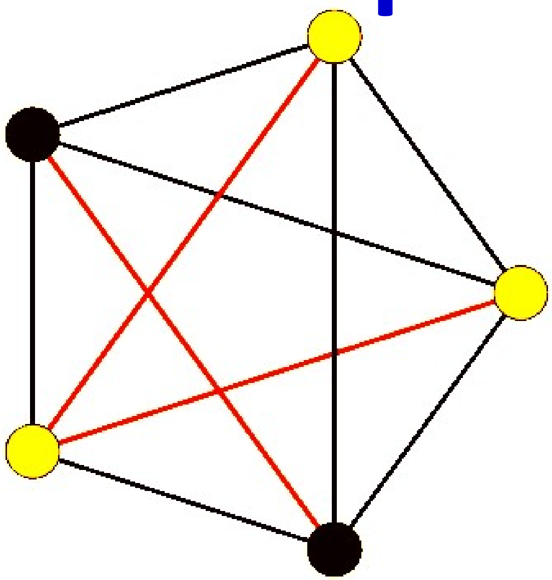
Neural Networks

Hopfield Nets and Boltzmann Machines

Recap: Hopfield network

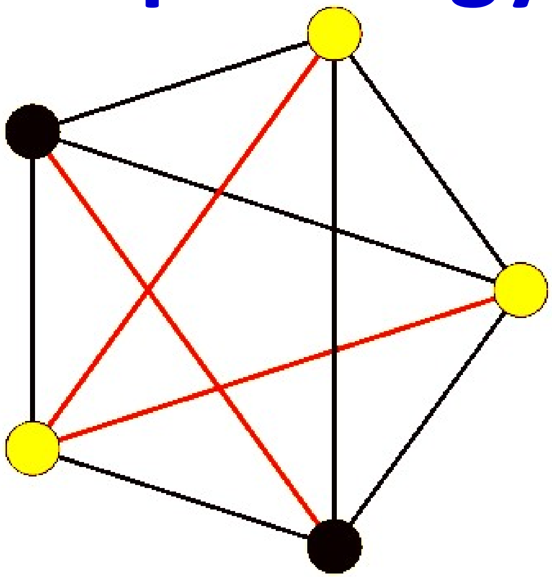


$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

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

Recap: Energy of a Hopfield Network



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

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

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

- The system will evolve until the energy hits a local minimum
- In vector form
  - Bias term may be viewed as an extra input pegged to 1.0

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

Recap: Hopfield net computation

1. Initialize network with initial pattern

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

2. Iterate until convergence

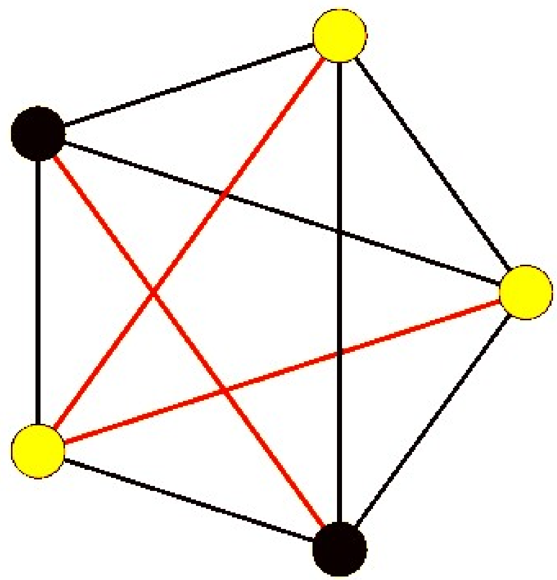
$$y_i(t+1) = \Theta\left(\sum_{j \neq i} w_{ji} y_j\right), \qquad 0 \le i \le N-1$$

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

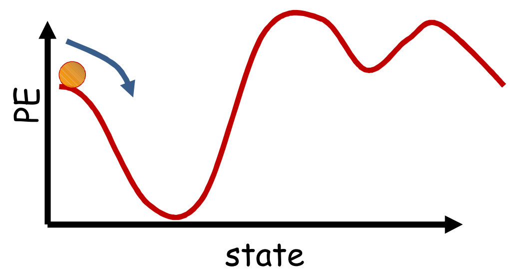
$$E = -\sum_i \sum_{j>i} w_{ji} y_j y_i$$

does not change significantly any more

Recap: Evolution

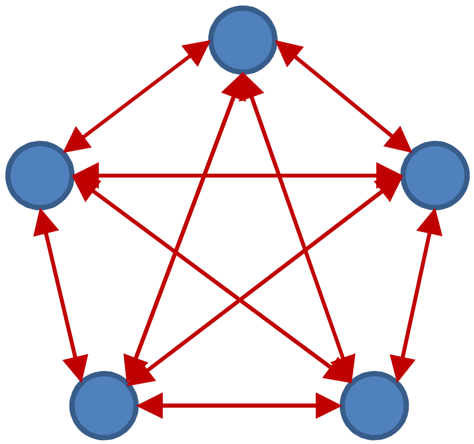


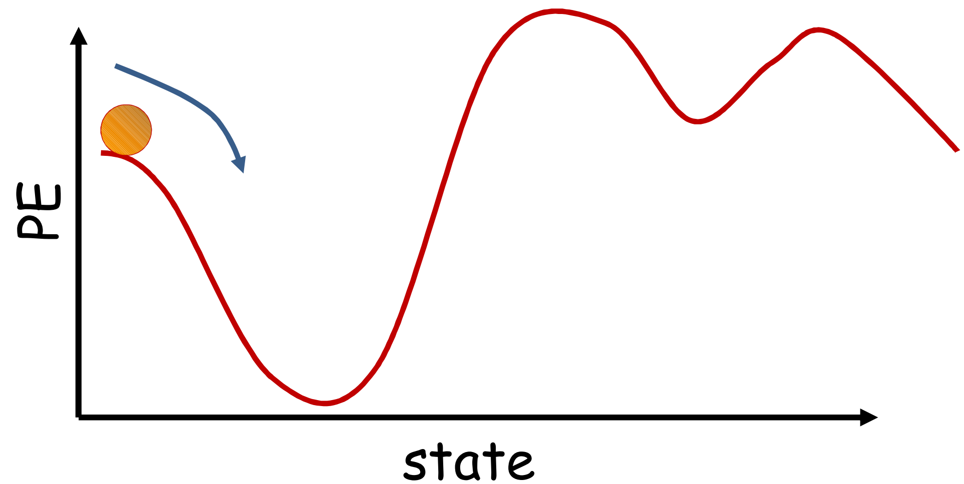
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$



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

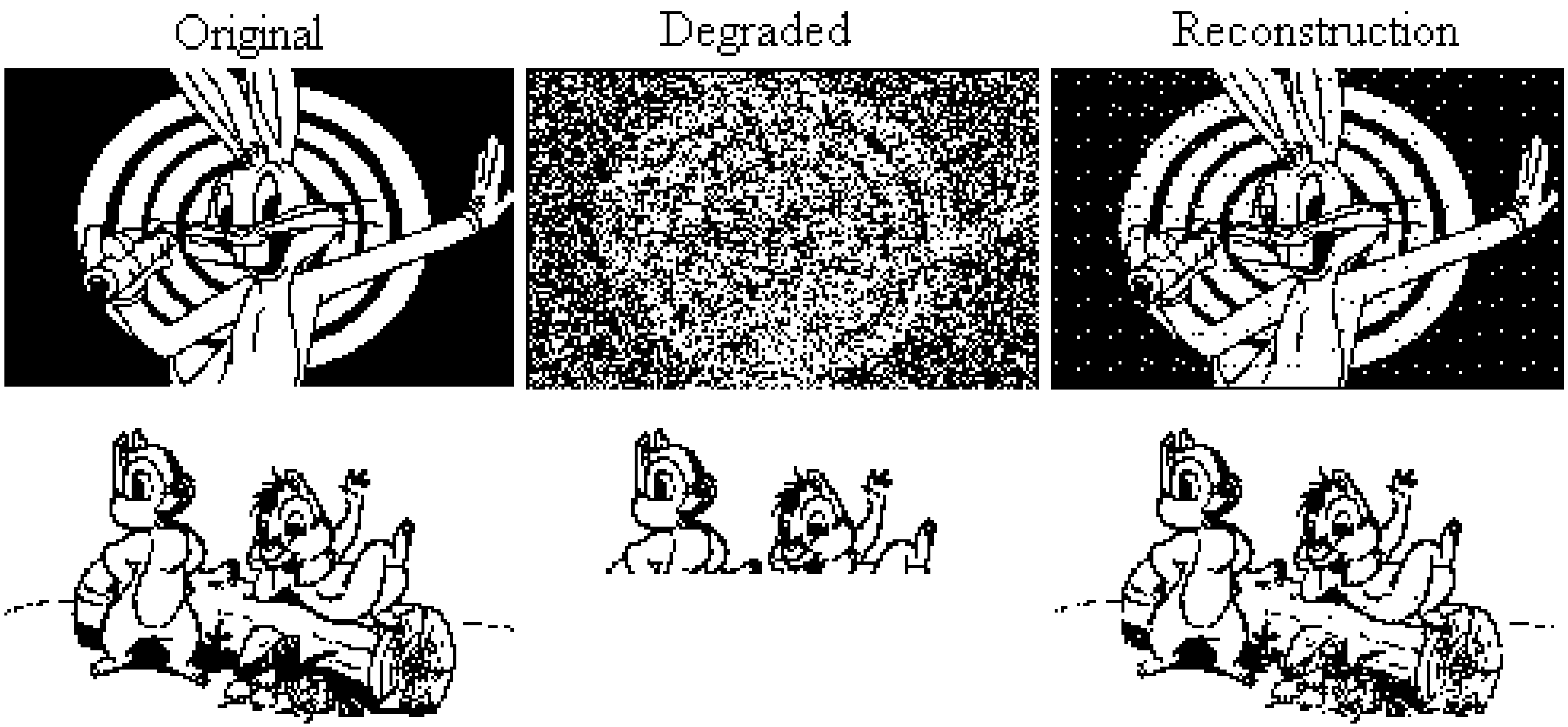
Recap: Content-addressable memory





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

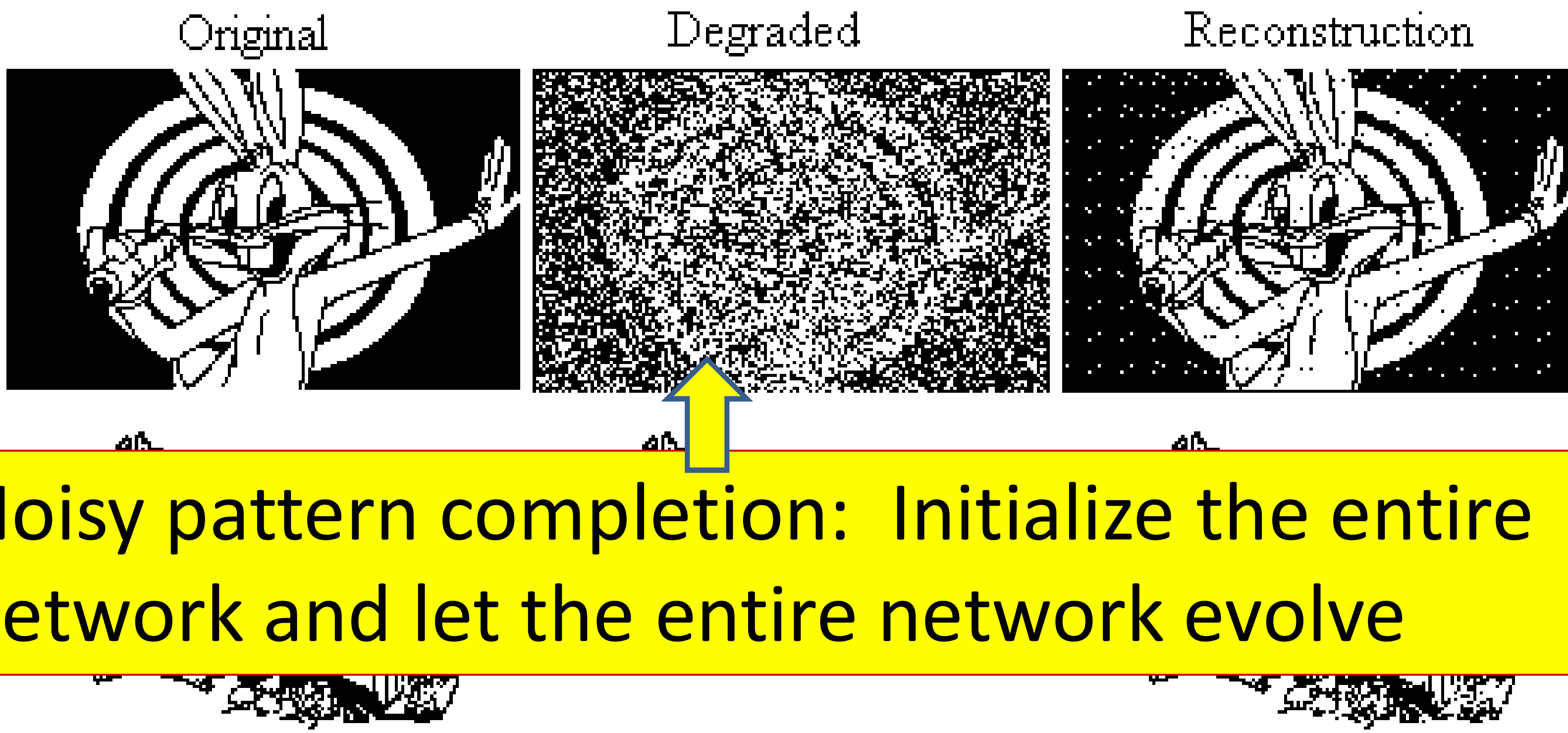
Examples: Content addressable memory



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

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

Examples: Content addressable memory

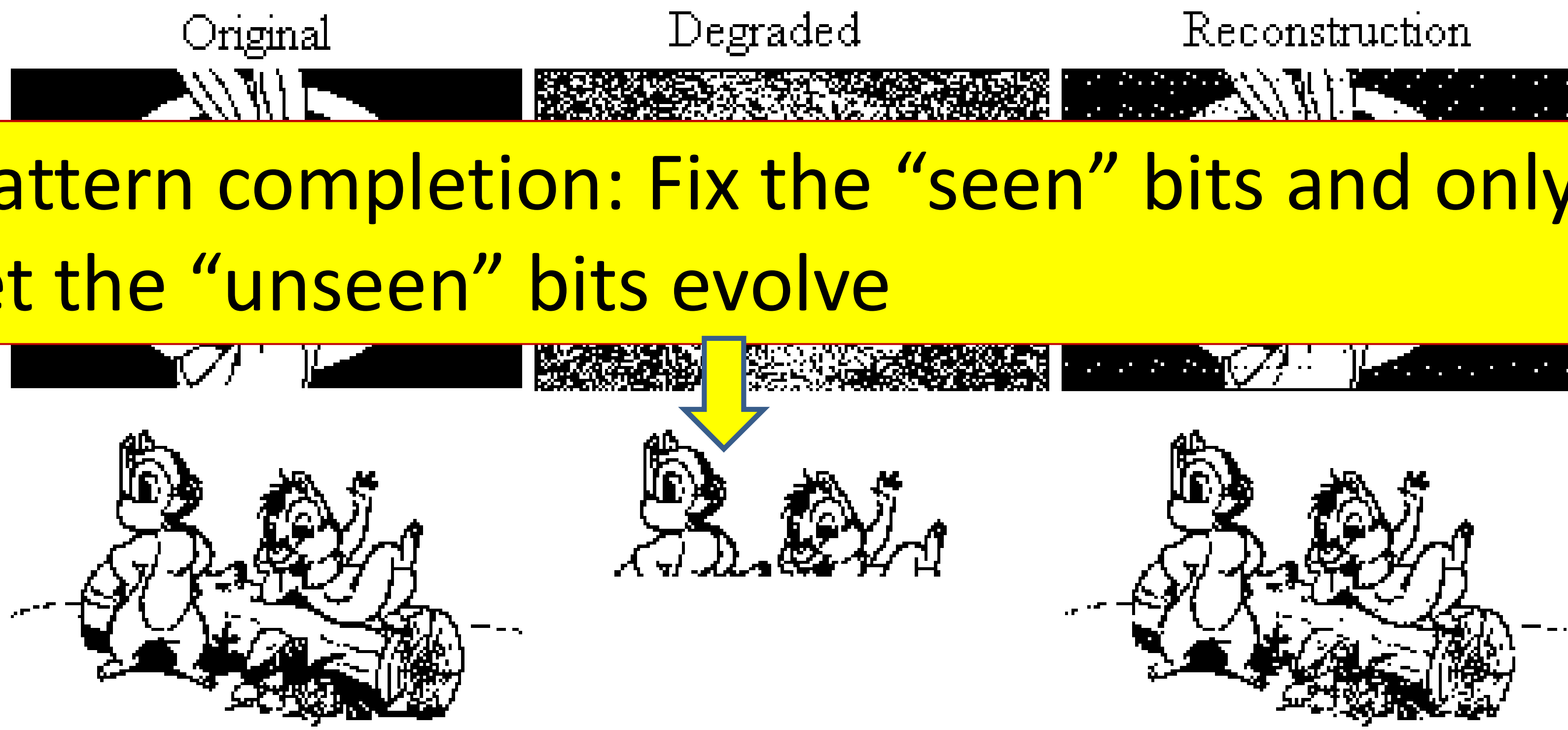


Noisy pattern completion: Initialize the entire network and let the entire network evolve

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

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

Examples: Content addressable memory



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

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

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

Training a Hopfield Net to "Memorize" target patterns

- The Hopfield network can be *trained* to remember specific "target" patterns
  - E.g. the pictures in the previous example
- This can be done by setting the weights $\mathbf{W}$ appropriately

Random Question:
Can you use *backprop* to train Hopfield nets?

