Neural Networks

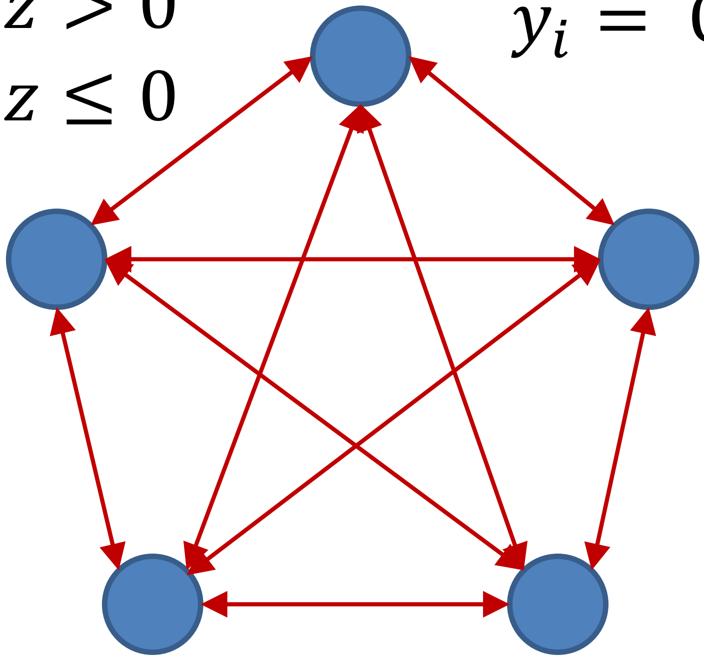
Hopfield Nets and Boltzmann Machines

Fall 2017

Recap: Hopfield network

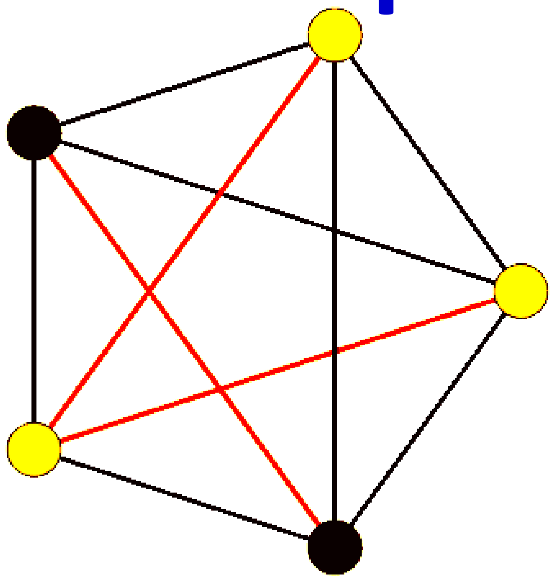
$$\Theta(z) = \begin{cases} +1 \; if \; z > 0 \\ -1 \; if \; z \leq 0 \end{cases}$$

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



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

Recap: Hopfield network

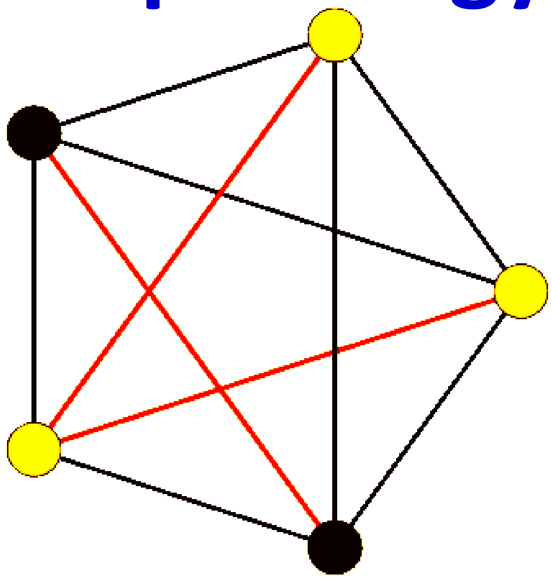


$$y_i = \Theta\left(\sum_{j \neq i} w_{ji} y_j + b_i\right)$$

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

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

Recap: Energy of a Hopfield Network



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

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

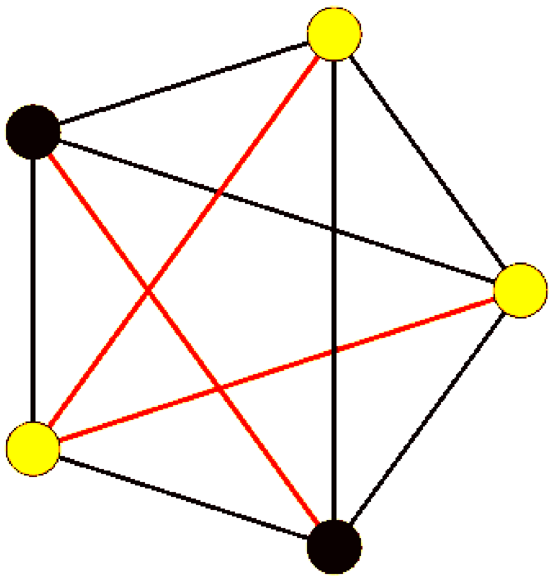
Not assuming node bias

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

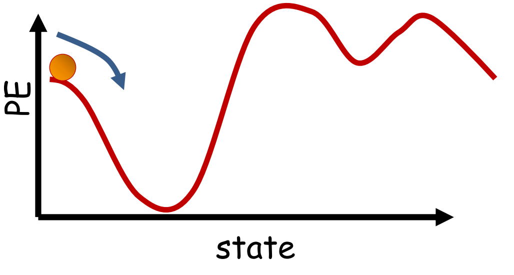
- The system will evolve until the energy hits a local minimum
- In vector form, including a bias term (not used in Hopfield nets)

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

Recap: Evolution

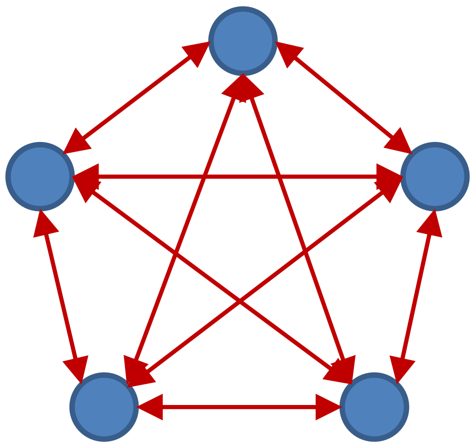


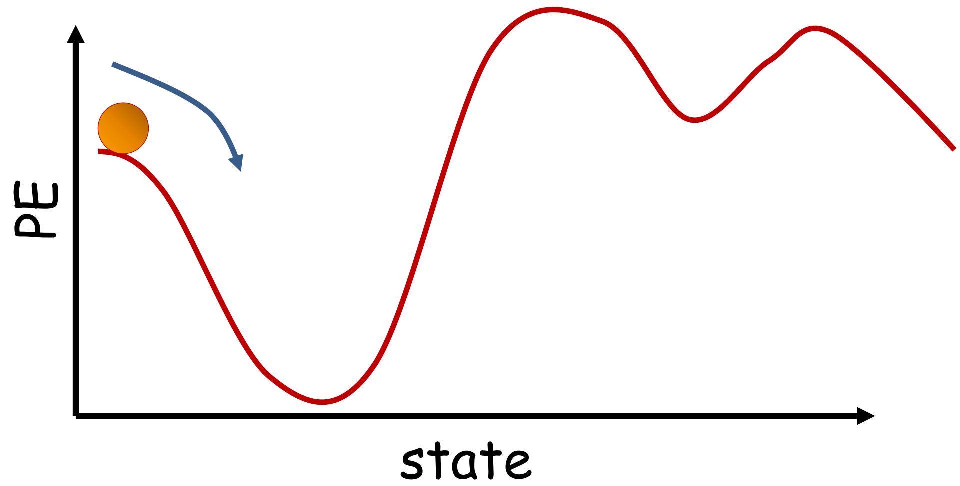
$$E = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}$$



