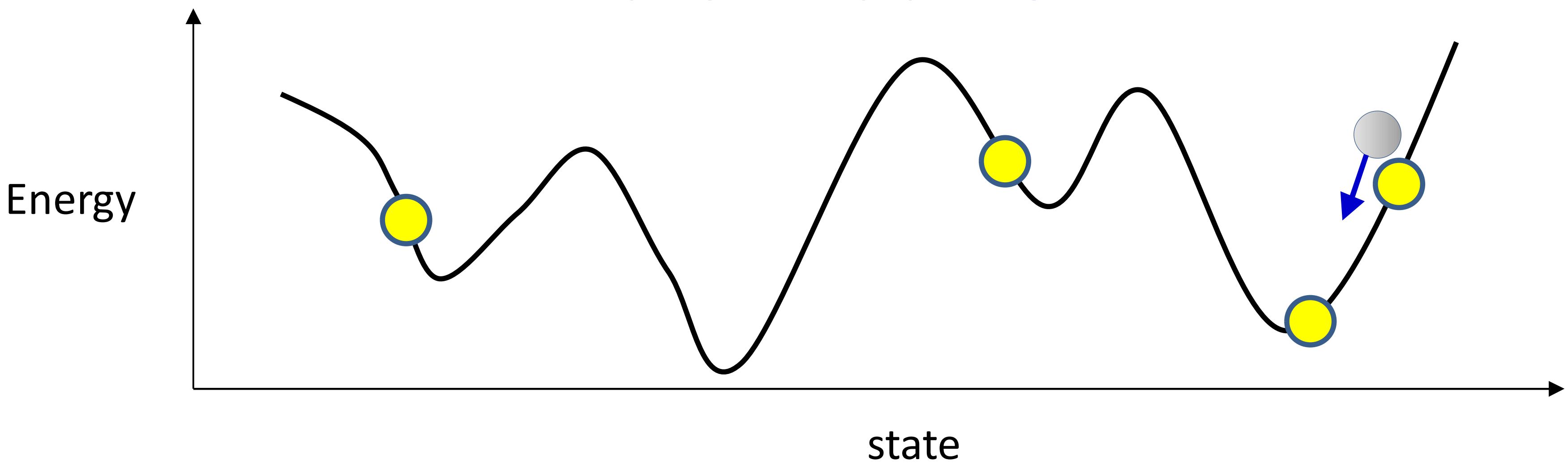
$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y} \qquad P(\mathbf{y}) = Cexp\left(\frac{-E(\mathbf{y})}{kT}\right)$$



$$C = \frac{1}{\sum_{\mathbf{y}} exp\left(\frac{-E(\mathbf{y})}{kT}\right)}$$

- 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 the Boltzmann distribution



- Selecting a next state is analogous to drawing a sample from the Boltzmann distribution at $T = 1$, in a universe where $k = 1$
  - Energy landscape of a spin-glass model: Exploration and characterization, Zhou and Wang, Phys. Review E 79, 2009

Hopfield nets: 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:

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

More importance to more frequently presented memories

More importance to more attractive spurious memories

Hopfield nets: 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:

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

More importance to more frequently presented memories

More importance to more attractive spurious memories

THIS LOOKS LIKE AN EXPECTATION!

Hopfield nets: Optimizing W

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

- Update rule

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

$$\mathbf{W} = \mathbf{W} + \eta\left(E_{\mathbf{y}\sim\mathbf{Y}_P}\mathbf{y}\mathbf{y}^T - E_{\mathbf{y}\sim Y}\mathbf{y}\mathbf{y}^T\right)$$

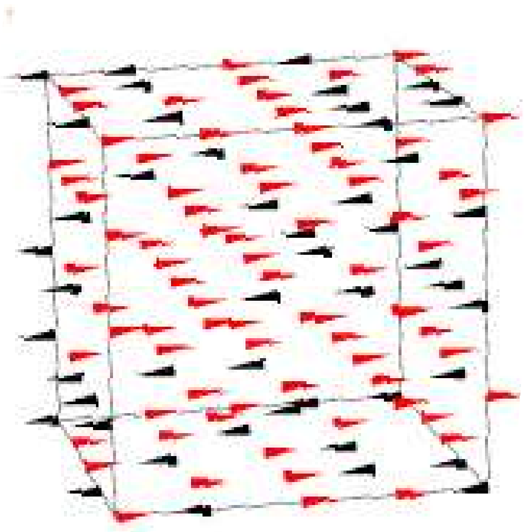
Natural distribution for variables: The Boltzmann Distribution

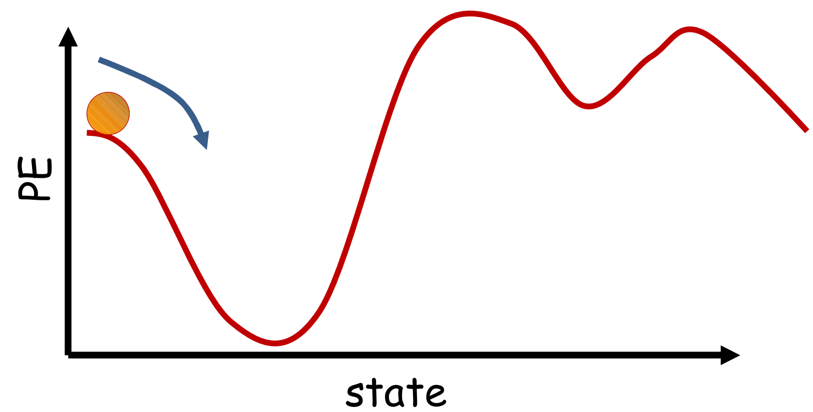
From Analogy to Model

- The behavior of the Hopfield net is analogous to annealed dynamics of a spin glass characterized by a Boltzmann distribution

- So, let's explicitly model the Hopfield net as a distribution..

Revisiting Thermodynamic Phenomena





- Is the system actually in a specific state at any time?
- No – the state is actually continuously changing
  - Based on the temperature of the system
    - At higher temperatures, state changes more rapidly
- What is actually being characterized is the *probability* of the state
  - And the *expected* value of the state

The Helmholtz Free Energy of a System

- A thermodynamic system at temperature T can exist in one of many states
  - Potentially infinite states
  - At any time, the probability of finding the system in state s at temperature T is $P_T(s)$
- At each state s it has a potential energy E_s
- The *internal energy* of the system, representing its capacity to do work, is the average:

$$U_T = \sum_s P_T(s)\, E_s$$

The Helmholtz Free Energy of a System

- The capacity to do work is counteracted by the internal disorder of the system, i.e. its entropy

$$H_T = -\sum_s P_T(s) \log P_T(s)$$

- The *Helmholtz* free energy of the system combines the two terms

$$F_T = U_T + kTH_T$$

$$= \sum_s P_T(s)\, E_s - kT \sum_s P_T(s) \log P_T(s)$$

The Helmholtz Free Energy of a System

$$F_T = \sum_s P_T(s)\, E_s - kT \sum_s P_T(s) \log P_T(s)$$

- A system held at a specific temperature *anneals* by varying the rate at which it visits the various states, to reduce the free energy in the system, until a minimum free-energy state is achieved

- The probability distribution of the states at steady state is known as the *Boltzmann distribution*

The Helmholtz Free Energy of a System

$$F_T = \sum_s P_T(s)\, E_s - kT \sum_s P_T(s) \log P_T(s)$$

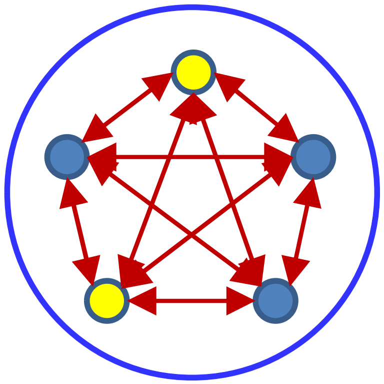
- Minimizing this w.r.t $P_T(s)$, we get

$$P_T(s) = \frac{1}{Z} exp\left(\frac{-E_s}{kT}\right)$$

  – Also known as the *Gibbs* distribution
  – Z is a normalizing constant
  – Note the dependence on T
  – A T = 0, the system will always remain at the lowest-energy configuration with prob = 1.

The Energy of the Network

Visible Neurons



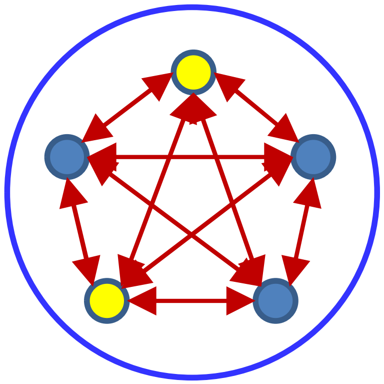
$$E(S) = -\sum_{i<j} w_{ij} s_i s_j - b_i s_i$$

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

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

The Hopfield net is a distribution

Visible Neurons



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

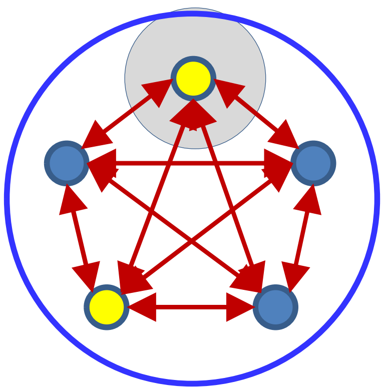
$$P(S) = \frac{exp(-E(S))}{\sum_{S'} exp(-E(S'))}$$

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

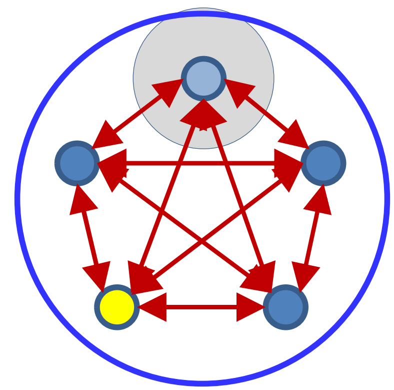
- The probability that (at equilibrium) the network will be in any state is $P(S)$
  - It is a *generative* model: generates states according to $P(S)$

The field at a single node

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



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

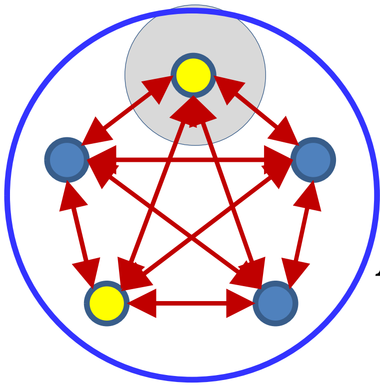


$$logP(S) - logP(S') = logP\left(s_i = 1 \middle| s_{j\neq i}\right) - logP\left(s_i = -1 \middle| s_{j\neq i}\right)$$

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

The field at a single node

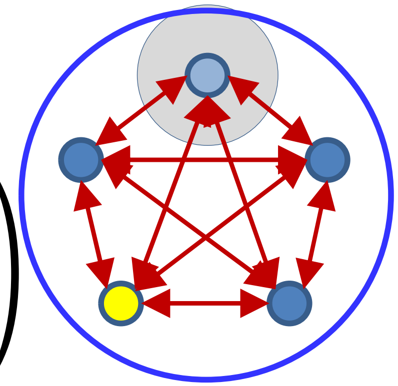
- Let S and S' be the states with the ith bit in the $+1$ and -1 states



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

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

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



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

The field at a single node

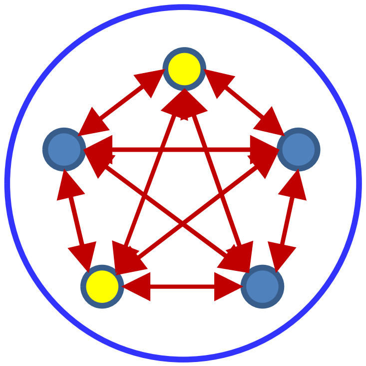
$$log\left(\frac{P(s_i = 1|s_{j\neq i})}{1 - P(s_i = 1|s_{j\neq i})}\right) = \sum_{j\neq i} w_j s_j + b_i$$

- Giving us

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

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

Redefining the network

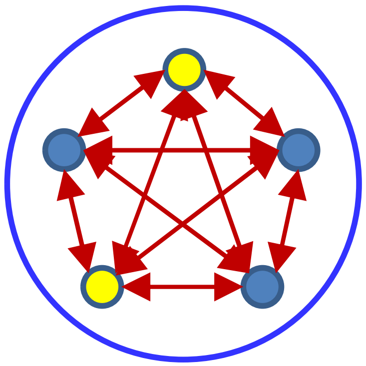


$$z_i = \sum_j w_{ji} s_j + b_i$$

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

- First try: Redefine a regular Hopfield net as a stochastic system
- Each neuron is *now a stochastic unit* with a binary state s_i, which can take value 0 or 1 with a probability that depends on the local field
  - Note the slight change from Hopfield nets
  - Not actually necessary; only a matter of convenience

The Hopfield net is a distribution

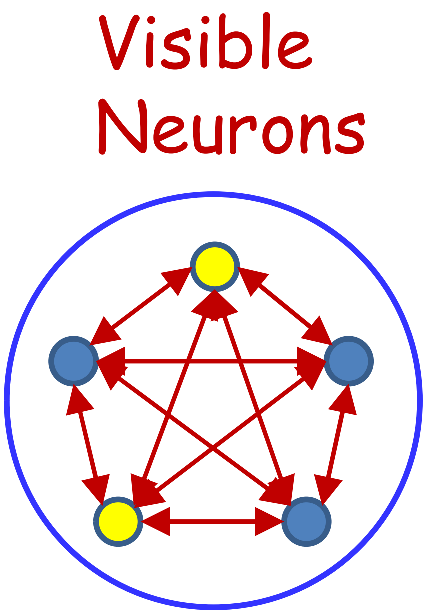


$$z_i = \sum_j w_{ji} s_j + b_i$$

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

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

Running the network



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

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

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

Evolution of a stochastic Hopfield net

1. Initialize network with initial pattern

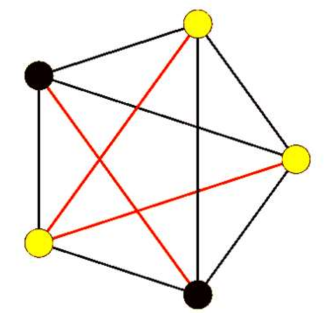
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