Hint: Think RNN

Training a Hopfield Net to "Memorize" target patterns

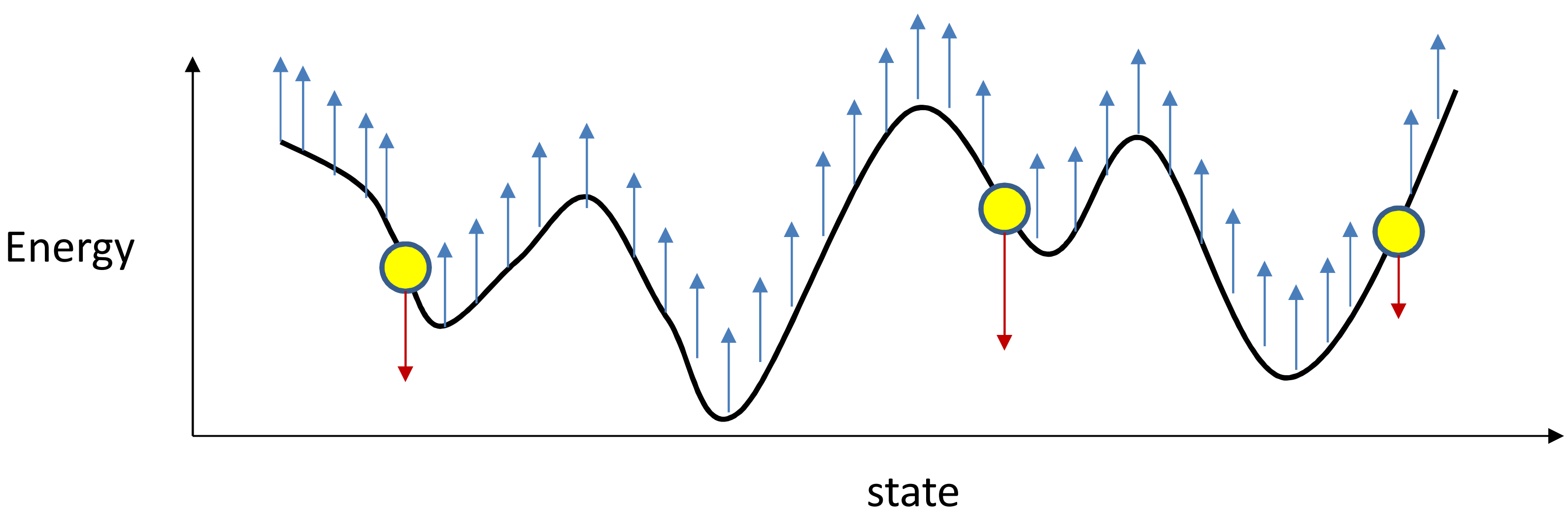
- The Hopfield network can be *trained* to remember specific "target" patterns
  - E.g. the pictures in the previous example
- A Hopfield net with N neurons can designed to store up to N target N-bit memories
  - But can store an exponential number of unwanted "parasitic" memories along with the target patterns
- **Training the network:** Design weights matrix $\mathbf{W}$ such that the energy of …
  - Target patterns is minimized, so that they are in energy wells
  - *Other untargeted* potentially parasitic patterns is maximized so that they don't become parasitic

Training the network

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

Minimize energy of target patterns

Maximize energy of all other 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:

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

Minimize energy of target patterns

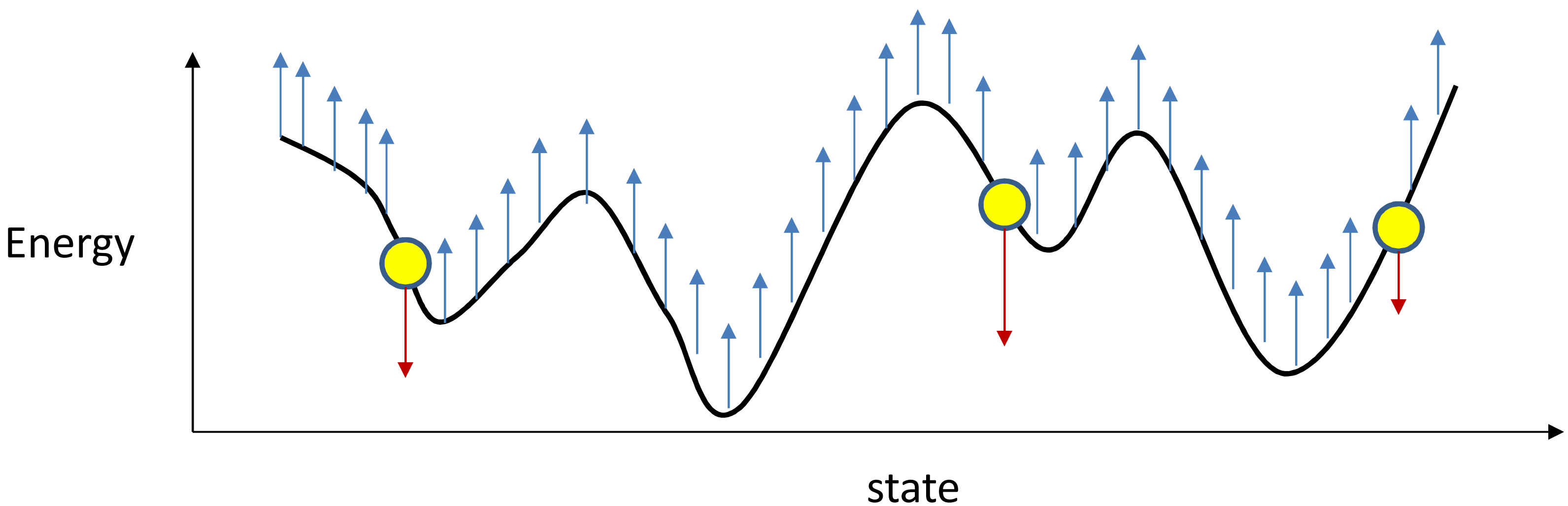
Maximize energy of all other patterns

Training the network

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

Minimize energy of target patterns

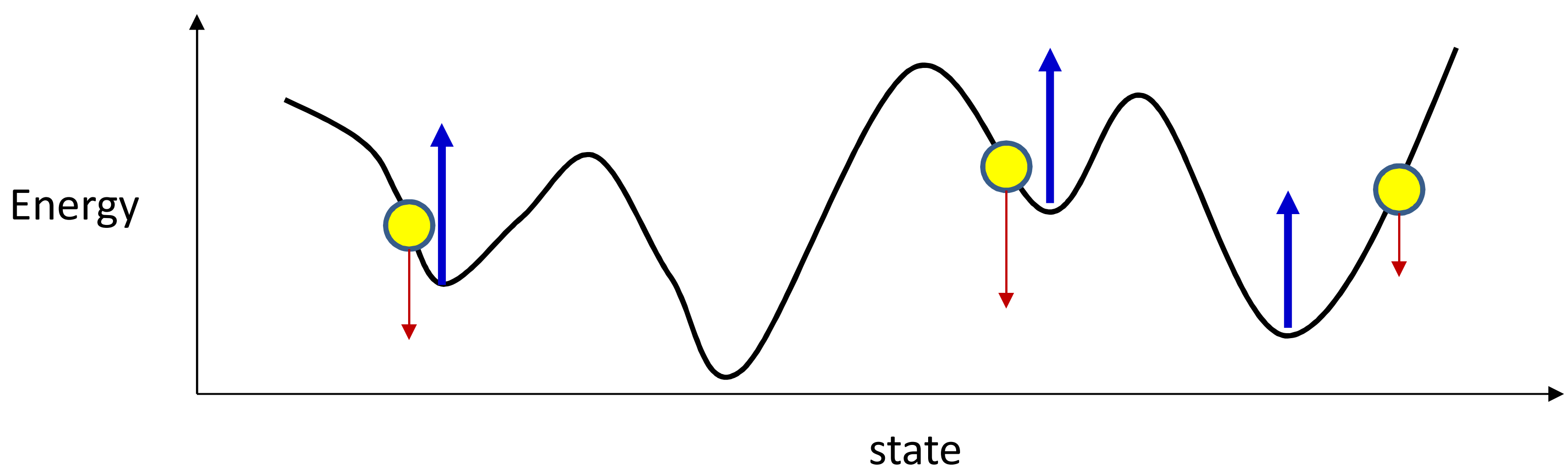
Maximize energy of all other patterns



Simpler: Focus on confusing parasites

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

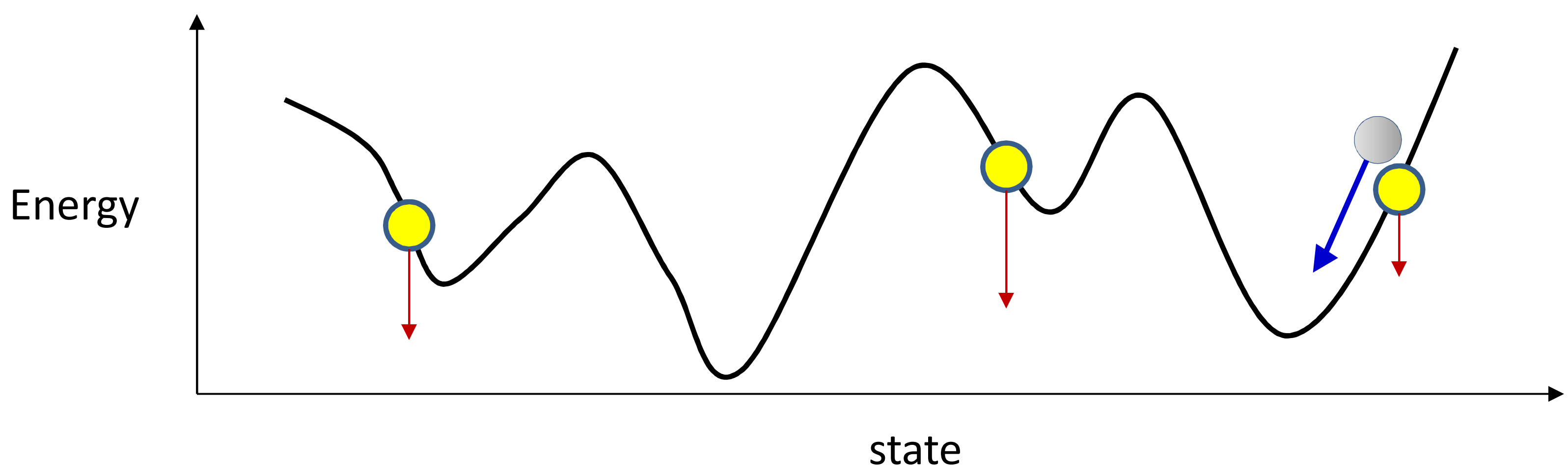
- Focus on minimizing parasites that can prevent the net from remembering target patterns
  - Energy valleys in the neighborhood of target patterns



Training to maximize memorability of target patterns

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

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



Training the Hopfield network

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

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

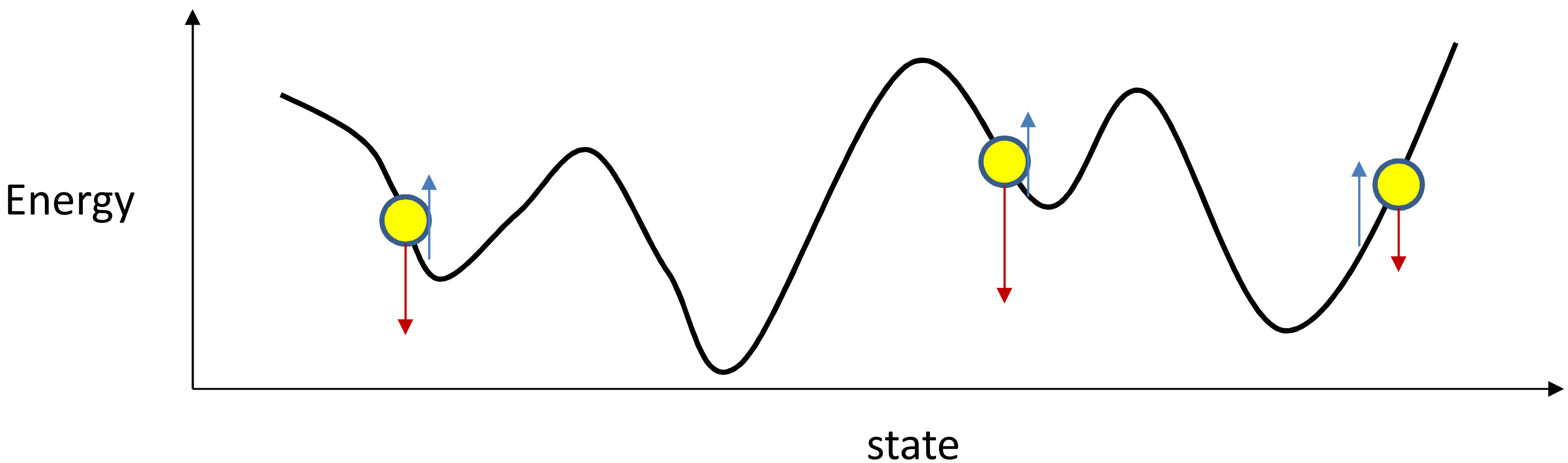
Training the Hopfield network: SGD version