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

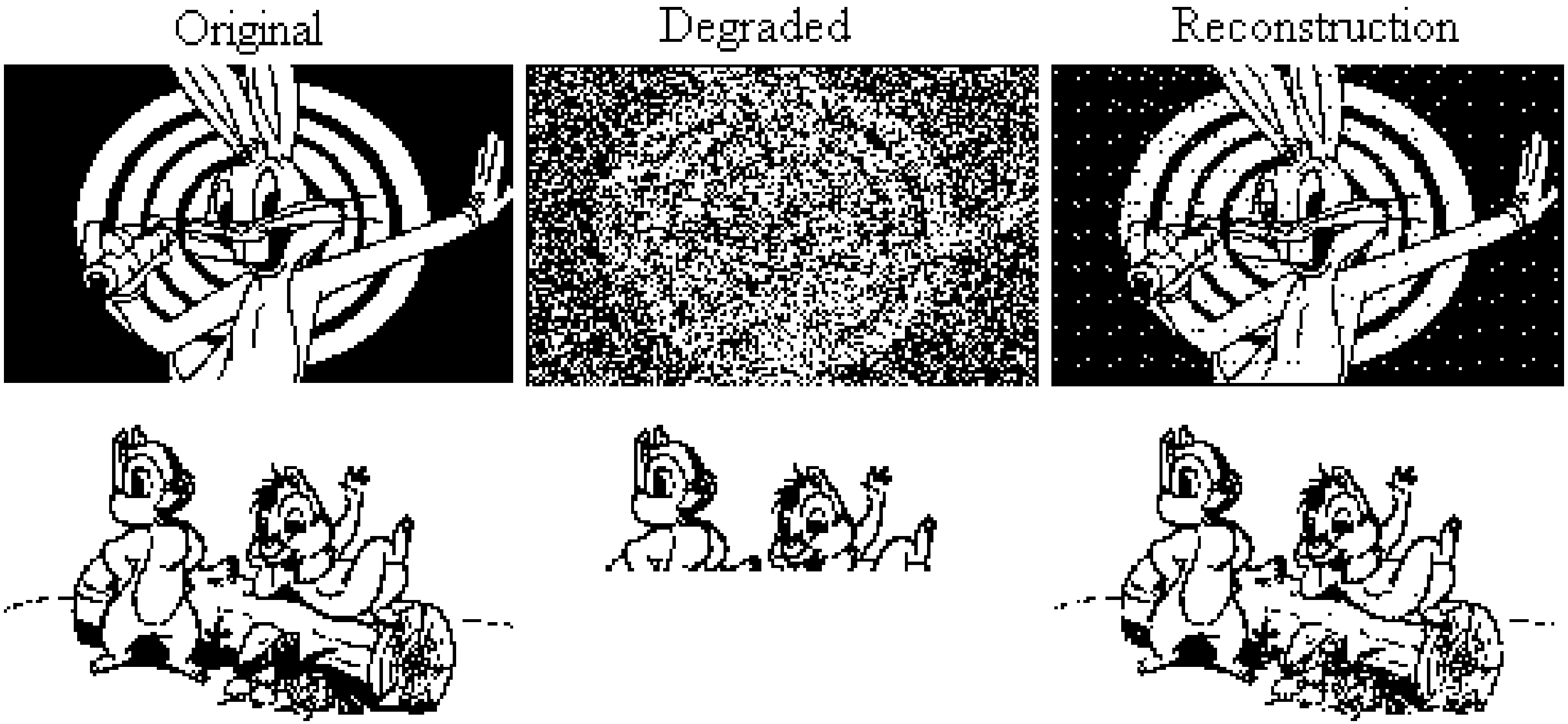
Recap: Content-addressable memory





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

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/

The bottom line

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

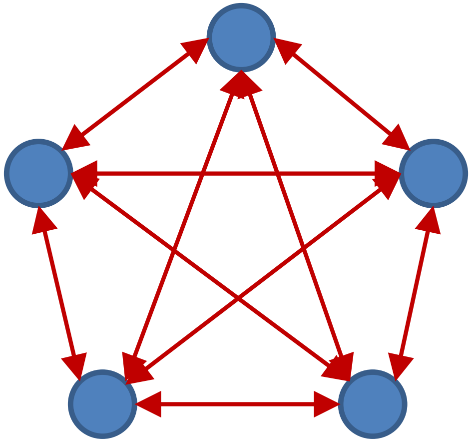
- For a *specific* set of K patterns, we can *always* build a network for which all K patterns are stable provided $K \leq N$
  - Mostafa and St. Jacques 85'
    - For large N, the upper bound on K is actually $N/4logN$
      - McElice et. Al. 87'
  - **But this may come with many "parasitic" memories**

Training the Net

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

- Secondary question
  - How many patterns can we store?

Consider the energy function



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

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

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
  - Which could be repeated to emphasize their importance
- Maximize the total energy of all *non-target* patterns
  - Which too could be repeated to emphasize their importance

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:

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

Various versions of choosing $\mathbf{y} \in \mathbf{Y}_P$ let us assign importance to $\mathbf{y}$

Various versions of choosing $\mathbf{y} \notin \mathbf{Y}_P$ gave us different learning algorithms

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:

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

Weighted average (weights sum to 1.0)
Weights capture importance

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:

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

Weighted average (weights sum to 1.0)
Weights capture importance

THIS LOOKS LIKE AN EXPECTATION!

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:

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

Desideratum: The weights should ideally reflect confusability
Lower-energy patterns (according to the current weights) should be more important to pull "up"

If you want the dependence on energy to be exponential..

A probabilistic interpretation

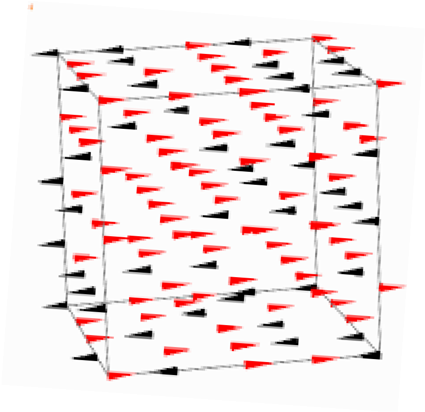
$$E(\mathbf{y}) = \frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} \qquad P(\mathbf{y}) = Cexp\left(-\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y}\right)$$

