# Neural Networks

## Hopfield Nets, Auto Associators, *Boltzmann machines*

## Fall 2025

# Story so far: 2024 Nobel Prize in Physics

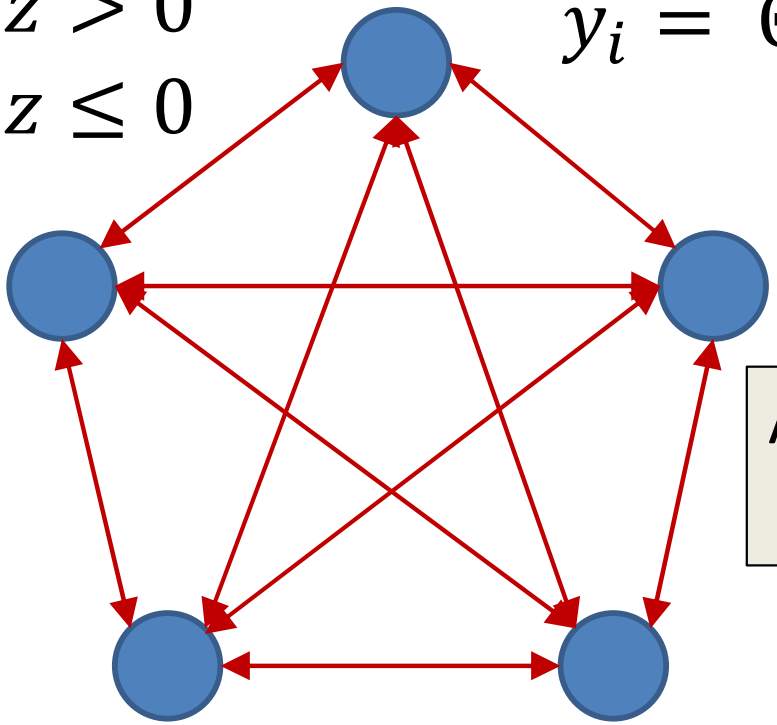
## They trained artificial neural networks using physics

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

# Hopfield Net

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

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



A symmetric network:

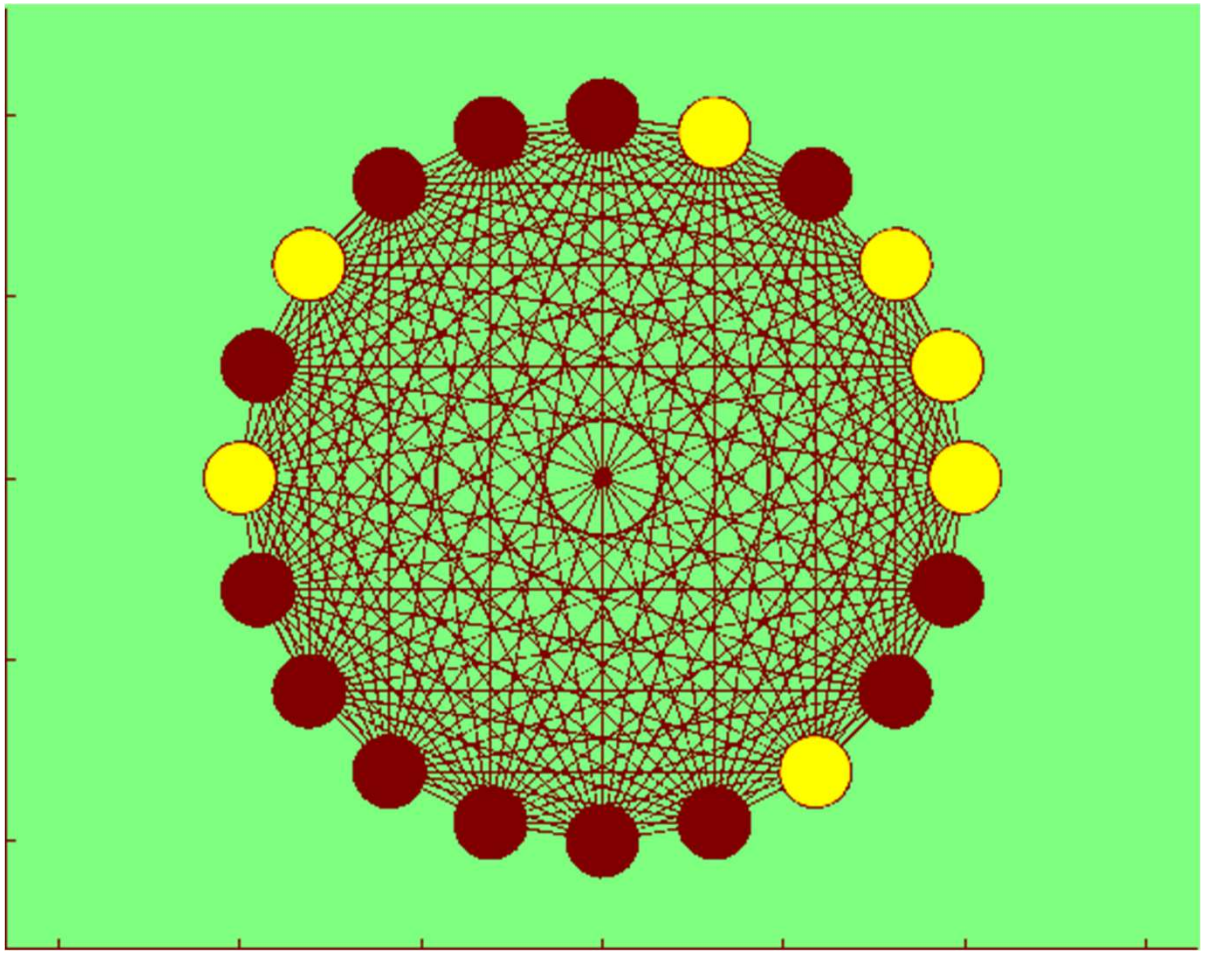
$$w_{ij} = w_{ji}$$

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

# 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

# 120 evolutions of a loopy net

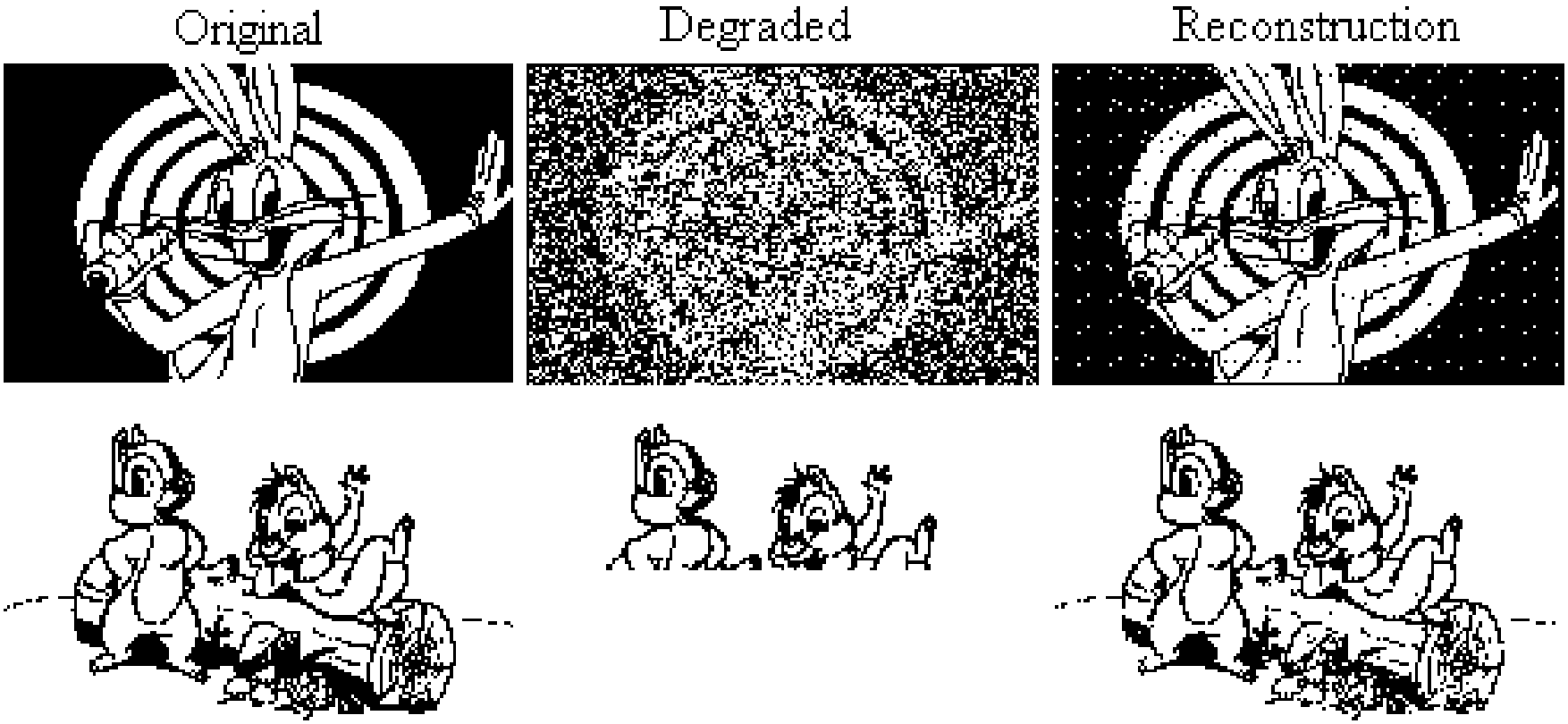


- All neurons which do not “align” with the local field “flip”

# Story so far

- 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

# 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/

# Story so far

- The network must be designed to store the desired memories
  - Memory patterns must be *stationary* and *stable* on the energy contour
- Network memory can be trained by Hebbian learning
  - Guarantees that a network of N bits trained via Hebbian learning can store 0.14N random patterns with less than 0.4% probability that they will be unstable
- However, empirically it appears that we may sometimes be able to store *more than* 0.14N patterns

# Story so far

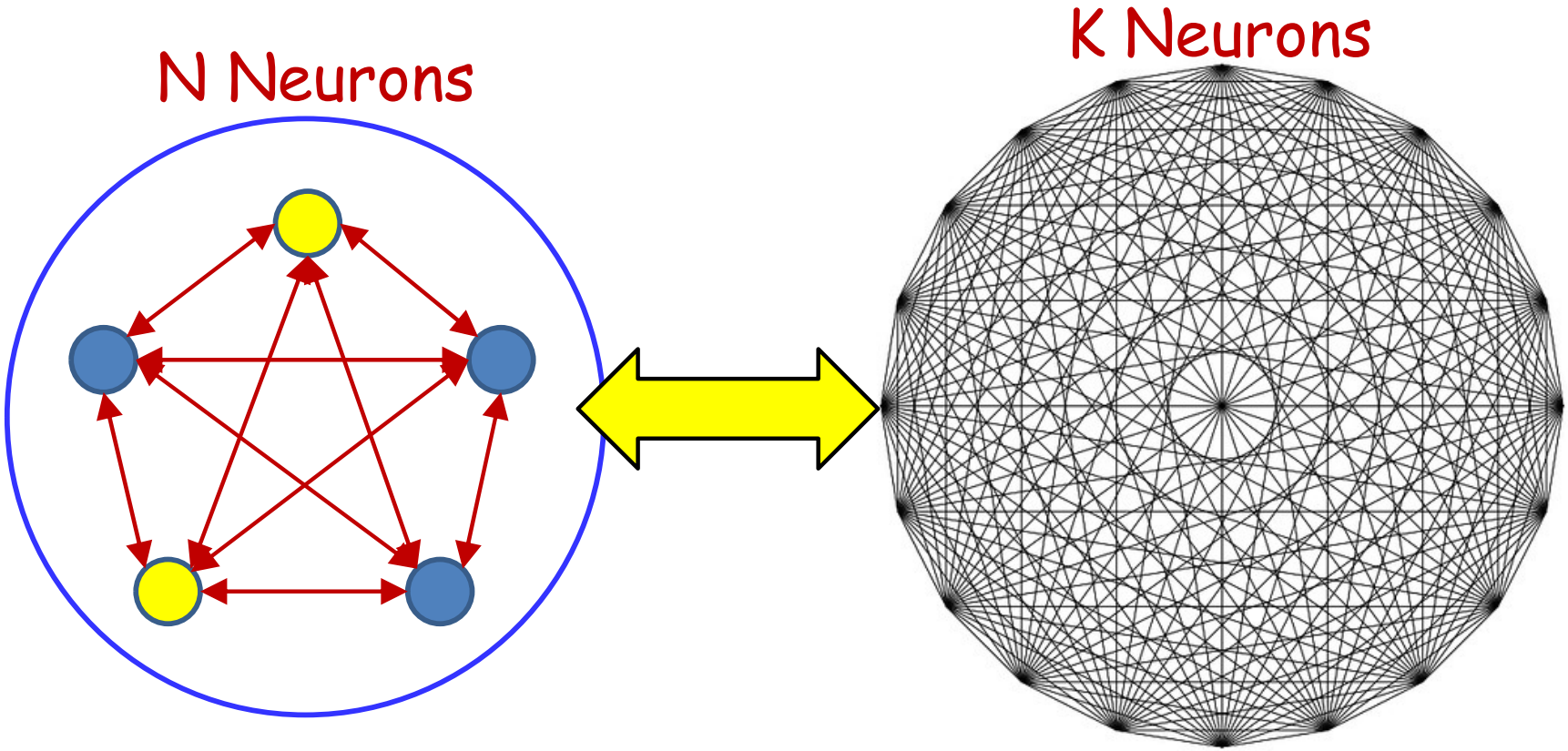
- Hopfield nets with $N$ neurons can store up to $O(N)$ random patterns
  - But comes with many parasitic memories

- 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

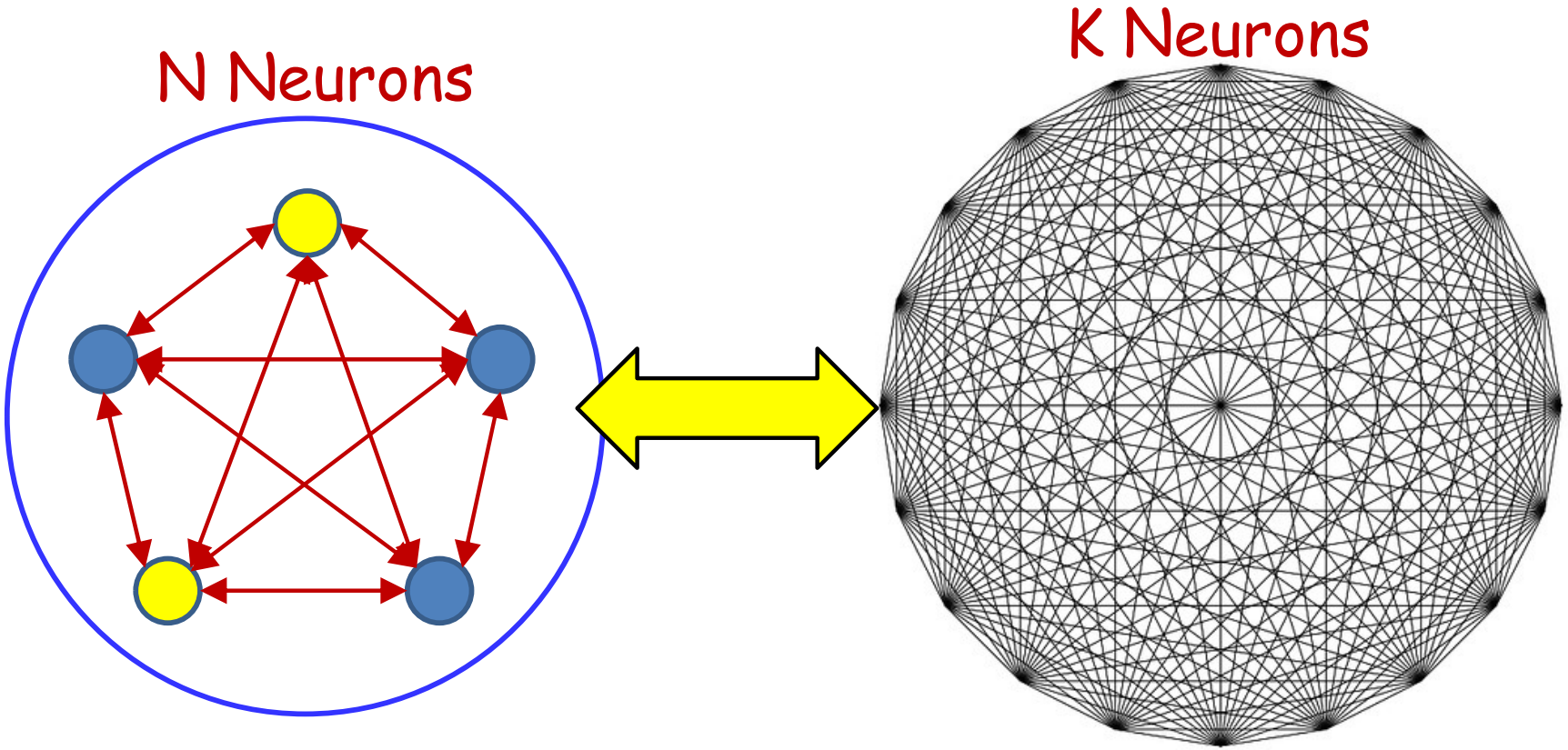
- The memory capacity of an $N$-bit network is at most $N$
  - Stable patterns (not necessarily even stationary)
    - Abu Mustafa and St. Jacques, 1985
    - Although "information capacity" is $\mathcal{O}(N^3)$
- How do we increase the capacity of the network
  - How to store more than $N$ patterns