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

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

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

Assuming T = 1



Evolution of a stochastic Hopfield net

1. Initialize network with initial pattern

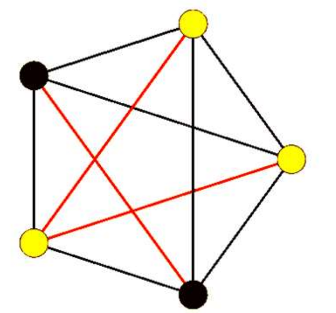
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

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

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

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

Assuming T = 1



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

Evolution of a stochastic Hopfield net

1. Initialize network with initial pattern

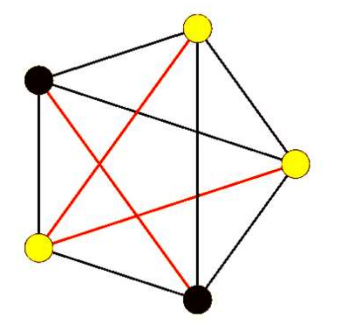
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

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

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

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

Assuming T = 1



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

Evolution of a stochastic Hopfield net

1. Initialize network with initial pattern

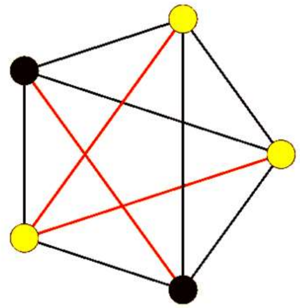
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

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

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

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

Assuming T = 1



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

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

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

Evolution of the stochastic network

1. Initialize network with initial pattern

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

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

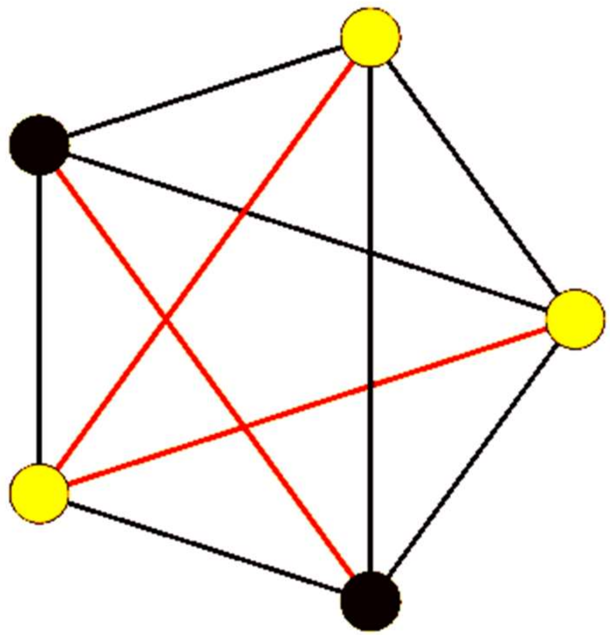
Noisy pattern completion: Initialize the entire network and let the entire network evolve

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

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

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

Including a "Temperature" term



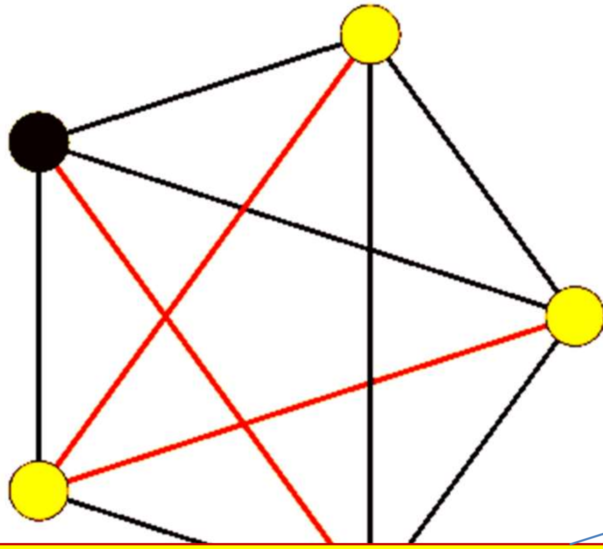
$$z_i = \frac{1}{T}\sum_{j\neq i} w_{ij} y_j$$

$$P(y_i = 1) = \sigma(z_i)$$

$$P(y_i = 0) = 1 - \sigma(z_i)$$

- Including a temperature term in computing the local field
  - This is much more in accord with Thermodynamic models
- At $T = \infty$ the energy "surface" will be flat. At $T = 1$ the surface will be the usual energy surface
  - This can be used to improve the likelihood of finding good (or optimal) minimum-energy states

Recap: Stochastic Hopfield Nets



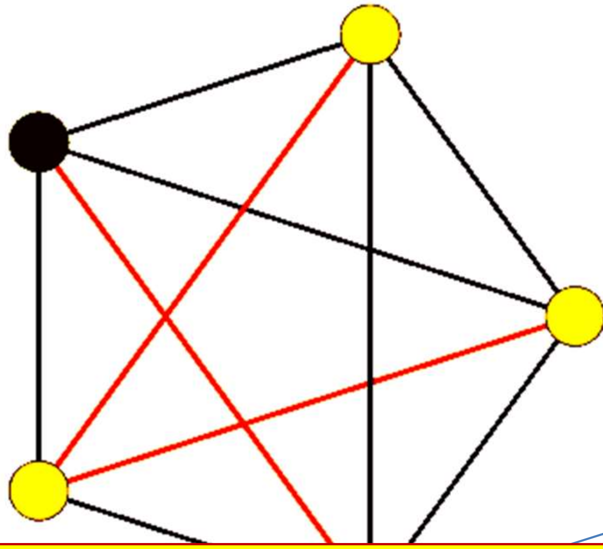
$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ji} y_j$$

$$P(y_i = 1) = \sigma(z_i)$$

The field quantifies the energy difference obtained by flipping the current unit

- Including a temperature term in computing the local field
  - This is much more in accord with Thermodynamic models
- At $T = \infty$ the energy "surface" will be flat. At $T = 1$ the surface will be the usual energy surface
  - This can be used to improve the likelihood of finding good (or optimal) minimum-energy states

Recap: Stochastic Hopfield Nets



$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ji} y_j$$

$$P(y_i = 1) = \sigma(z_i)$$

The field quantifies the energy difference obtained by flipping the current unit

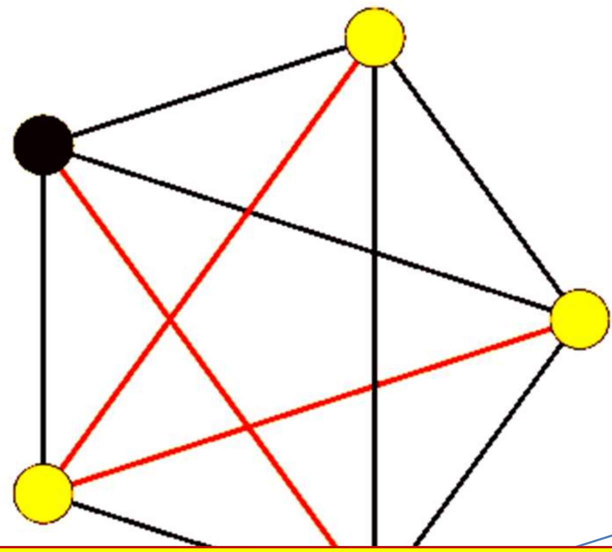
- Including a temperature term in computing the local field

If the difference is not large, the probability of flipping approaches 0.5

  – This is much more in accord with Thermodynamic models

- At $T = \infty$ the energy "surface" will be flat. At $T = 1$ the surface will be the usual energy surface
  – This can be used to improve the likelihood of finding good (or optimal) minimum-energy states

Recap: Stochastic Hopfield Nets



$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ji} y_j$$

$$P(y_i = 1) = \sigma(z_i)$$

The field quantifies the energy difference obtained by flipping the current unit

- Including a temperature term in computing the local field

If the difference is not large, the probability of flipping approaches 0.5

  – This is much more in accord with Thermodynamic models

T is a "temperature" parameter: increasing it moves the probability of the bits towards 0.5
At T=1.0 we get the traditional definition of field and energy
At T = 0, we get deterministic Hopfield behavior

  – This can be used to improve the likelihood of finding good (or optimal) minimum-energy states

Annealing

1. Initialize network with initial pattern

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

2. For $T = T_0$ *down to* T_{min}
   i. For iter $1..L$
      a) For $0 \le i \le N-1$

$$P = \sigma\left(\frac{1}{T}\sum_{j \ne i} w_{ji} y_j\right)$$

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

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

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

Evolution of a stochastic Hopfield net

1. Initialize network with initial pattern

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

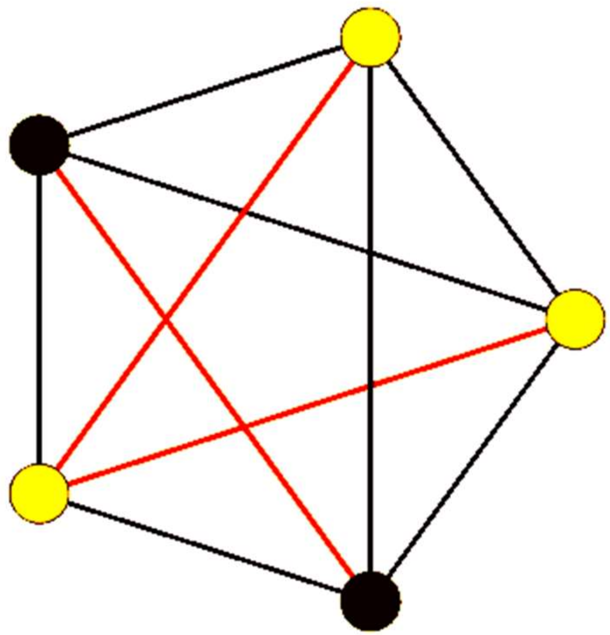
2. For $T = T_0$ *down to* T_{min}
   i. For iter $1..L$
      a) For $0 \le i \le N-1$

      $$P = \sigma\left(\frac{1}{T}\sum_{j \ne i} w_{ji} y_j\right)$$

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

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

Recap: Stochastic Hopfield Nets

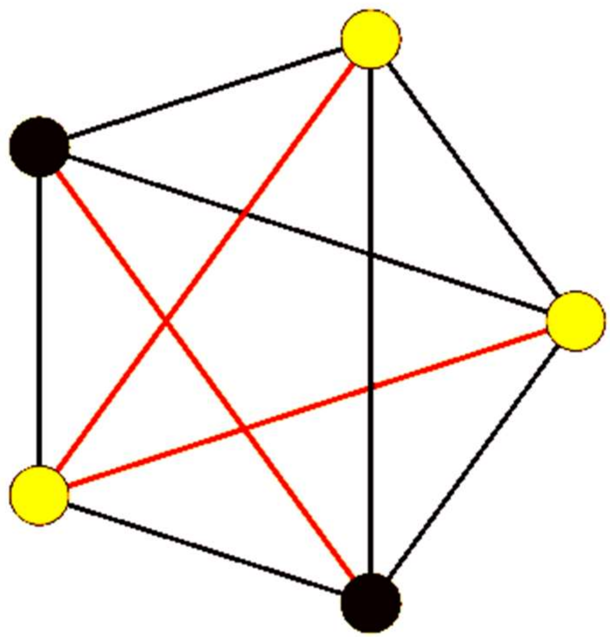


$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ji} y_j$$

$$P\left(y_i = 1 | y_{j \neq i}\right) = \sigma(z_i)$$

- The probability of each neuron is given by a *conditional* distribution
- What is the overall probability of *the entire set of neurons* taking any configuration $\mathbf{y}$

The overall probability



$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ji} y_j$$

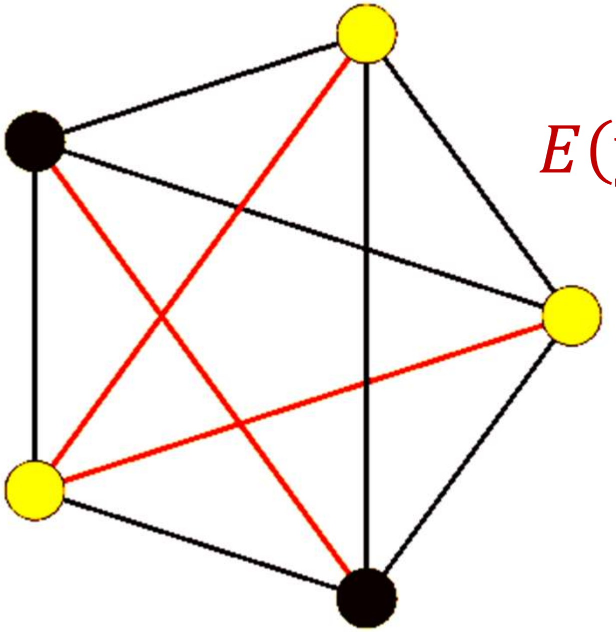
$$P\left(y_i = 1 | y_{j \neq i}\right) = \sigma(z_i)$$

- The probability of any state $\mathbf{y}$ can be shown to be given by the *Boltzmann distribution*

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

  – Minimizing energy maximizes log likelihood

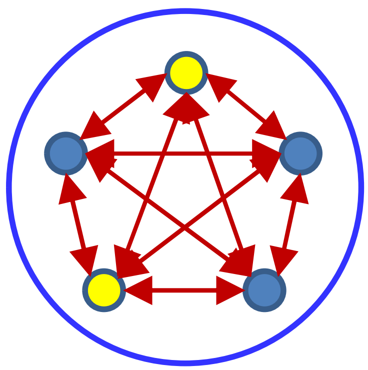
The overall probability



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

- Stop when the running average of the log probability of patterns stops increasing
  - I.e. when the (running average) of the energy of the patterns stops decreasing

The Hopfield net is a distribution



$$z_i = \frac{1}{T}\sum_j w_{ji}s_j$$

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

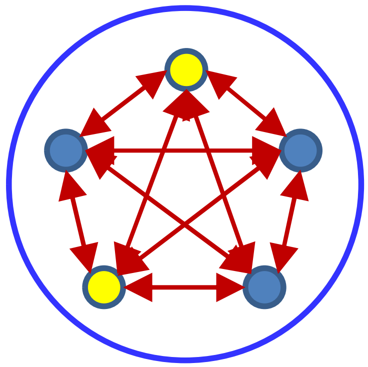
- The Hopfield net is a probability distribution over binary sequences
  - The Boltzmann distribution

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

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

  - The parameter of the distribution is the weights matrix $\mathbf{W}$
- The *conditional* distribution of individual bits in the sequence is a logistic
- We will call this a Boltzmann machine

The Boltzmann Machine



$$z_i = \frac{1}{T}\sum_j w_{ji} s_j$$

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

- The entire model can be viewed as a *generative model*
- Has a probability of producing any binary vector $\mathbf{y}$:

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

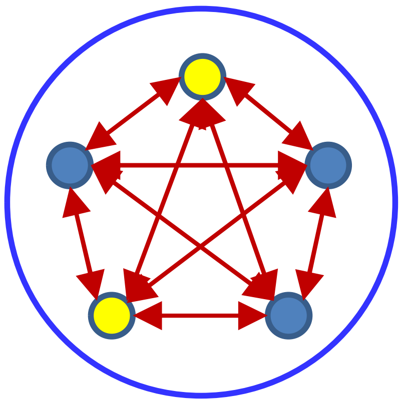
$$P(\mathbf{y}) = Cexp\left(-\frac{E(\mathbf{y})}{T}\right)$$

Training the model

- How does the probabilistic view affect how we train the model?