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

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

The Boltzmann Distribution

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

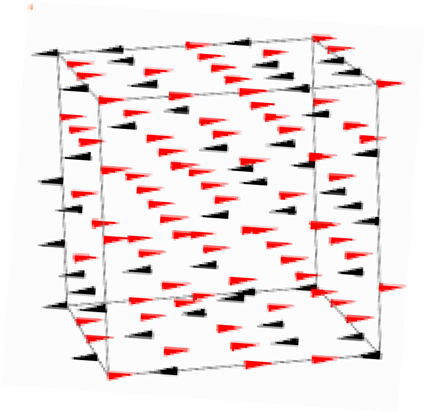


$$C = \frac{1}{\sum_{\mathbf{y}} P(\mathbf{y})}$$

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

Continuing the Boltzmann analogy

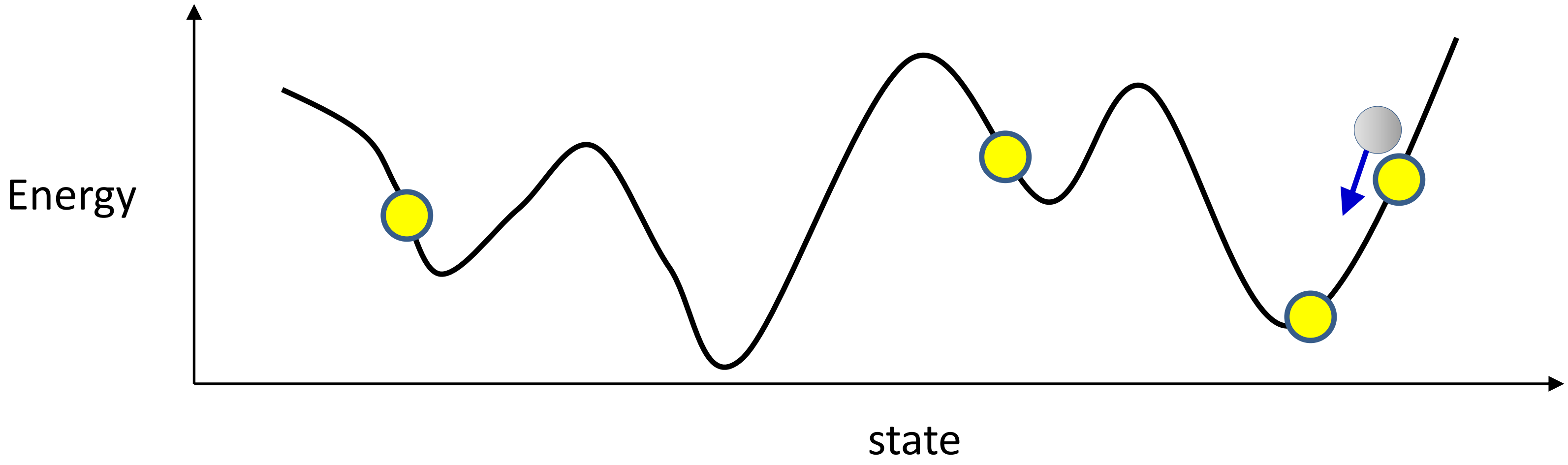
$$E(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} - \mathbf{b}^T\mathbf{y} \qquad P(\mathbf{y}) = Cexp\left(\frac{-E(\mathbf{y})}{kT}\right)$$



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

- At each instant the system *probabilistically* moves to a new state, greatly favoring states with lower energy
  - The lower the T, the more it favors low-energy states
  - With infinitesimally slow cooling, at $T = 0$, it arrives at the global minimal state

Spin glasses and Hopfield nets



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

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:

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

THIS LOOKS LIKE AN EXPECTATION!

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

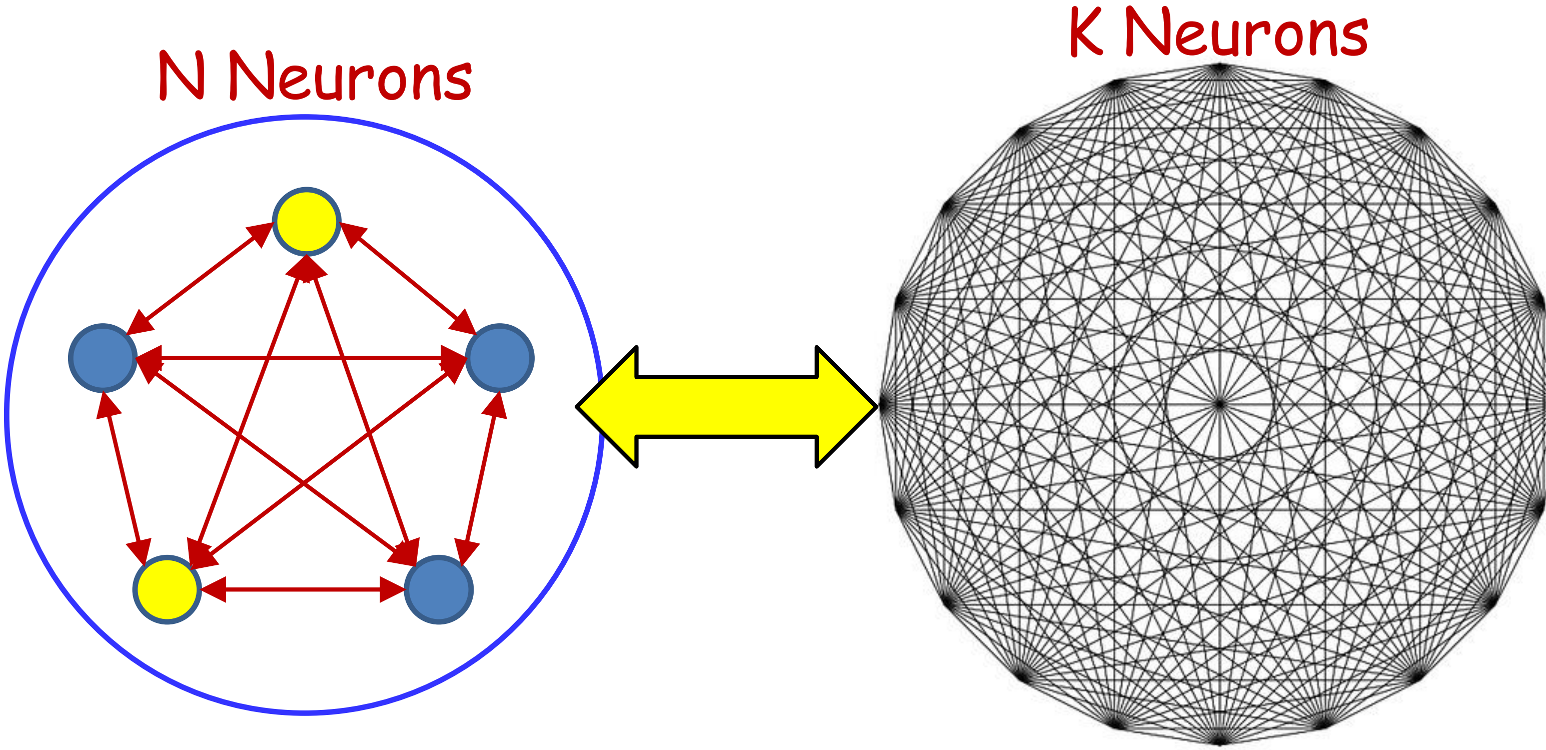
Continuing on..