# Expanding the network



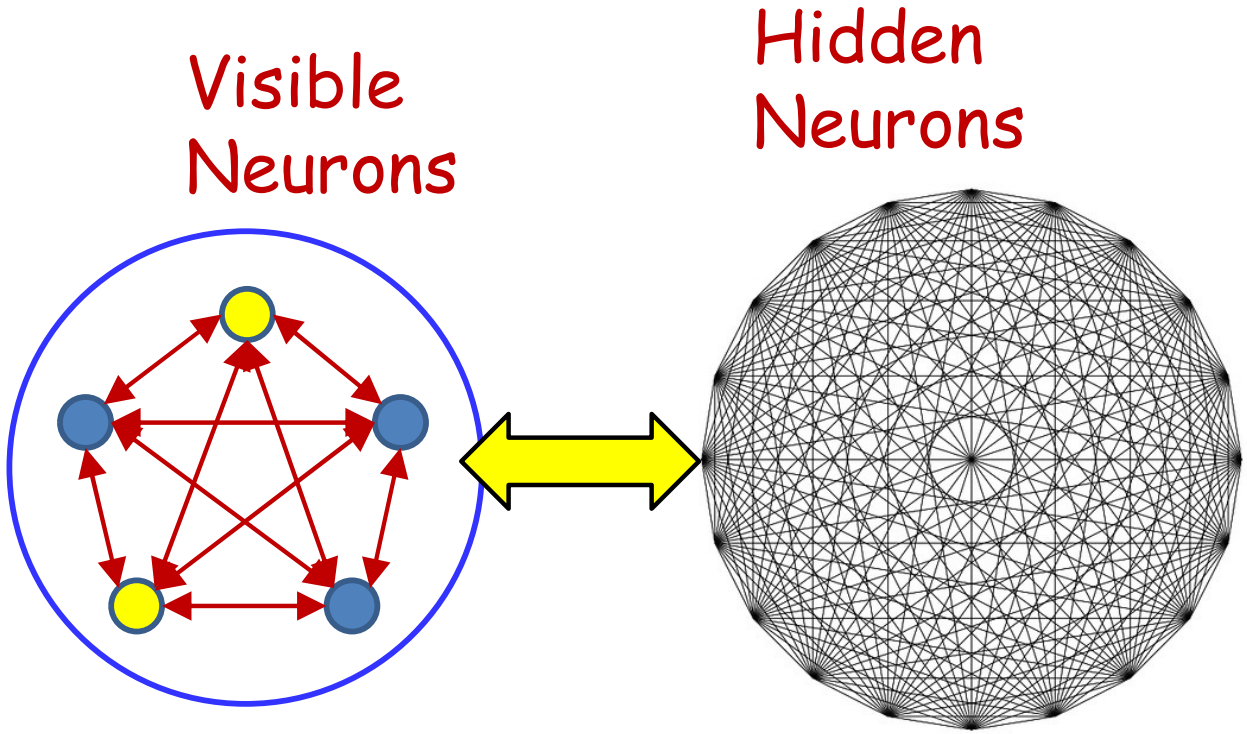

- Add a large number of neurons whose actual values you don't care about!

# Expanded Network



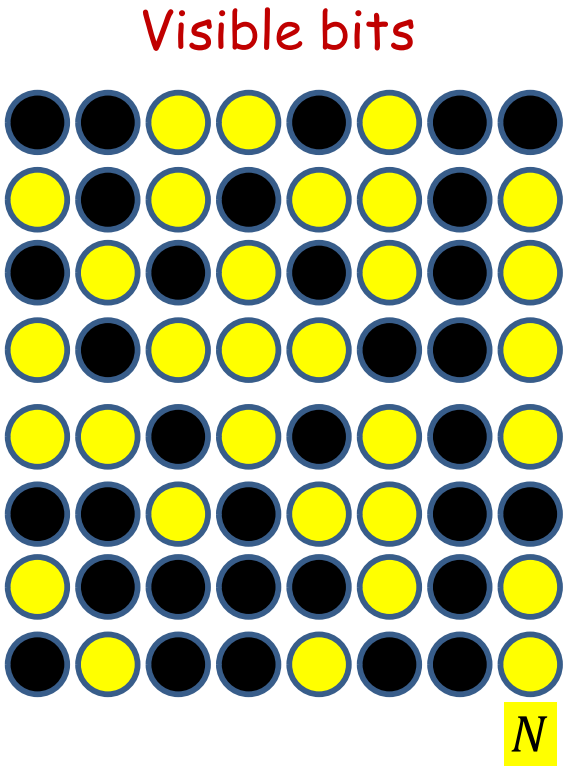

- New capacity: $\sim(N + K)$ patterns
  - Although we only care about the pattern of the first N neurons
  - We're interested in *N-bit* patterns

# Terminology



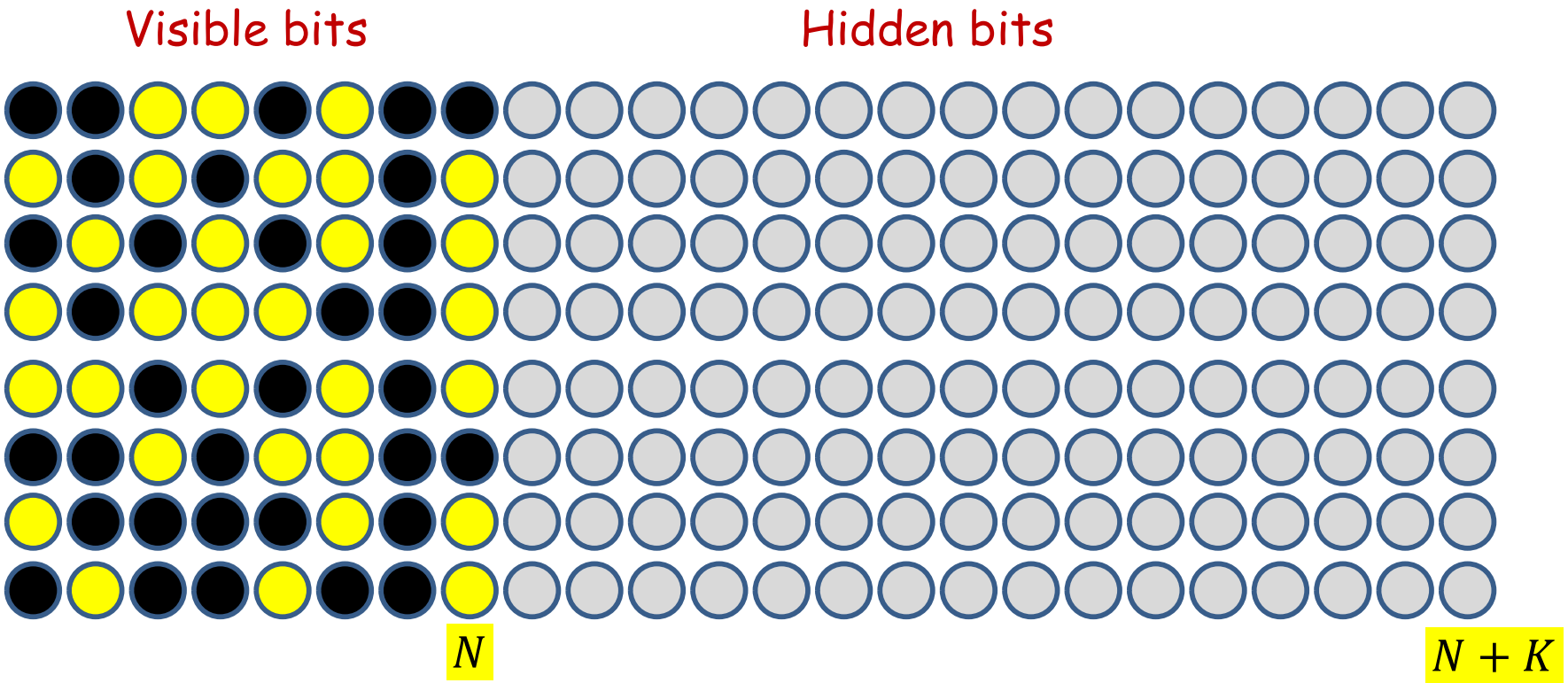

- Terminology:
  - The neurons that store the actual patterns of interest: *Visible neurons*
  - The neurons that only serve to increase the capacity but whose actual values are not important: *Hidden neurons*
  - These can be set to anything in order to store a visible pattern

# Increasing the capacity: bits view




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

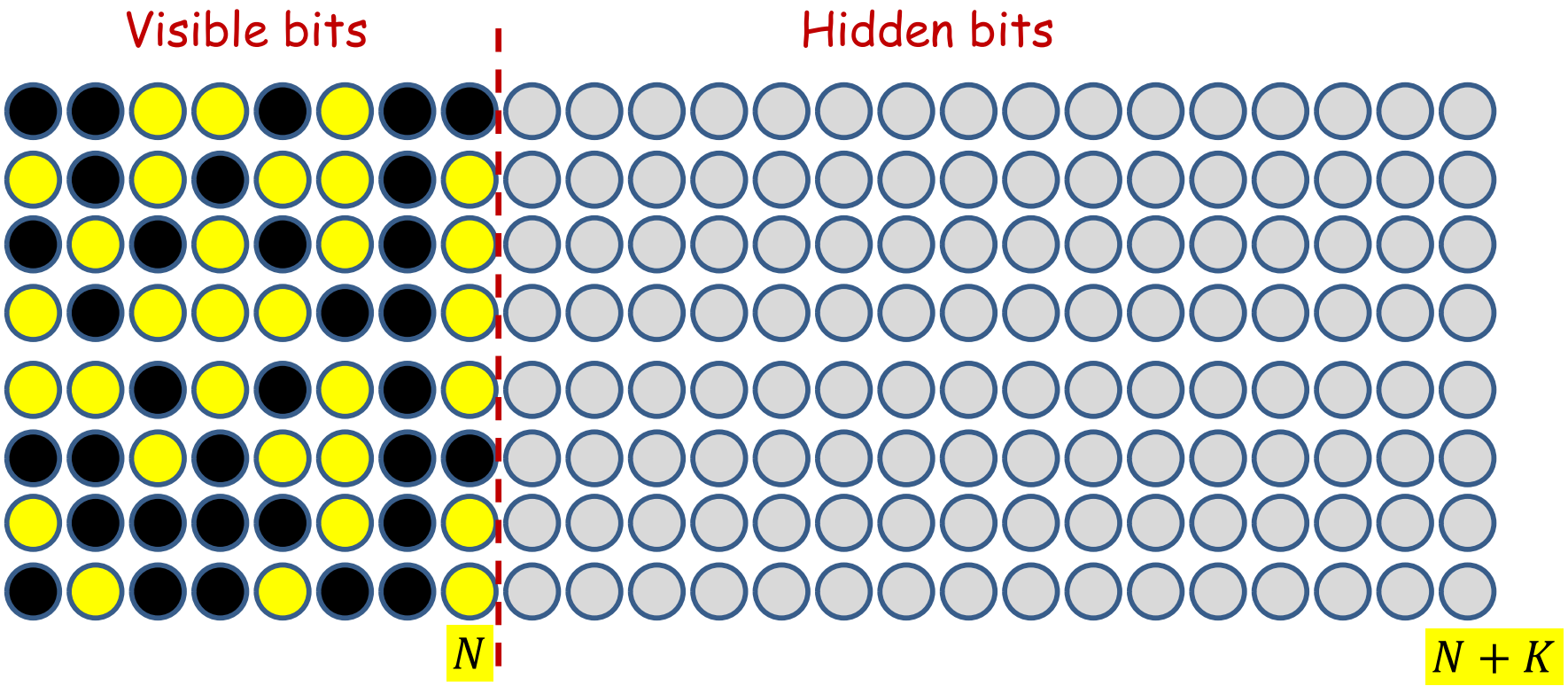
# Increasing the capacity: bits view




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

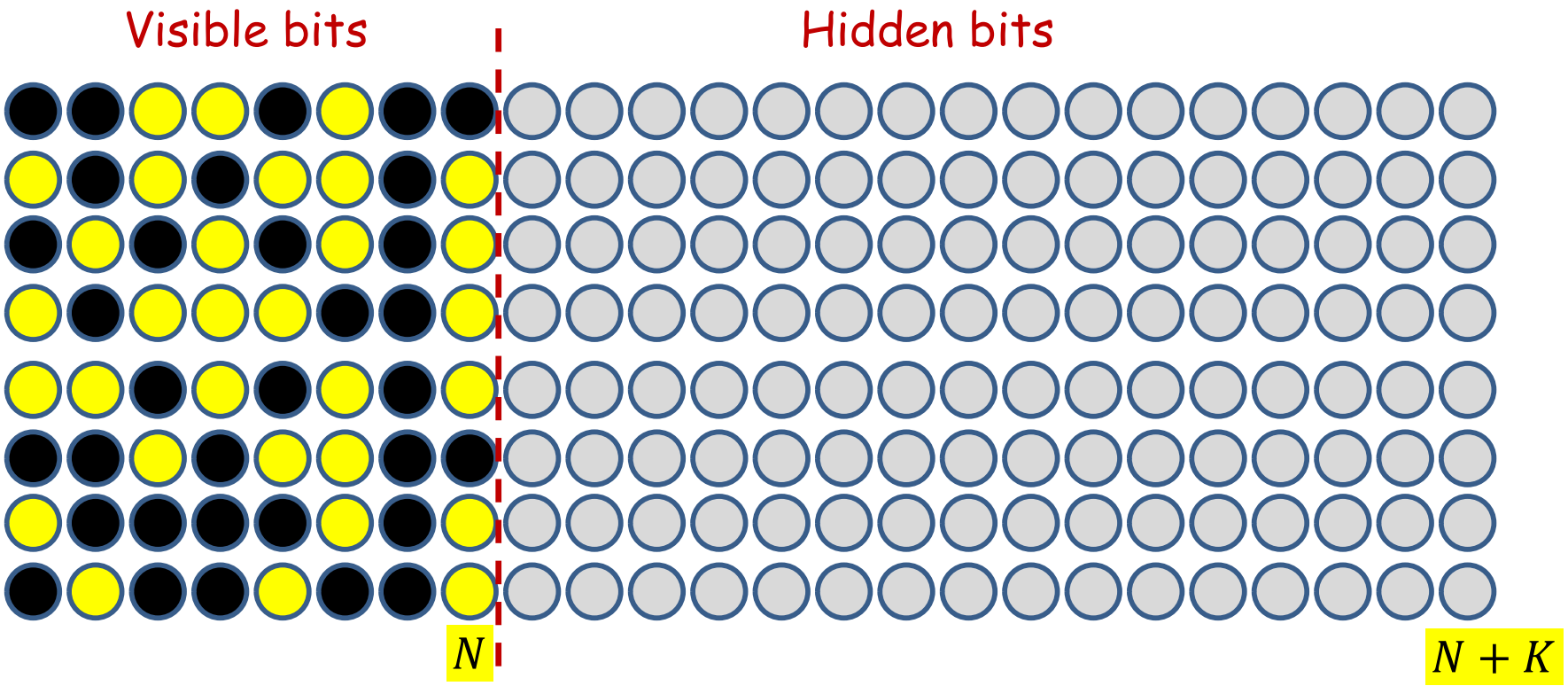
- So, let's *pad* the patterns with *K* "don't care" bits
  - The new width of the patterns is N+K
  - Now we can store N+K patterns!