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

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

More efficient training

- 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

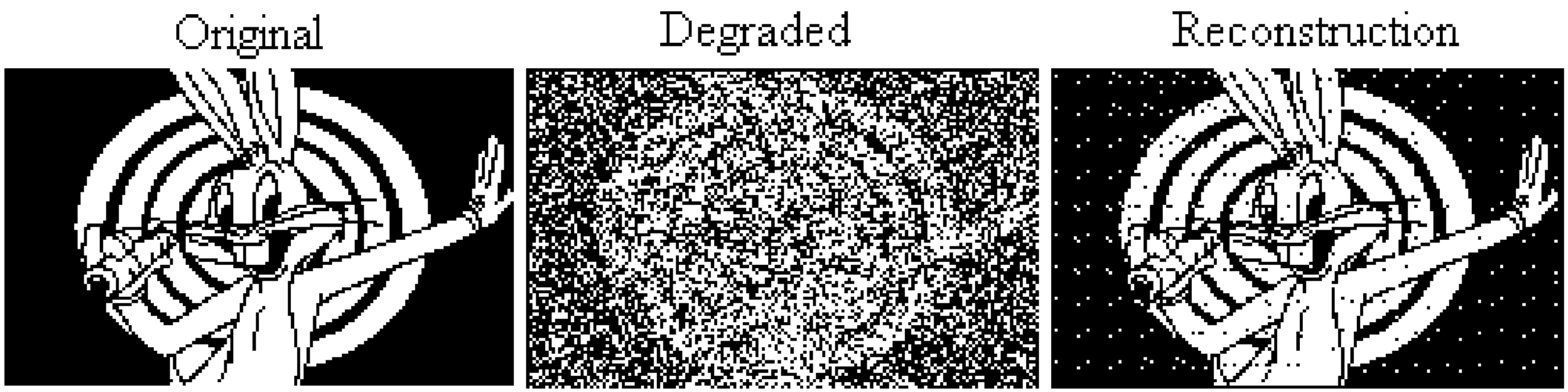


Training the Hopfield network: SGD version

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

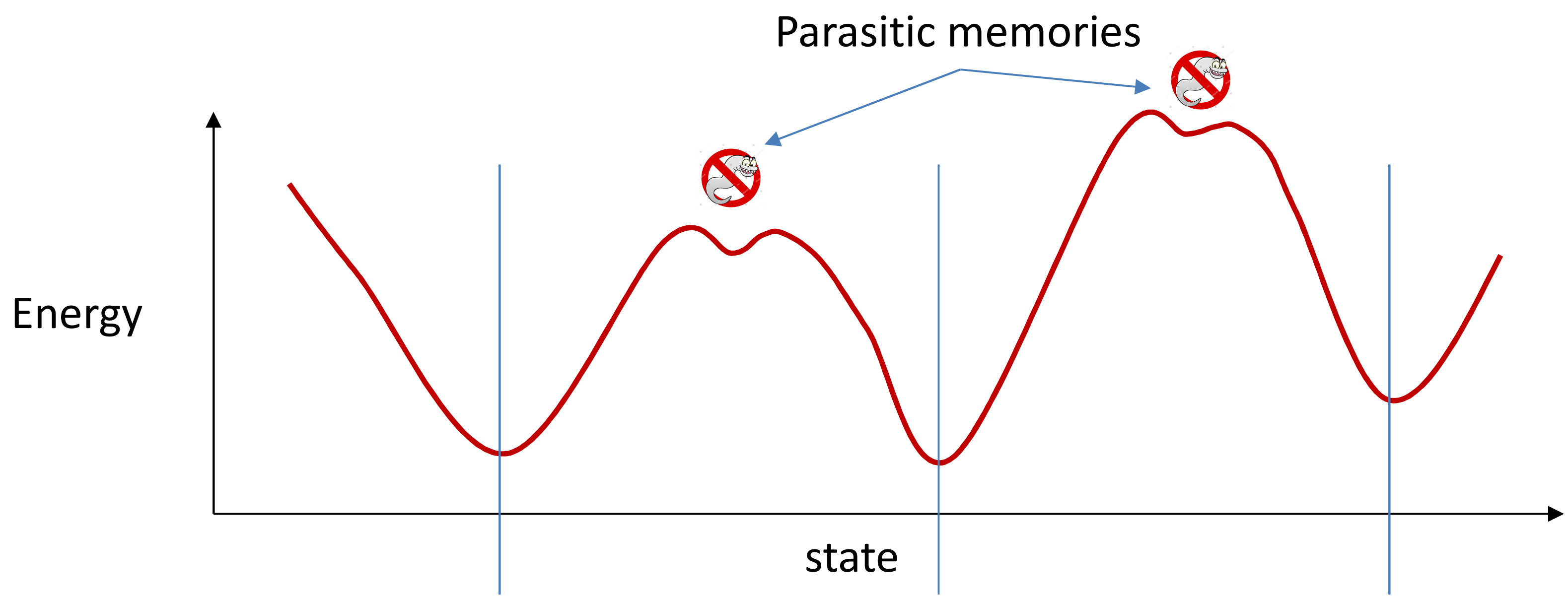
- Initialize $\mathbf{W}$
- Do until convergence, satisfaction, or death from boredom:
  - Sample a target pattern $\mathbf{y}_p$
    - Sampling frequency of pattern must reflect importance of pattern
  - Initialize the network at $\mathbf{y}_p$ and let it evolve ***a few steps (2-4)***
    - And arrive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

Problem with Hopfield net



- Why is the recalled pattern not perfect?

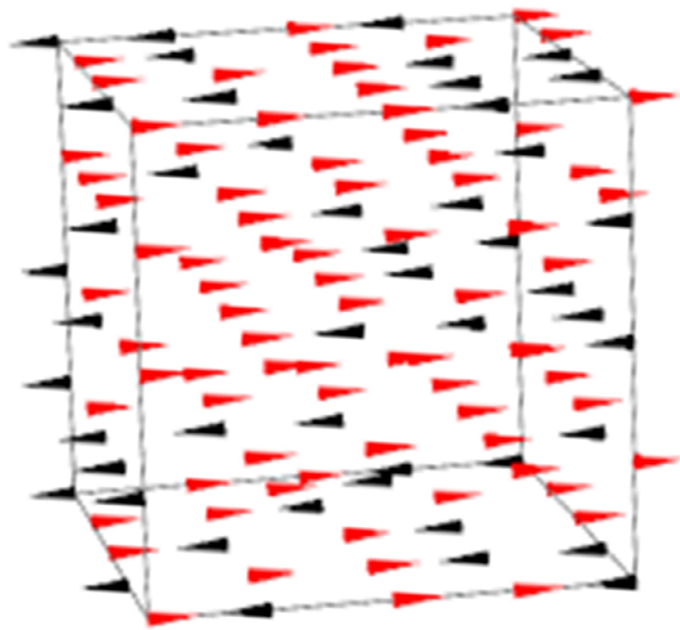
A Problem with Hopfield Nets



- Many local minima
  - Parasitic memories

- May be escaped by adding some *noise* during evolution
  - Permit changes in state even if energy increases..
    - Particularly if the increase in energy is small

Recap – Analogy: Spin Glasses



Total field at current dipole:

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

Response of current diplose

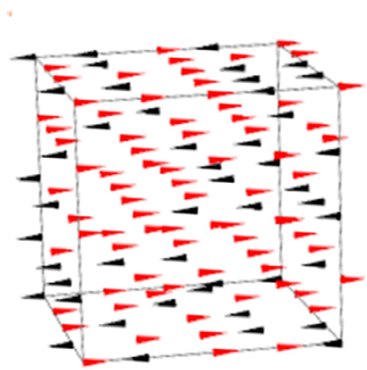
$$x_i = \begin{cases} x_i \; if \; sign\big(x_i \, f(p_i)\big) = 1 \\ -x_i \; otherwise \end{cases}$$

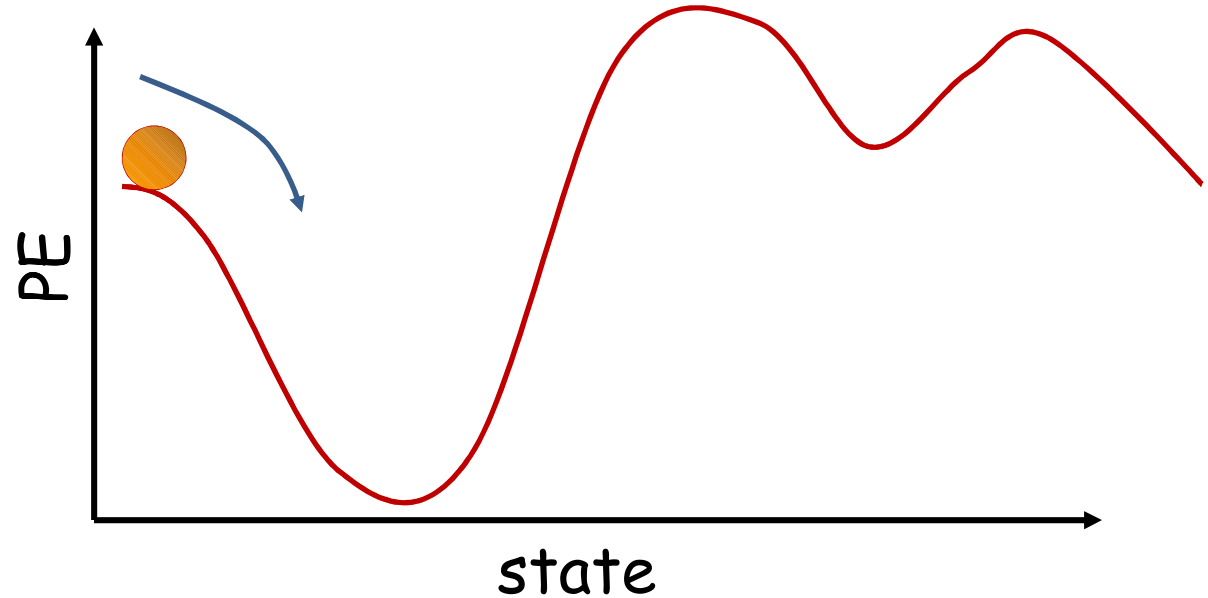
- The total energy of the system

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

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

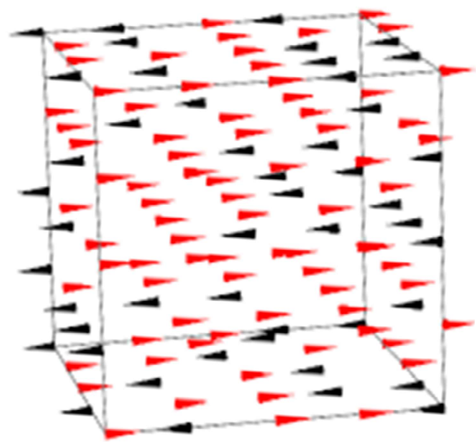
Recap : Spin Glasses

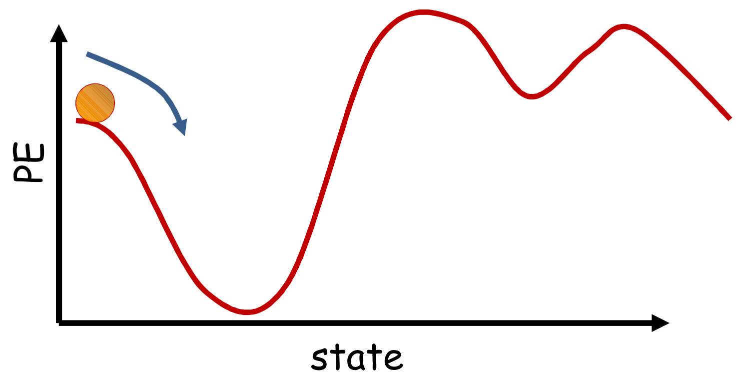




- The system stops at one of its *stable* configurations
  - Where energy is a local minimum

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 at equilibrium
  - The system "prefers" low energy states
  - Evolution of the system favors transitions towards lower-energy states

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 measures the *useful* work derivable from it and 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

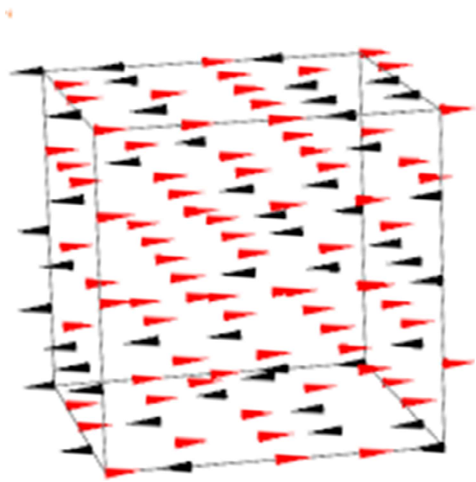
$$F_T = \sum_s P_T(s)\, E_s - kT \sum_s P_T(s) \log P_T(s)$$

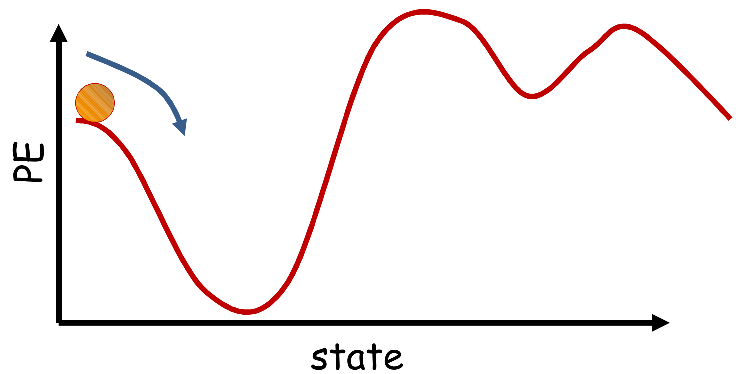
- Minimizing this w.r.t $P_T(s)$, we get

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

  - 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.

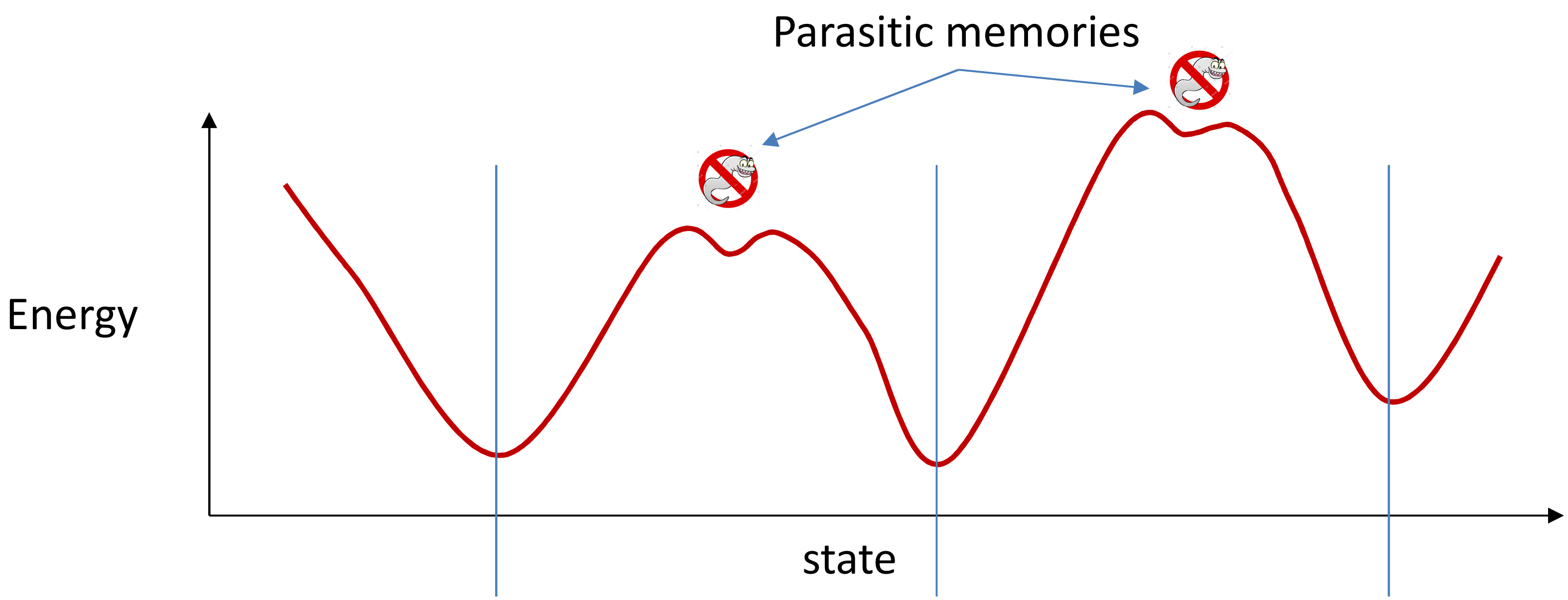
Revisiting Thermodynamic Phenomena





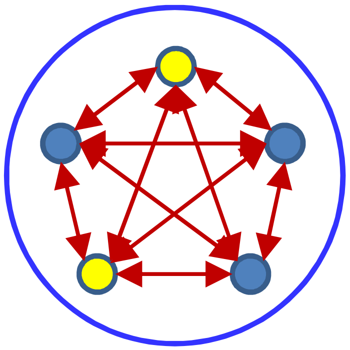
- The evolution of the system is actually *stochastic*
- At equilibrium the system visits various states according to the Boltzmann distribution
  - The probability of any state is inversely related to its energy
    - and also temperatures: $P(s) \propto exp\left(\frac{-E_s}{kT}\right)$
- The most likely state is the lowest energy state

Returning to the problem with Hopfield Nets



- Many local minima
  - Parasitic memories
- May be escaped by adding some *noise* during evolution
  - Permit changes in state even if energy increases..
    - Particularly if the increase in energy is small

The Hopfield net as a distribution



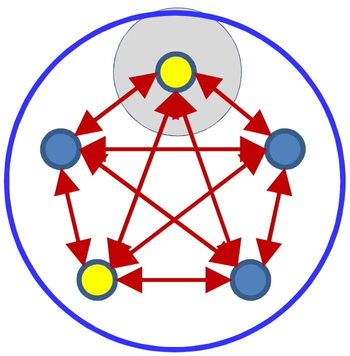
$$E(S) = -\sum_{i<j} w_{ij} s_i s_j - b_i s_i$$