- Adding capacity to a Hopfield network
  - And the Boltzmann analogy

Storing more than N patterns

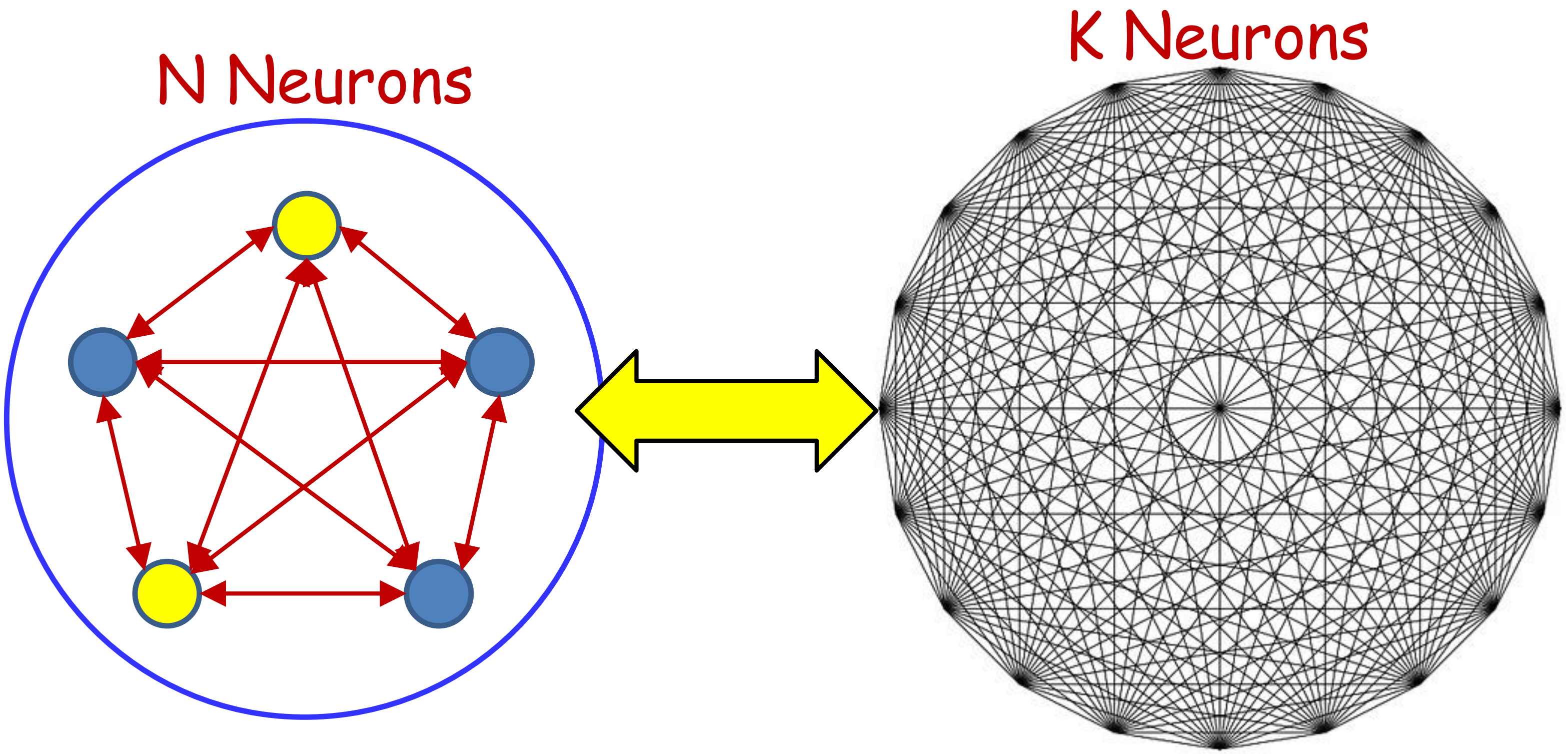
- The memory capacity of an N-bit network is at most N
  - Stable patterns (not necessarily even stationary)
    - Abu Mustafa and St. Jacques, 1985
    - Although “information capacity” is $\mathcal{O}(N^3)$
- How do we increase the capacity of the network
  - Store more patterns

Expanding the network



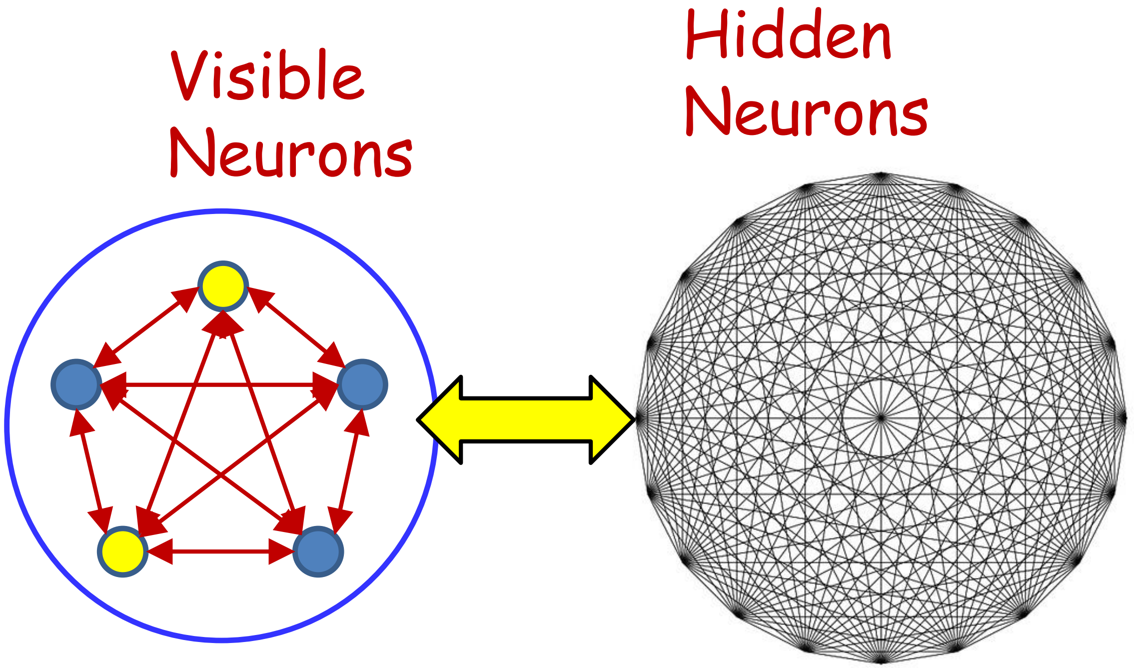
- Add a large number of neurons whose actual values you don't care about!

Expanded Network



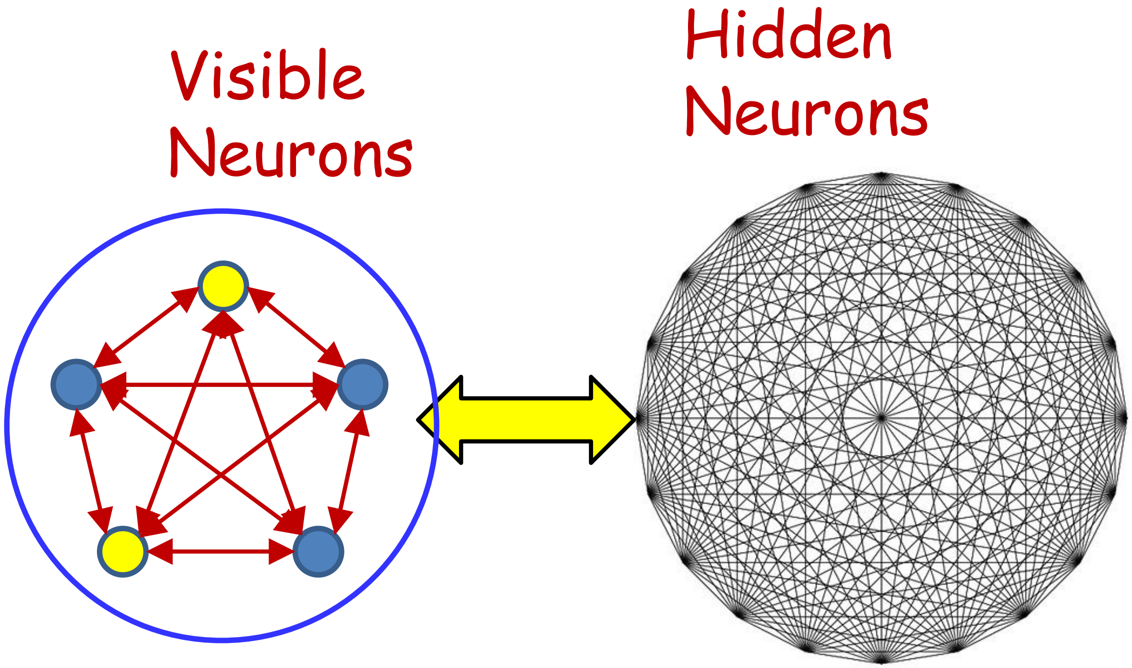
- New capacity: $\sim(N + K)$ patterns
  - Although we only care about the pattern of the first N neurons
  - We're interested in *N-bit* patterns