# Issues: Storage




- What patterns do we fill in the don't care bits?
  - Simple option: Randomly
    - Flip a coin for each bit
  - Optimize
- 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

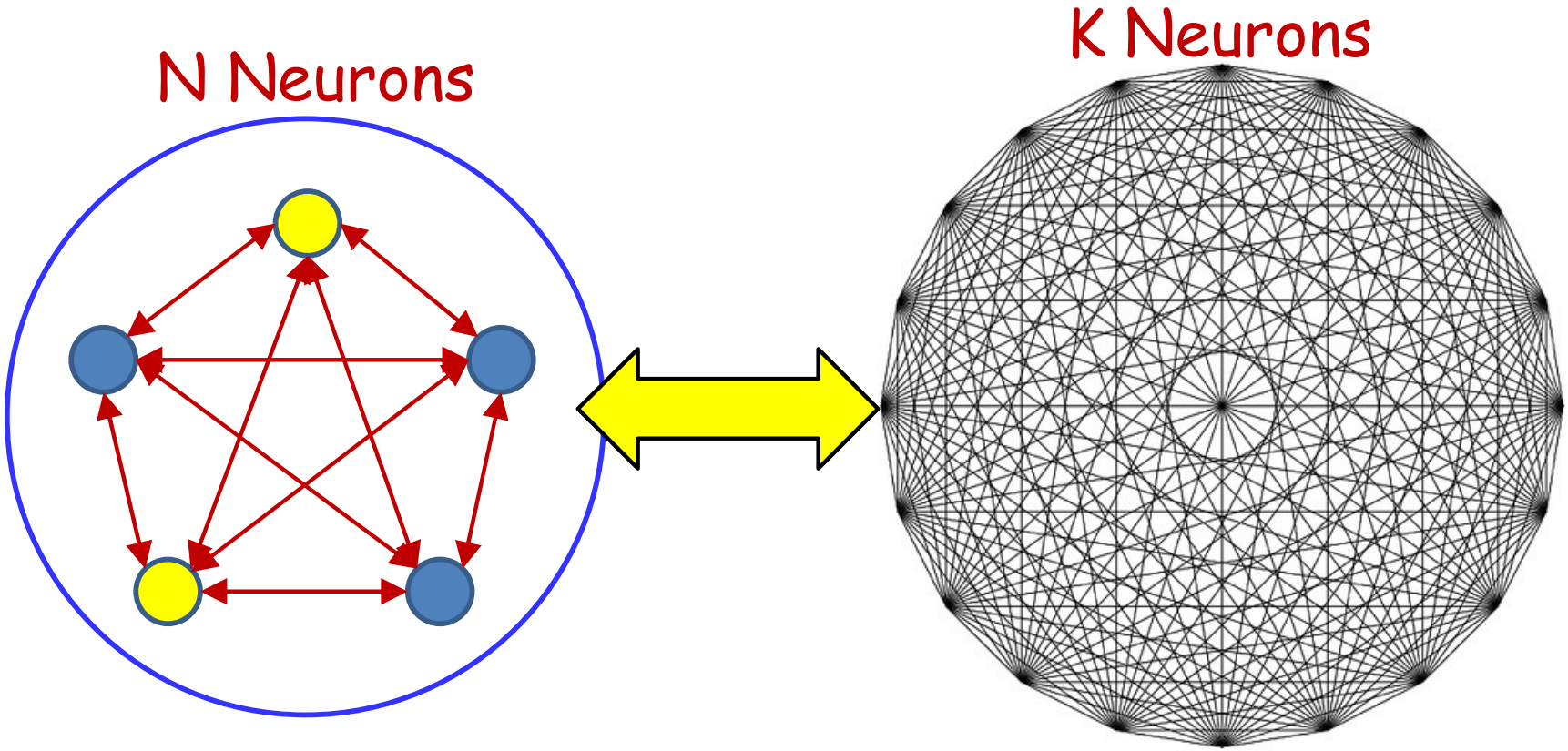
# Poll 1

- Which of the following is true of the memory capacity of an N-bit Hopfield net
  - It can store 0.14N patterns perfectly
  - It can store O(N) patterns
  - Patterns can be trained into the network using gradient descent
- How can we increase the memory capacity of a Hopfield net from N to N+K
  - By appending K irrelevant bits to the patterns to increase pattern length (and the size of the net) to N+K
  - By using alternate optimization rules that increase the mathematical capacity of the network
- If we try to increase the capacity of the network by adding K irrelevant bits to the patterns, it is important for the network to recall these additional K bits exactly to recall the stored patterns
  - Yes
  - No

# Poll 1

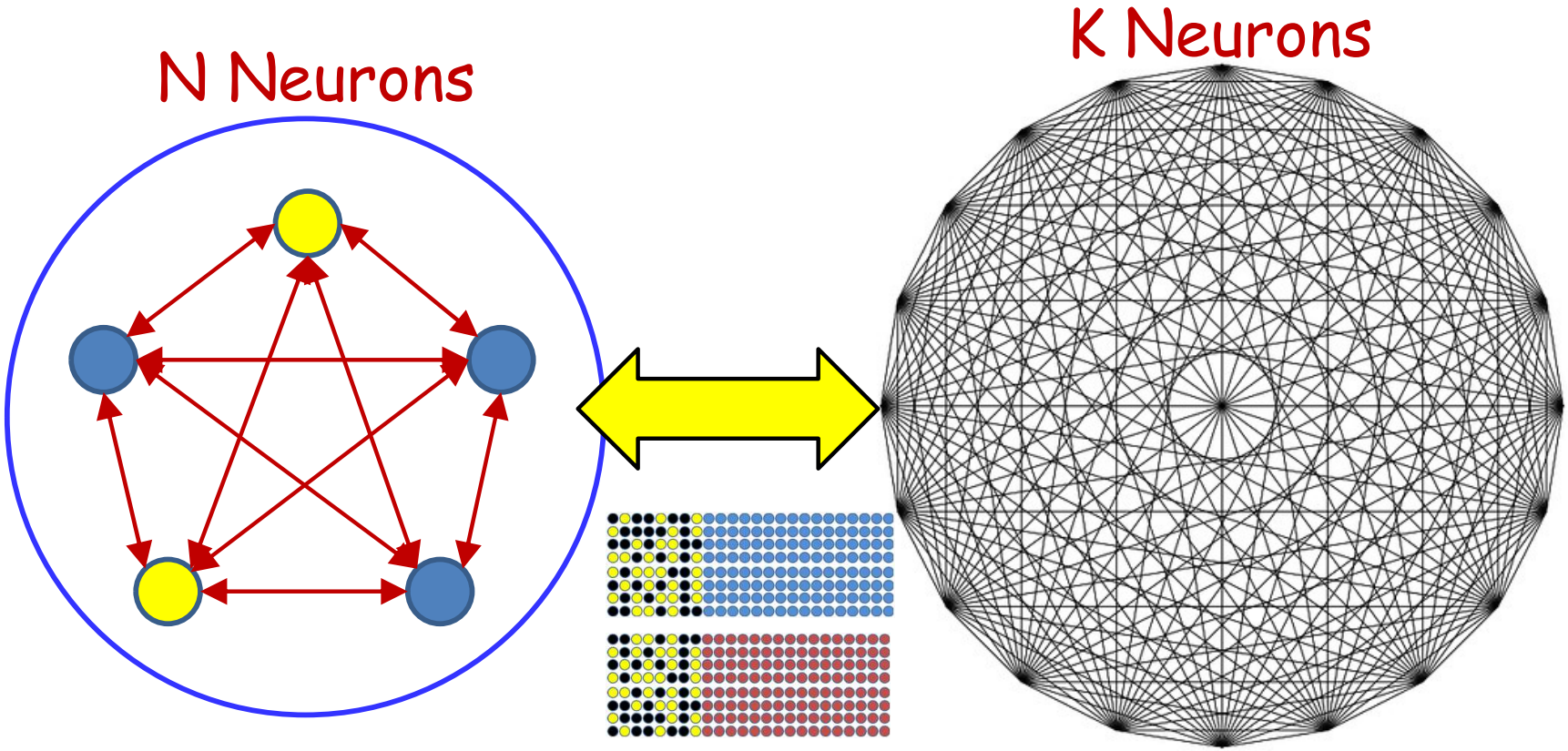
- Which of the following is true of the memory capacity of an N-bit Hopfield net
  - It can store 0.14N patterns perfectly
  - **It can store O(N) patterns**
  - **Patterns can be trained into the network using gradient descent**
- How can we increase the memory capacity of a Hopfield net from N to N+K
  - **By appending K irrelevant bits to the patterns to increase pattern length (and the size of the net) to N+K**
  - By using alternate optimization rules that increase the mathematical capacity of the network
- If we try to increase the capacity of the network by adding K irrelevant bits to the patterns, it is important for the network to recall these additional K bits exactly to recall the stored patterns
  - Yes
  - **No**

# 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

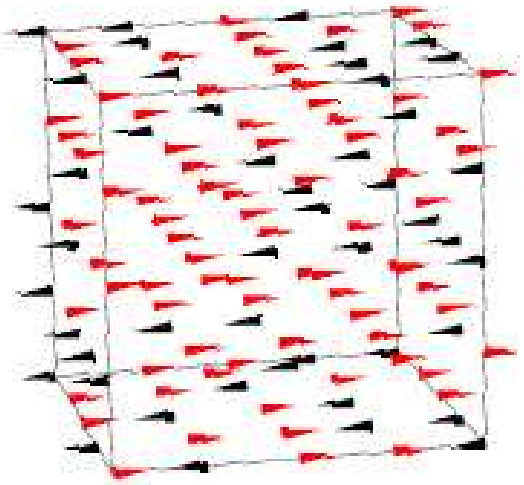
- However, it doesn't sufficiently exploit the redundancy of the don't care bits
  - Possible to set the don't care bits such that the overall pattern (and hence the "visible" bits portion of the pattern) is more memorable
  - Also, may have multiple don't-care patterns for a target pattern
    - Multiple valleys, in which the visible bits remain the same, but don't care bits vary

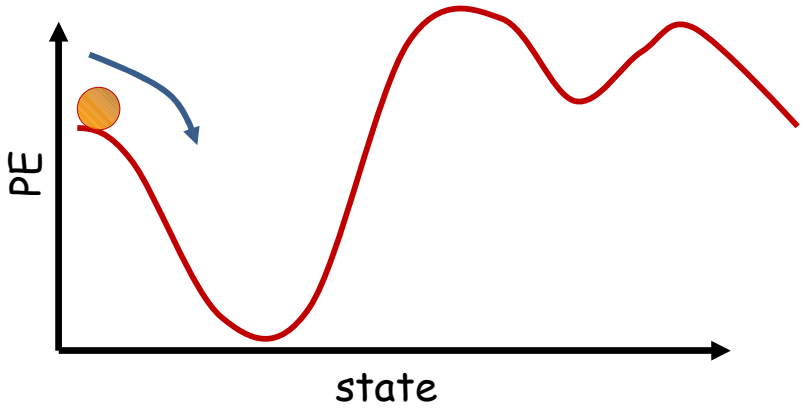
- To exploit it properly, it helps to view the Hopfield net differently: as a probabilistic machine

# A return to physics…

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

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

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

$$F_T = U_T - TH_T$$

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

# 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, while applying a Lagrangian

$$F_T = \sum_s P_T(s)\, E_s + kT \sum_s P_T(s) \log P_T(s) + \lambda(\sum_s P_T(s) - 1)$$

- Differentiating w.r.t. $P_T(s)$

$$\frac{dF_T}{dP_T(s)} = E_s + kt(1 + \log P_T(s)) + \lambda = 0$$

- Solving

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

# The Helmholtz Free Energy of a System

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

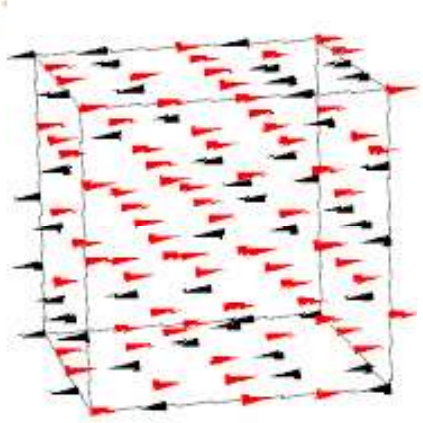
- Minimizing this w.r.t $P_T(s)$, we get

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

  - 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 Boltzmann Distribution

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



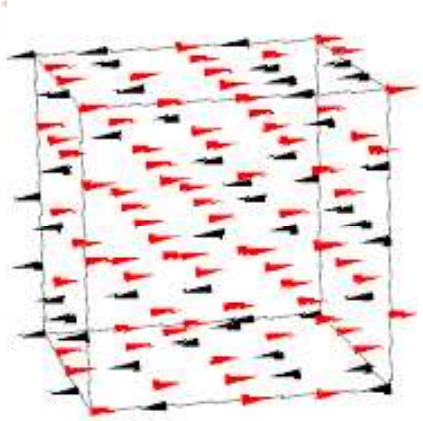
$$C = \frac{1}{\sum_{\mathbf{y}} exp\left(\frac{-E(\mathbf{y})}{kT}\right)}$$

- $k$ 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..

# The evolution of the system

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

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



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

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

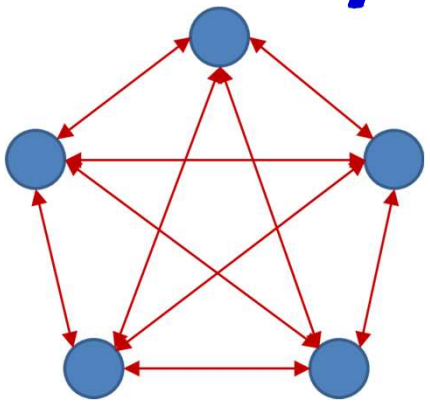
# Poll 2

- A Hopfield network is a computational analogue of a deterministic ferroelectric material
  - True
  - False
- The true behavior of thermodynamic systems is stochastic
  - True
  - False
- Hopfield networks too can be modified to emulate the stochastic behavior of thermodynamic systems
  - True
  - False

# Poll 2

- A Hopfield network is a computational analogue of a deterministic ferroelectric material
  - **True**
  - False
- The true behavior of thermodynamic systems is stochastic
  - **True**
  - False
- Hopfield networks too can be modified to emulate the stochastic behavior of thermodynamic systems
  - **True**
  - False

# A probabilistic interpretation of Hopfield Nets with thermodynamic analogy



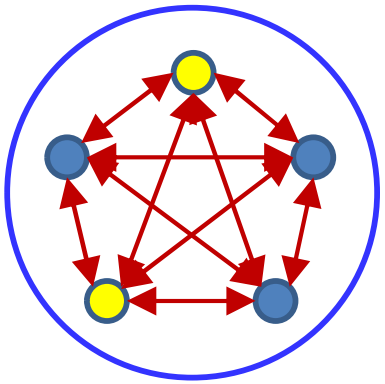
- Instead of viewing the network as a deterministic machine, model it as a *statistical* system that is in any state $y$ with probability $P(y)$ following thermodynamic principles

- For *binary* y the energy of a pattern is the analog of the negative log likelihood of a *Boltzmann distribution*
  - **The system has a native energy for each state, which is countered by entropy**
  - **The system "evolves" to minimize the *expected* energy of the system**
  - **Minimizing energy gives us a Boltzmann distribution over states with T=1**

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

# The Equilibrium Distribution

Visible Neurons



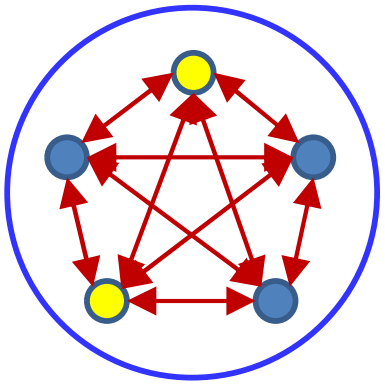
$$E(S) = -\sum_{i<j} w_{ij} s_i s_j - b_i s_i$$

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

- *Neurons are stochastic*, with disorder or entropy
  - The network evolves to minimize the energy, while maximizing the entropy
- The ***equilibrium* 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

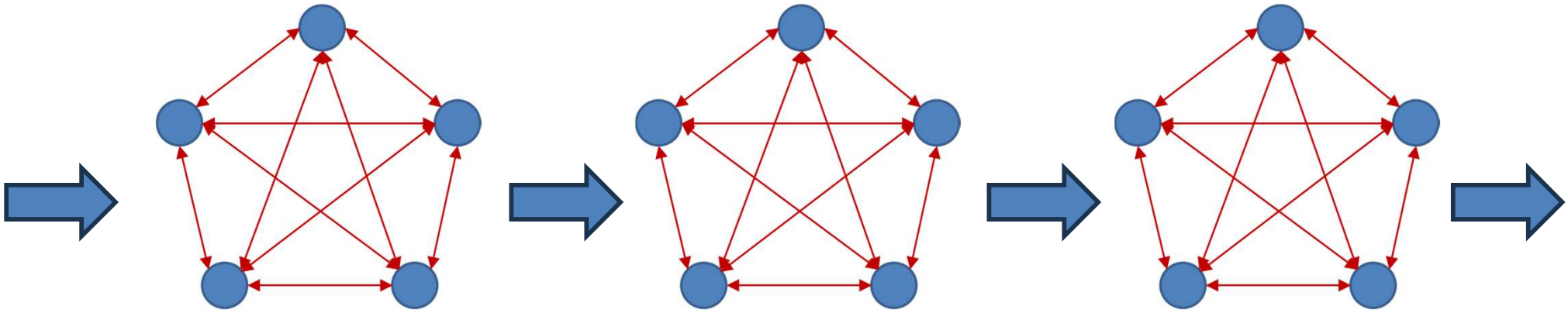


$$E(S) = -\sum_{i<j} w_{ij} s_i s_j - b_i s_i$$

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

- The stochastic Hopfield network models a ***probability distribution*** over states
  - Where a state is a binary string
  - Specifically, at Equilibrium, 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 $P(S)$

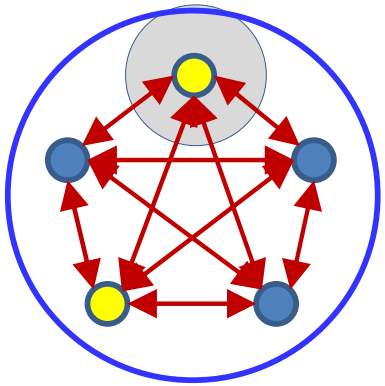
# How does a stochastic Hopfield Net evolve



- How does this stochastic (probabilistic) network evolve?
  - What is the rule by which neurons flip?