- Not much…

Training the network



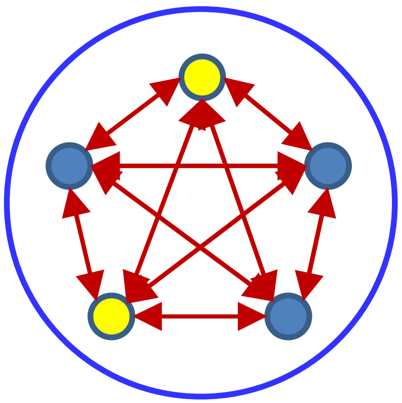
$$E(S) = -\sum_{i<j} w_{ij} s_i s_j$$

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

$$P(S) = \frac{exp\left(\sum_{i<j} w_{ij} s_i s_j\right)}{\sum_{S'} exp\left(\sum_{i<j} w_{ij} s'_i s'_j\right)}$$

- Training a Hopfield net: Must learn weights to "remember" target states and "dislike" other states
  - **"State" == binary pattern of all the neurons**
- Training Boltzmann machine: Must learn weights to assign a desired probability distribution to states
  - (vectors $\mathbf{y}$, which we will now calls S because I'm too lazy to normalize the notation)
  - This should assign more probability to patterns we "like" (or try to memorize) and less to other patterns

Training the network



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

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

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

- Must train the network to assign a desired probability distribution to states
- Given a set of "training" inputs $S_1, \dots, S_N$
  - Assign higher probability to patterns seen more frequently
  - Assign lower probability to patterns that are not seen at all
- Alternately viewed: *maximize likelihood of stored states*

Maximum Likelihood Training

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

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

Average log likelihood of training vectors (to be maximized)

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

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

Maximum Likelihood Training

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

$$\frac{d\mathcal{L}}{dw_{ij}} = \frac{1}{N}\sum_{S} s_i s_j - ???$$

- We will use gradient ascent, but we run into a problem..
- The first term is just the average $s_i s_j$ over all training patterns
- But the second term is summed over *all* states
  - Of which there can be an exponential number!

The second term

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

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

$$\frac{d\log\left(\sum_{S'} exp\left(\sum_{i<j} w_{ij}s_i's_j'\right)\right)}{dw_{ij}} = \sum_{S'} \frac{exp\left(\sum_{i<j} w_{ij}s_i's_j'\right)}{\sum_{S''} exp\left(\sum_{i<j} w_{ij}s_i^{''}s_j^{''}\right)} s_i's_j'$$

The second term

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

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

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

$P(S')$

The second term

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

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

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

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

The second term

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

- The second term is simply the *expected value* of $s_i s_j$, over all possible values of the state
- We cannot compute it exhaustively, but we can compute it by sampling!

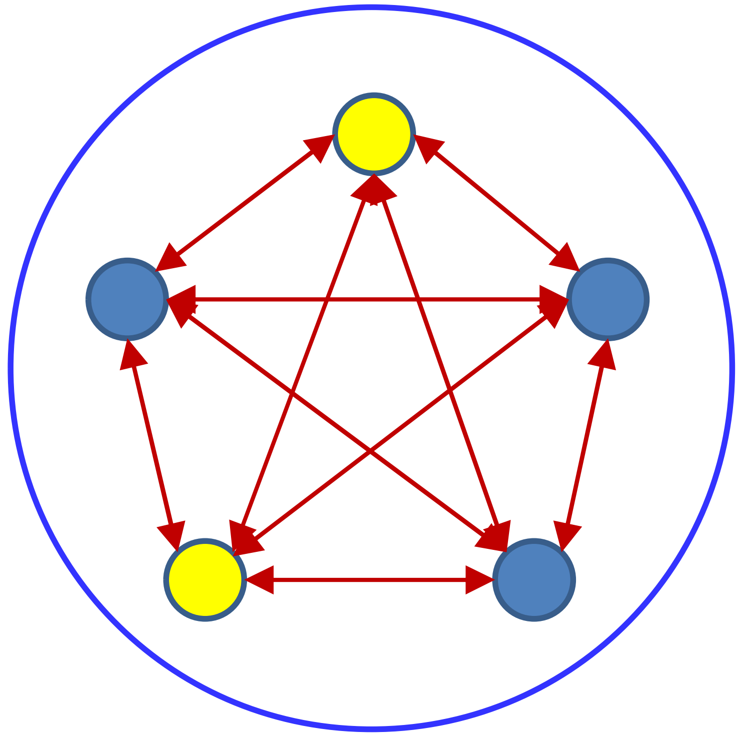
Estimating the second term

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

$$\sum_{S'} P(S')s_i's_j' \approx \frac{1}{M} \sum_{S' \in \mathbf{S}_{samples}} s_i's_j'$$

- The expectation can be estimated as the average of samples drawn from the distribution
- Question: How do we draw samples from the Boltzmann distribution?
  - How do we draw samples from the network?

The simulation solution



- Initialize the network randomly and let it "evolve"
  - By probabilistically selecting state values according to our model
- After many many epochs, take a snapshot of the state
- Repeat this many many times
- Let the collection of states be

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

The simulation solution for the second term

$$\frac{d\log\left(\sum_{S'} exp\left(\sum_{i<j} w_{ij} s'_i s'_j\right)\right)}{dw_{ij}} = \sum_{S'} P(S') s'_i s'_j$$

$$\sum_{S'} P(S') s'_i s'_j \approx \frac{1}{M} \sum_{S' \in \mathbf{S}_{simul}} s'_i s'_j$$

- The second term in the derivative is computed as the average of sampled states when the network is running "freely"

Maximum Likelihood Training

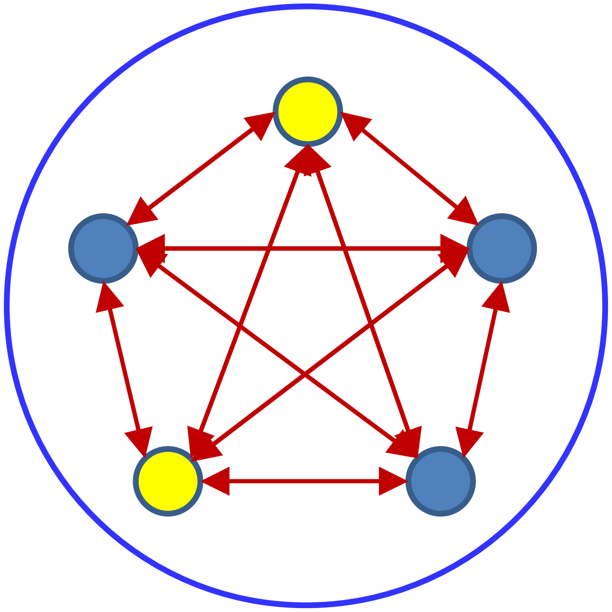
Sampled estimate

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

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

- The overall gradient ascent rule

Overall Training

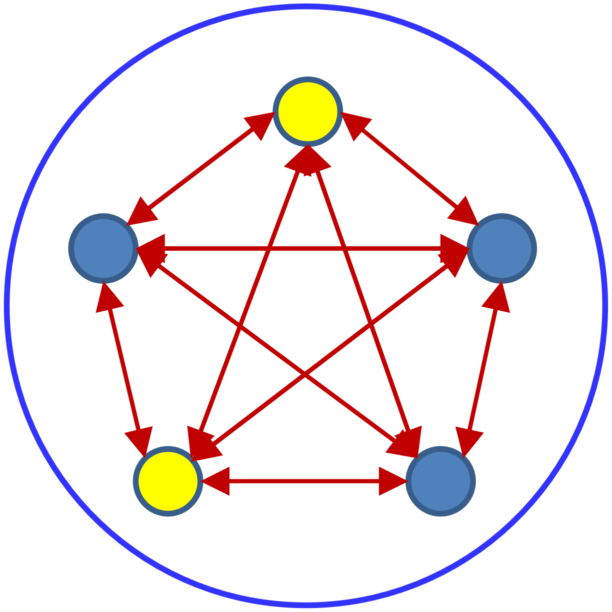


$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{N}\sum_{S} s_i s_j - \frac{1}{M}\sum_{S' \in \mathbf{S}_{simul}} s'_i s'_j$$

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

- Initialize weights
- Let the network run to obtain simulated state samples
- Compute gradient and update weights
- Iterate

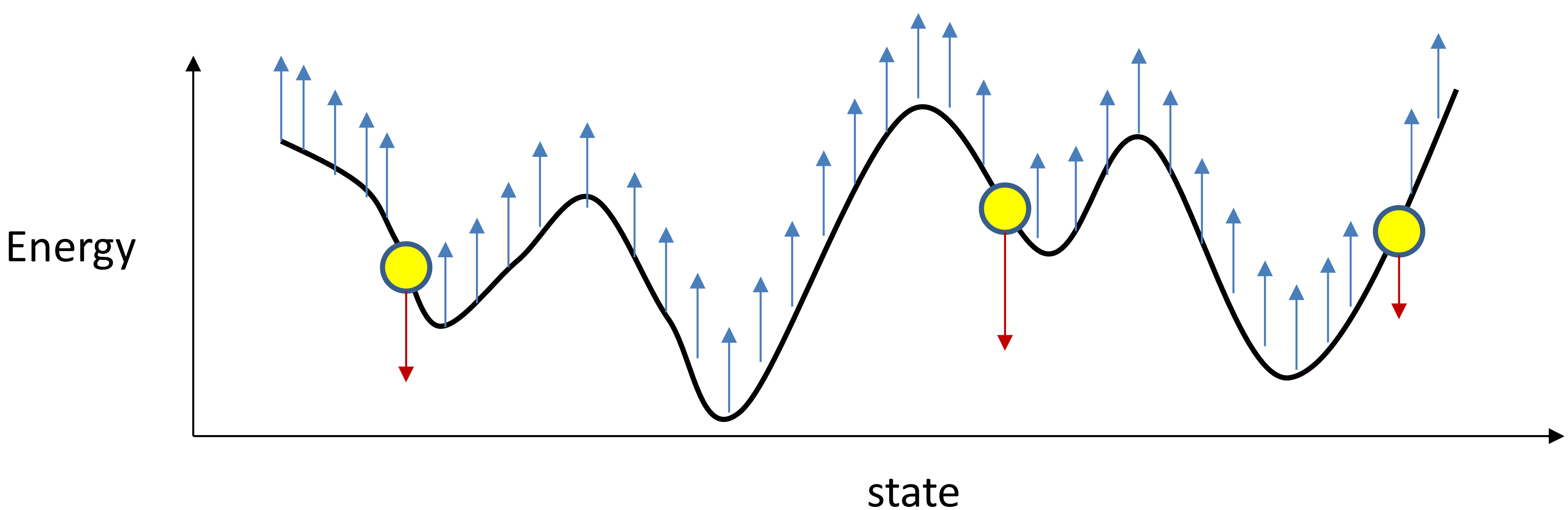
Overall Training



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

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

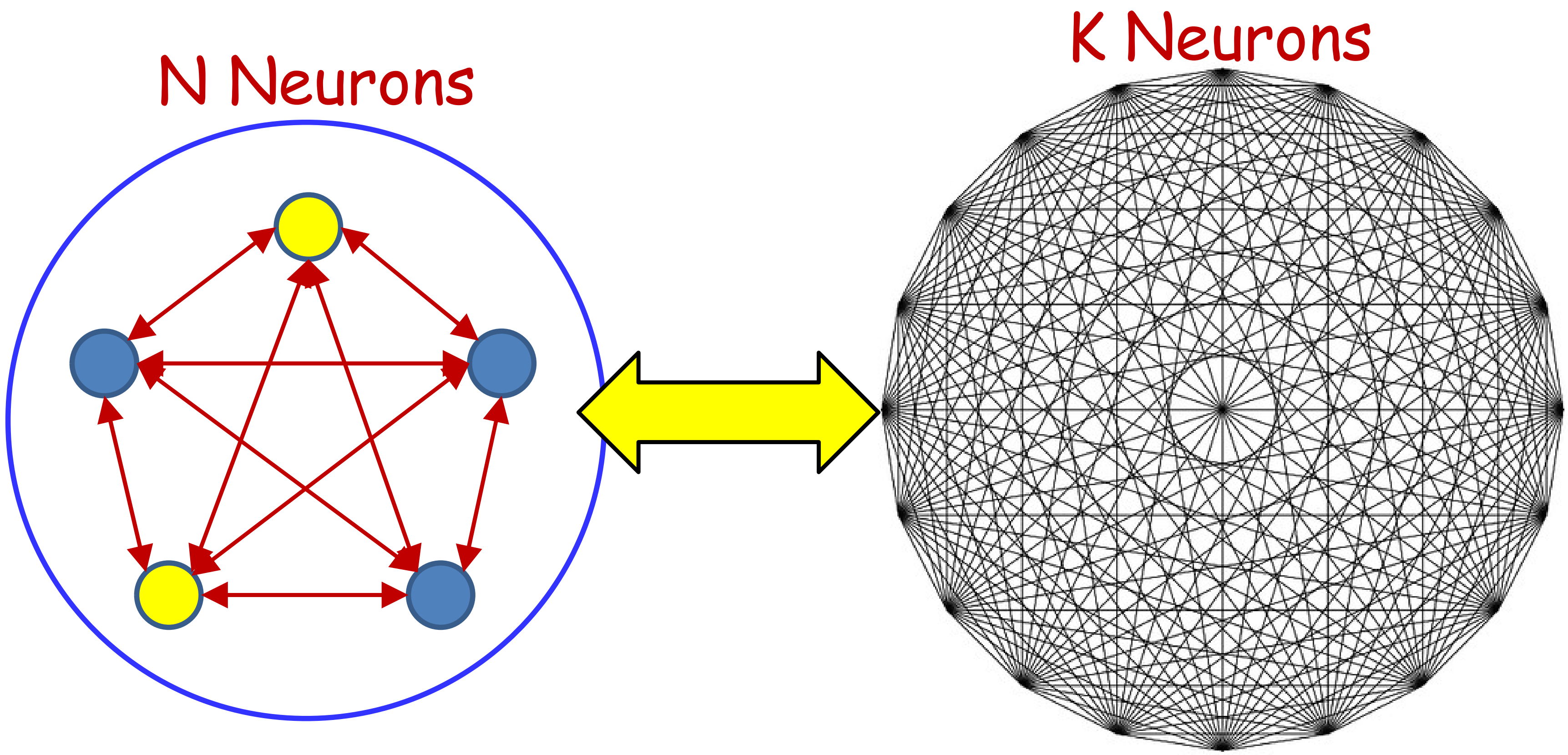
Note the similarity to the update rule for the Hopfield network