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

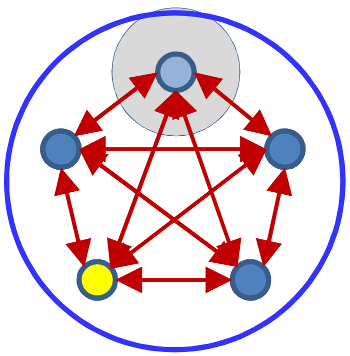
- Mimics the Spin glass system
- The stochastic Hopfield network models a ***probability distribution*** over states
  - Where a state is a binary string
  - Specifically, it models a *Boltzmann distribution*
  - **The parameters of the model are the weights of the network**
- The probability that (at 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



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

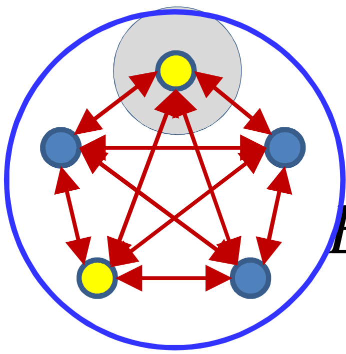


$$logP(S) - logP(S') = logP(s_i = 1 | s_{j\neq i}) - logP(s_i = -1 | s_{j\neq i})$$

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

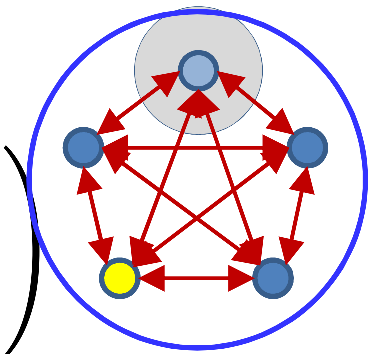
The field at a single node

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



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

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



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

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

The field at a single node

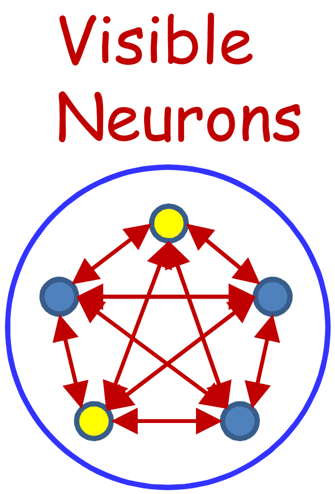
$$log\left(\frac{P(s_i = 1|s_{j\neq i})}{1 - P(s_i = 1|s_{j\neq i})}\right) = \sum_{j\neq i} w_{ij}s_j + b_i$$

- Giving us

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

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

Redefining the network

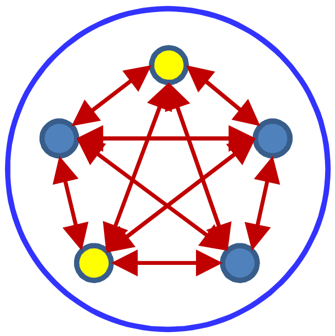


$$z_i = \sum_j w_{ij} s_j + b_i$$

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

- 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

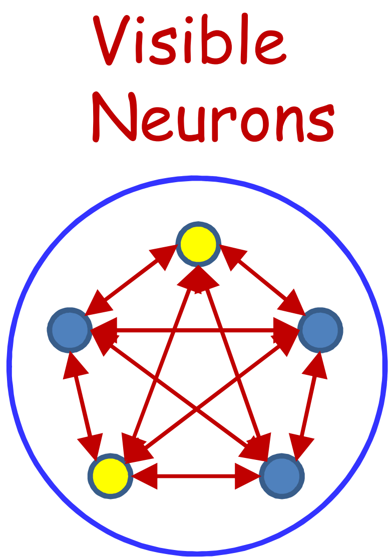


$$z_i = \sum_j w_{ij} s_j + b_i$$

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

- 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

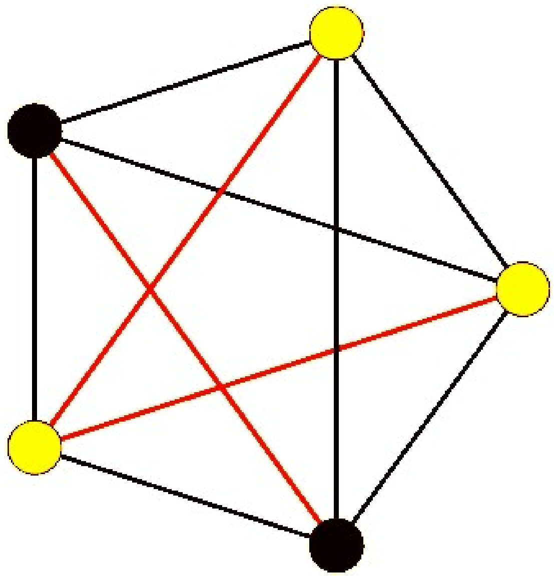


$$z_i = \sum_j w_{ij} s_j + b_i$$

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

- Initialize the neurons
- Cycle through the neurons and randomly set the neuron to 1 or 0 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

Recap: Stochastic Hopfield Nets



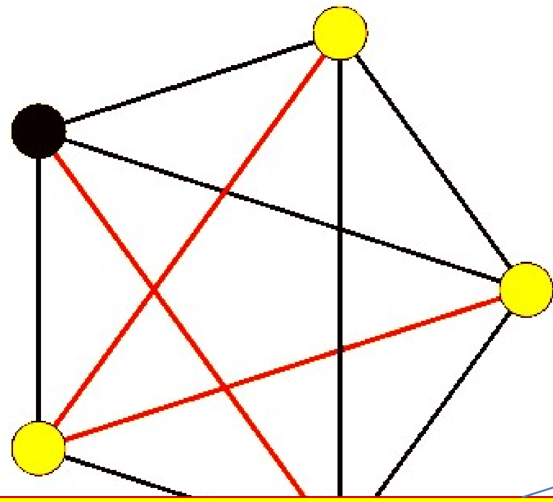
$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ij} y_j$$

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

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

- The evolution of the Hopfield net can be made *stochastic*
- Instead of deterministically responding to the sign of the local field, each neuron responds *probabilistically*
  - This is much more in accord with Thermodynamic models
  - The evolution of the network is more likely to escape spurious "weak" memories

Recap: Stochastic Hopfield Nets



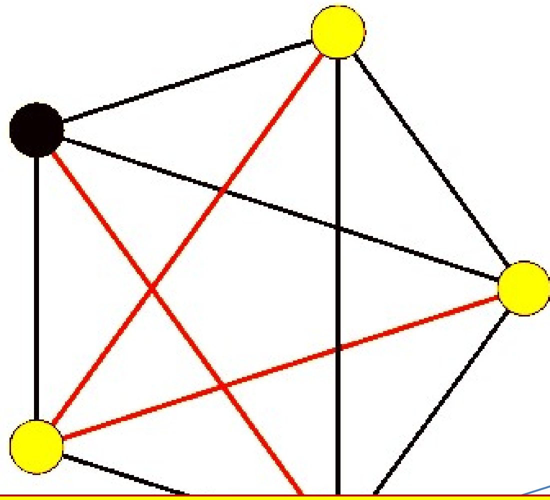
$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ji} y_j$$

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

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

- The evolution of the Hopfield net can be made *stochastic*
- Instead of deterministically responding to the sign of the local field, each neuron responds *probabilistically*
  - This is much more in accord with Thermodynamic models
  - The evolution of the network is more likely to escape spurious "weak" memories

Recap: Stochastic Hopfield Nets



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

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

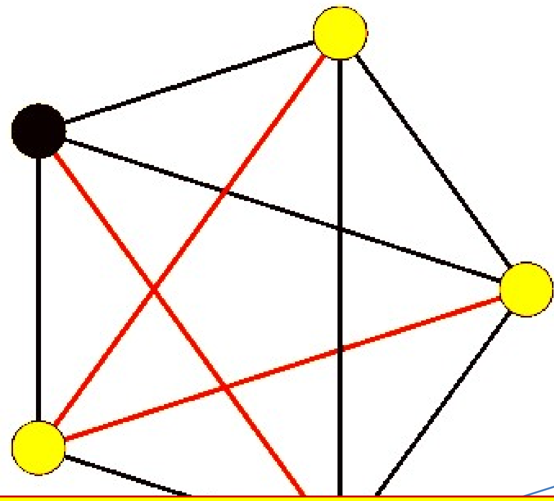
The field quantifies the energy difference obtained by flipping the current unit

- The evolution of the Hopfield net can be made *stochastic*

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

- Instead of deterministically responding to the sign of the local field, each neuron responds *probabilistically*
  - This is much more in accord with Thermodynamic models
  - The evolution of the network is more likely to escape spurious "weak" memories

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

- The evolution of the Hopfield net can be made *stochastic*

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

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

  - The evolution of the network is more likely to escape spurious "weak" memories

Evolution of a stochastic Hopfield net

1. Initialize network with initial pattern

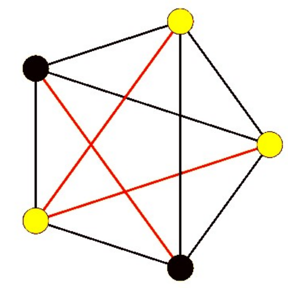
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

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

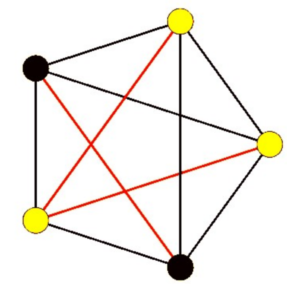
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

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

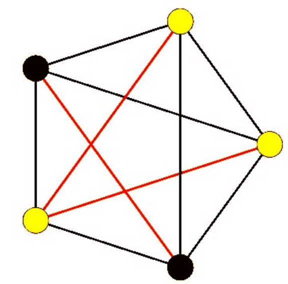
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

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

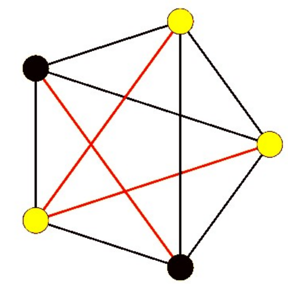
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

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

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

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

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 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

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

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 $0 \le i \le N-1$

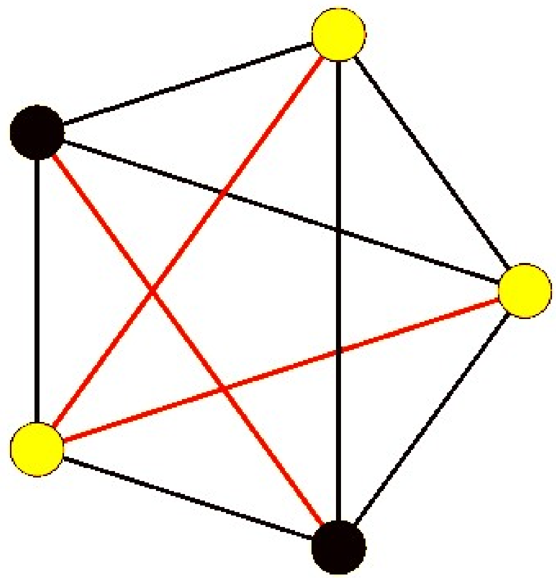
$$P = \sigma\left(\sum_{j \ne i} w_{ji} y_j\right)$$

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

Assuming T = 1

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

Recap: Stochastic Hopfield Nets

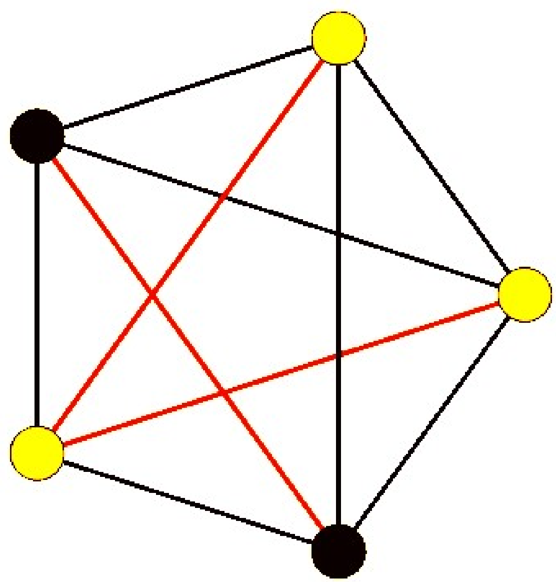


$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ji} y_j$$

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

- 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



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

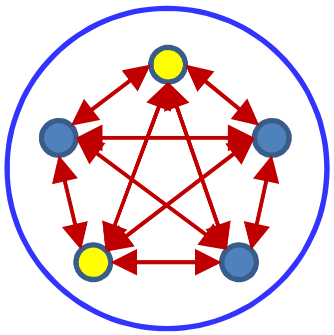
$$P(y_i = 1 | y_{j \neq i}) = \sigma(z_i)$$

- The probability of any state **y** can be shown to be given by the *Boltzmann distribution*

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

  – Minimizing energy maximizes log likelihood

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

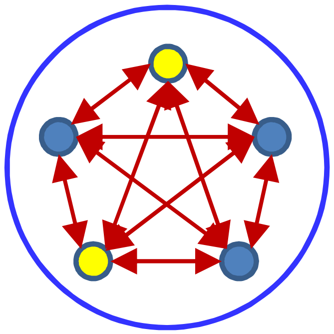
$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$

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

  - 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



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

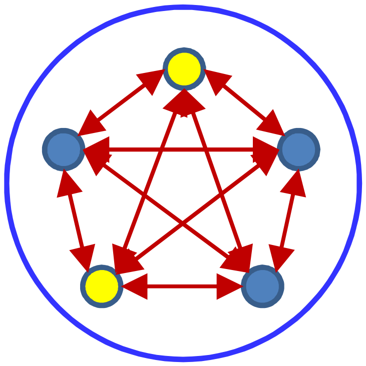
$$P(s_i = 1 | s_{j \neq i}) = \frac{1}{1 + e^{-z_i}}$$

- 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 network



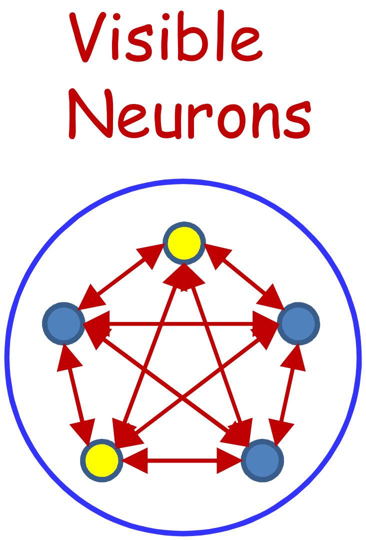
$$E(S) = -\sum_{i<j} w_{ij} s_i s_j$$

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

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

- 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



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

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

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

- Must train the network to assign a desired probability distribution to states
- Given a set of "training" inputs $S_1, \ldots, S_N$
  - 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

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

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

- The second term 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!