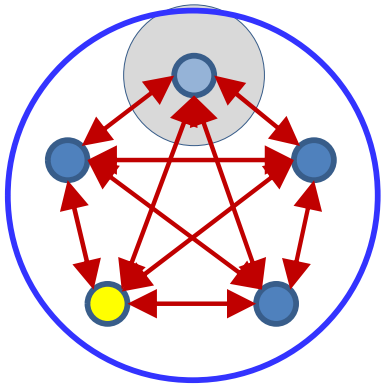
# The field at a single node

- 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 = $-1$



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

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

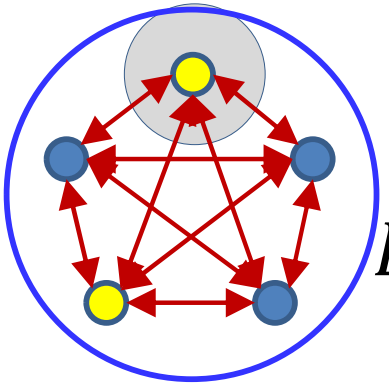


$$logP(S) - logP(S') = logP\left(s_i = 1 \middle| s_{j\neq i}\right) - logP\left(s_i = -1 \middle| s_{j\neq i}\right)$$

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

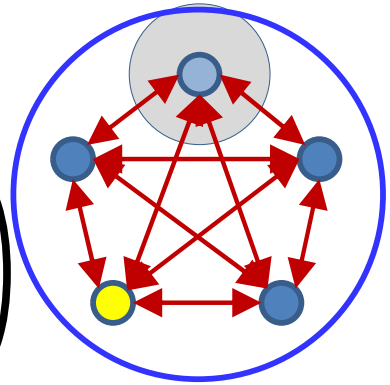
# The field at a single node

- 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

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

- Exponentiating and expanding

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

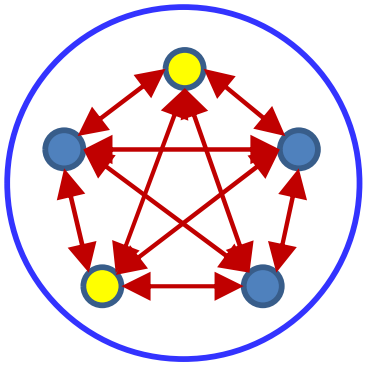
- Giving us

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

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

# Redefining the network

Visible Neurons



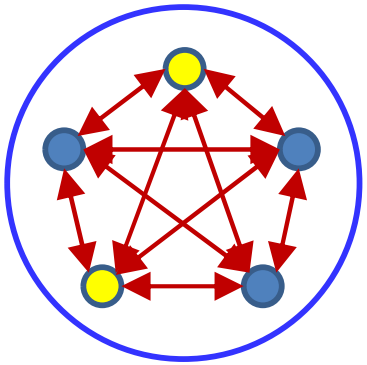
$$z_i = \sum_j w_{ji} s_j + b_i$$

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

- 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

Visible Neurons



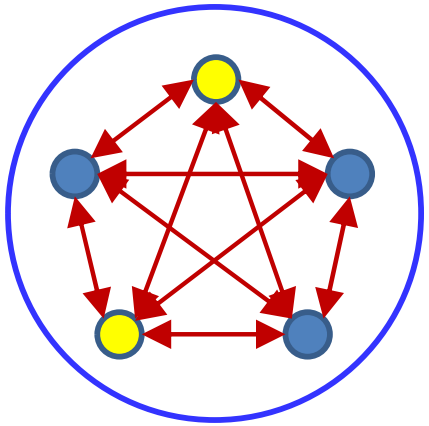
$$z_i = \sum_j w_{ji} s_j + b_i$$

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

- 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

Visible Neurons



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

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

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

# Evolution of a stochastic Hopfield net

1. Initialize network with initial pattern

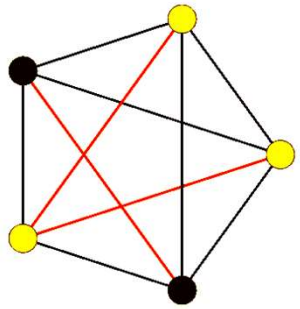
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

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

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

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

Assuming T = 1



# Evolution of a stochastic Hopfield net

1. Initialize network with initial pattern

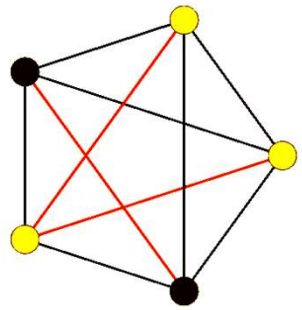
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

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

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

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

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

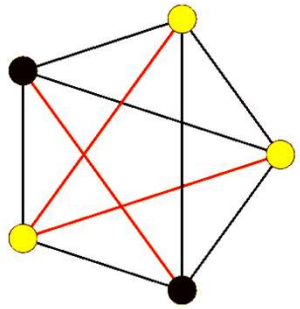
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

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

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

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

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

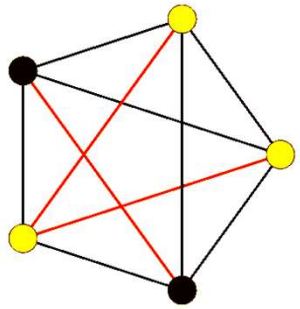
$$y_i(0) = x_i, \qquad 0 \le i \le N-1$$

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

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

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

Assuming T = 1

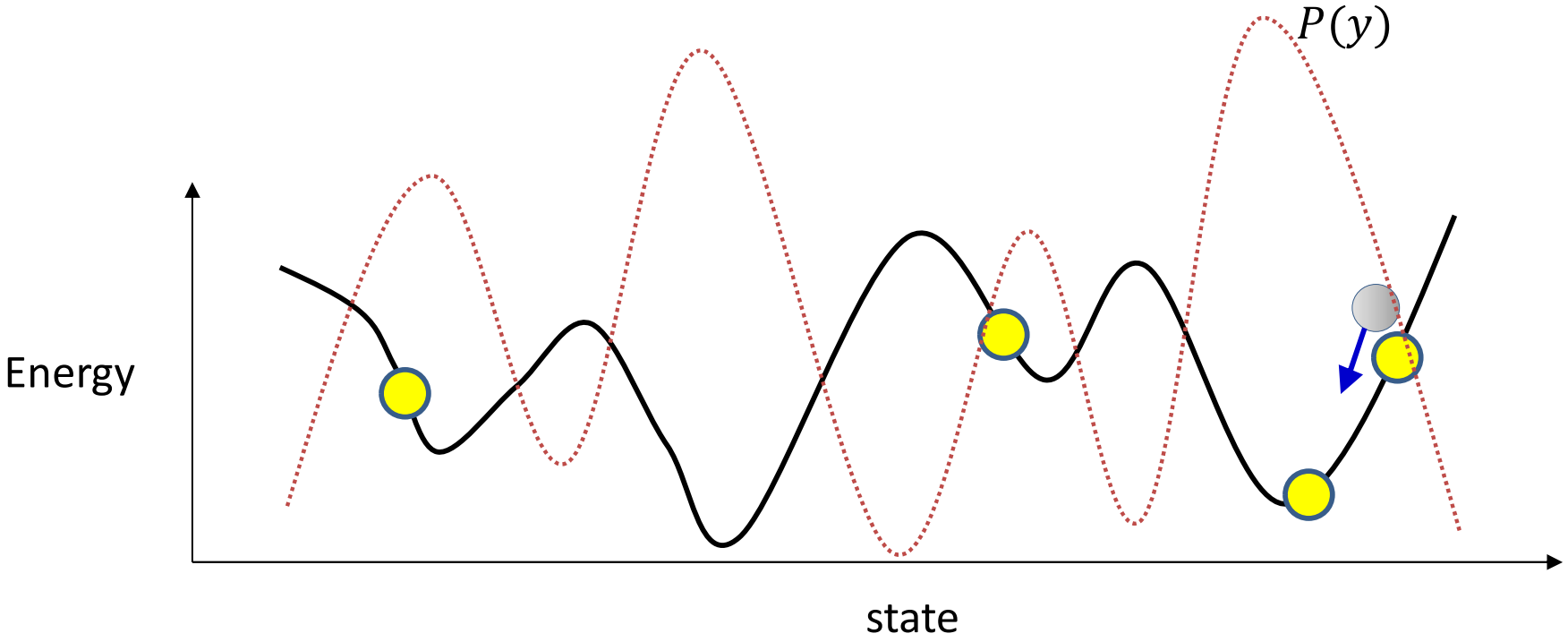


- Let the system evolve to "equilibrium"
- Let $\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_L$ be the sequence of values ($L$ large)
- Final predicted configuration: from the average of the final few iterations

$$\mathbf{y} = \left(\frac{1}{M}\sum_{t=L-M+1}^{L} \mathbf{y}_t\right) > 0.5?$$

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

# The "Boltzmann" Machine




- 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

# 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* $T_{min}$

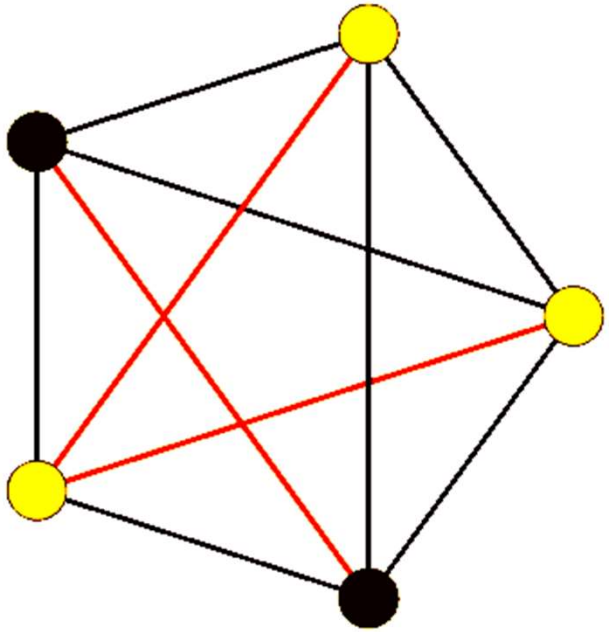
Noisy pattern completion: Initialize the entire network and let the entire network evolve

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

$$\mathbf{y} = \left( \frac{1}{M} \sum_{t=L-M+1}^{L} \mathbf{y}_t \right) > 0.5?$$

# Including a "Temperature" term



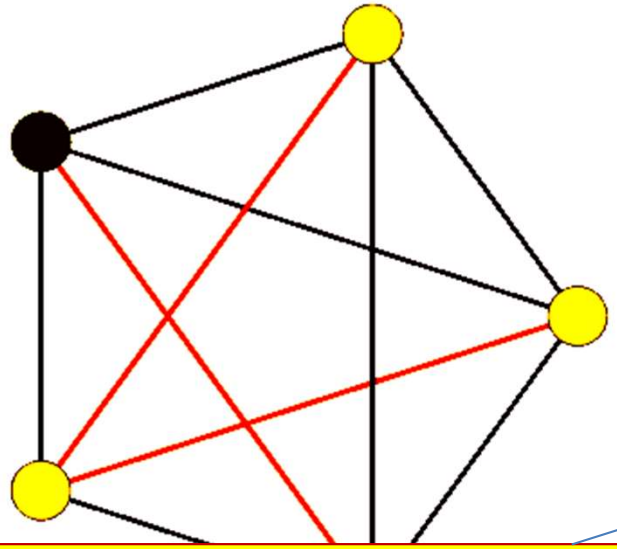
$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ij} y_j$$

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

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

- 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



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

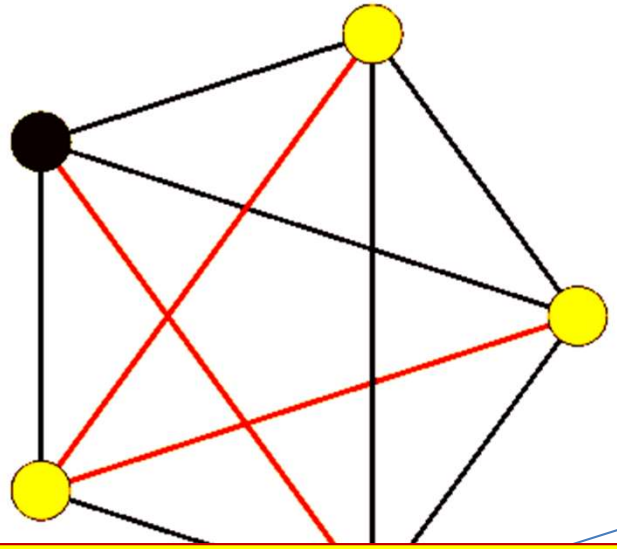
$$P(y_i = 1) = \sigma(z_i)$$

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



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

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

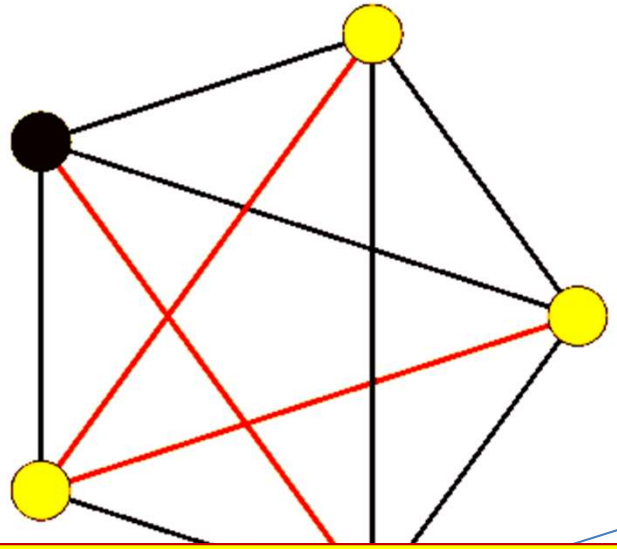
The field quantifies the energy difference obtained by flipping the current unit

- Including a temperature term in computing the local field
  - This is much more in accord with thermodynamic models