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

Training the network

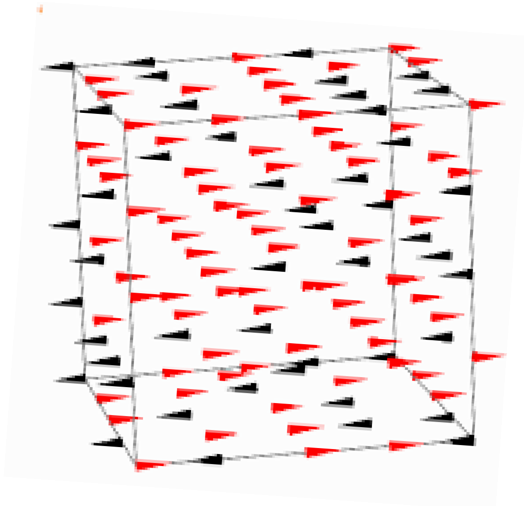


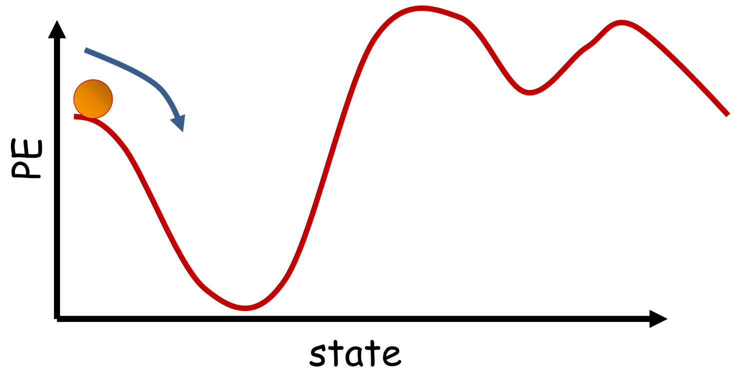
- For a given pattern of *visible* neurons, there are any number of *hidden* patterns (2^K)
- Which of these do we choose?
  - Ideally choose the one that results in the lowest energy
  - But that's an exponential search space!
    - Solution: Combinatorial optimization
      - Simulated annealing

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?

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 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

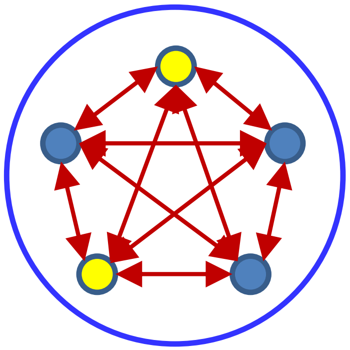
$$F_T = \sum_s P_T(s)\, E_s - kT \sum_s P_T(s) \log P_T(s)$$

- 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.

The Energy of the Network



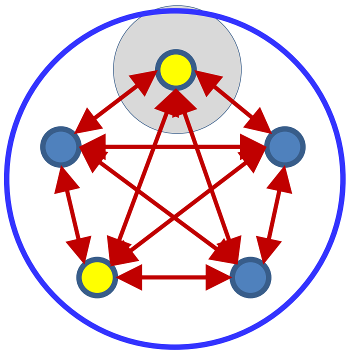
$$E(S) = -\sum_{i<j} w_{ij} s_i s_j - b_i s_i$$

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

- We can define the energy of the system as before
- Since each neuron are stochastic, there is disorder or entropy (with T = 1)
- 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*

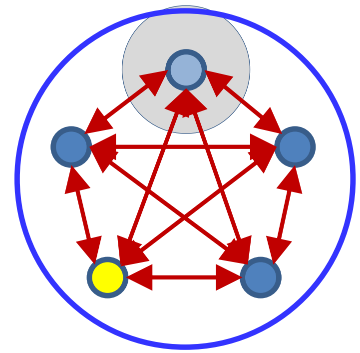
The field at a single node

- 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



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

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

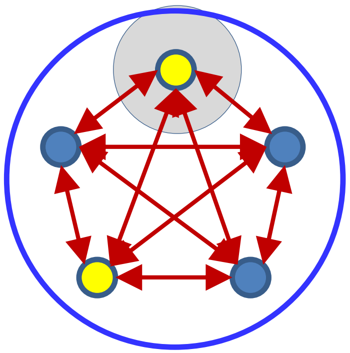


$$logP(S) - logP(S') = logP\left(s_i = 1 \middle| s_{j\neq i}\right) - logP\left(s_i = 0 \middle| s_{j\neq i}\right)$$

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

The field at a single node

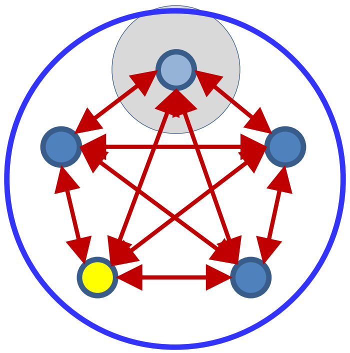
- Let S and S' be the states with the ith bit in the $+1$ and -1 states



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

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

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



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

The field at a single node

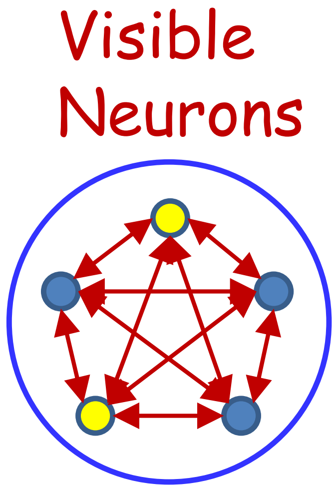
$$log\left(\frac{P\left(s_i = 1 \middle| s_{j\neq i}\right)}{1 - P\left(s_i = 1 \middle| s_{j\neq i}\right)}\right) = \sum_{j\neq i} w_j s_j + b_i$$

- Giving us

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

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

Redefining the network

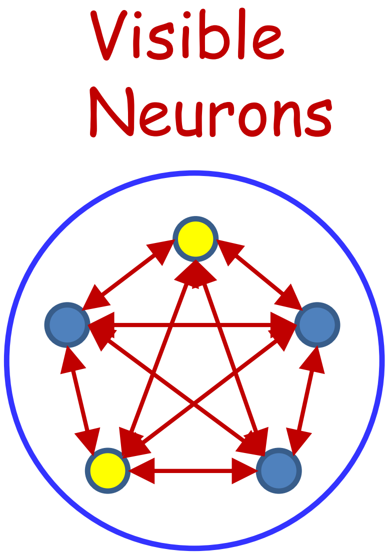


$$z_i = \sum_j w_{ji} s_j + b_i$$

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

- 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

Running the network

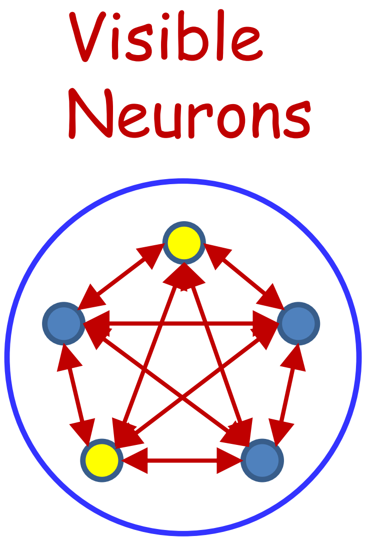


$$z_i = \sum_j w_{ji} s_j + b_i$$

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

- 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

Training the network



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

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

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

- As in Hopfield nets, in order to train the network, we need to select weights such that those states are more probable than other states
  - Maximize the likelihood of the “stored” states

Maximum Likelihood Training

$$\log(P(S)) = \left(\sum_{i<j} w_{ij}s_is_j + b_is_i\right) - \log\left(\sum_{S\prime} exp\left(\sum_{i<j} w_{ij}s'_is'_j + b_is'_i\right)\right)$$

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

$$= \frac{1}{N}\sum_{S}\left(\sum_{i<j} w_{ij}s_is_j + b_is_i(S)\right) - \log\left(\sum_{S\prime} exp\left(\sum_{i<j} w_{ij}s'_is'_j + b_is'_i\right)\right)$$

- Maximize the average log likelihood of all "training" vectors $\mathbf{S} = \{S_1, S_2, \dots, SN\}$
  - 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

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

$$\frac{d\langle \log(P(\mathbf{S})) \rangle}{dw_{ij}} = \frac{1}{N} \sum_{S} s_i s_j - ???$$

- We will use gradient descent, 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!