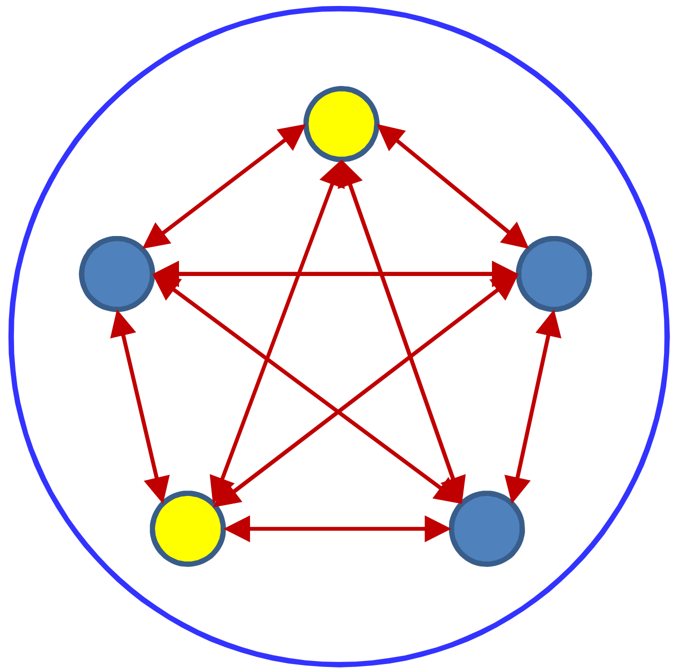
Estimating the second term

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

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

- 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

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

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

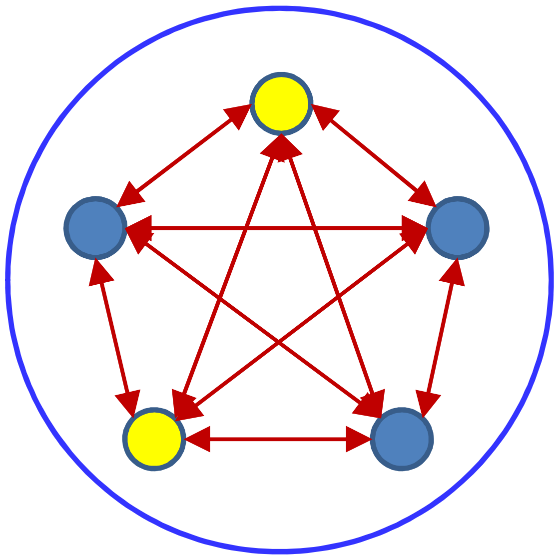
$$\langle \log(P(\mathbf{S})) \rangle = \frac{1}{N}\sum_{S}\left(\sum_{i<j} w_{ij}s_i s_j\right) - \log\left(\sum_{S' \in \mathbf{S}_{simul}} exp\left(\sum_{i<j} w_{ij}s'_i s'_j\right)\right)$$

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

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

- The overall gradient ascent rule

Overall Training

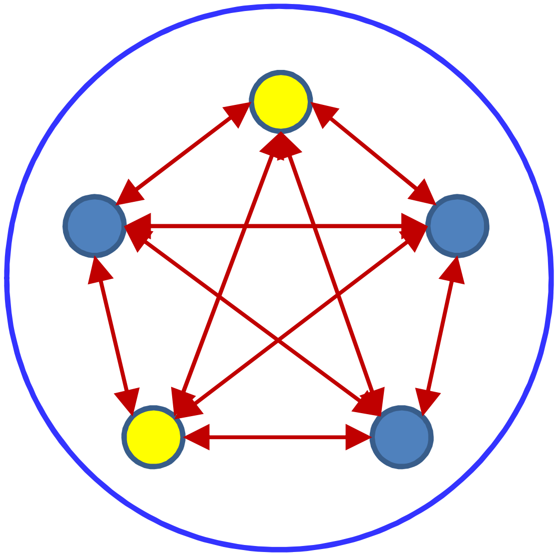


$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{N}\sum_{S} s_i s_j - \frac{1}{M}\sum_{S' \in \mathbf{S}_{simul}} s_i' s_j'$$

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

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

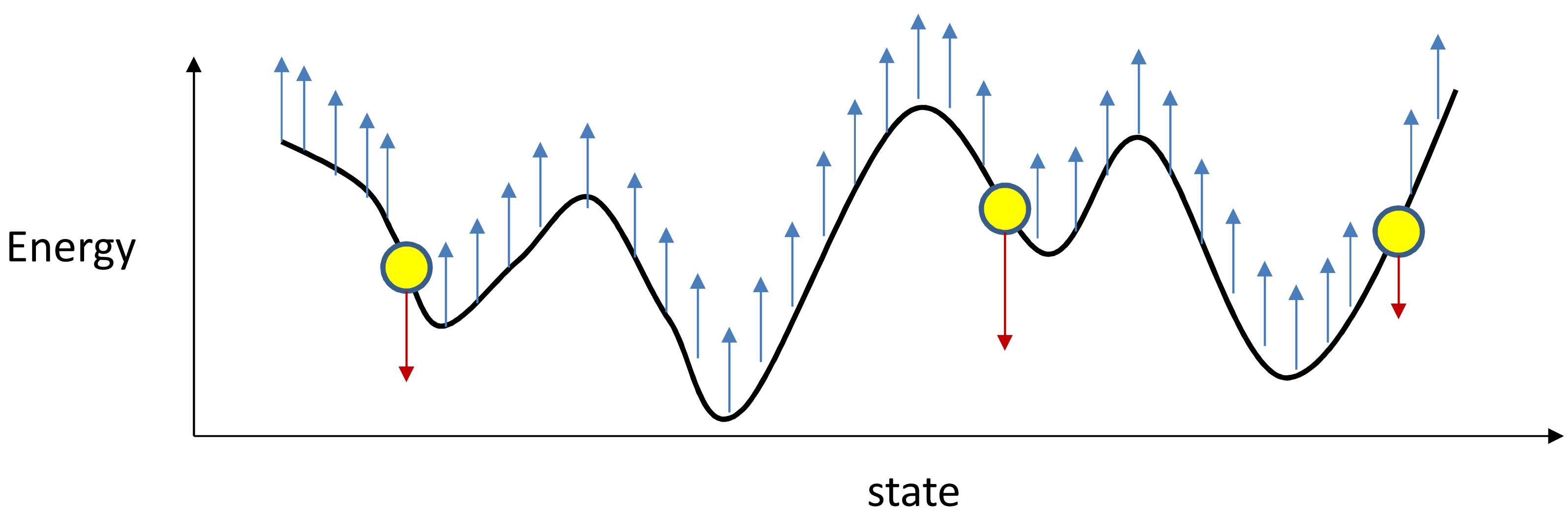
Overall Training



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

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

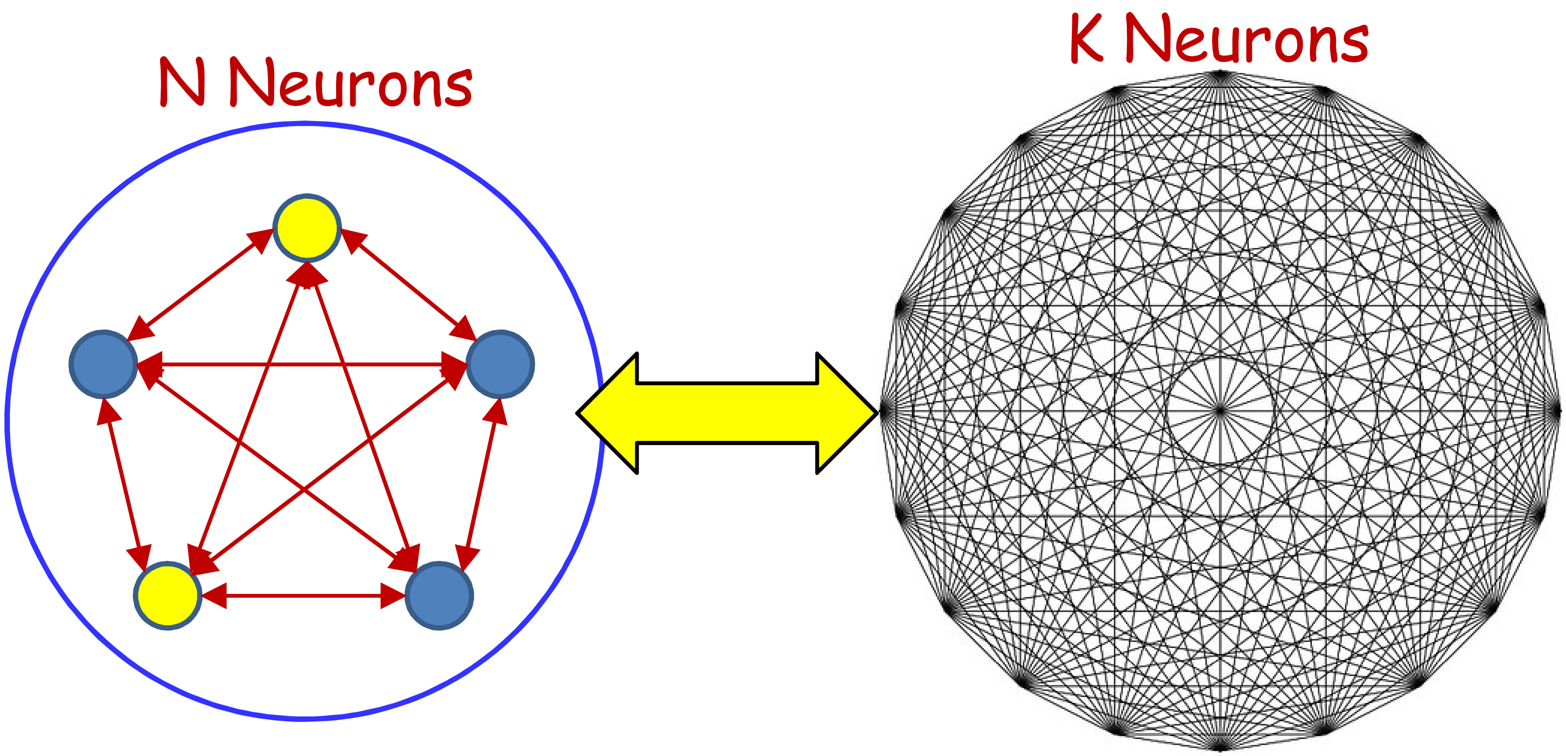
Note the similarity to the update rule for the Hopfield network



Adding Capacity to the Hopfield Network / Boltzmann Machine

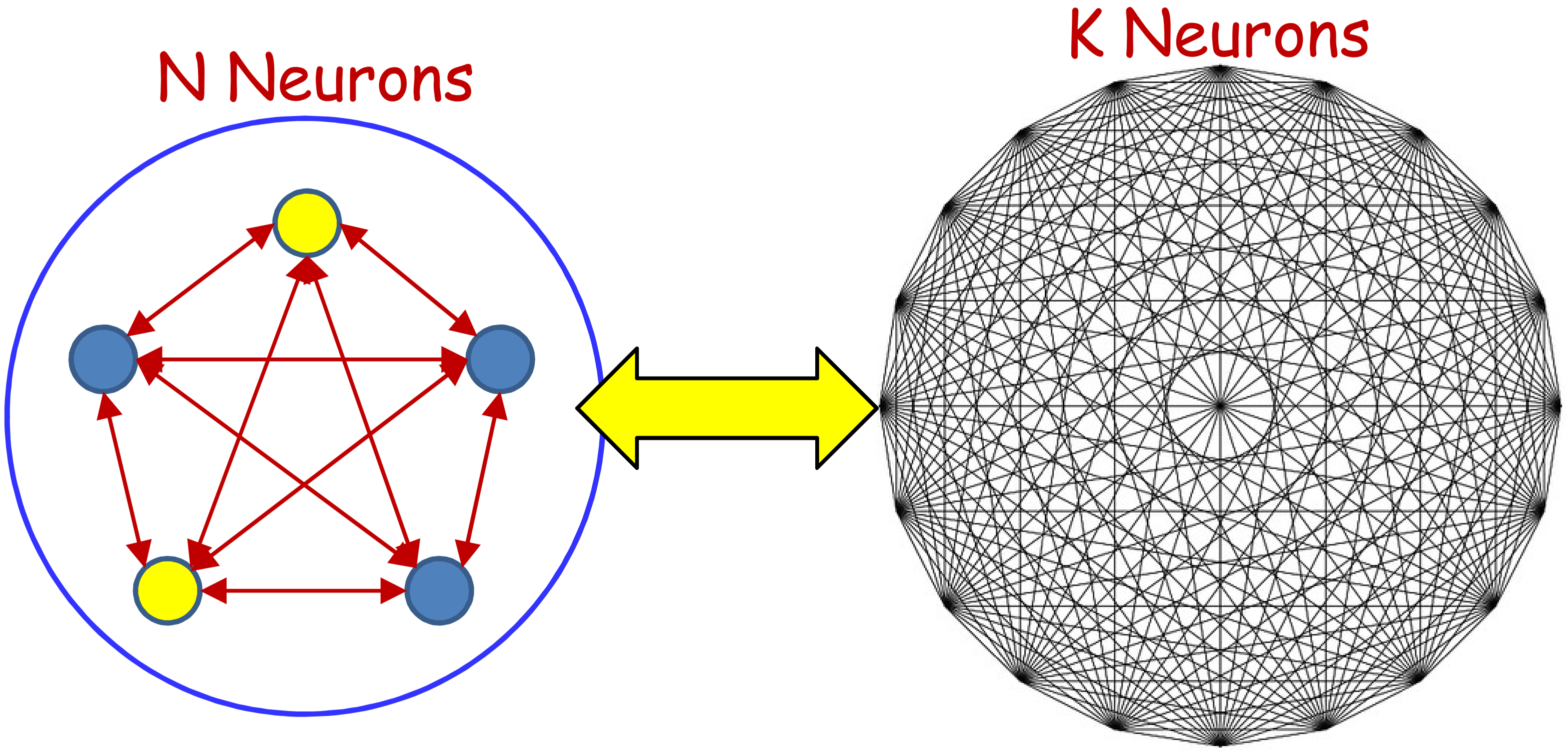
- The network can store up to N N-bit patterns
- How do we increase the capacity

Expanding the network



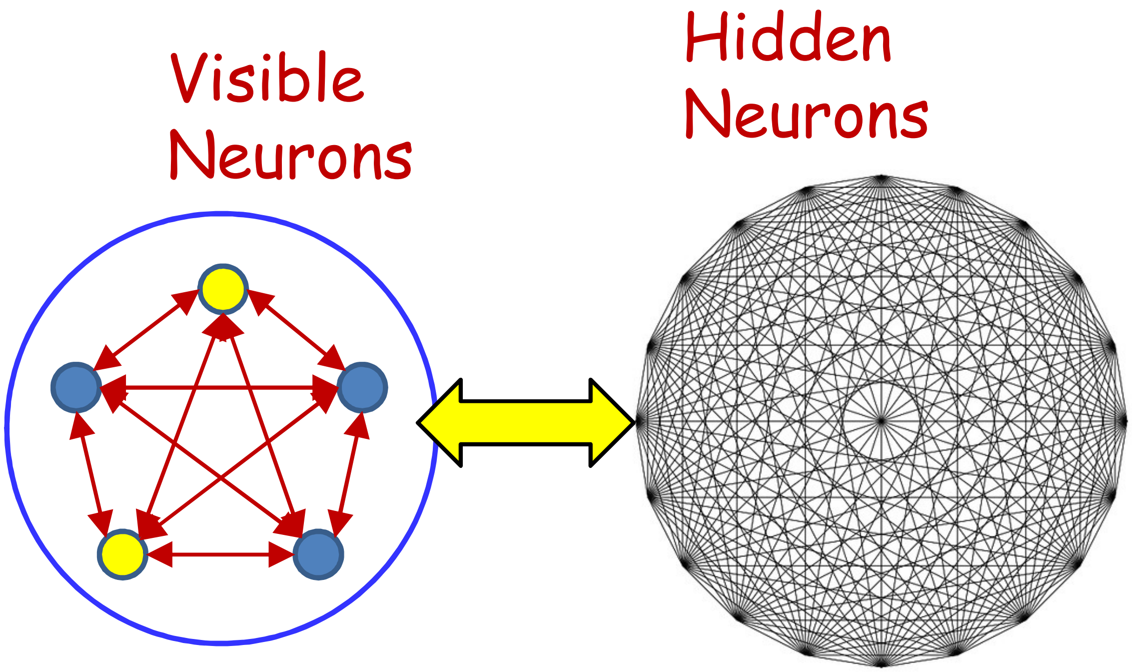
- Add a large number of neurons whose actual values you don't care about!