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

- 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

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

  - 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 $0 \le i \le N-1$

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

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

- Let the system evolve to "equilibrium"
- Let $\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_L$ 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. For $T = T_0 \; down \; to \; T_{min}$

   i. For iter $1..L$

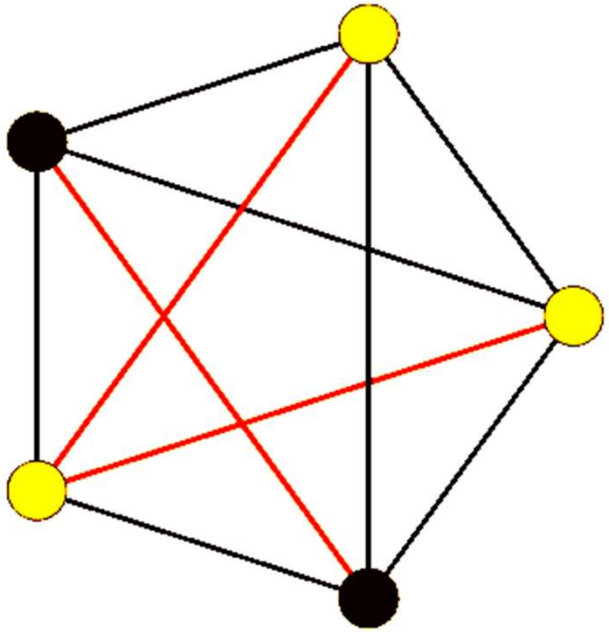
      a) For $0 \le i \le N-1$

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

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

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

# Recap: Stochastic Hopfield Nets

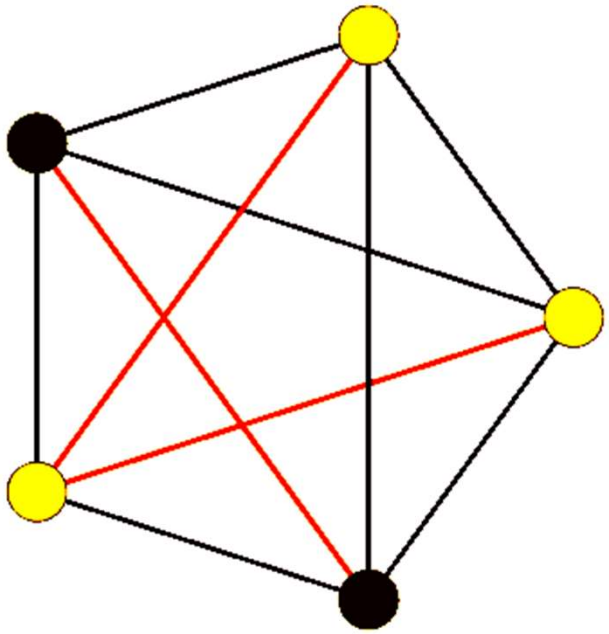


$$z_i = \frac{1}{T}\sum_{j \neq i} w_{ji} y_j$$

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

- 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



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

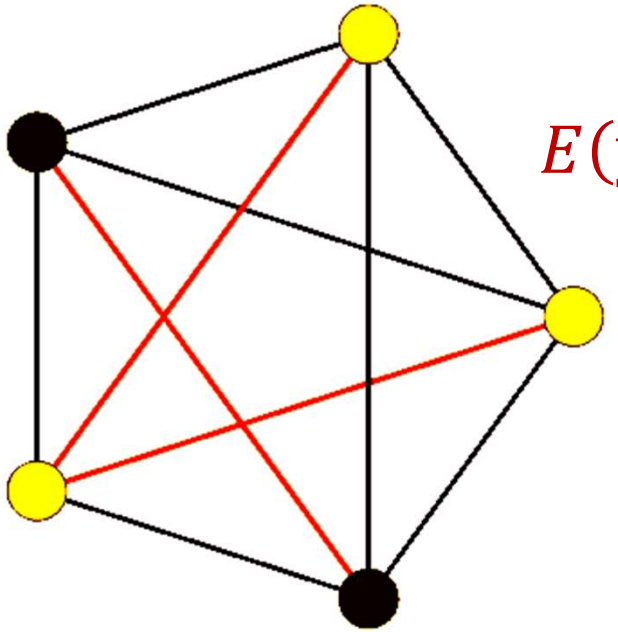
$$P(y_i = 1 | y_{j \neq i}) = \sigma(z_i)$$

- The probability of any state $\mathbf{y}$ 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

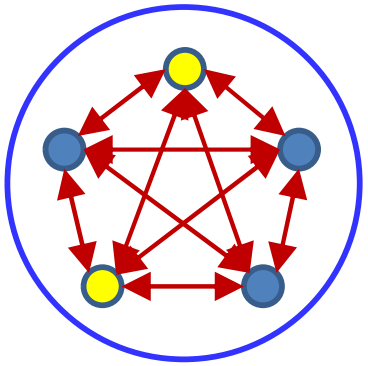
# Poll 3

- The stochastic Hopfield net flips neurons according to a probability computed by a logistic activation at the neuron
  - True
  - False
- The stochastic Hopfield net can flip a neuron even if doing so increases the energy
  - True
  - False

# Poll 3

- The stochastic Hopfield net flips neurons according to a probability computed by a logistic activation at the neuron
  - **True**
  - False
- The stochastic Hopfield net can flip a neuron even if doing so increases the energy
  - **True**
  - False

# Recap: The Hopfield net is a distribution



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

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

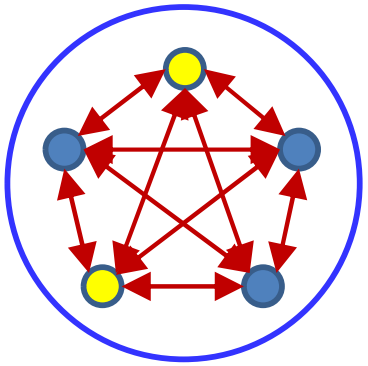
- The Hopfield net is a probability distribution over binary sequences
  - The Boltzmann distribution

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

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

  - 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 $\mathbf{y}$:

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

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

# Training the model

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

- Not much…

# 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:

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

More importance to more frequently presented memories

More importance to more attractive spurious memories

THIS LOOKS LIKE AN EXPECTATION!

# Hopfield nets: Optimizing W

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

- Update rule

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

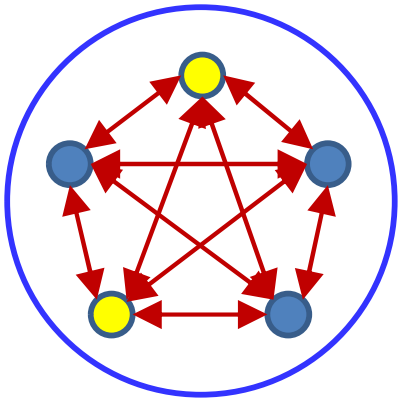
$$\mathbf{W} = \mathbf{W} + \eta\left(E_{\mathbf{y}\sim\mathbf{Y}_P}\mathbf{y}\mathbf{y}^T - E_{\mathbf{y}\sim Y}\mathbf{y}\mathbf{y}^T\right)$$

Natural distribution for variables: The Boltzmann Distribution

We can arrive at the same result a bit more formally…

# *Training* the network

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



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

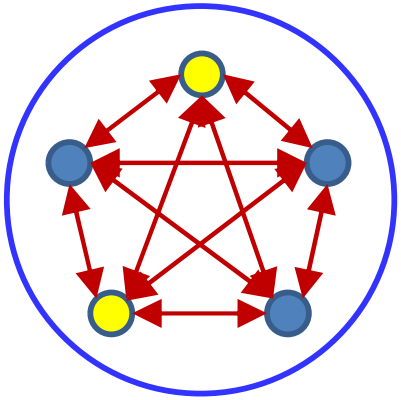
$$P(S) = \frac{exp\left(\sum_{i<j} w_{ij} s_i s_j\right)}{\sum_{S'} exp\left(\sum_{i<j} w_{ij} s'_i s'_j\right)}$$

- 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

Visible Neurons



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

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

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

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

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

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

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 $\mathbf{S} = \{S_1, S_2, \ldots, SN\}$
  - 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

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

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

- 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!

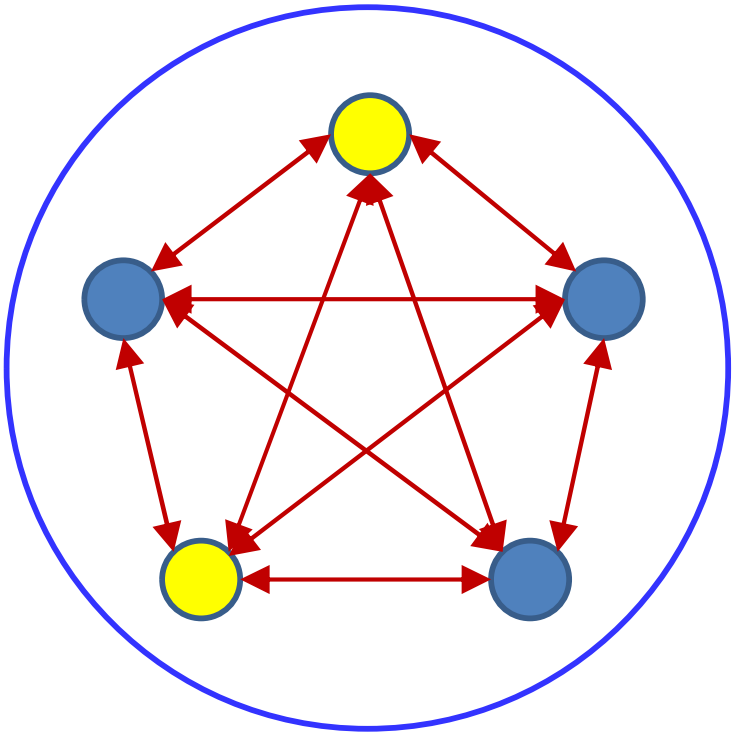
# *Estimating the second term*

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

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

- 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

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

# 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*

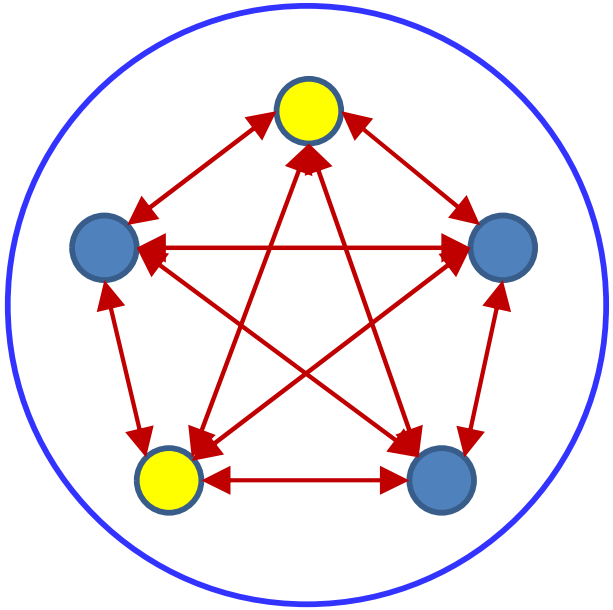
Sampled estimate

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

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

- The overall gradient ascent rule

# Overall Training

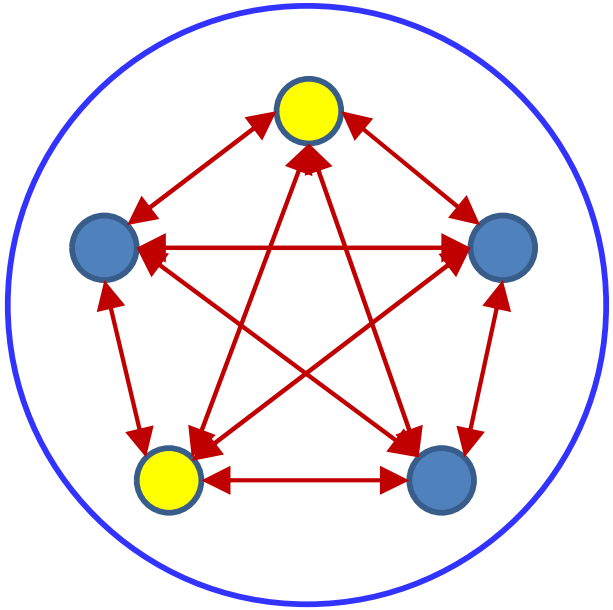


$$\frac{d\langle \log(P(\mathbf{S}))\rangle}{dw_{ij}} = \frac{1}{N}\sum_{S} s_i s_j - \frac{1}{M}\sum_{S' \in \mathbf{S}_{simul}} s'_i s'_j$$

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

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

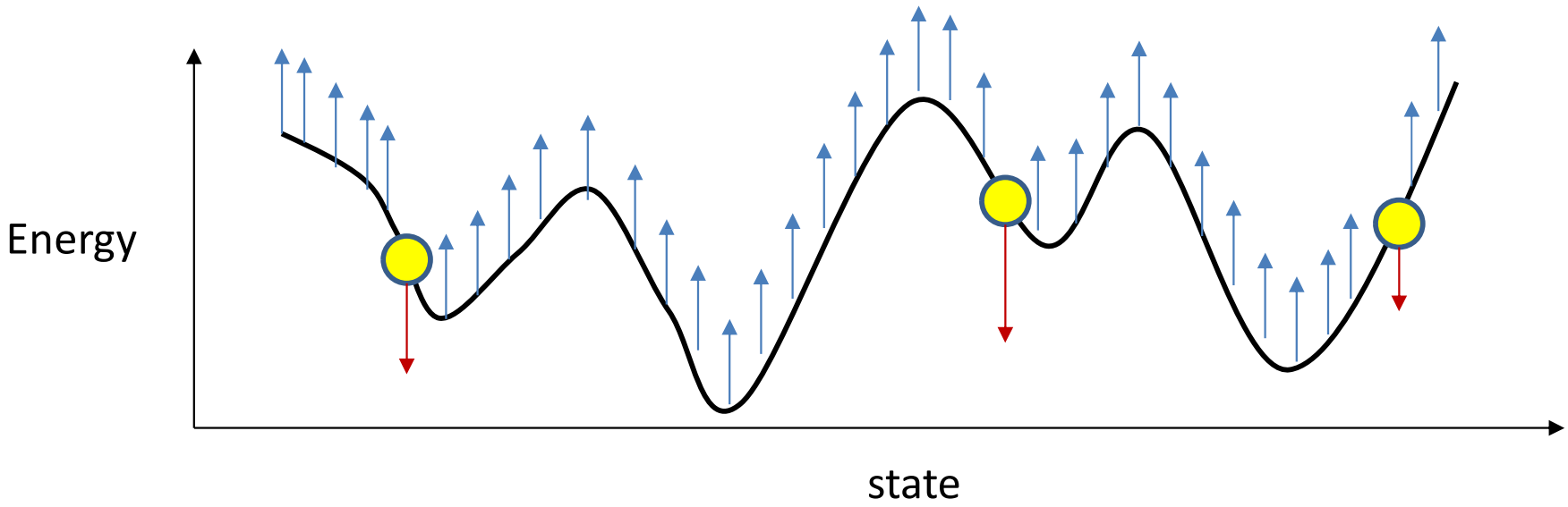
# Overall Training



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

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

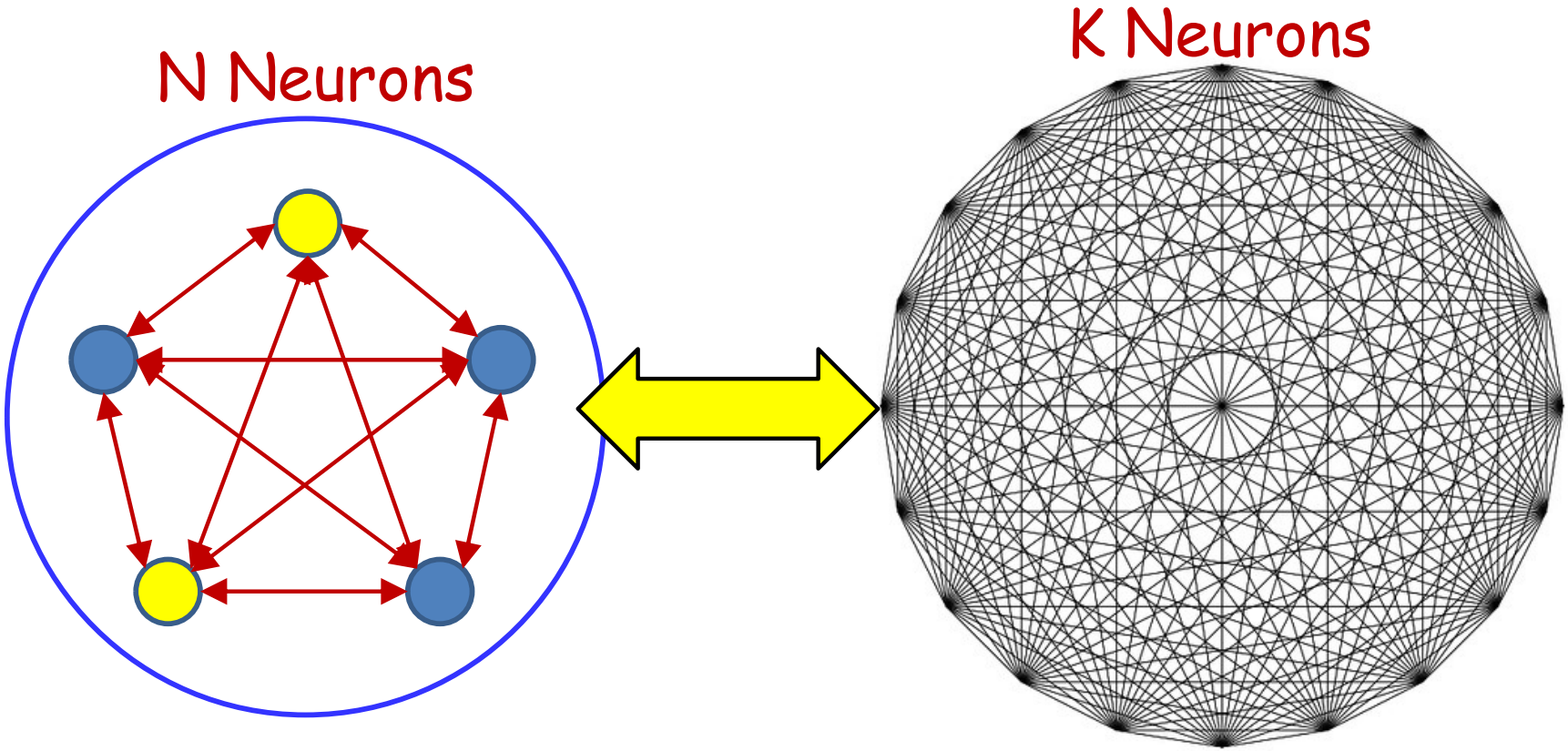
Note the similarity to the update rule for the Hopfield network