Adding Capacity to the Hopfield Network / Boltzmann Machine

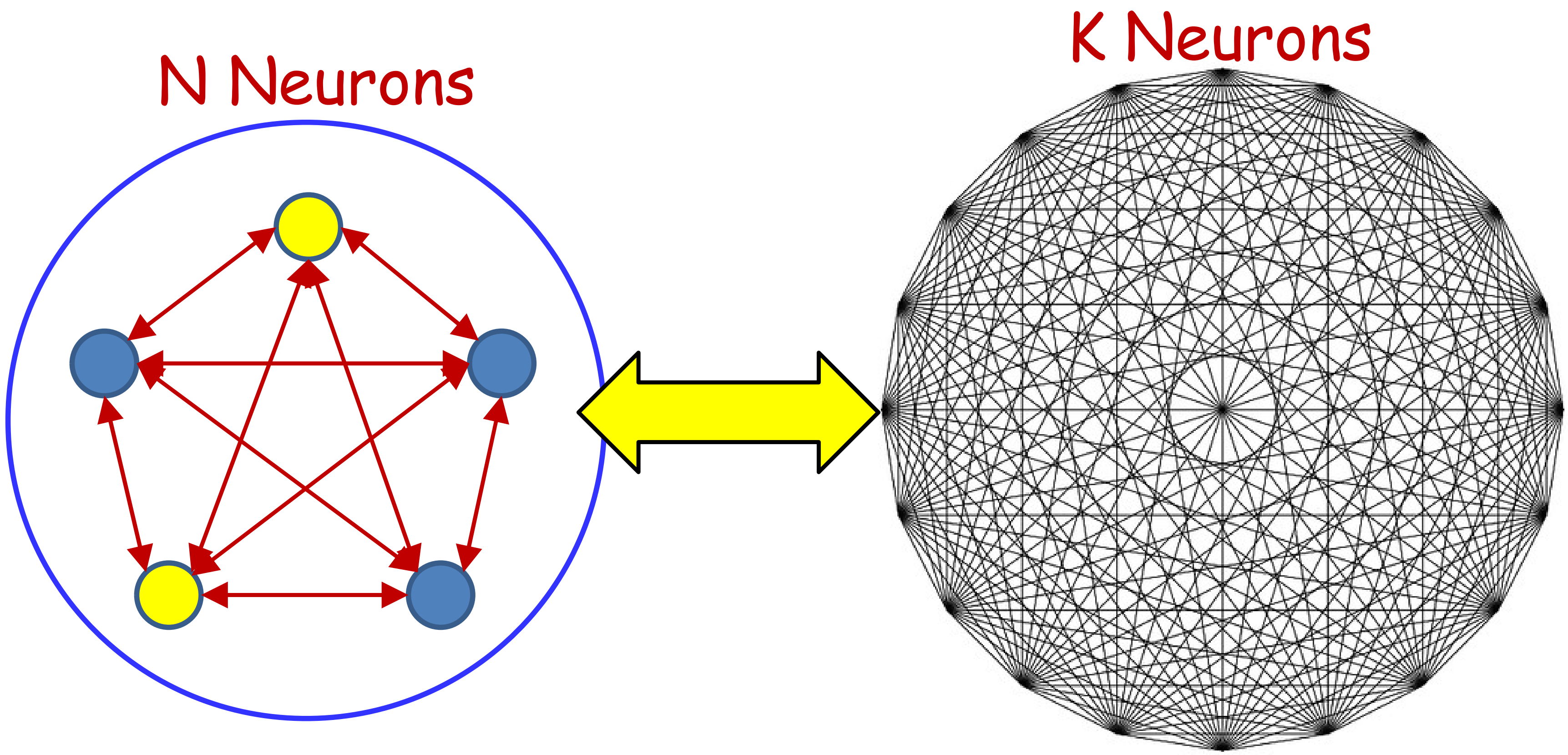
- The network can store up to N N-bit patterns
- How do we increase the capacity

Expanding the network



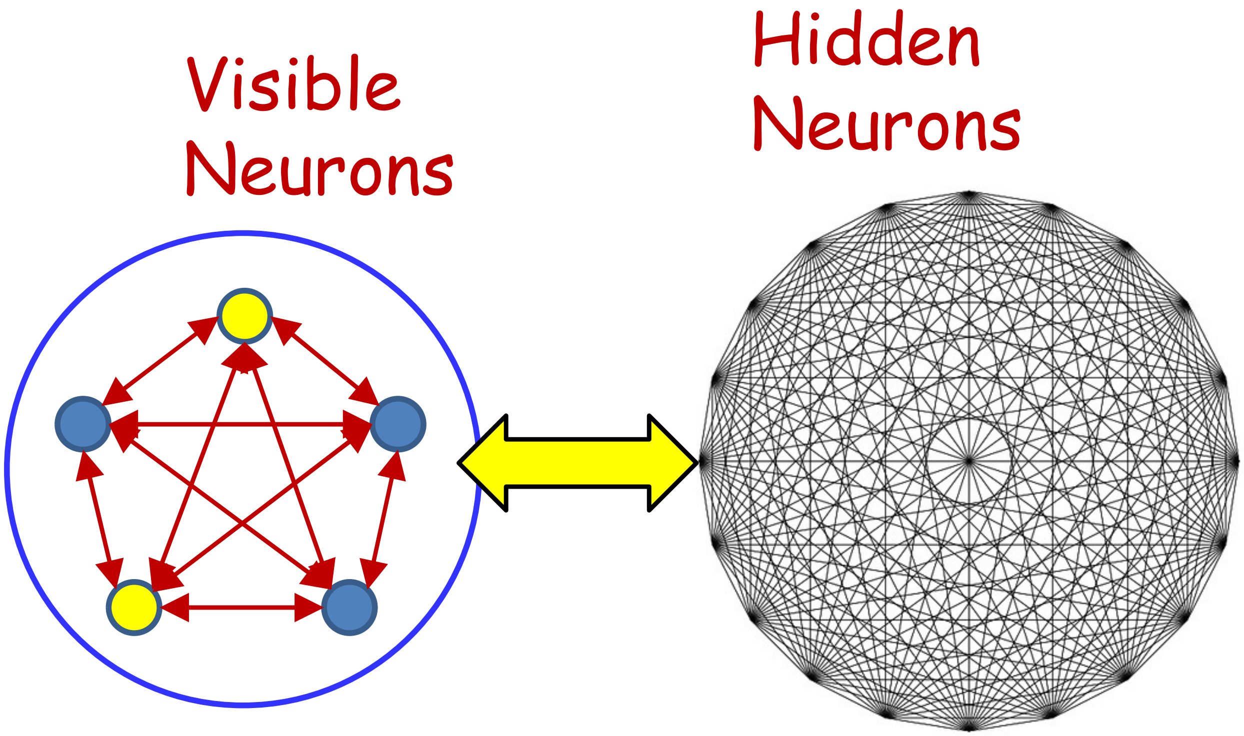
- Add a large number of neurons whose actual values you don't care about!

Expanded Network



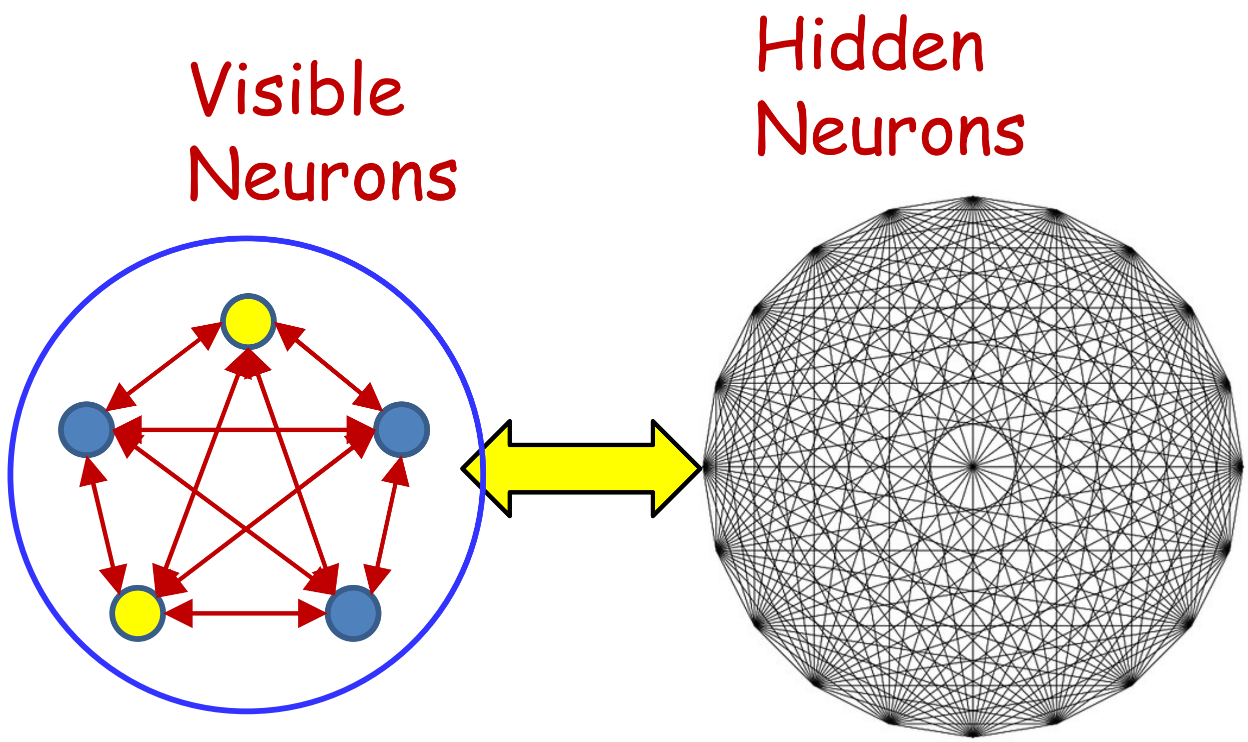
- New capacity: $\sim(N + K)$ patterns
  - Although we only care about the pattern of the first N neurons
  - We're interested in *N-bit* patterns

Terminology



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

Training the network



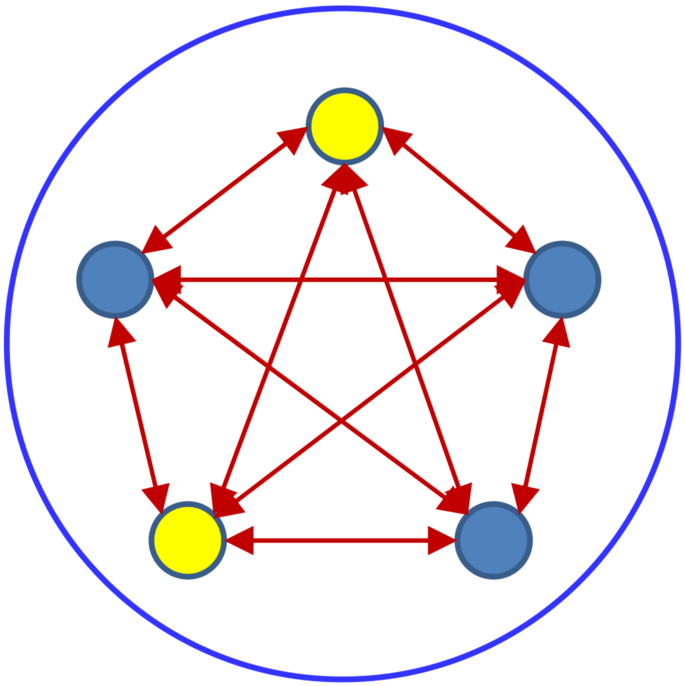
- For a given pattern of *visible* neurons, there are any number of *hidden* patterns (2^K)
- Which of these do we choose?
  - Ideally choose the one that results in the lowest energy
  - But that's an exponential search space!

The patterns

- In fact we could have *multiple* hidden patterns coupled with any visible pattern
  - These would be multiple stored patterns that all give the same visible output
  - How many do we permit

- Do we need to specify one or more particular hidden patterns?
  - How about *all* of them
  - What do I mean by this bizarre statement?