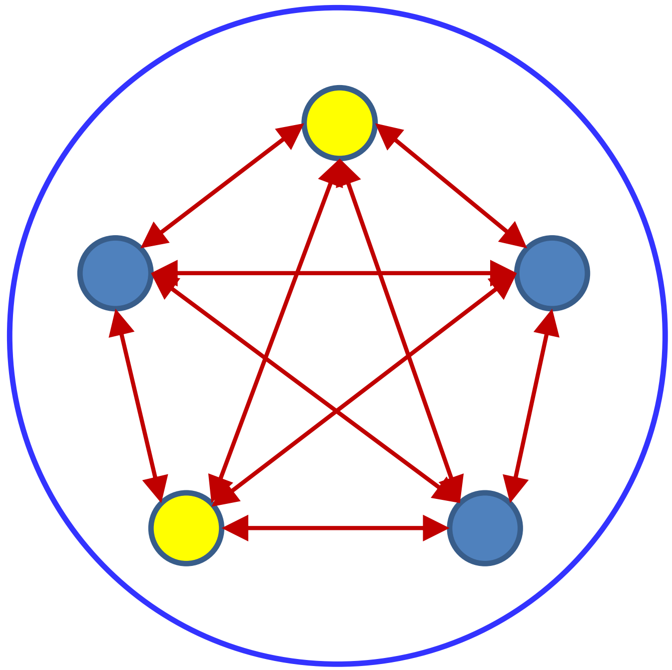
The second term

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

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

- 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!

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

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

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

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

Maximum Likelihood Training

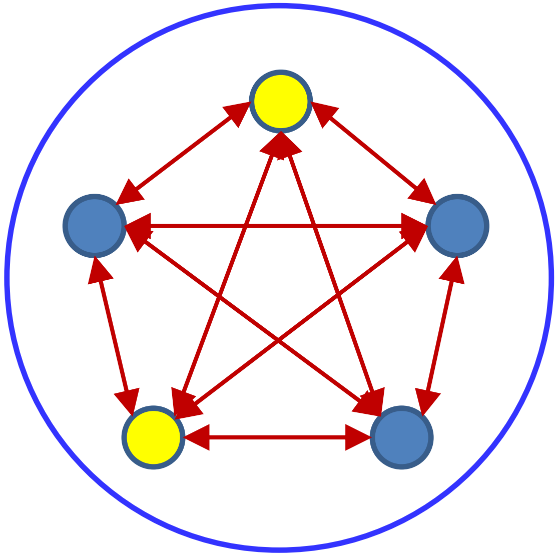
$$\langle \log(P(\mathbf{S}))\rangle = \frac{1}{N}\sum_{S}\left(\sum_{i<j} w_{ij}s_i s_j + b_i s_i(S)\right) - \log\left(\sum_{S'} exp\left(\sum_{i<j} w_{ij}s'_i s'_j + b_i s'_i\right)\right)$$

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

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

- The overall gradient ascent rule

Overall Training

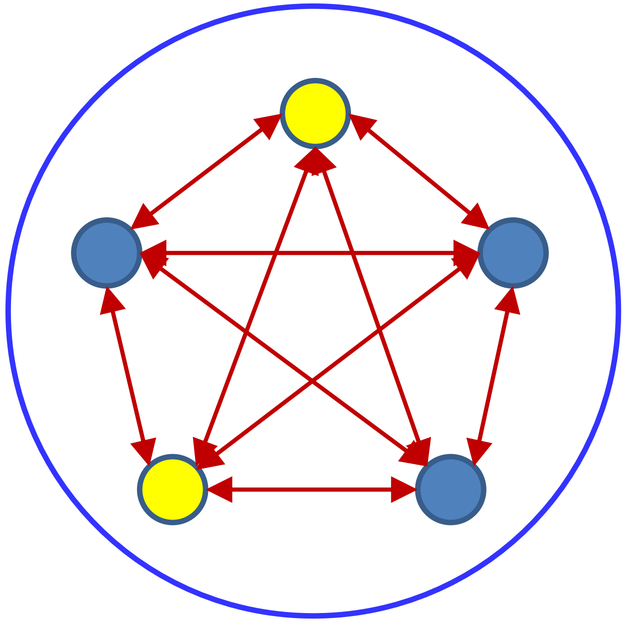


$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{N}\sum_{S} s_i s_j - \frac{1}{M}\sum_{S' \in \mathbf{S}_{simul}} s'_i s'_j$$

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

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

But this is missing hidden nodes

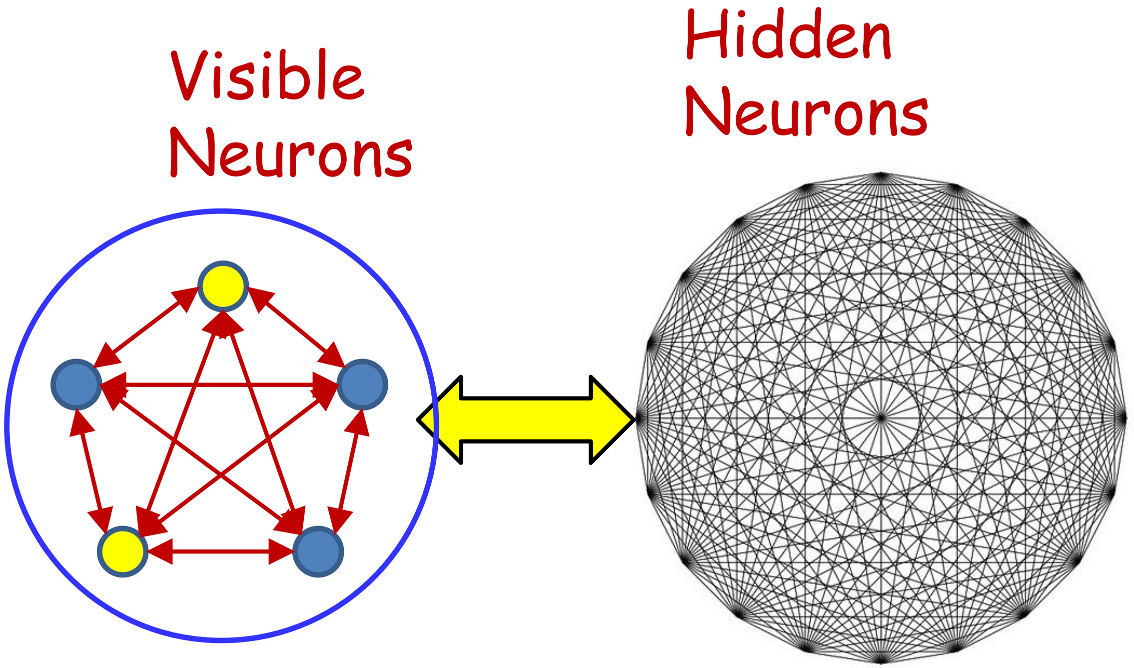


$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{N}\sum_{S} s_i s_j - \frac{1}{M}\sum_{S\prime \in \mathbf{S}_{simul}} s_i' s_j'$$

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

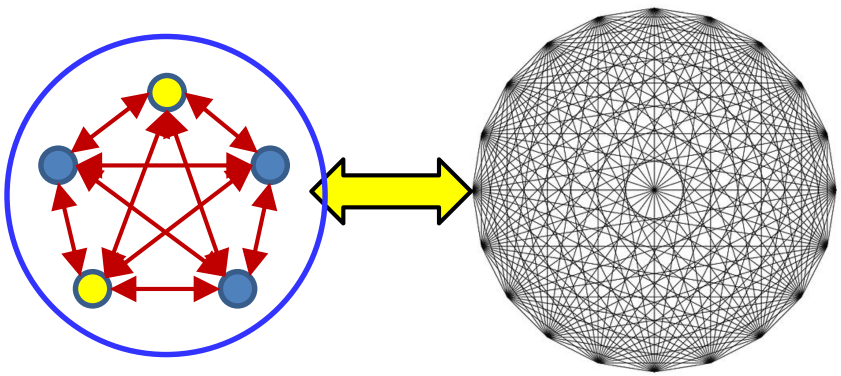
- This framework only works for networks with only visible nodes
- We wanted *hidden* nodes
- How do we extend the paradigm?