# Adding Capacity to the Hopfield Network / Boltzmann Machine

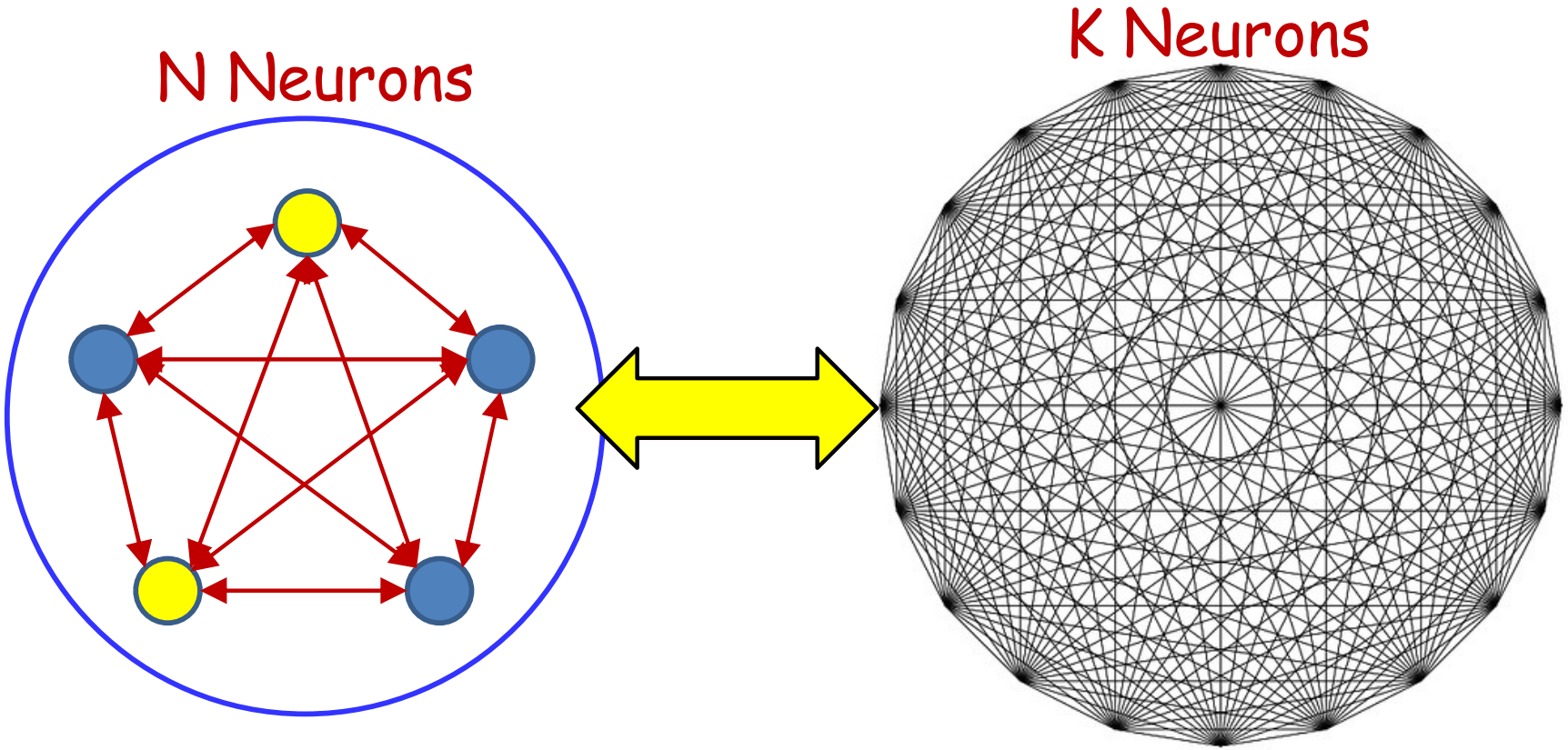
- The network can store up to $N$ $N$-bit patterns
- How do we increase the capacity

# Expanding the network



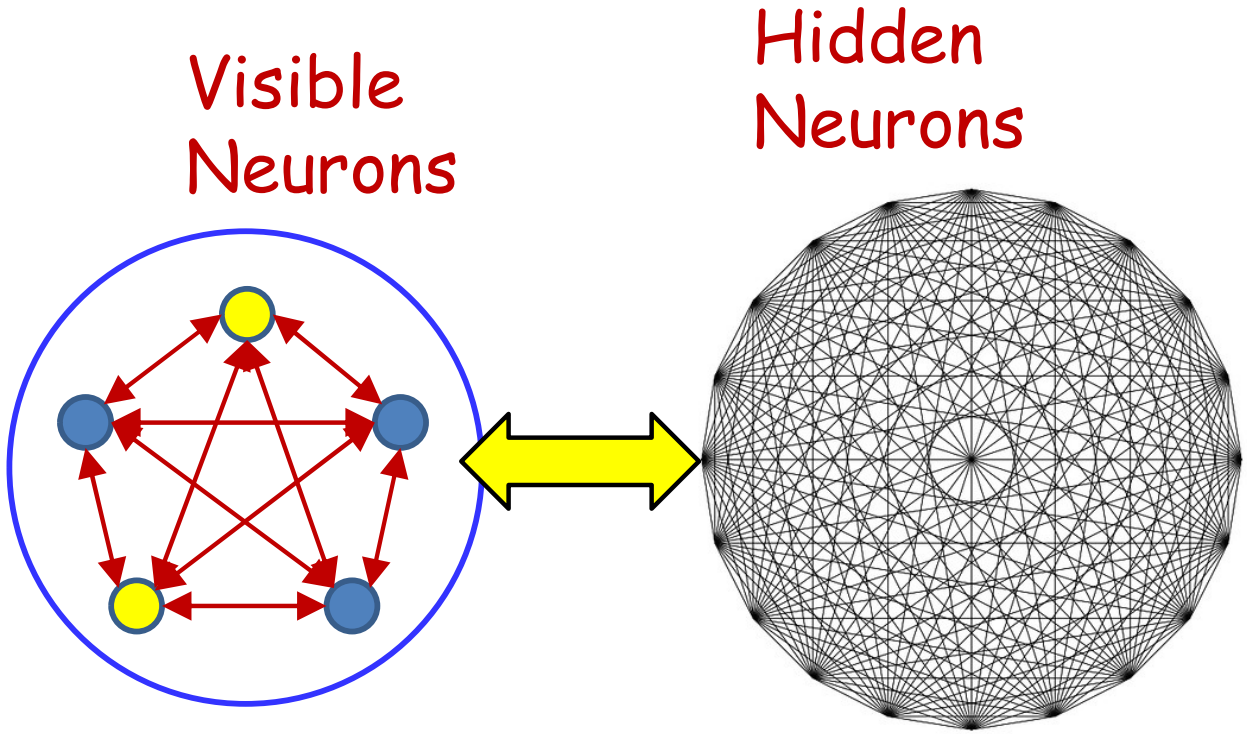

- Add a large number of neurons whose actual values you don't care about!

# Expanded Network



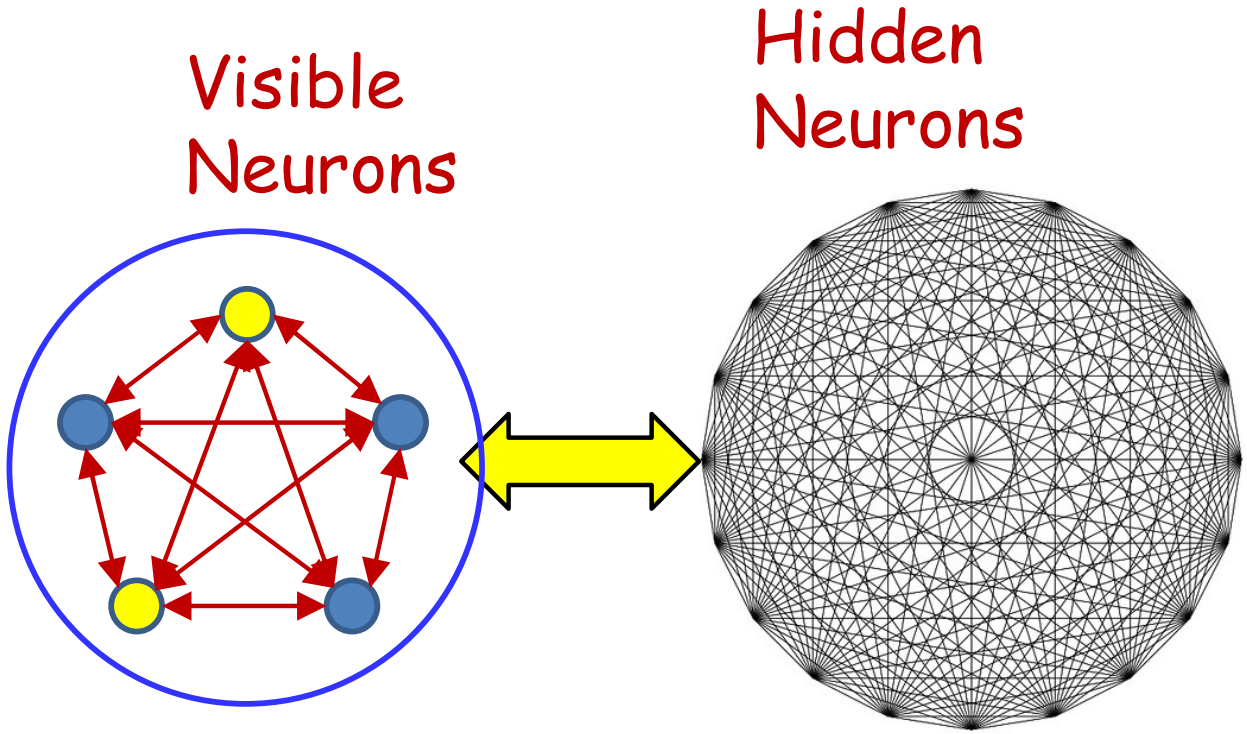

- New capacity: $\sim(N + K)$ patterns
  - Although we only care about the pattern of the first N neurons
  - We're interested in *N-bit* patterns

# 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

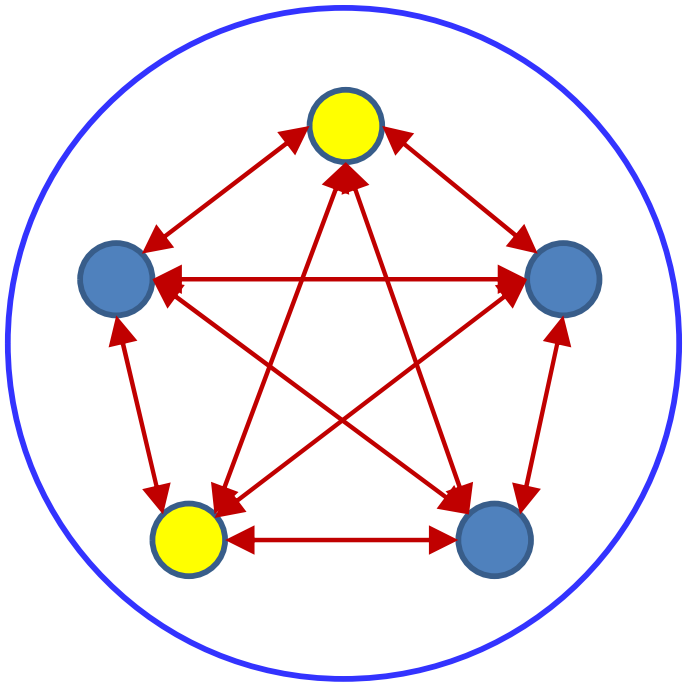



- For a given pattern of *visible* neurons, there are any number of *hidden* patterns ($2^K$)
- Which of these do we choose?
  - Ideally choose the one that results in the lowest energy
  - But that's an exponential search space!

# 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

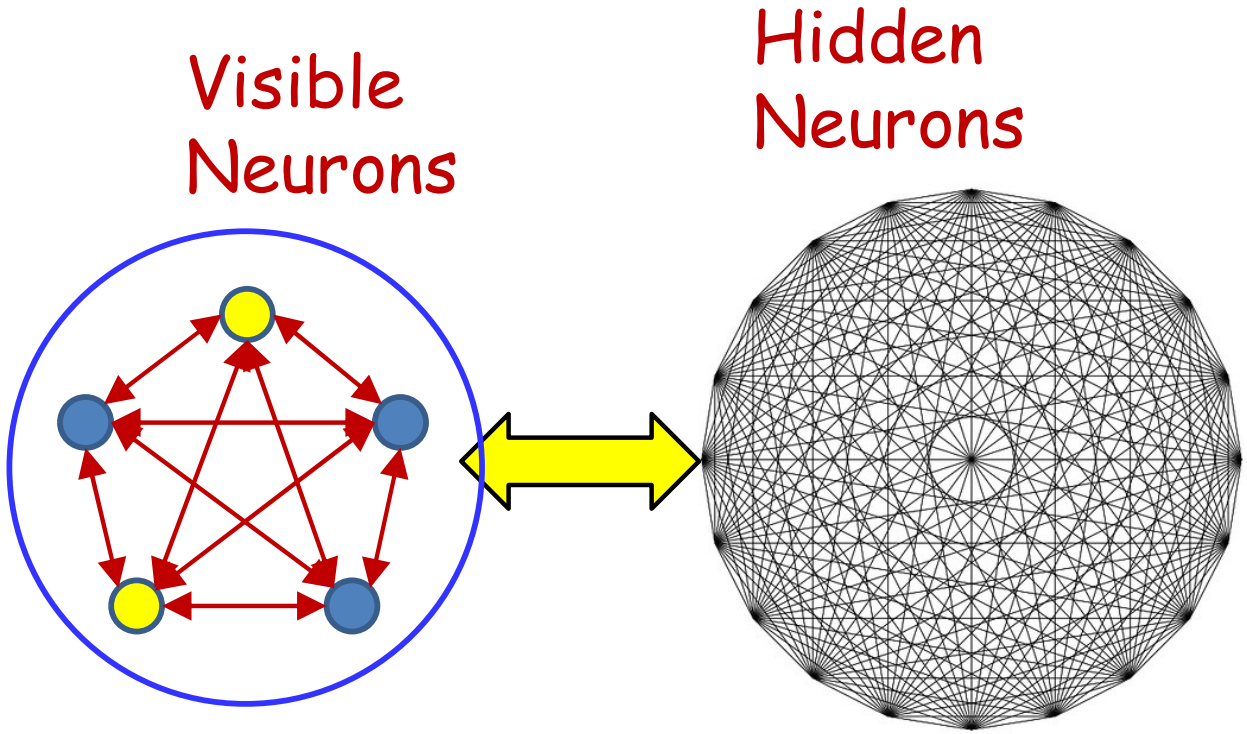


$$\frac{d\langle \log(P(\mathbf{S})) \rangle}{dw_{ij}} = \frac{1}{N}\sum_{S} s_i s_j - \frac{1}{M}\sum_{S' \in \mathbf{S}_{simul}} s'_i s'_j$$