Boltzmann machine without hidden units

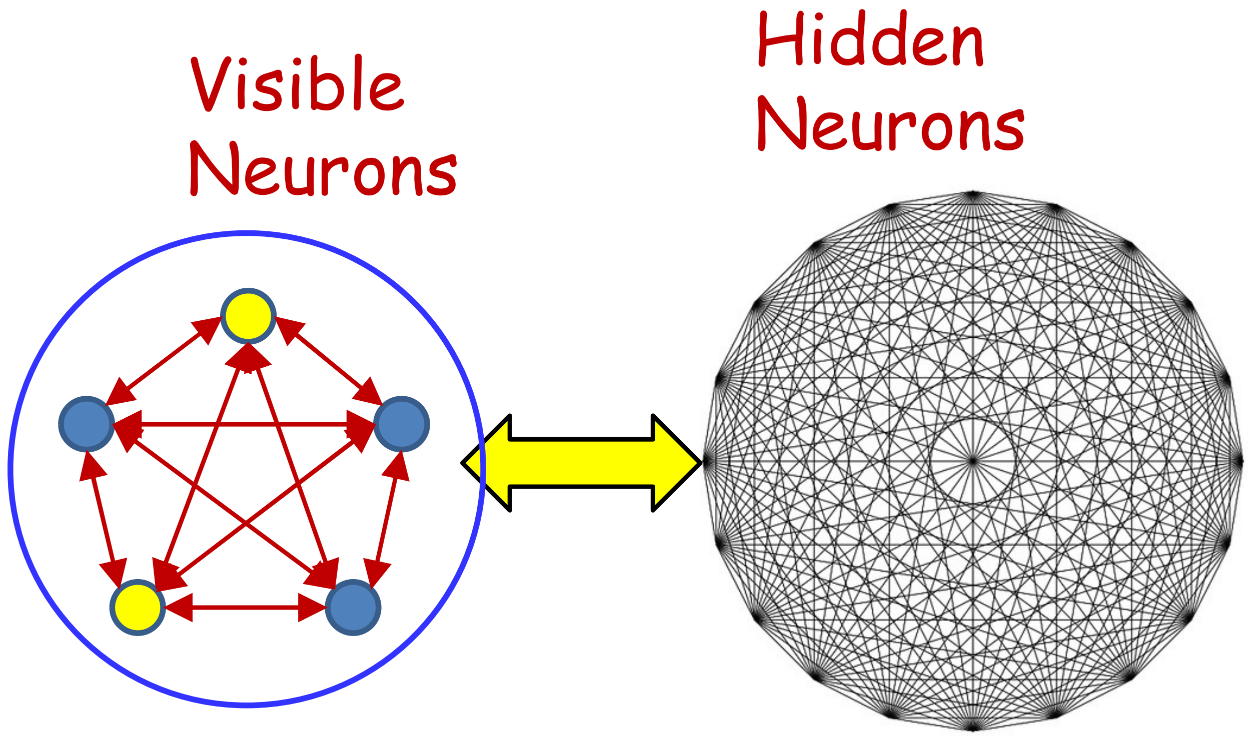


$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{N}\sum_{S} s_i s_j - \frac{1}{M}\sum_{S' \in \mathbf{S}_{simul}} s'_i s'_j$$

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

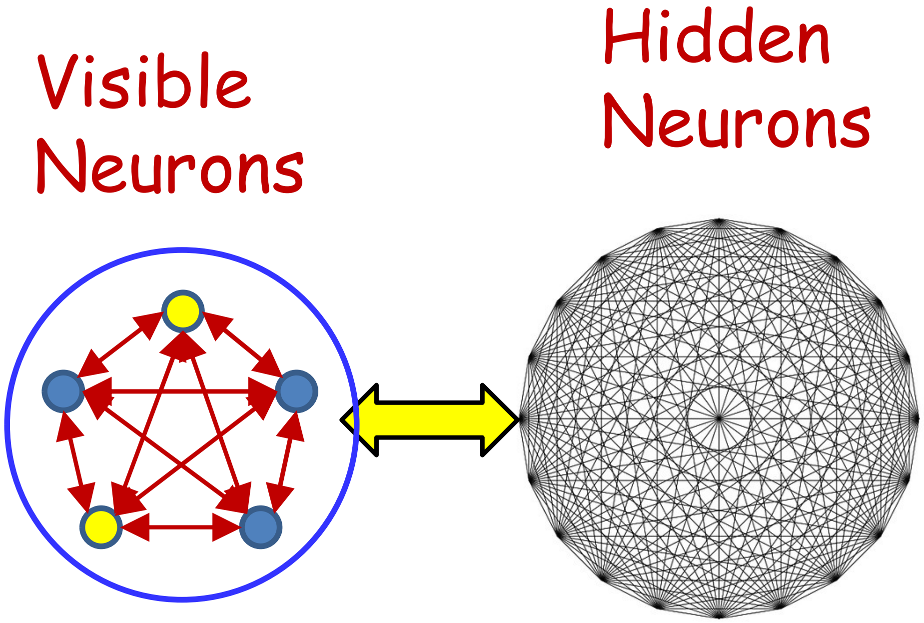
- This basic framework has no hidden units
- Extended to have hidden units

With hidden neurons



- Now, with hidden neurons the complete state pattern for even the *training* patterns is unknown
  – Since they are only defined over visible neurons

With hidden neurons



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

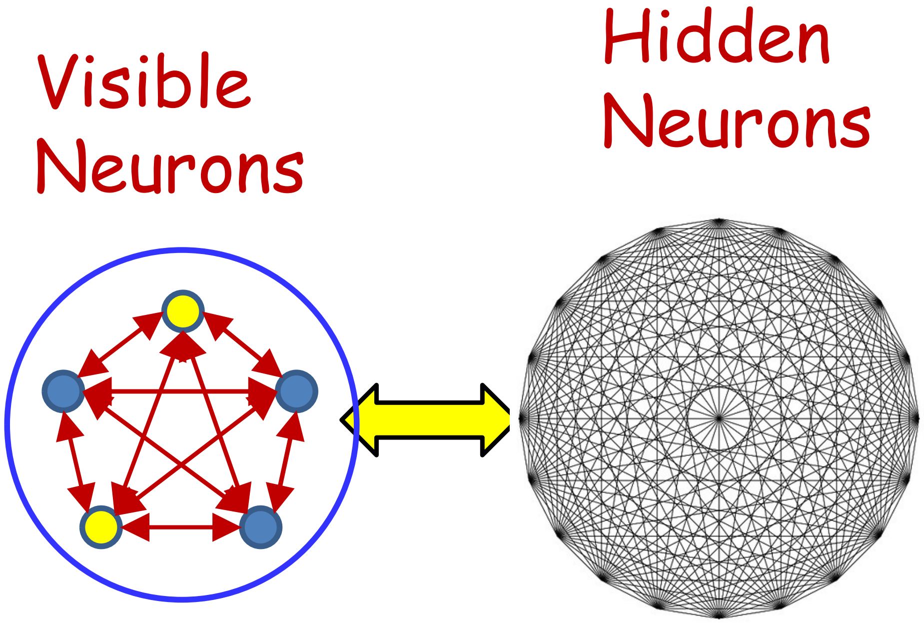
$$P(S) = P(V, H)$$

$$P(V) = \sum_{H} P(S)$$

- We are interested in the *marginal* probabilities over *visible* bits
  - We want to learn to represent the visible bits
  - The hidden bits are the "latent" representation learned by the network

- $S = (V, H)$
  - V = visible bits
  - H = hidden bits

With hidden neurons



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

$$P(S) = P(V,H)$$

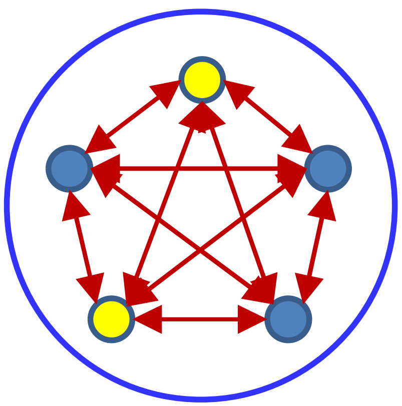
$$P(V) = \sum_{H} P(S)$$

- We are interested in the *marginal* probabilities over *visible* bits
  - We want to learn to represent the visible bits
  - The hidden bits are the "latent" representation learned by the network

- $S = (V,H)$
  - V = visible bits
  - H = hidden bits

Must train to maximize probability of desired patterns of *visible* bits

Training the network



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

$$P(S) = \frac{exp\left(\sum_{i<j} w_{ij} s_i s_j\right)}{\sum_{S'} exp\left(\sum_{i<j} w_{ij} s'_i s'_j\right)}$$

$$P(V) = \sum_{H} \frac{exp\left(\sum_{i<j} w_{ij} s_i s_j\right)}{\sum_{S'} exp\left(\sum_{i<j} w_{ij} s'_i s'_j\right)}$$

- Must train the network to assign a desired probability distribution to *visible* states
- Probability of visible state sums over all hidden states

Maximum Likelihood Training

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

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

Average log likelihood of training vectors (to be maximized)

$$= \frac{1}{N}\sum_{V\in\mathbf{V}} \log\left(\sum_{H} exp\left(\sum_{i<j} w_{ij} s_i s_j\right)\right) - \log\left(\sum_{S'} exp\left(\sum_{i<j} w_{ij} s'_i s'_j\right)\right)$$

- Maximize the average log likelihood of all visible bits of "training" vectors $\mathbf{V} = \{V_1, V_2, \dots, V_N\}$
  - The first term also has the same format as the second term
    - Log of a sum
  - Derivatives of the first term will have the same form as for the second term

Maximum Likelihood Training

$$\mathcal{L} = \frac{1}{N}\sum_{V\in\mathbf{V}} \log\left(\sum_{H} exp\left(\sum_{i<j} w_{ij}s_i s_j\right)\right) - \log\left(\sum_{S\prime} exp\left(\sum_{i<j} w_{ij}s_i' s_j'\right)\right)$$

$$\frac{d\mathcal{L}}{dw_{ij}} = \frac{1}{N}\sum_{V\in\mathbf{V}}\sum_{H} \frac{exp(\sum_{k<l} w_{kl}s_k s_l)}{\sum_{H\prime} exp(\sum_{k<l} w_{kl}s_k^{"}s_l^{"})} s_i s_j - \sum_{S\prime} \frac{exp(\sum_{k<l} w_{kl}s_k' s_l')}{\sum_{S"} exp(\sum_{k<l} w_{ij}s_k^{"}s_l^{"})} s_i' s_j'$$

$$\frac{d\mathcal{L}}{dw_{ij}} = \frac{1}{N}\sum_{V\in\mathbf{V}}\sum_{H} P(S|V)s_i s_j - \sum_{S\prime} P(S')s_i' s_j'$$

- We’ve derived this math earlier
- But now *both* terms require summing over an exponential number of states
  - The first term fixes visible bits, and sums over all configurations of hidden states for each visible configuration in our training set
  - But the second term is summed over *all* states

The simulation solution

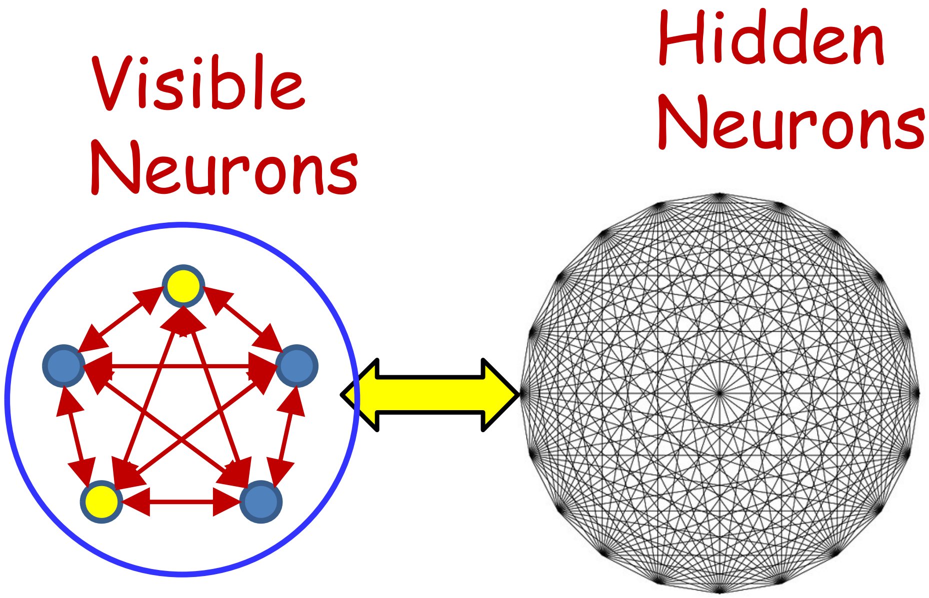
$$\frac{d\mathcal{L}}{dw_{ij}} = \frac{1}{N}\sum_{V\in\mathbf{V}}\sum_{H} P(S|V)s_i s_j - \sum_{S'} P(S')s'_i s'_j$$

$$\sum_{H} P(S|V)s_i s_j \approx \frac{1}{K}\sum_{H\in\mathbf{H}_{simul}} s_i s_j$$

$$\sum_{S'} P(S')s'_i s'_j \approx \frac{1}{M}\sum_{S'\in\mathbf{S}_{simul}} s'_i s'_j$$

- The first term is computed as the average sampled *hidden* state with the visible bits fixed
- The second term in the derivative is computed as the average of sampled states when the network is running “freely”

More simulations

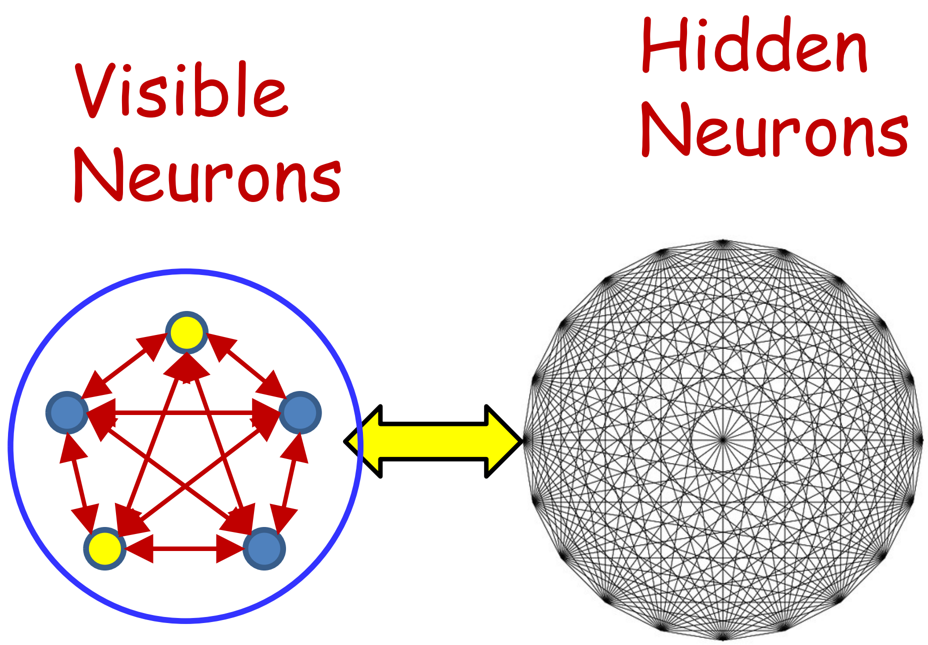


$$P(S) = \frac{exp(-E(S))}{\sum_{S'} exp(-E(S'))}$$

$$P(V) = \sum_{H} P(S)$$

- Maximizing the marginal probability of V requires summing over all values of H
  - An exponential state space
  - So we will use simulations again