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

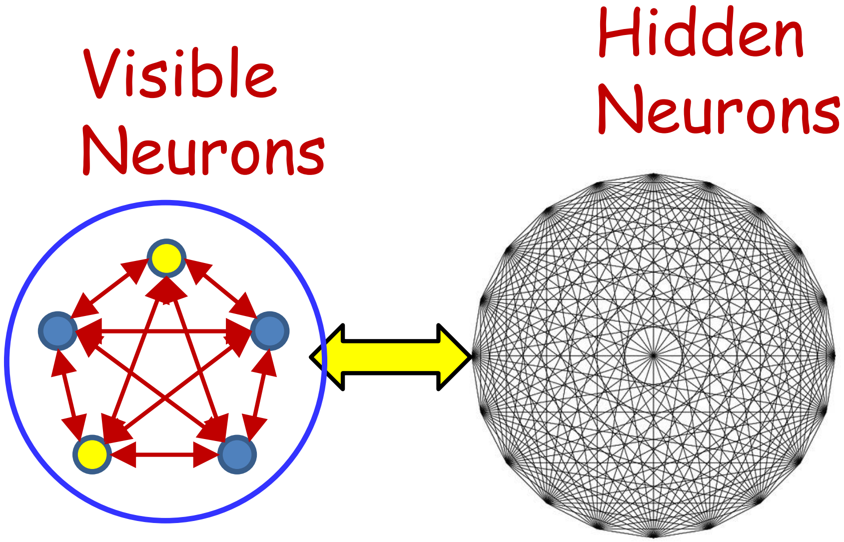


$$P(S) = \frac{exp(-E(S))}{\sum_{S'} exp(-E(S'))}$$

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

- We will now only maximize *marginal* probabilities over visible bits
- $S = (V, H)$
  - V = visible bits
  - H = hidden bits

More simulations

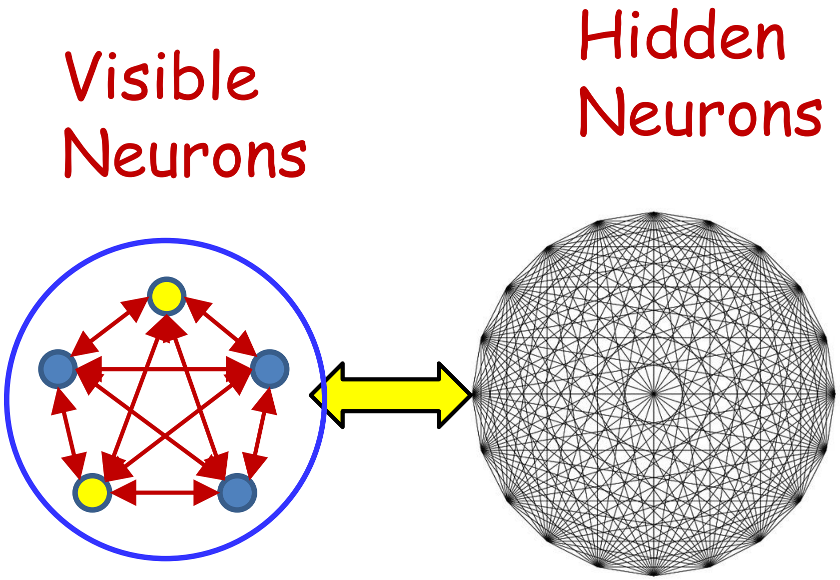


$$P(S) = \frac{exp(-E(S))}{\sum_{S'} exp(-E(S'))}$$

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

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

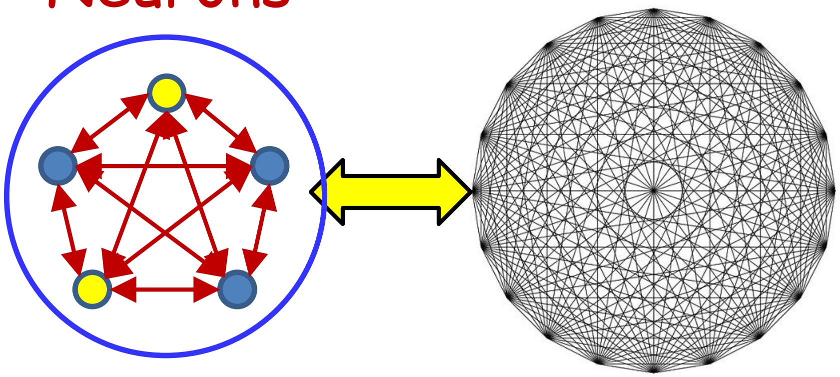
Step 1



- 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

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

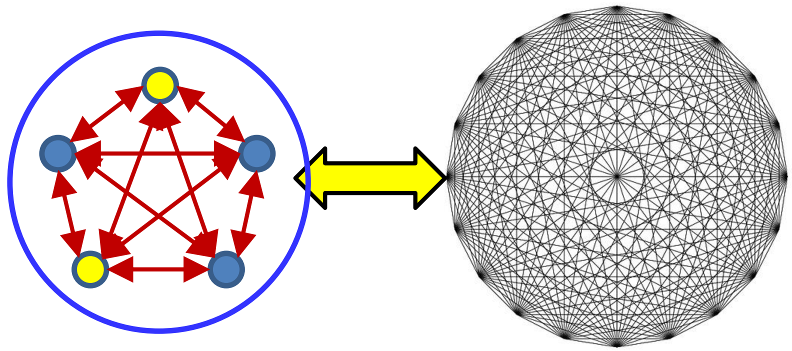
Step 2



- 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

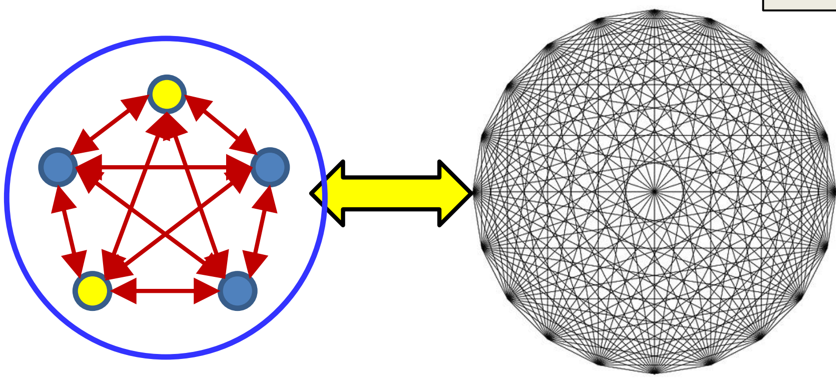


$$\frac{d\left\langle \log\left(P(\mathbf{S})\right)\right\rangle}{dw_{ij}} = \frac{1}{NK}\sum_{\boldsymbol{S}} s_i s_j - \frac{1}{M}\sum_{S' \in \mathbf{S}_{simul}} s_i' s_j'$$