Expanded Network



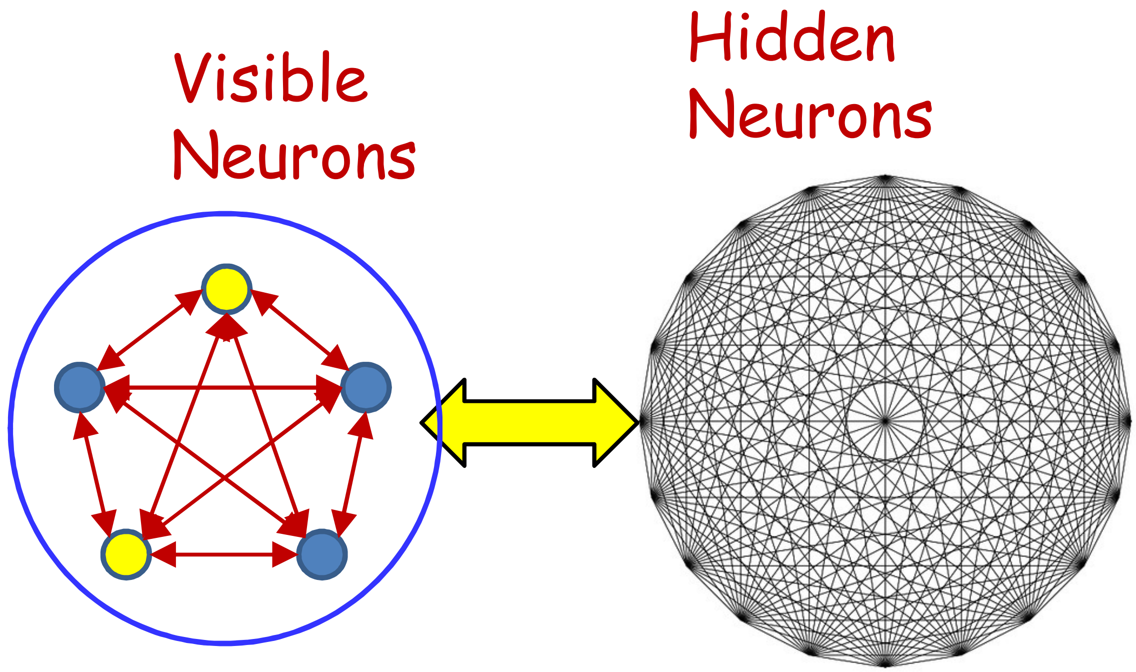
- New capacity: $\sim(N + K)$ patterns
  - Although we only care about the pattern of the first N neurons
  - We're interested in *N-bit* patterns

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

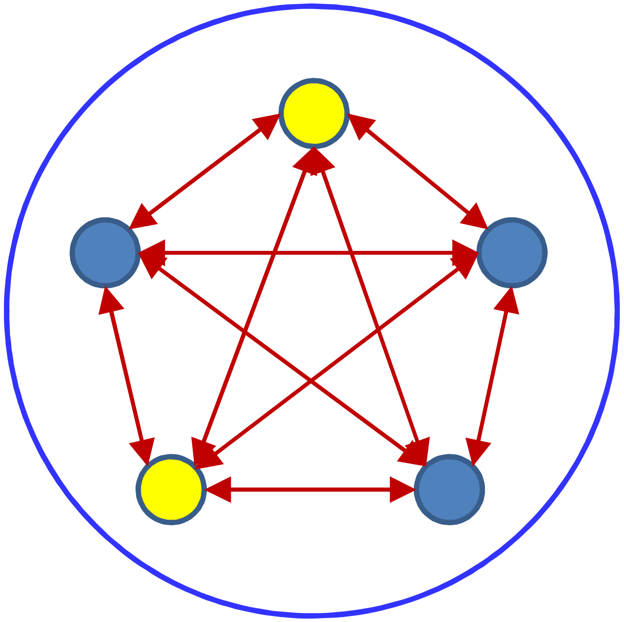


- For a given pattern of *visible* neurons, there are any number of *hidden* patterns (2^K)
- Which of these do we choose?
  - Ideally choose the one that results in the lowest energy
  - But that's an exponential search space!

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

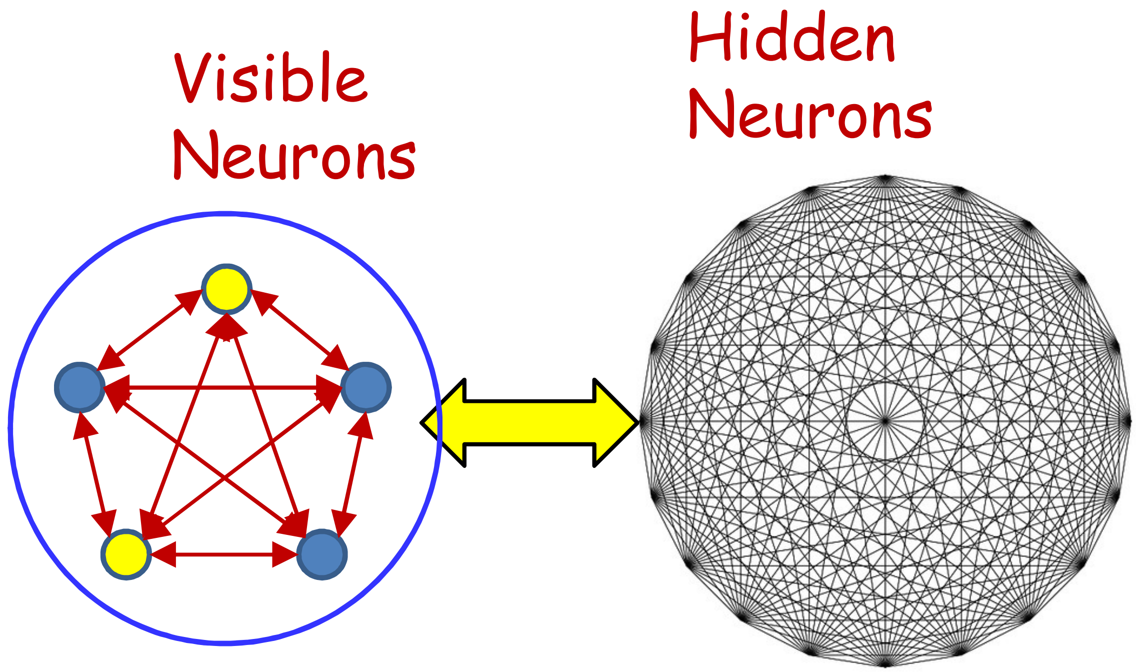


$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{N}\sum_{S} s_i s_j - \frac{1}{M}\sum_{S' \in \mathbf{S}_{simul}} s'_i s'_j$$

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

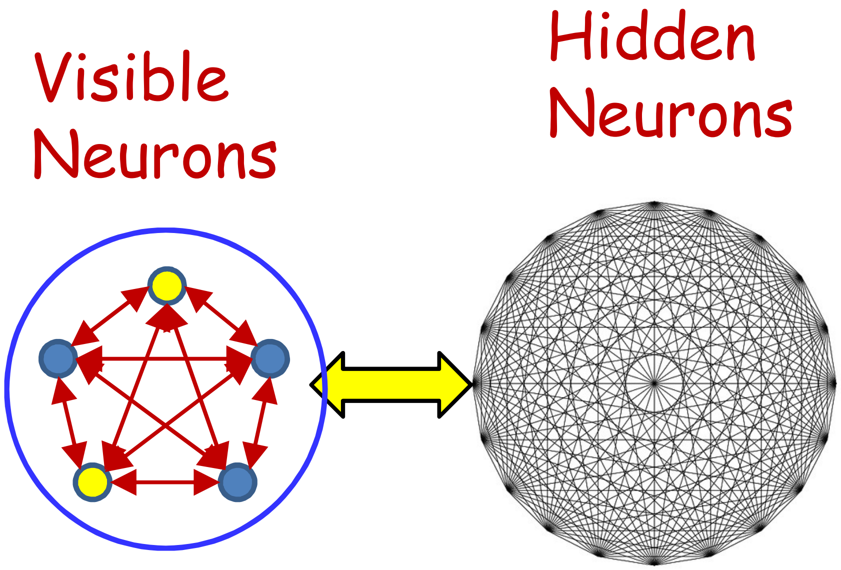
- This basic framework has no hidden units
- Extended to have hidden units

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



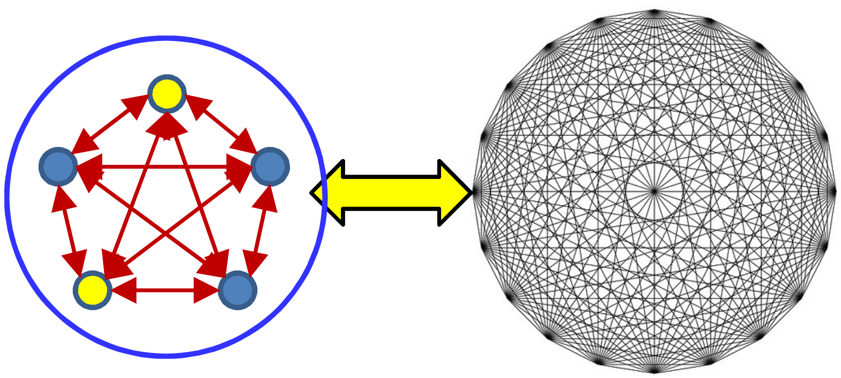
$$P(S) = \frac{exp(-E(S))}{\sum_{S'} exp(-E(S'))}$$

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

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

- 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



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

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

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

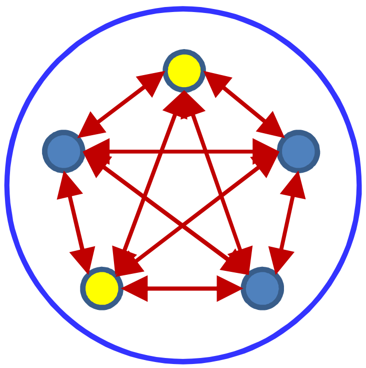
- We are interested in the *marginal* probabilities over *visible* bits
  - We want to learn to represent the visible bits
  - The hidden bits are the "latent" representation learned by the network

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

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

Training the network

Visible Neurons



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

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

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

- 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 $\mathbf{V} = \{V_1, V_2, \ldots, V_N\}$
  - 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

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

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

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

- 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

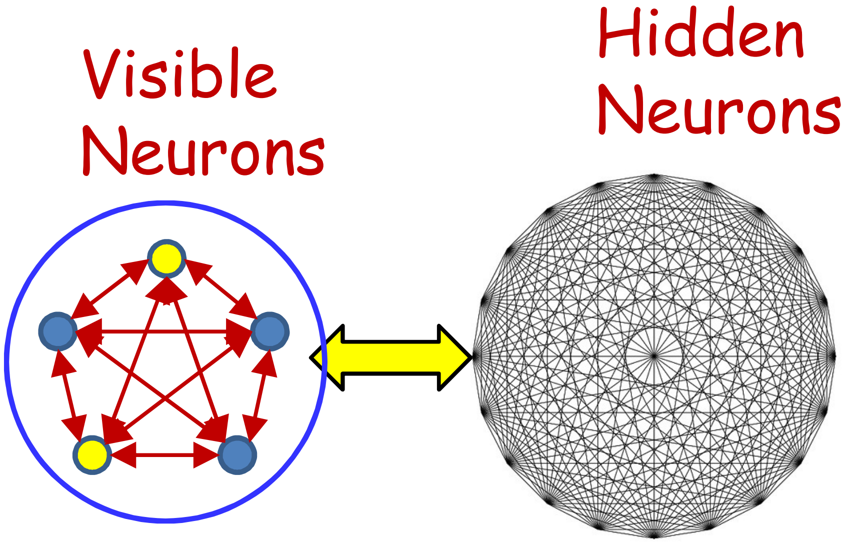
$$\frac{d\mathcal{L}}{dw_{ij}} = \frac{1}{N}\sum_{V\in\mathbf{V}}\sum_{H} P(S|V)s_i s_j - \sum_{S'} P(S')s'_i s'_j$$

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

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

- 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

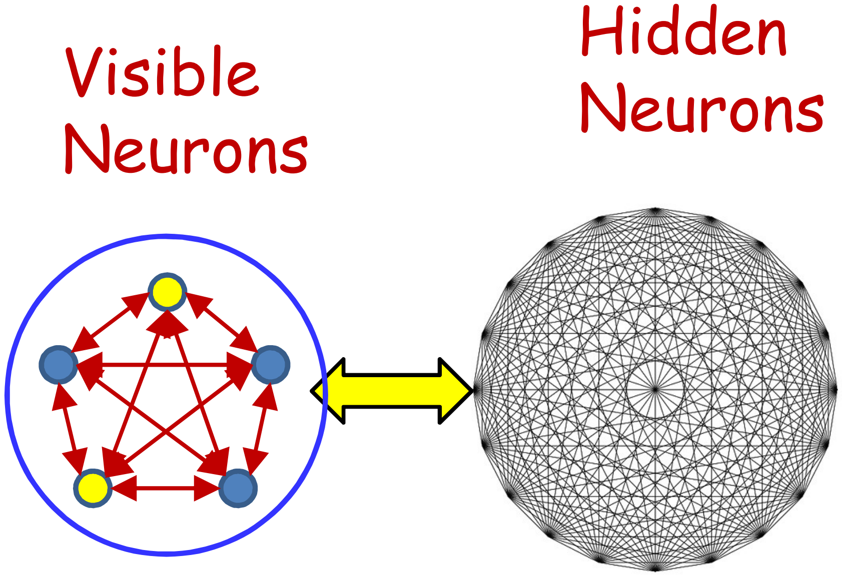


$$P(S) = \frac{exp(-E(S))}{\sum_{S'} exp(-E(S'))}$$

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

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

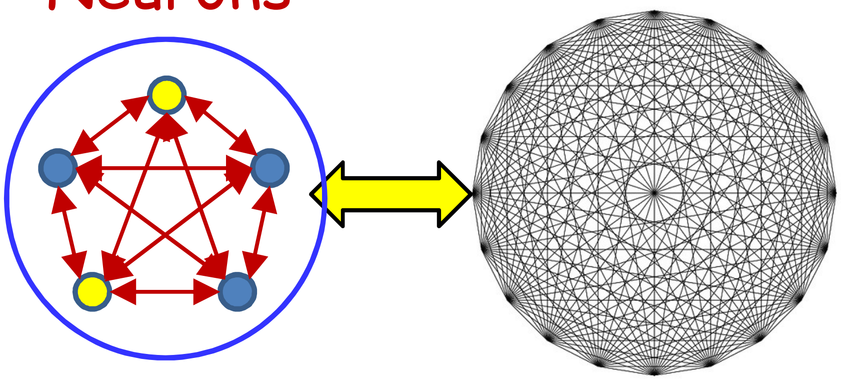
Step 1



- 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

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

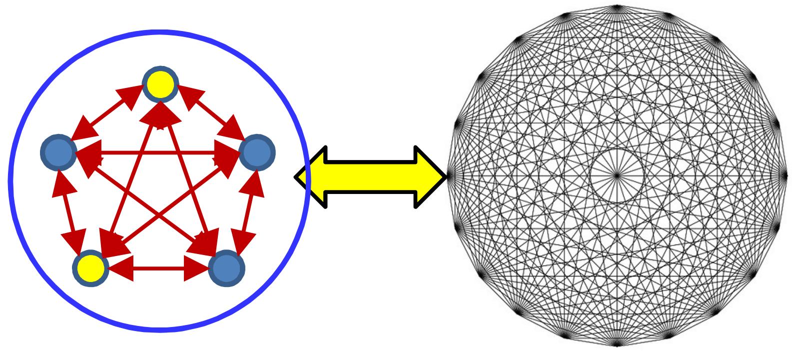
Step 2



- 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

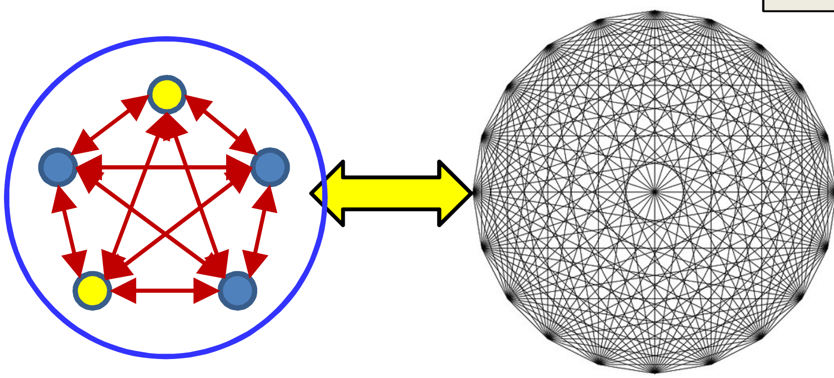


$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{NK}\sum_{\mathbf{S}} s_i s_j - \frac{1}{M}\sum_{S\prime \in \mathbf{S}_{simul}} s_i' s_j'$$