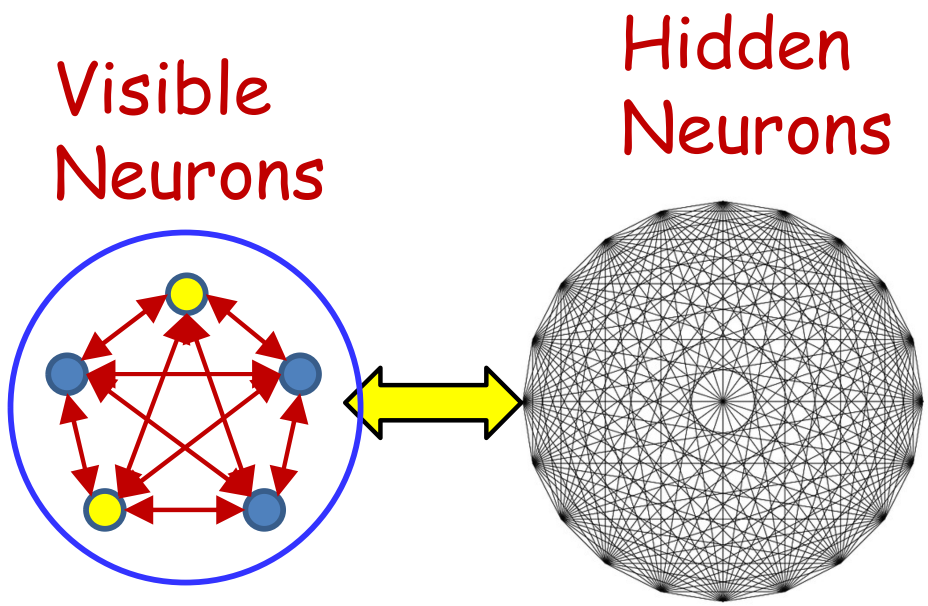
Step 1



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

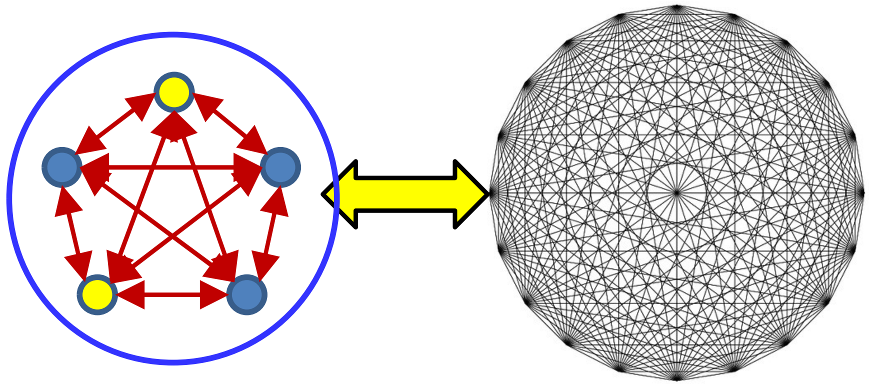
$$\mathbf{S} = \{S_{1,1}, S_{1,2}, \ldots, S_{1K}, S_{2,1}, \ldots, S_{N,K}\}$$

Step 2



- Now *unclamp* the visible units and let the entire network evolve several times to generate
  $\mathbf{S}_{simul} = \{S_{simul,1}, S_{simul,1=2}, \dots, S_{simul,M}\}$

Gradients

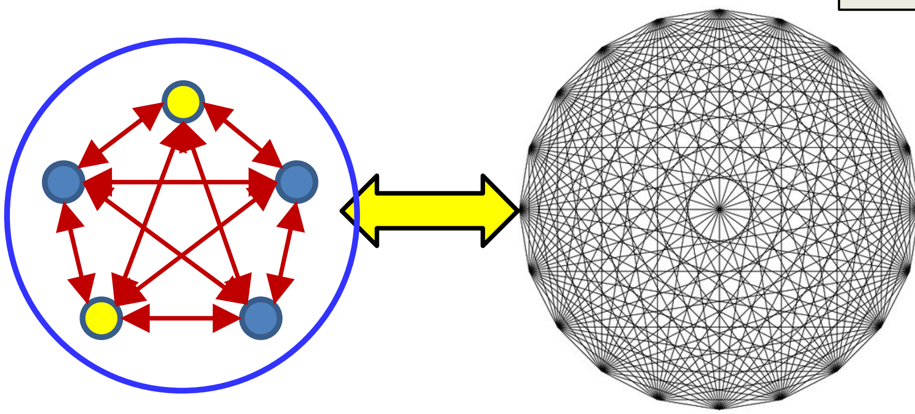


$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{NK}\sum_{\mathbf{S}} s_i s_j - \frac{1}{M}\sum_{S' \in \mathbf{S}_{simul}} s'_i s'_j$$

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

Overall Training

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



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

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

Boltzmann machines

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

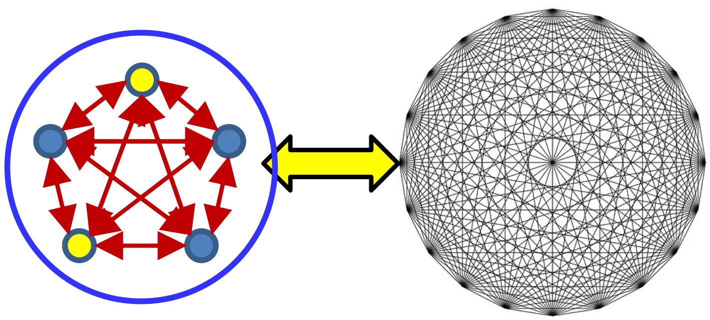
Boltzmann machines: Overall

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

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

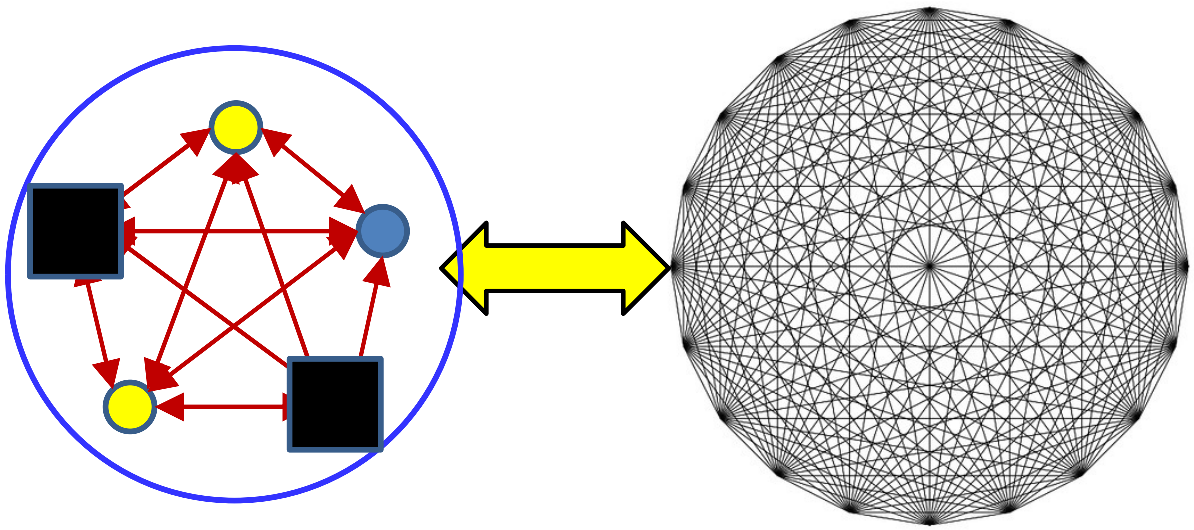
$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{NK} \sum_{\mathbf{S}} s_i s_j - \frac{1}{M} \sum_{S' \in \mathbf{S}_{simul}} s'_i s'_j$$

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



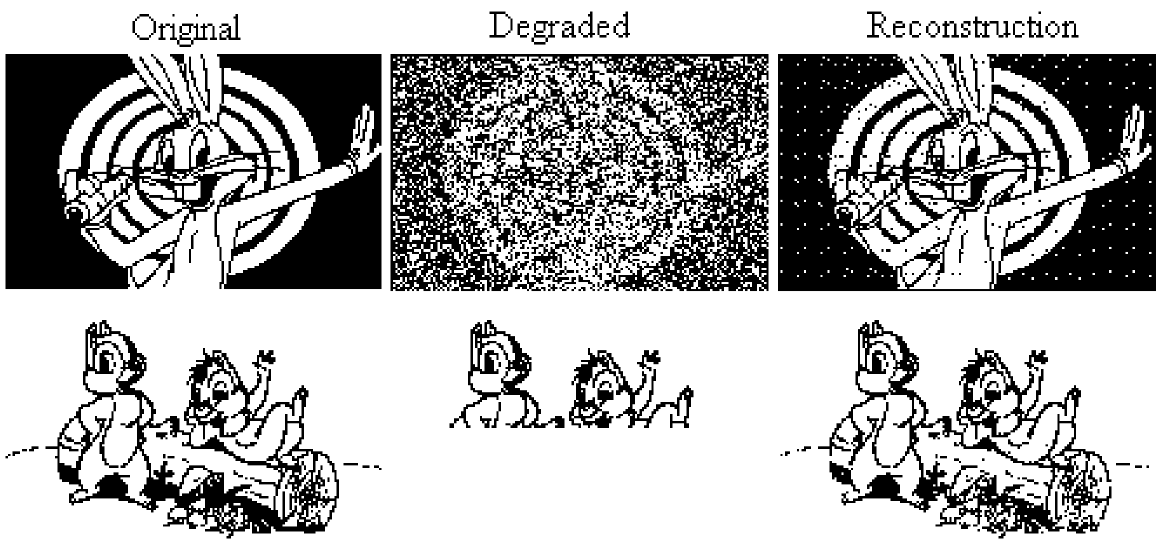
- **Training:** Given a set of training patterns
  - Which could be repeated to represent relative probabilities
- Initialize weights
- Run simulations to get clamped and unclamped training samples
- Compute gradient and update weights
- Iterate

Boltzmann machines: Overall



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

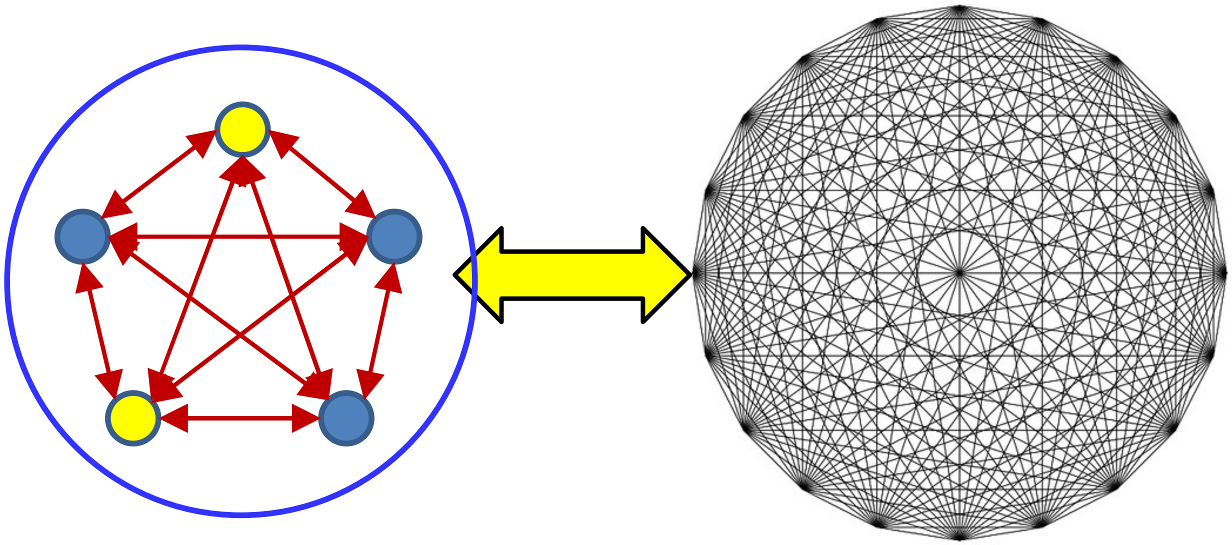
Applications



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

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

Boltzmann machines for classification

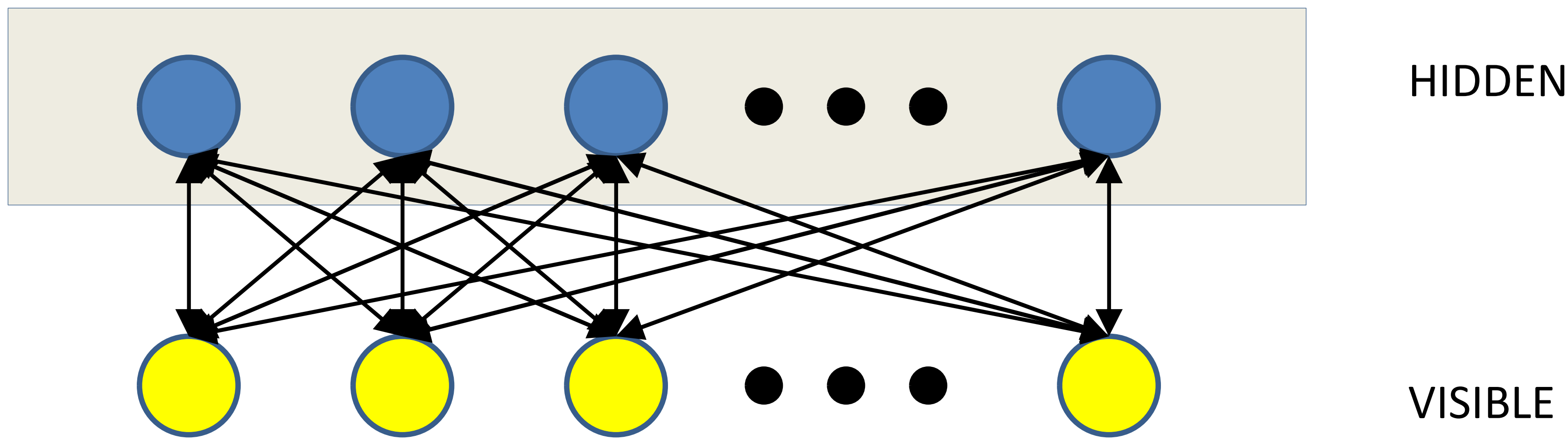


- Training patterns:
  - [f_1, f_2, f_3, …. , class]
  - Features can have binarized or continuous valued representations
  - Classes have "one hot" representation
- Classification:
  - Given features, anchor features, estimate a posteriori probability distribution over classes
    - Or choose most likely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