- 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

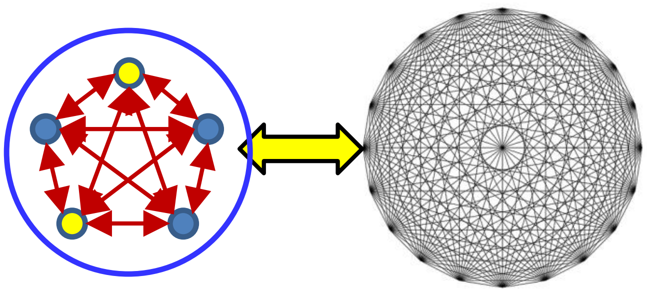
Boltzmann machines: Overall

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

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

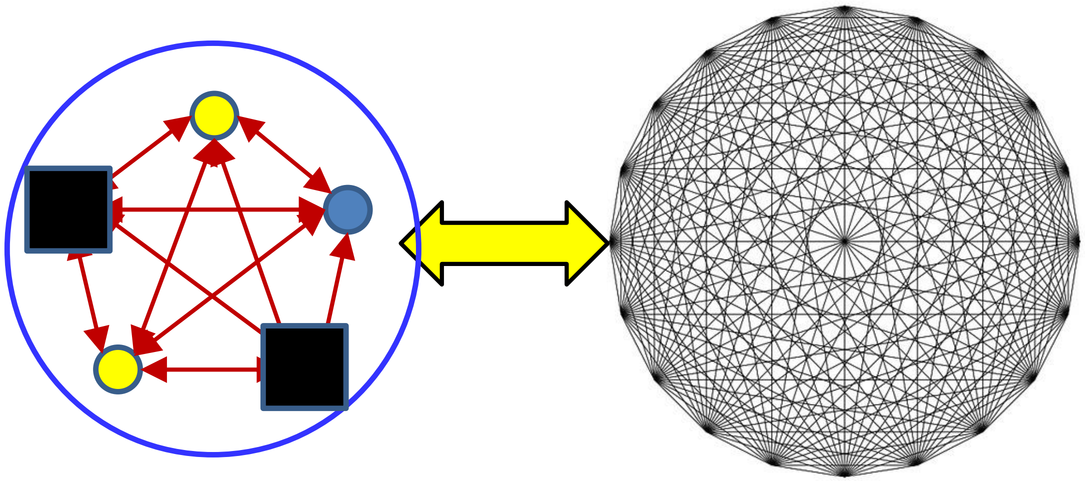
$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{NK}\sum_{\mathbf{S}} s_i s_j - \frac{1}{M}\sum_{S' \in \mathbf{S}_{simul}} s_i' s_j'$$

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



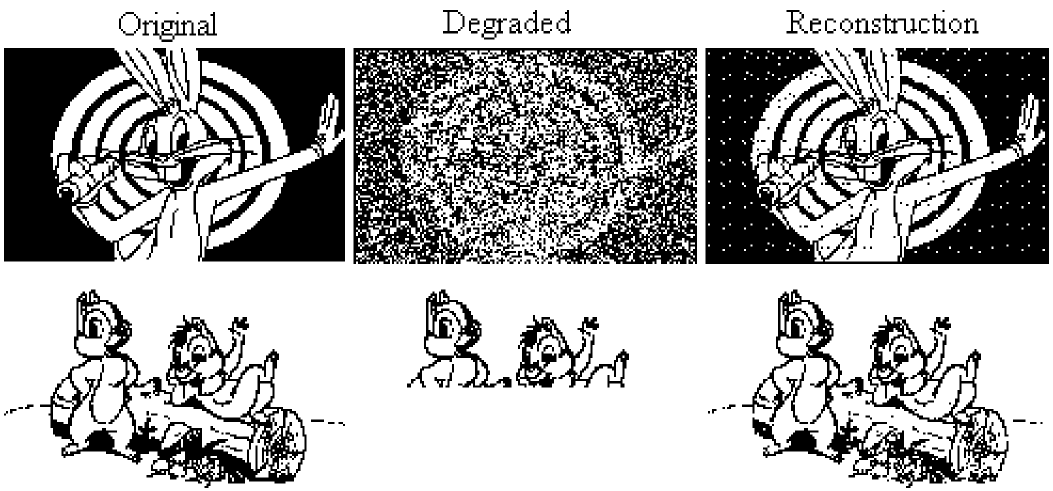
- **Training:** Given a set of training patterns
  - Which could be repeated to represent relative probabilities
- Initialize weights
- Run simulations to get clamped and unclamped training samples
- Compute gradient and update weights
- Iterate

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

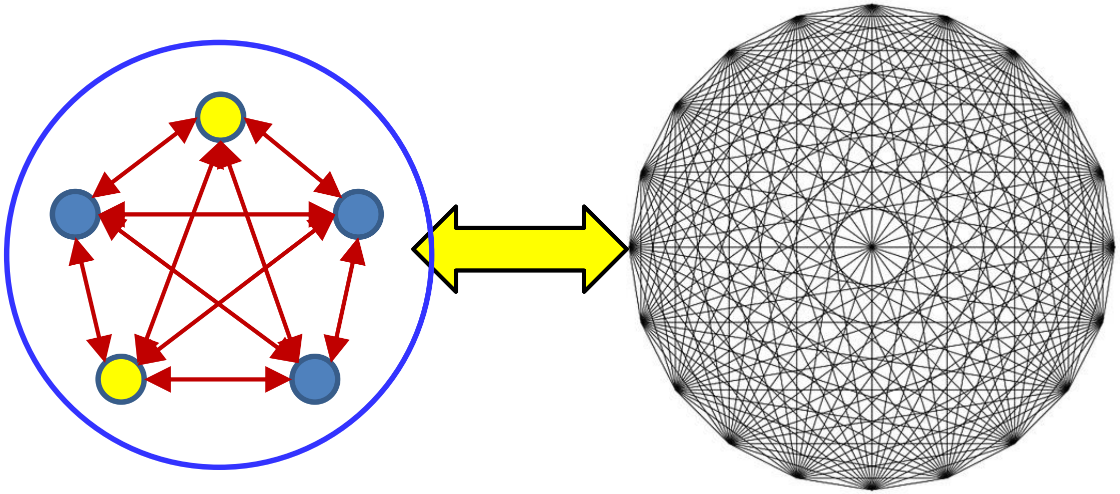
Applications



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

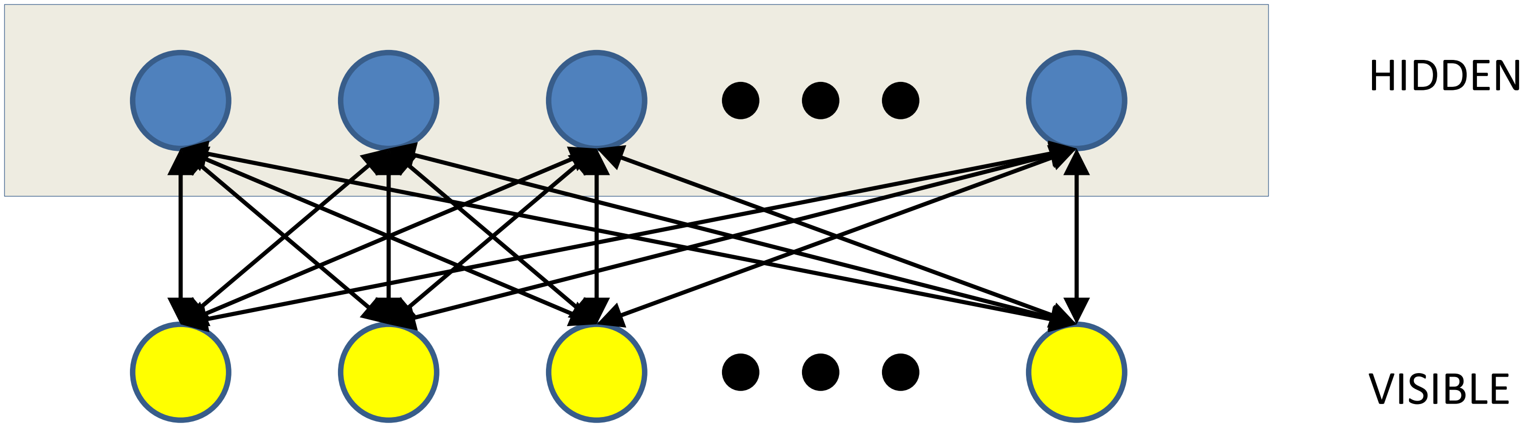


- Training patterns:
  - [f1, f2, f3, …. , class]
  - Features can have binarized or continuous valued representations
  - Classes have “one hot” representation
- Classification:
  - Given features, anchor features, estimate a posteriori probability distribution over classes
    - Or choose most likely 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..

Topics missed..

- The Boltzmann machine as a probability distribution
- RBMs
- Running RBMs
- Inference over RBMs
- RBMs as feature extractors
  - Pre training
- RBMs as generative models
- DBMs