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

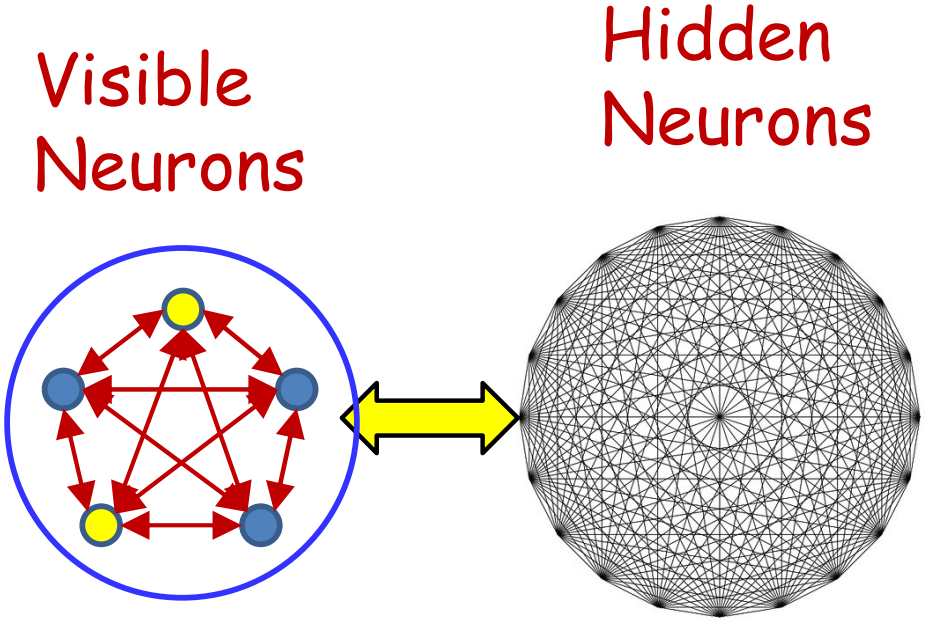
- This basic framework has no hidden units
- Extended to have hidden units

# 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




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

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

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

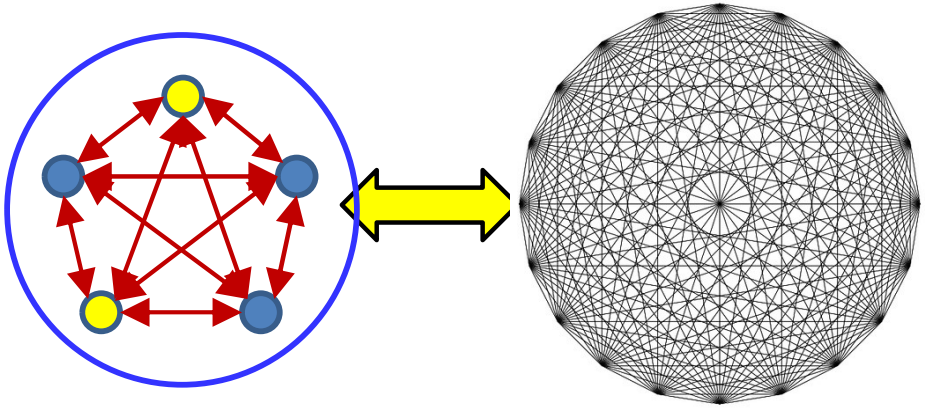
- We are interested in the *marginal* probabilities over *visible* bits
  - We want to learn to represent the visible bits
  - The hidden bits are the "latent" representation learned by the network

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

# With hidden neurons

Visible Neurons

Hidden Neurons



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

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

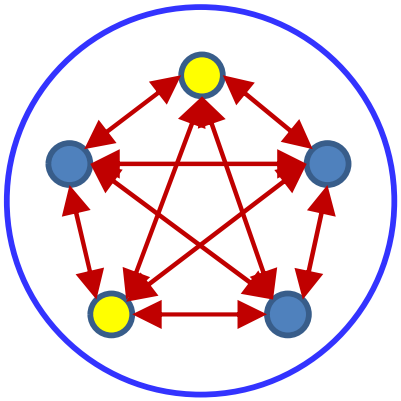
$$P(V) = \sum_{H} P(S)$$

- 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

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

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

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

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

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

- 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

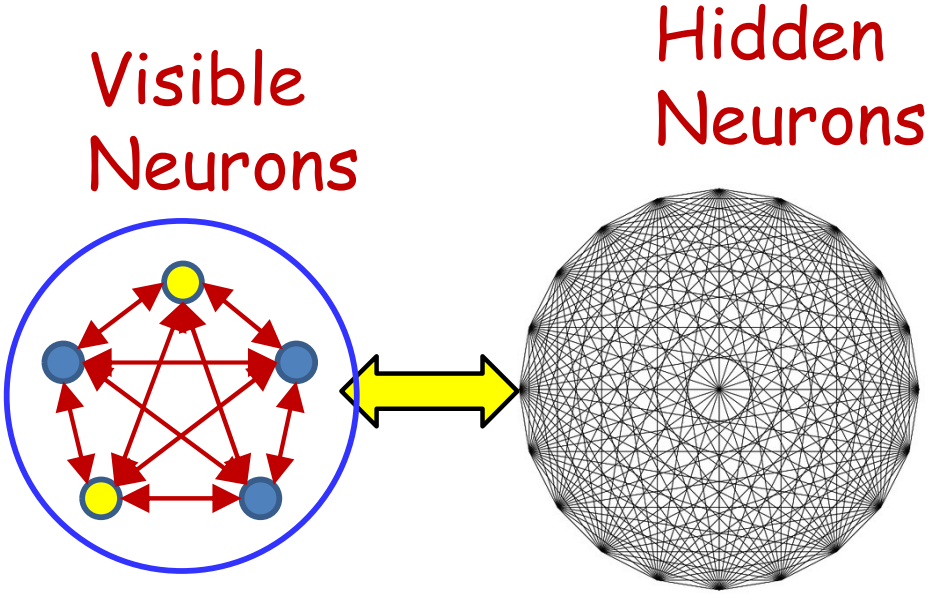
$$\frac{d\mathcal{L}}{dw_{ij}} = \frac{1}{N}\sum_{V\in\mathbf{V}}\sum_{H} P(S|V)s_i s_j - \sum_{S'} P(S')s'_i s'_j$$

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

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

- 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

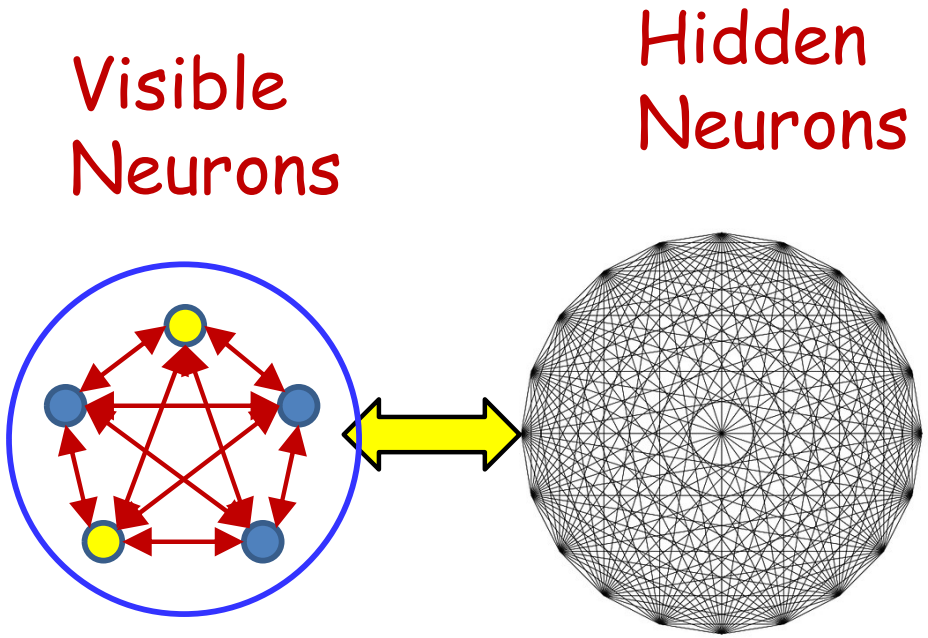



$$P(S) = \frac{exp(-E(S))}{\sum_{S'} exp(-E(S'))}$$

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

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

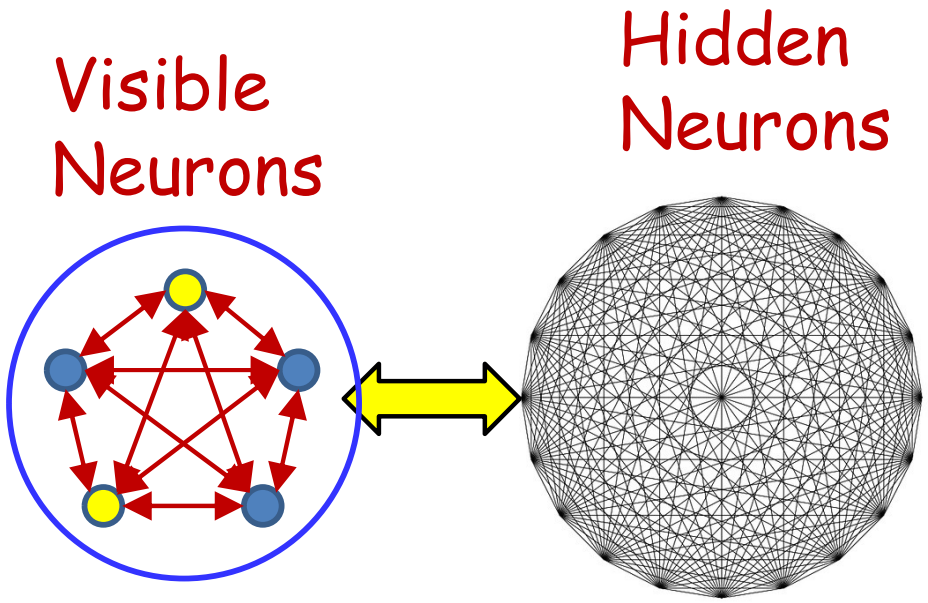
# Step 1




- 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

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

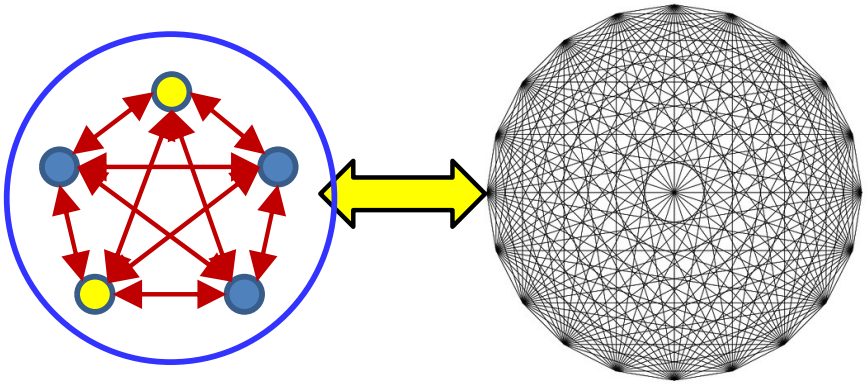
# Step 2




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

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

# Gradients

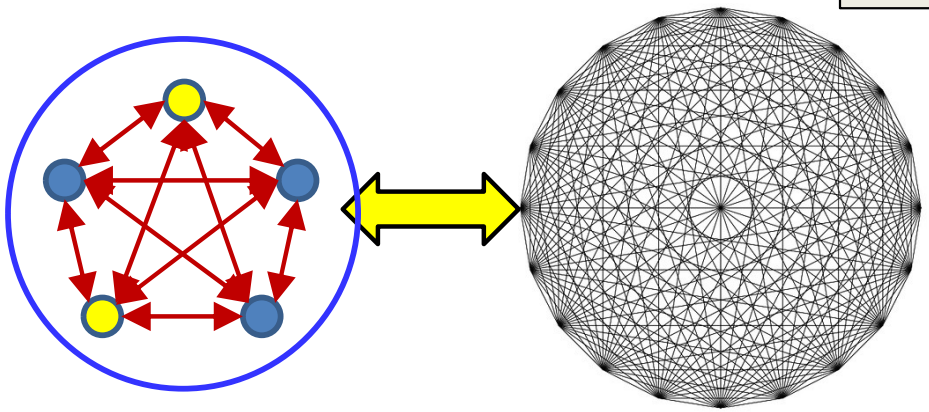


$$\frac{d\langle \log(P(\mathbf{S})) \rangle}{dw_{ij}} = \frac{1}{NK}\sum_{\mathbf{S}} s_i s_j - \frac{1}{M}\sum_{S' \in \mathbf{S}_{simul}} s'_i s'_j$$

- 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

# Poll 4

- The ‘irrelevant bits’ that we used to extend the Hopfield net’s capacity correspond to which components of the Boltzmann machine
  - The hidden neurons
  - The visible neurons
- The training paradigm of Boltzmann machines through gradient descent samples the hidden values to complete patterns
  - True
  - False

# Poll 4

- The ‘irrelevant bits’ that we used to extend the Hopfield net’s capacity correspond to which components of the Boltzmann machine
  - **The hidden neurons**
  - The visible neurons

- The training paradigm of Boltzmann machines through gradient descent samples the hidden values to complete patterns
  - **True**
  - False

# 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

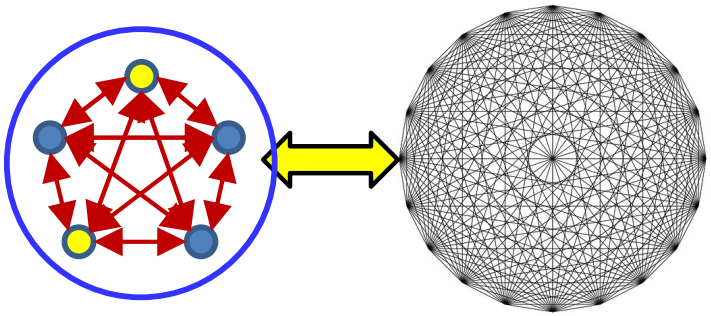
# Boltzmann machines: Overall

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

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

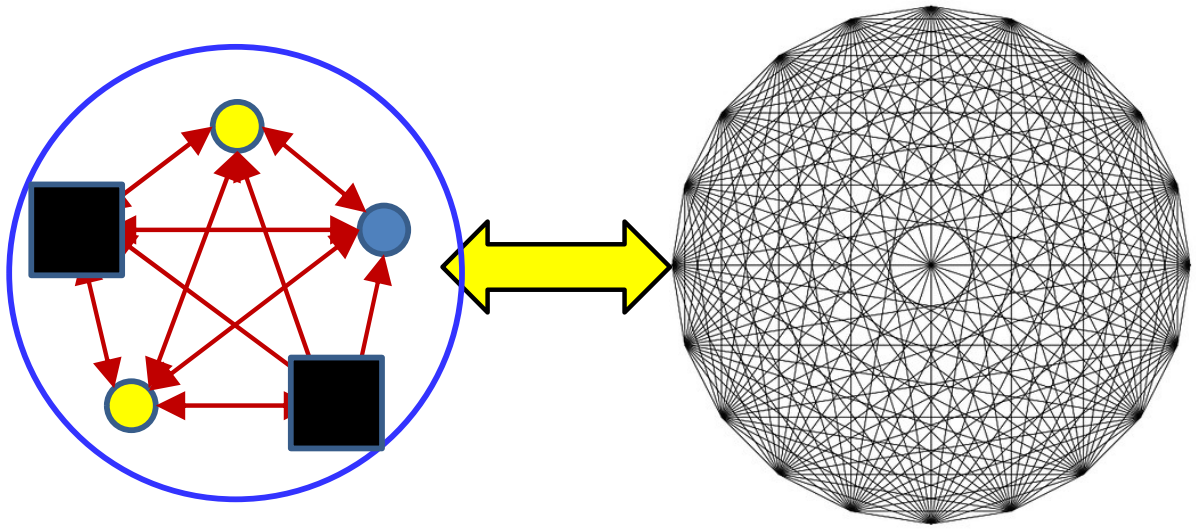
$$\frac{d\langle \log(P(\mathbf{S})) \rangle}{dw_{ij}} = \frac{1}{NK} \sum_{\boldsymbol{S}} s_i s_j - \frac{1}{M} \sum_{S' \in \mathbf{S}_{simul}} s'_i s'_j$$

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



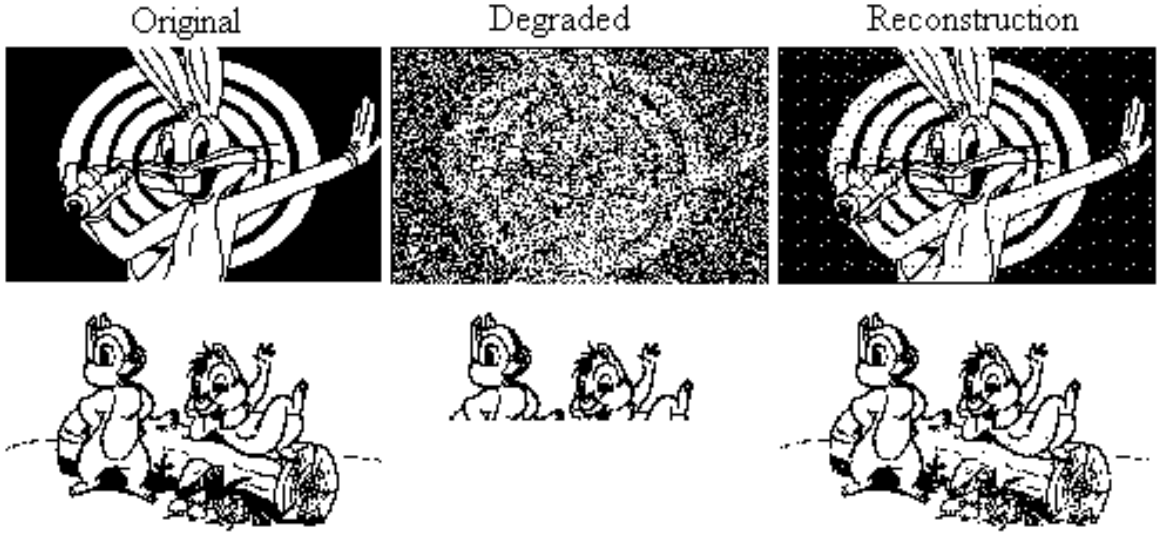
- **Training:** Given a set of training patterns
  - Which could be repeated to represent relative probabilities
- Initialize weights
- Run simulations to get clamped and unclamped training samples
- Compute gradient and update weights
- Iterate