Boltzmann machines: Issues

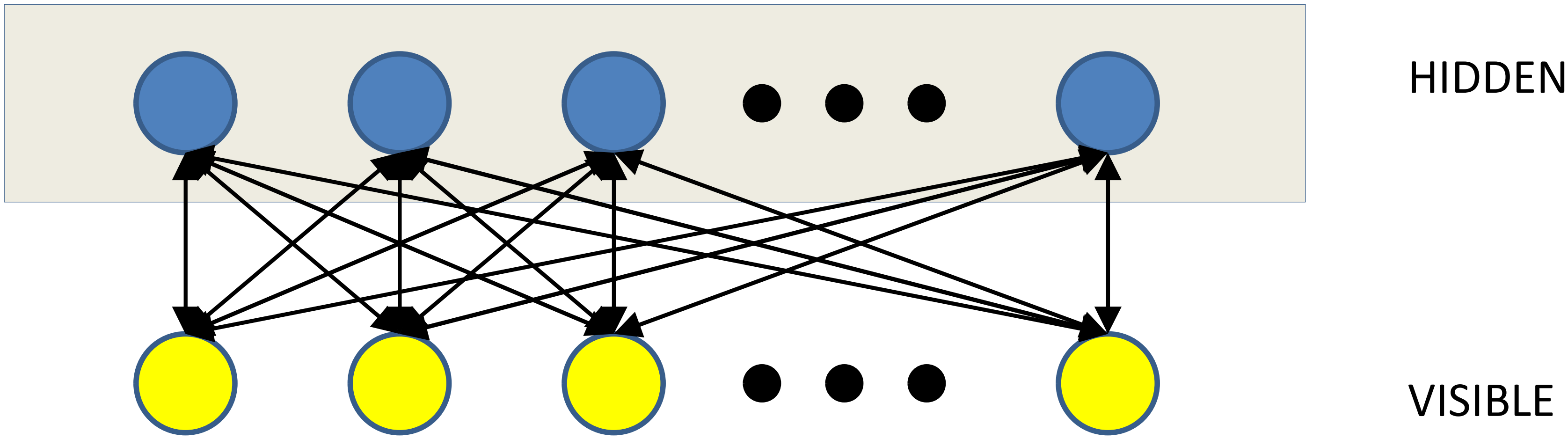
- Training takes for ever
- Doesn't really work for large problems
  - A small number of training instances over a small number of bits

Solution: *Restricted* Boltzmann Machines



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

Solution: *Restricted* Boltzmann Machines

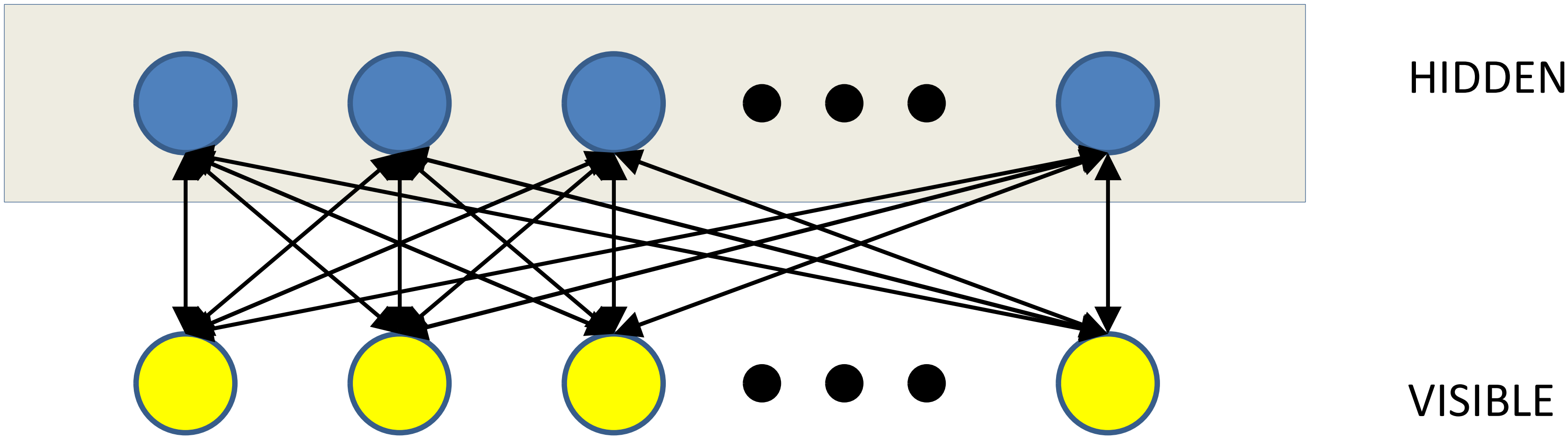


$$z_i = \sum_j w_{ji} s_i + b_i$$

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

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

Solution: *Restricted* Boltzmann Machines



HIDDEN

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

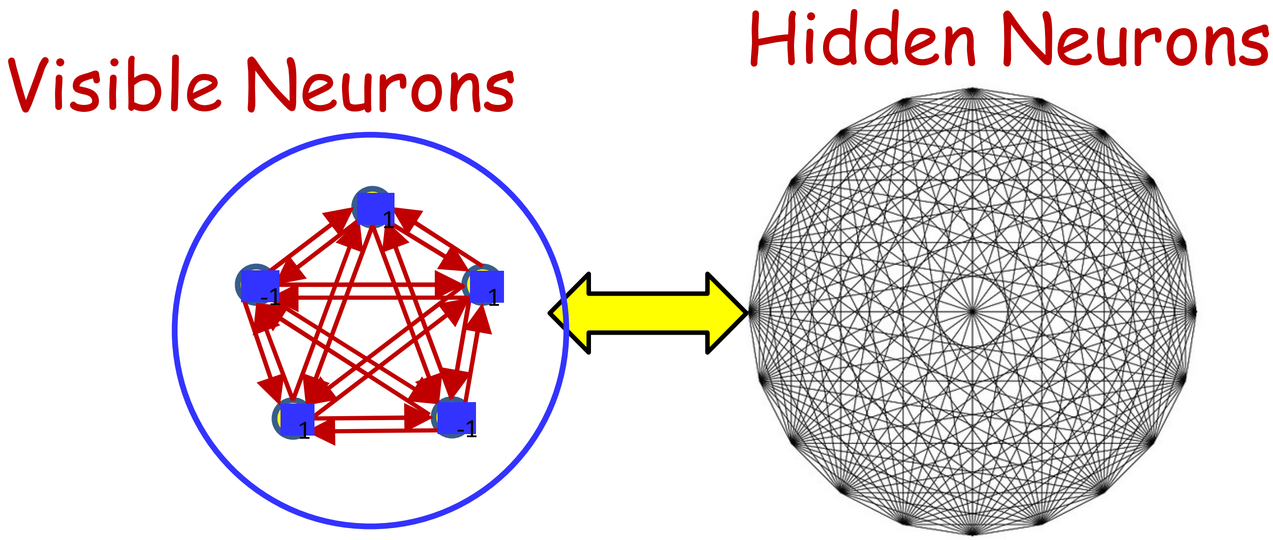
$$P(h_i = 1) = \frac{1}{1 + e^{-z_i}}$$

VISIBLE

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

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

Recap: Training full Boltzmann machines: Step 1

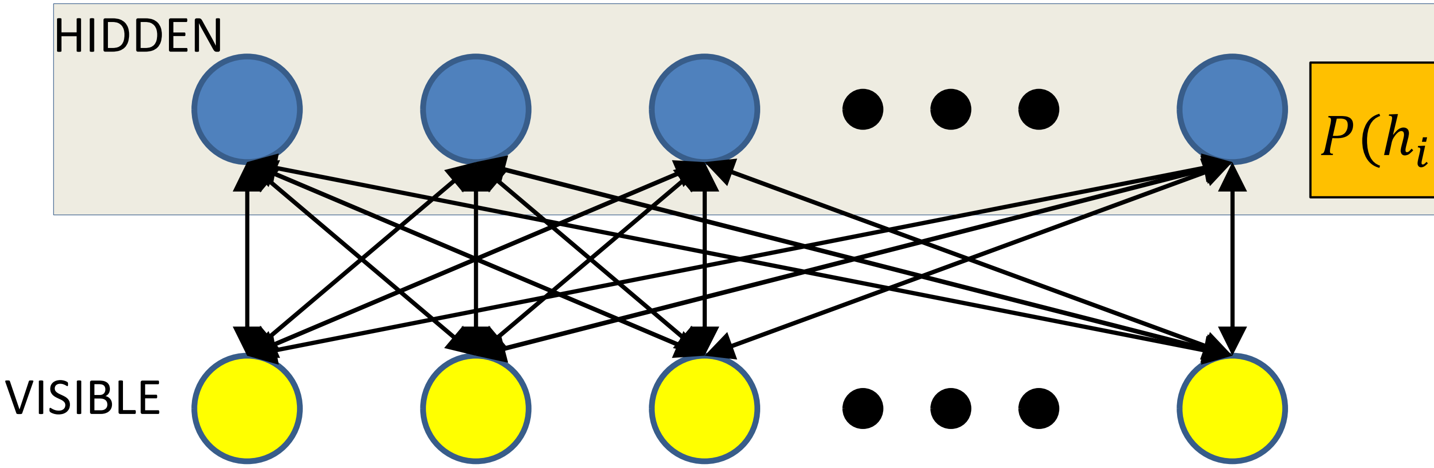


- For each training pattern V_i
  - Fix the visible units to V_i
  - Let the hidden neurons evolve from a random initial point to generate H_i
  - Generate $S_i = [V_i, H_i]$
- Repeat K times to generate synthetic training

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

Sampling: Restricted Boltzmann machine

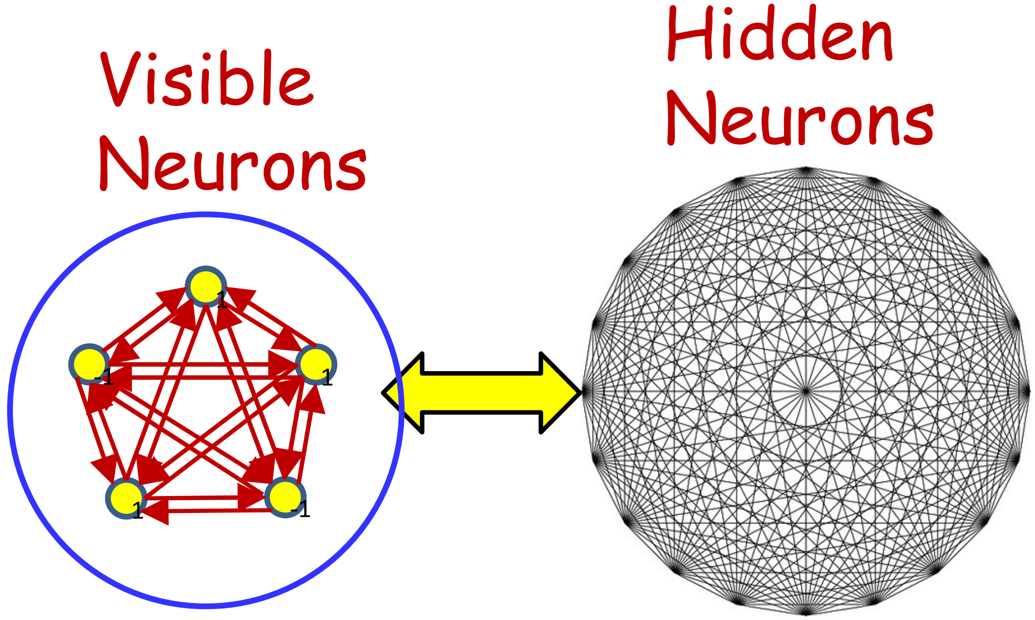
$$z_i = \sum_j w_{ji} v_i + b_i$$



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

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

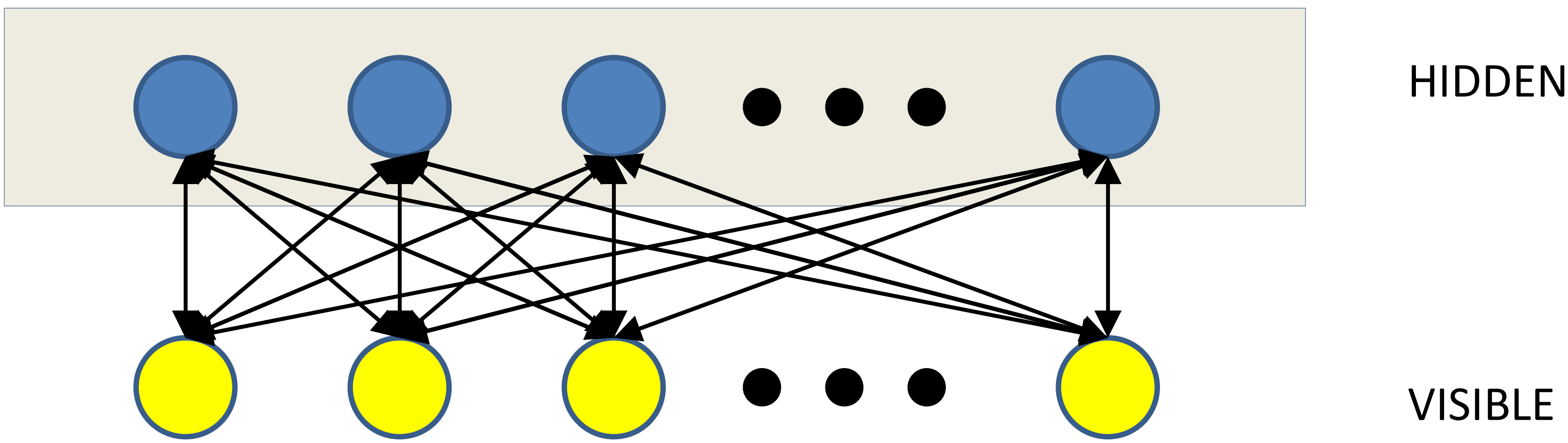
Recap: Training full Boltzmann machines: Step 2