- 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

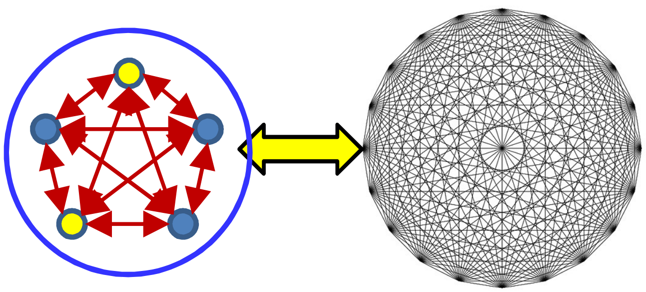
Boltzmann machines: Overall

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

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

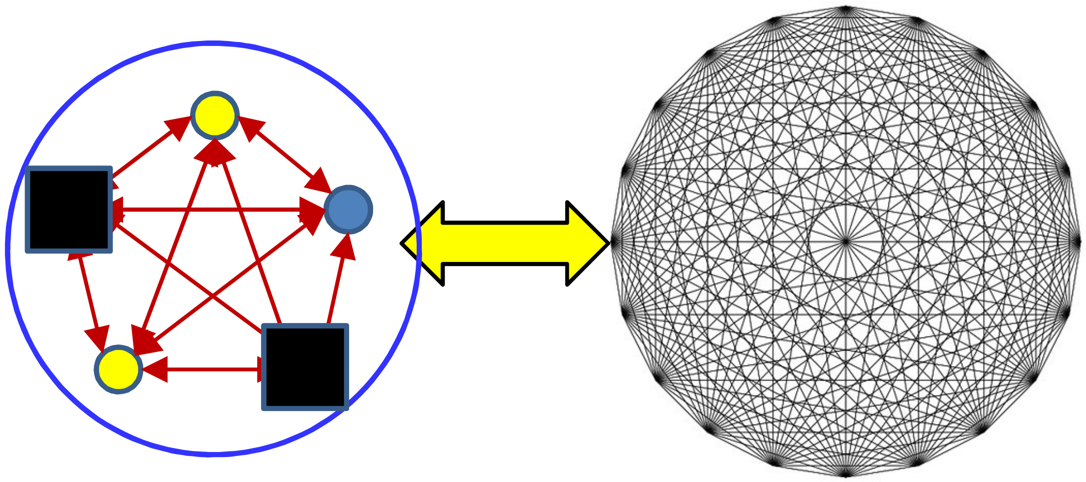
$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{NK} \sum_{\mathbf{S}} s_i s_j - \frac{1}{M} \sum_{S' \in \mathbf{S}_{simul}} s_i' s_j'$$

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



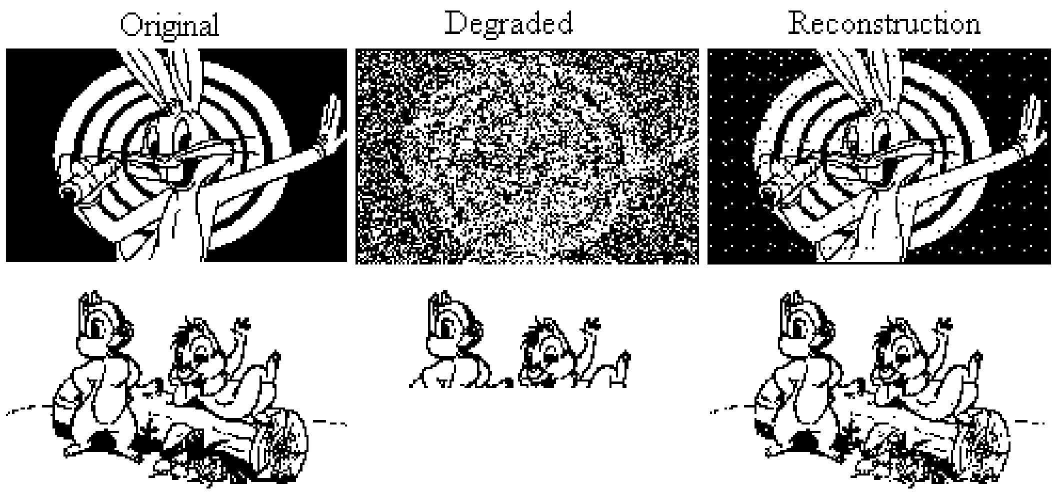
- **Training:** Given a set of training patterns
  - Which could be repeated to represent relative probabilities
- Initialize weights
- Run simulations to get clamped and unclamped training samples
- Compute gradient and update weights
- Iterate

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

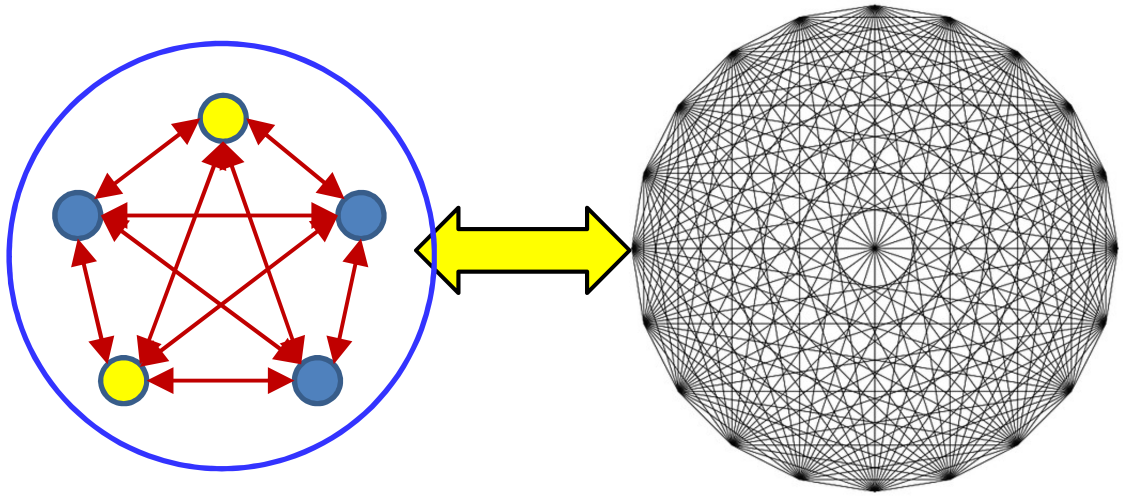
Applications



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

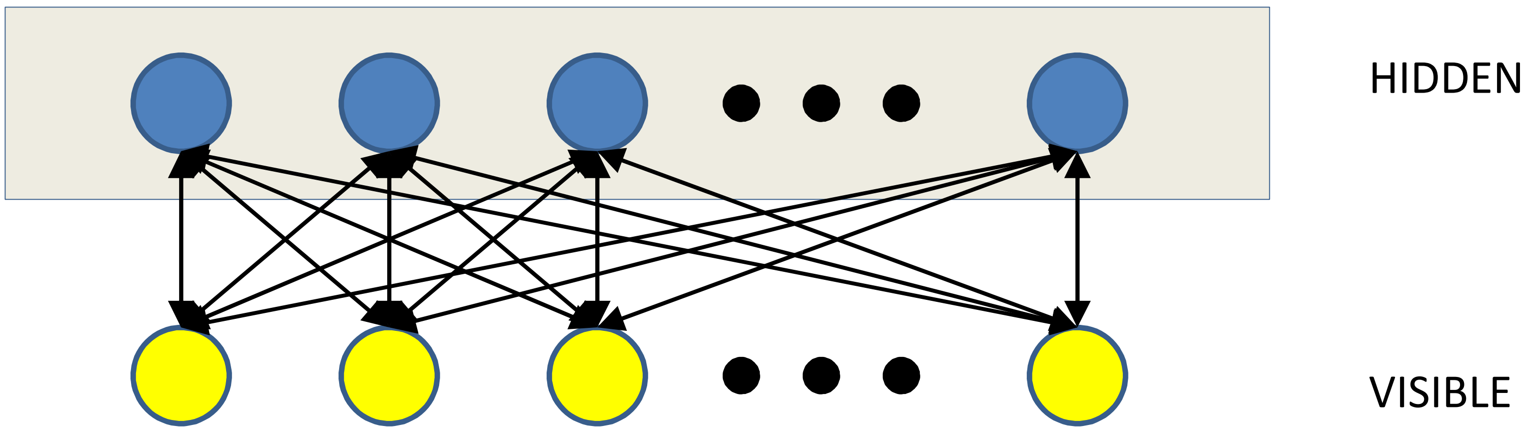


- Training patterns:
  - [f_1, f_2, f_3, …. , class]
  - Features can have binarized or continuous valued representations
  - Classes have "one hot" representation
- Classification:
  - Given features, anchor features, estimate a posteriori probability distribution over classes
    - Or choose most likely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

Boltzmann machines: Issues

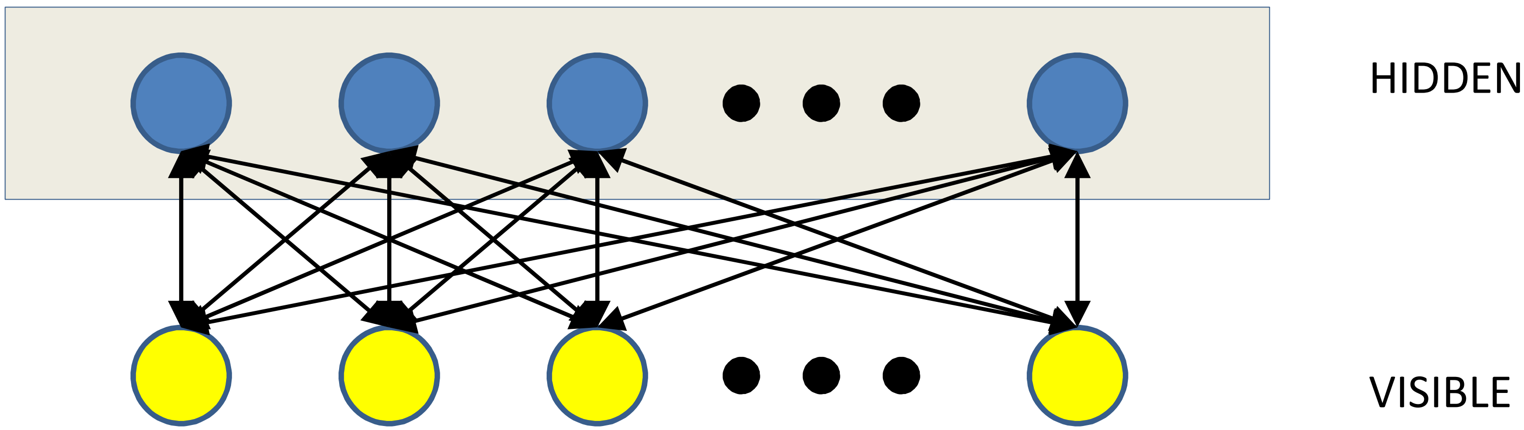
- Training takes for ever
- Doesn’t really work for large problems
  - A small number of training instances over a small number of bits

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

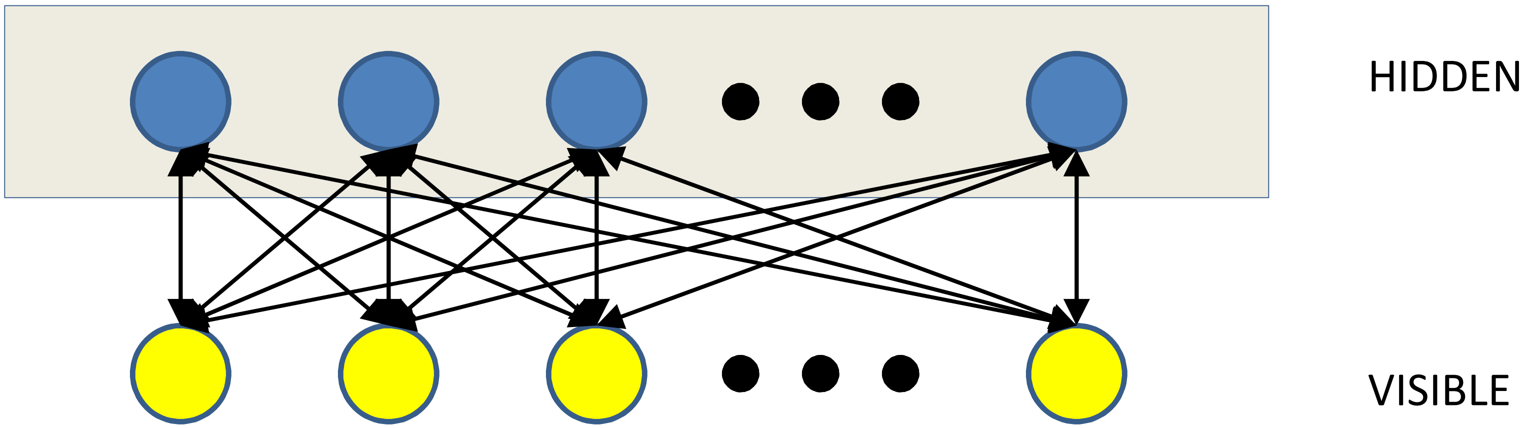


$$z_i = \sum_j w_{ji} s_i + b_i$$

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

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

Solution: *Restricted* Boltzmann Machines



HIDDEN

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

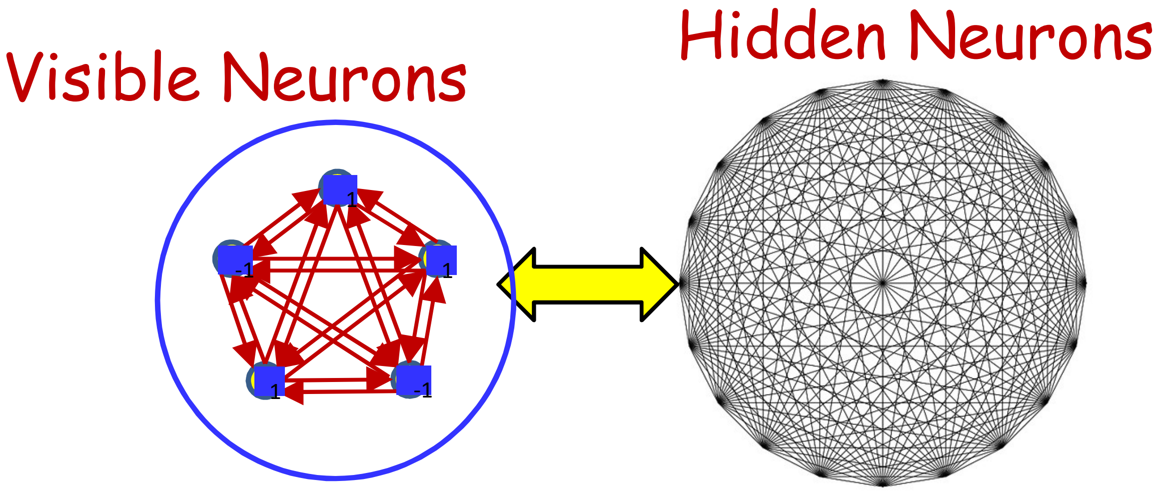
$$P(h_i = 1) = \frac{1}{1 + e^{-z_i}}$$

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

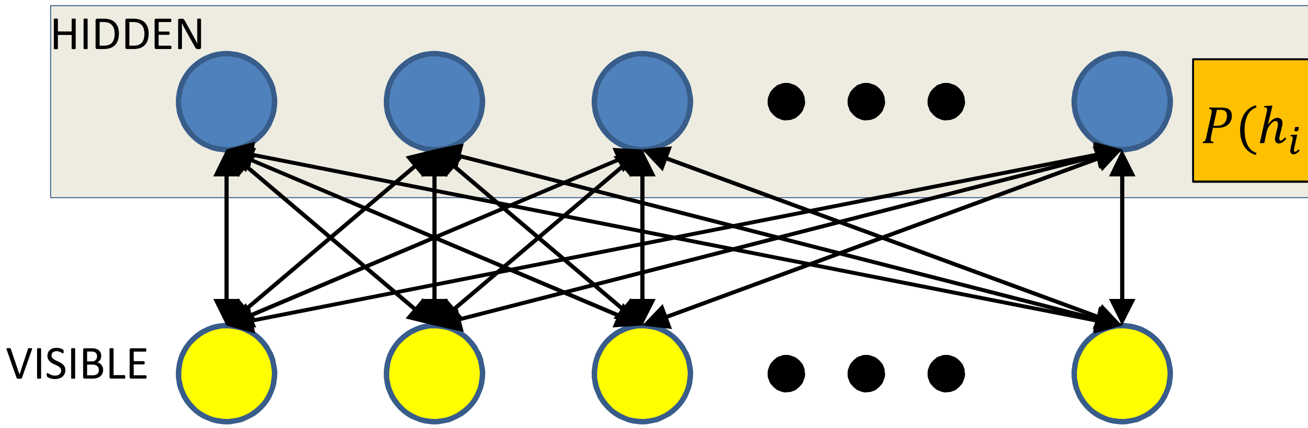


- For each training pattern V_i
  - Fix the visible units to V_i
  - Let the hidden neurons evolve from a random initial point to generate H_i
  - Generate $S_i = [V_i, H_i]$
- Repeat K times to generate synthetic training