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

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

Sampling: Restricted Boltzmann machine



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

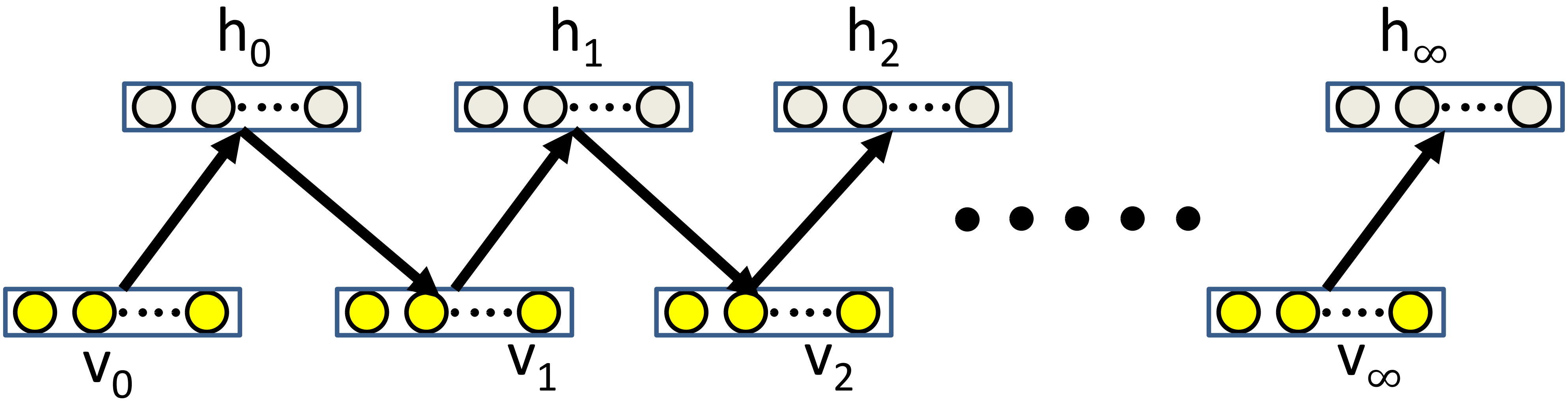
$$P(h_i = 1) = \frac{1}{1 + e^{-z_i}}$$

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

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

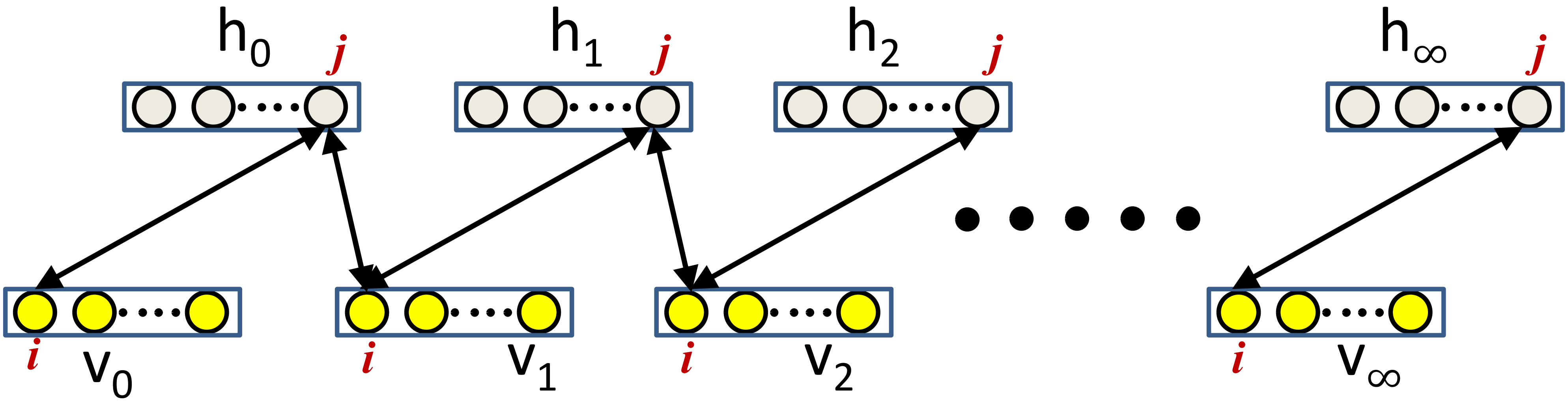
- For each sample:
  - Iteratively sample hidden and visible units for a long time
  - Draw final sample of both hidden and visible units

Pictorial representation of RBM training



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

Pictorial representation of RBM training



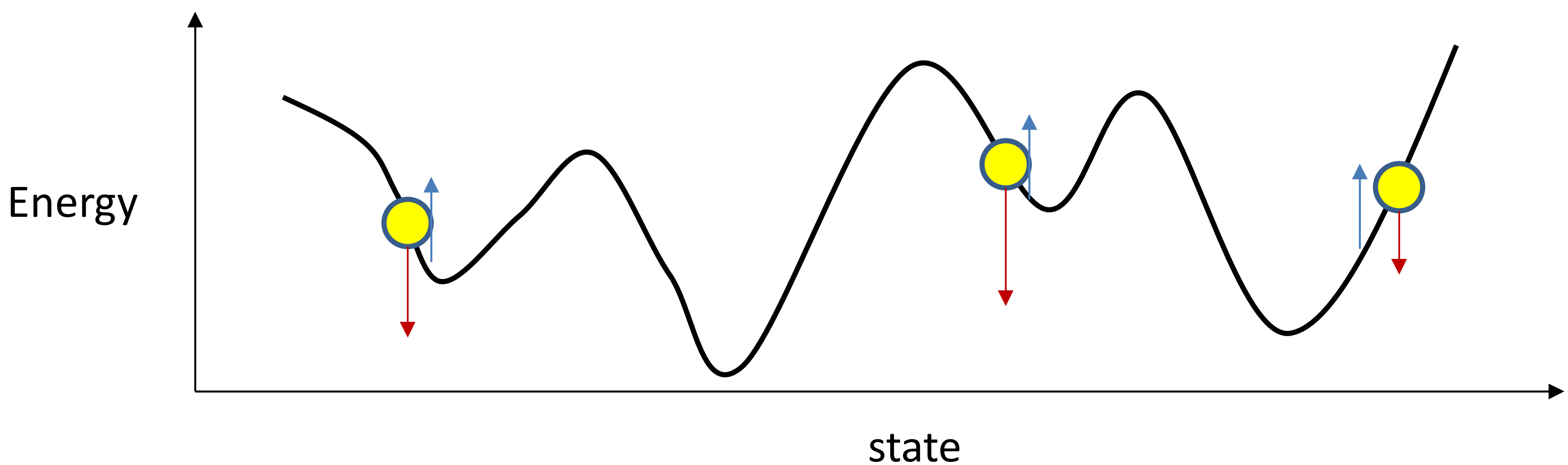
- Gradient (showing only one edge from visible node i to hidden node j)

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

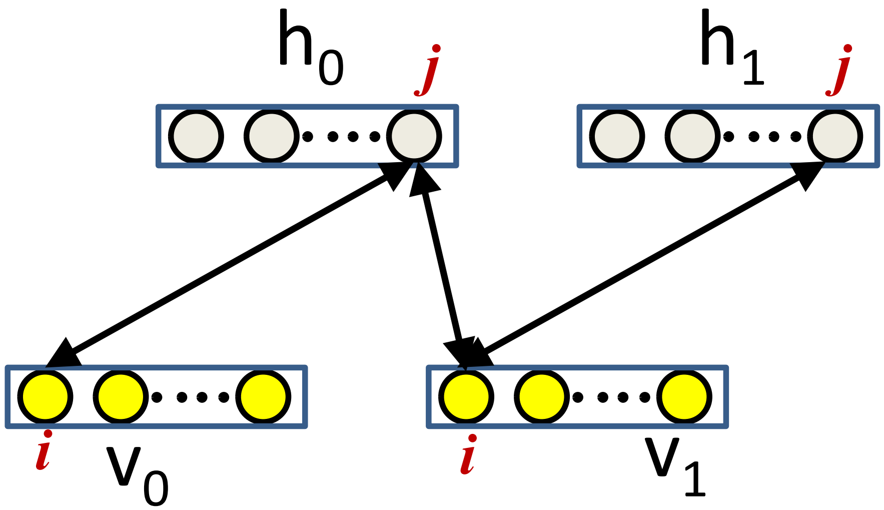
- $\langle v_i, h_j \rangle$ represents average over many generated training samples

Recall: Hopfield Networks

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



A Shortcut: Contrastive Divergence



- Sufficient to run one iteration!

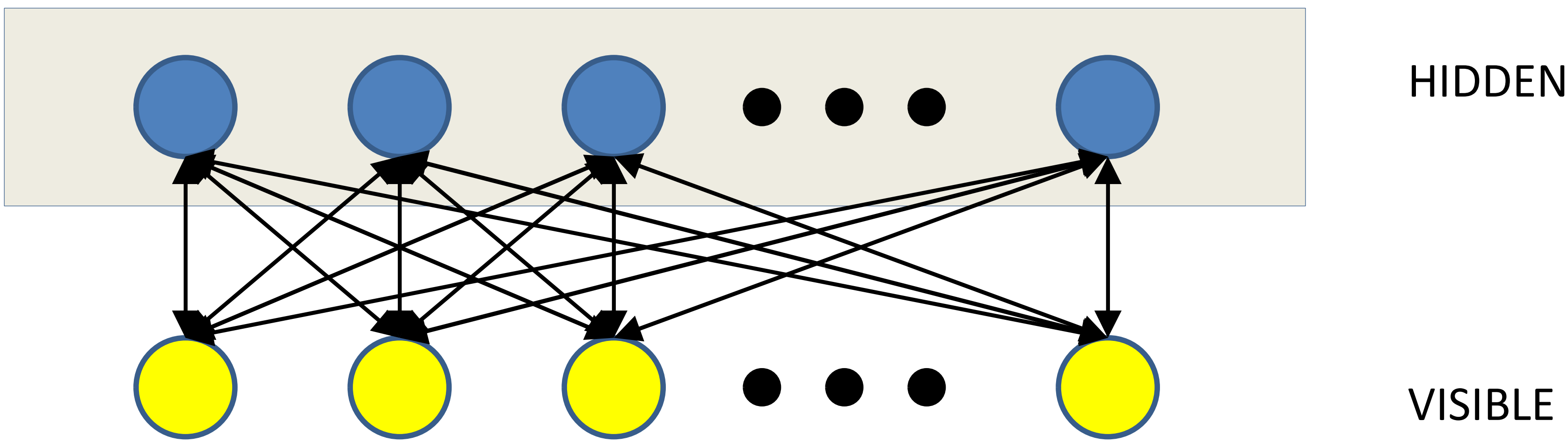
$$\frac{\partial \log p(v)}{\partial w_{ij}} = \langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1$$

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

Restricted Boltzmann Machines

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

Continuous-values RBMs



HIDDEN

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

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

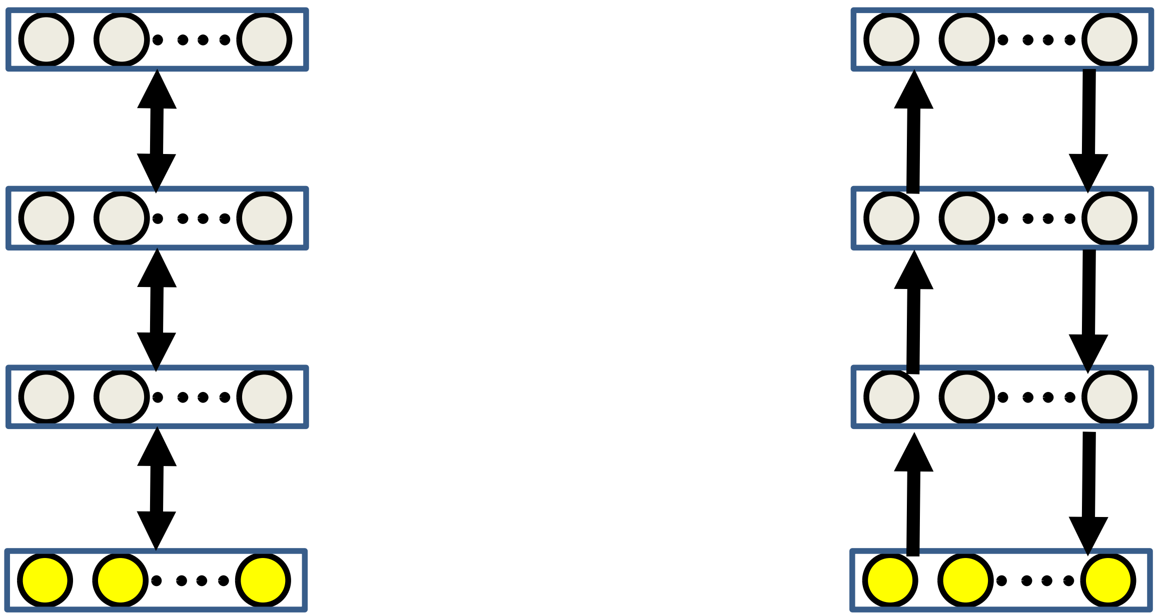
VISIBLE

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

$$P(v_i) = r(y_i) exp(y_i)$$

Hidden units may also be continuous values

Other variants



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

Topics missed..

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