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

Sampling: Restricted Boltzmann machine

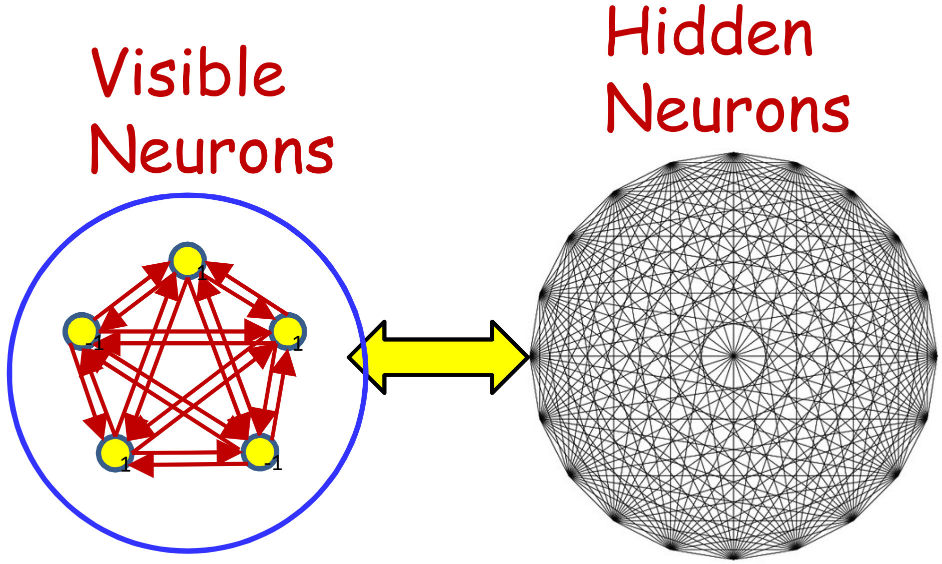
$$z_i = \sum_j w_{ji} v_i + b_i$$



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

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

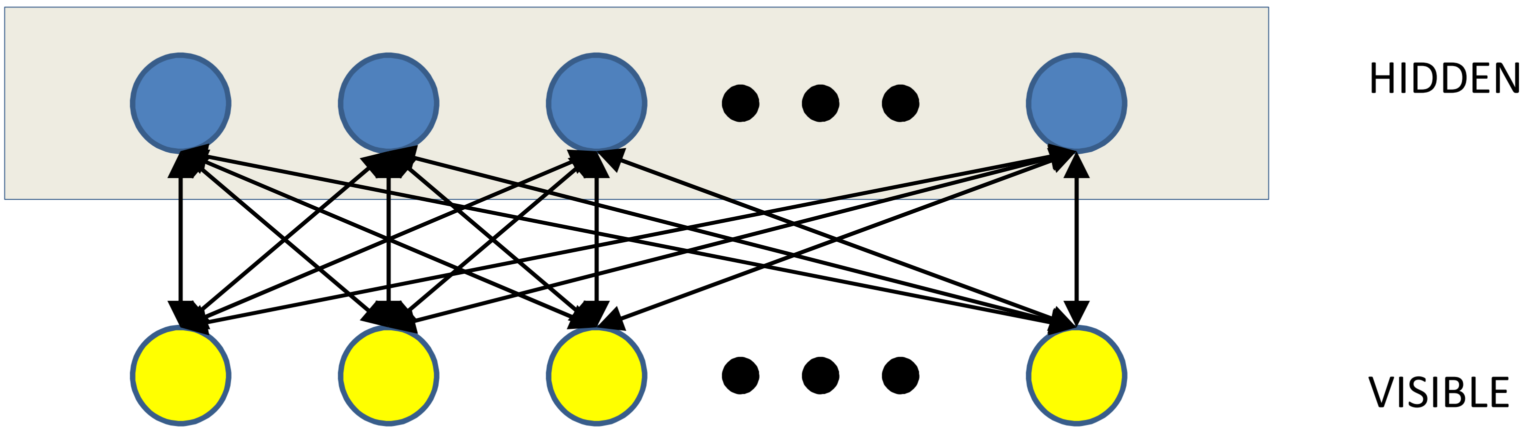
Recap: Training full Boltzmann machines: Step 2



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

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

Sampling: Restricted Boltzmann machine



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

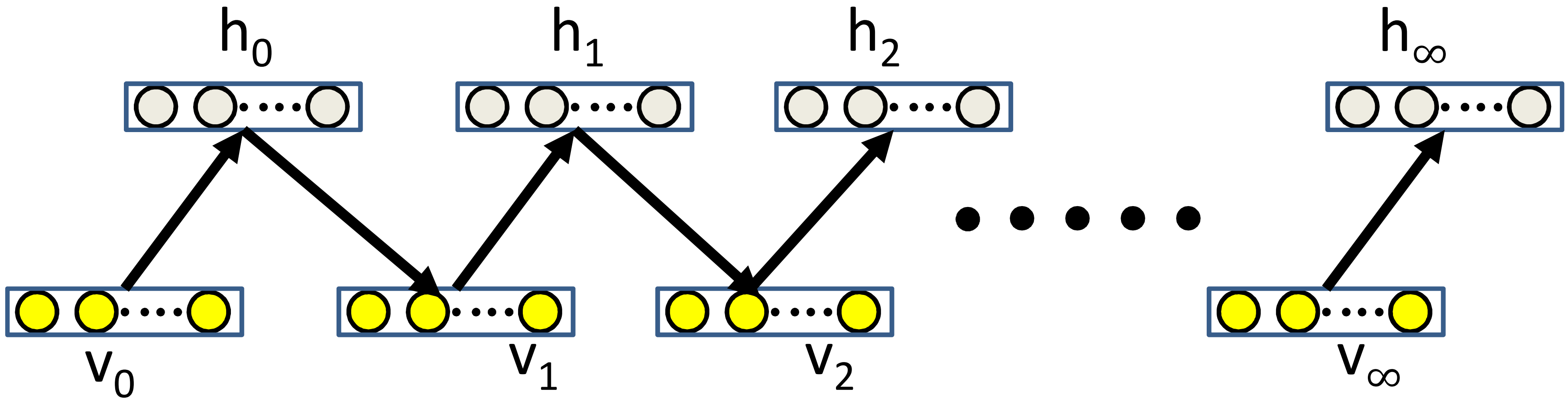
$$P(h_i = 1) = \frac{1}{1 + e^{-z_i}}$$

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

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

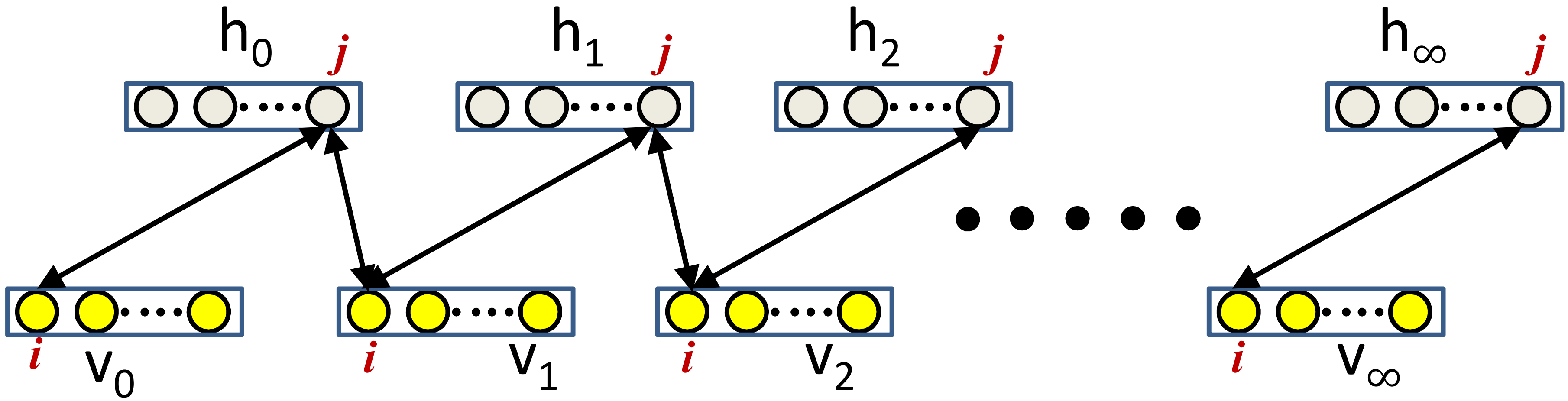
- For each sample:
  - Iteratively sample hidden and visible units for a long time
  - Draw final sample of both hidden and visible units

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



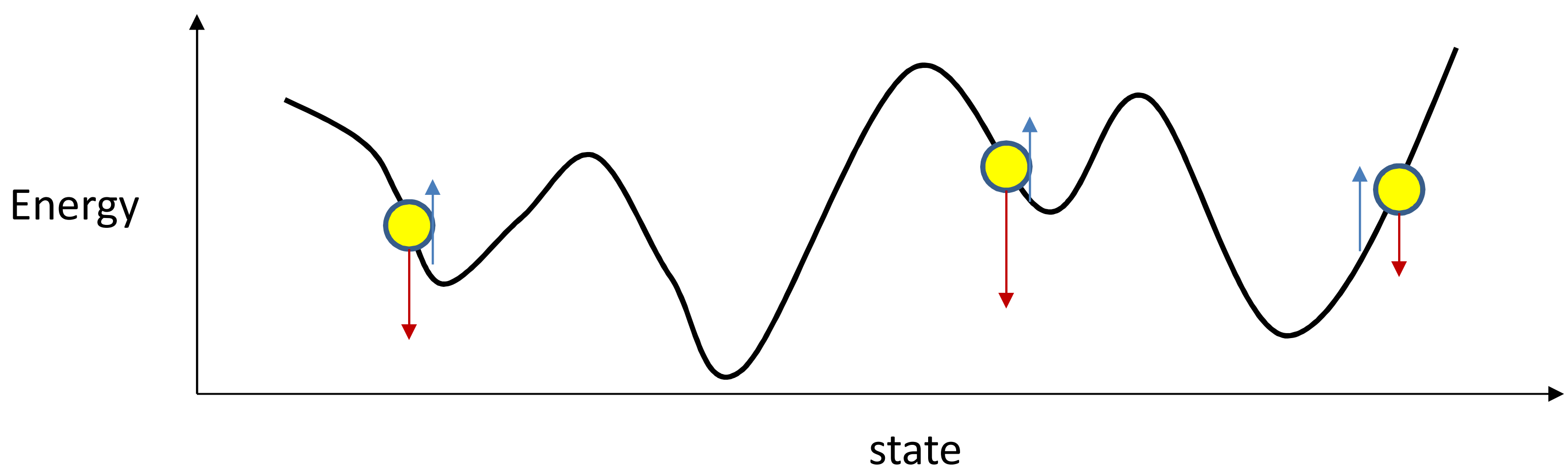
- Gradient (showing only one edge from visible node i to hidden node j)

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

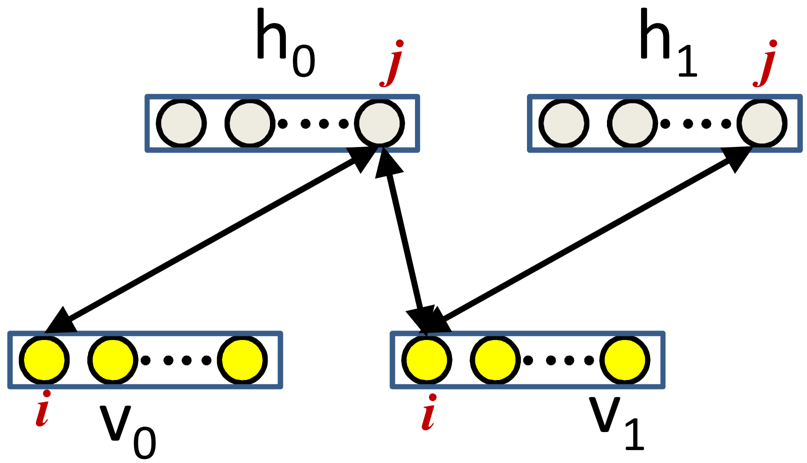
- $\langle v_i, h_j \rangle$ represents average over many generated training samples

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!

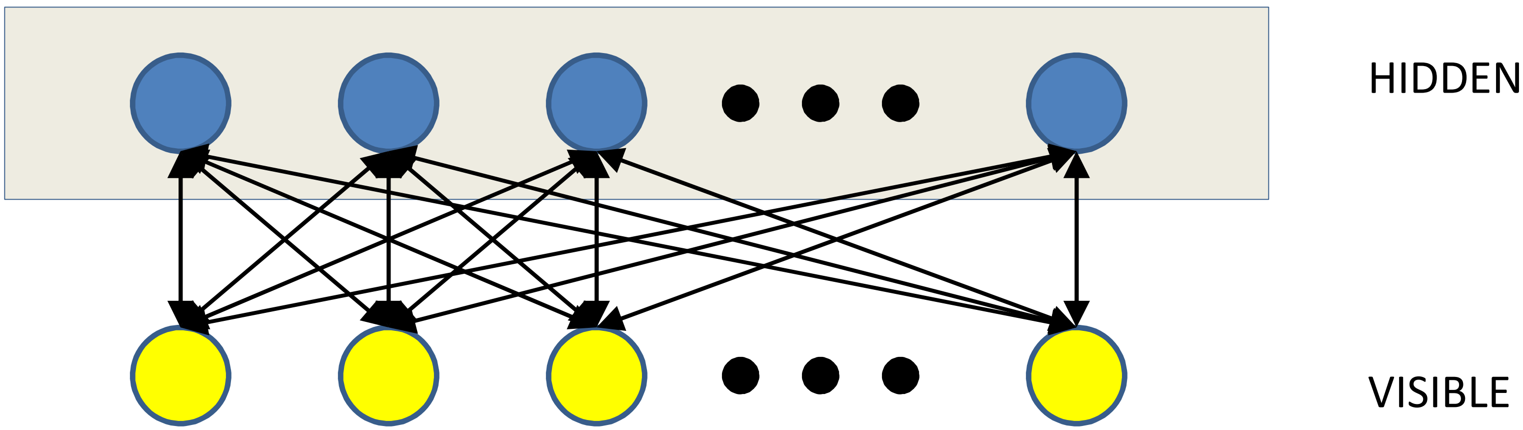
$$\frac{\partial \log p(v)}{\partial w_{ij}} = \langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1$$

- 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

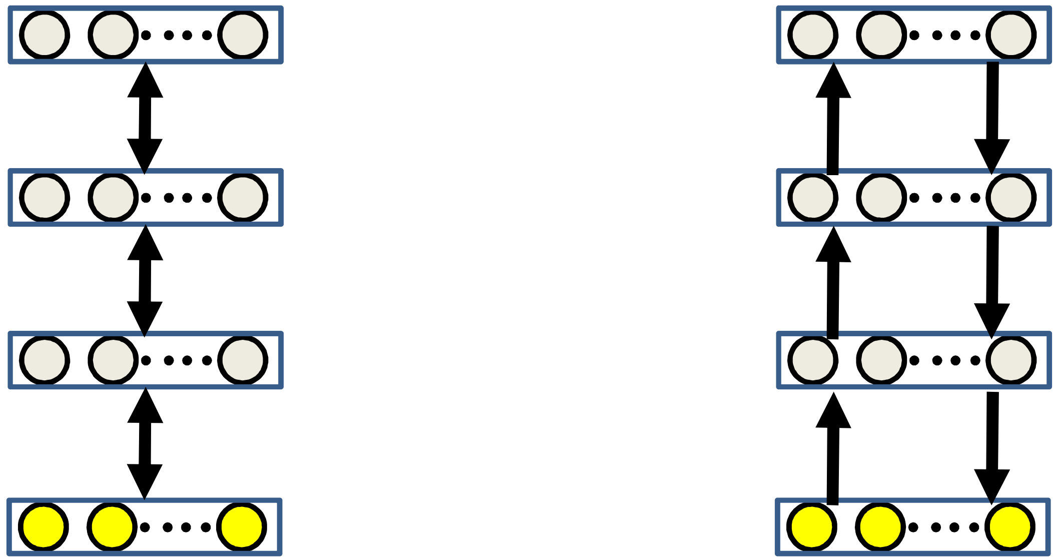


HIDDEN $$z_i = \sum_j w_{ji} v_i + b_i$$ $$P(h_i = 1) = \frac{1}{1 + e^{-z_i}}$$

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
- …