# 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

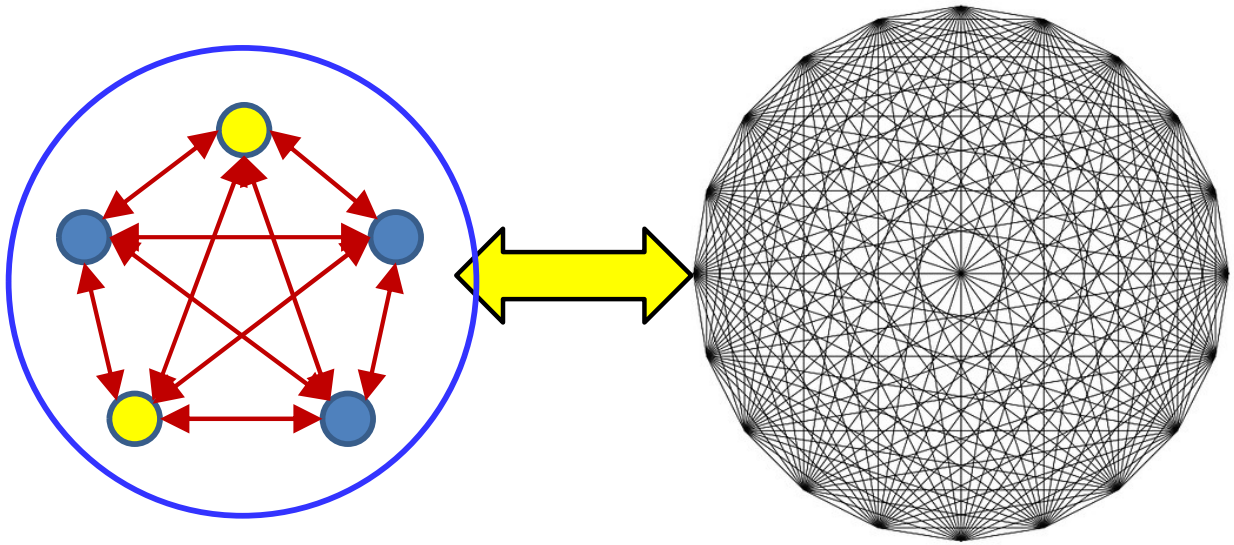
# Applications




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

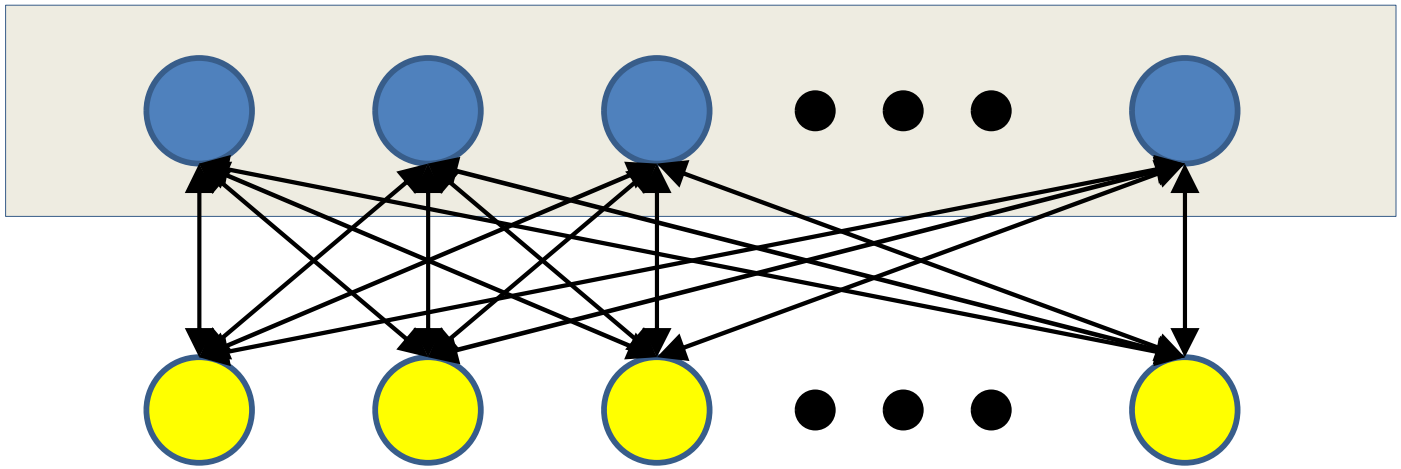


- Training patterns:
  - [$f_1$, $f_2$, $f_3$, …. , class]
  - Features can have binarized or continuous valued representations
  - Classes have “one hot” representation
- Classification:
  - Given features, anchor features, estimate a posteriori probability distribution over classes
    - Or choose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

# Boltzmann machines: Issues

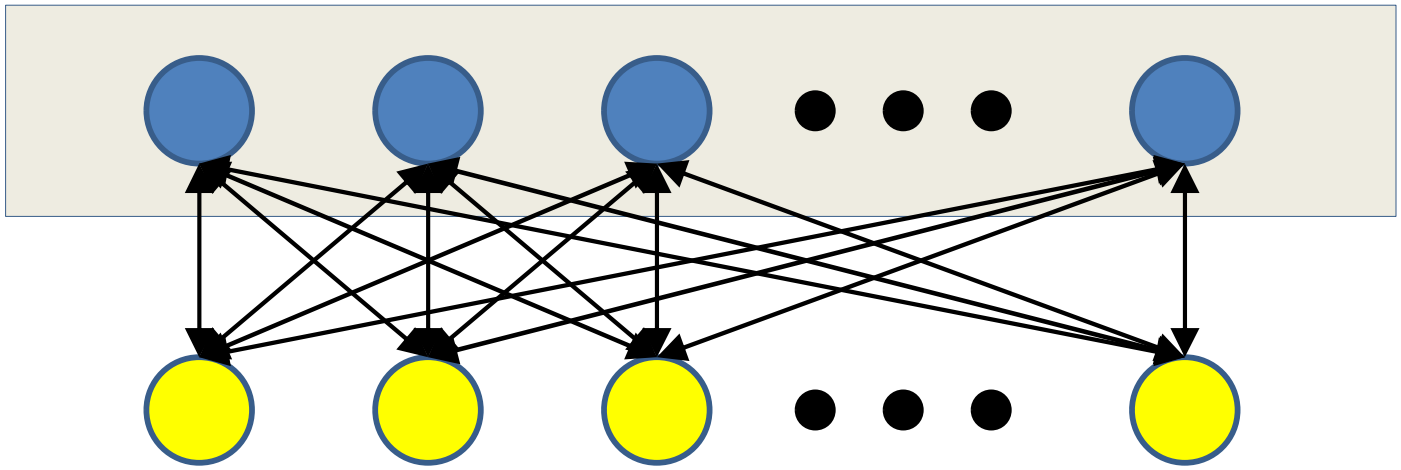
- Training takes for ever
- Doesn’t really work for large problems
  - A small number of training instances over a small number of bits

# 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

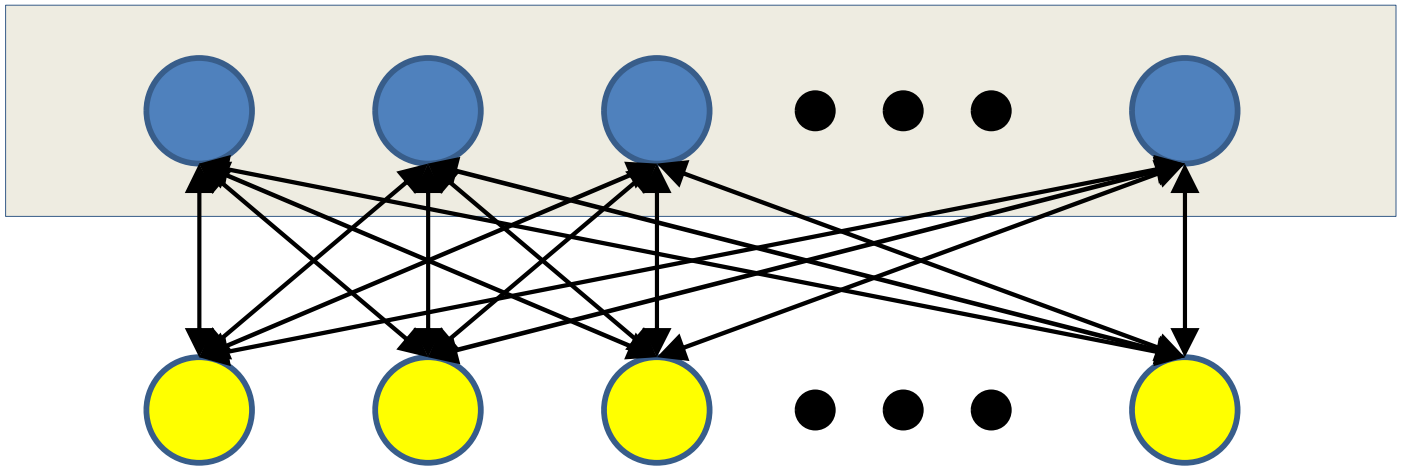



$$z_i = \sum_j w_{ji} s_i + b_i$$

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

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

# Solution: *Restricted* Boltzmann Machines




HIDDEN

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

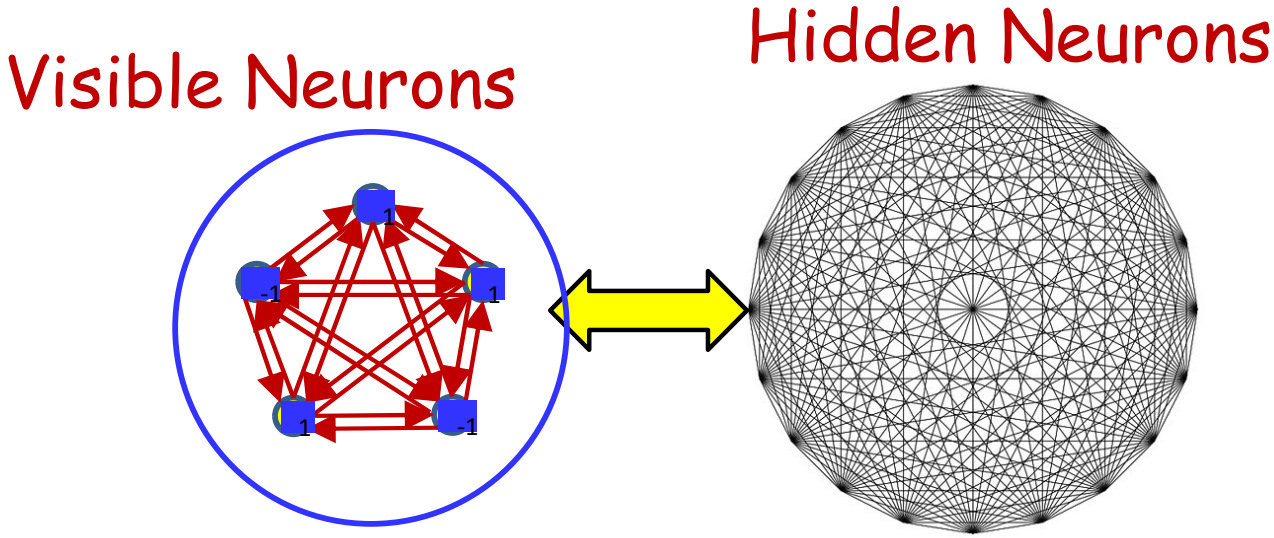
$$P(h_i = 1) = \frac{1}{1 + e^{-z_i}}$$

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

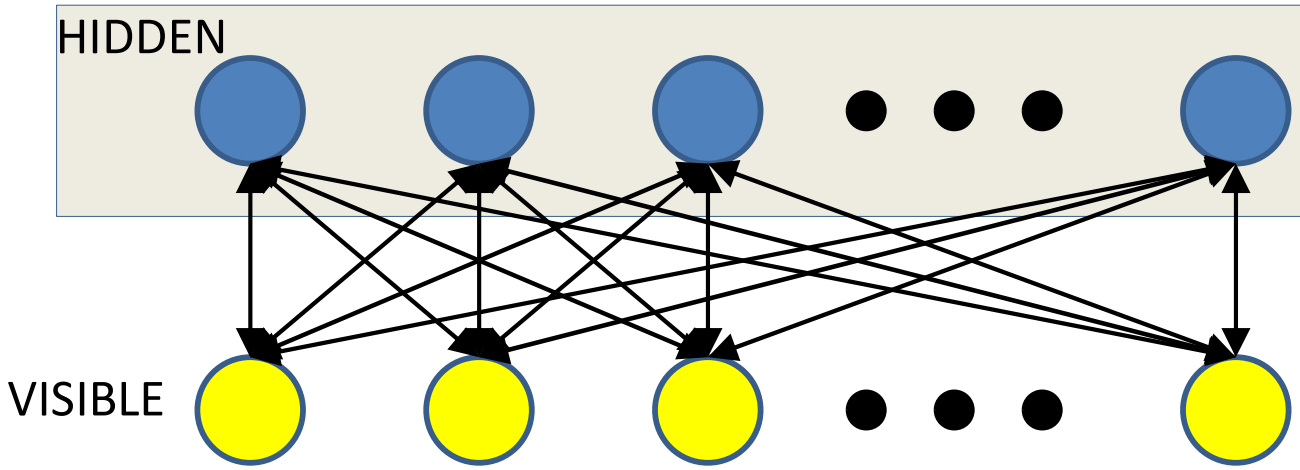



- For each training pattern $V_i$
  - Fix the visible units to $V_i$
  - Let the hidden neurons evolve from a random initial point to generate $H_i$
  - Generate $S_i = [V_i, H_i]$
- Repeat K times to generate synthetic training

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

# Sampling: Restricted Boltzmann machine

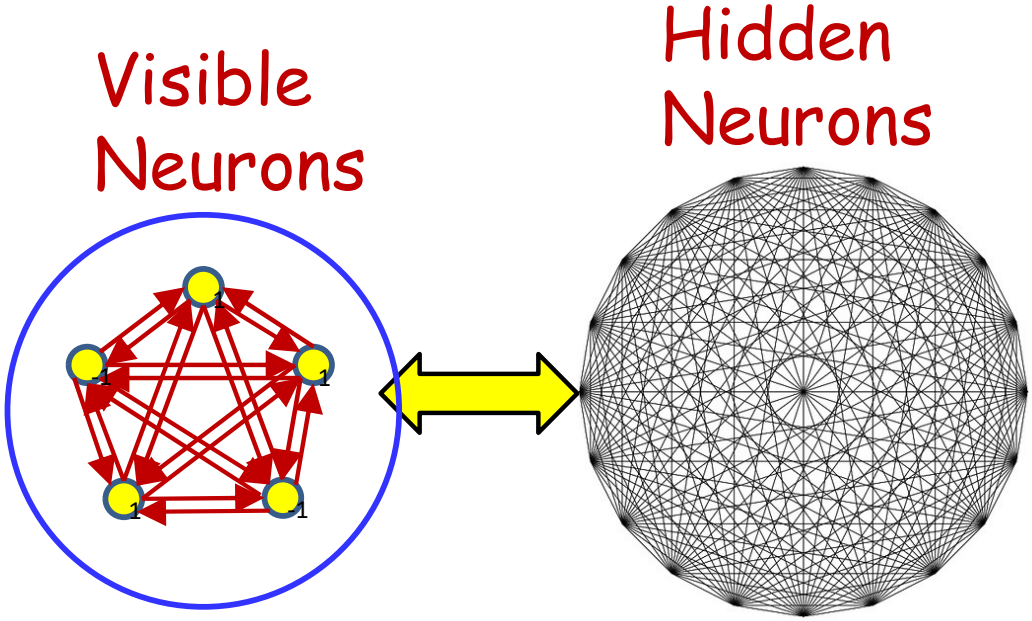
$$z_i = \sum_j w_{ji} v_i + b_i$$




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

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

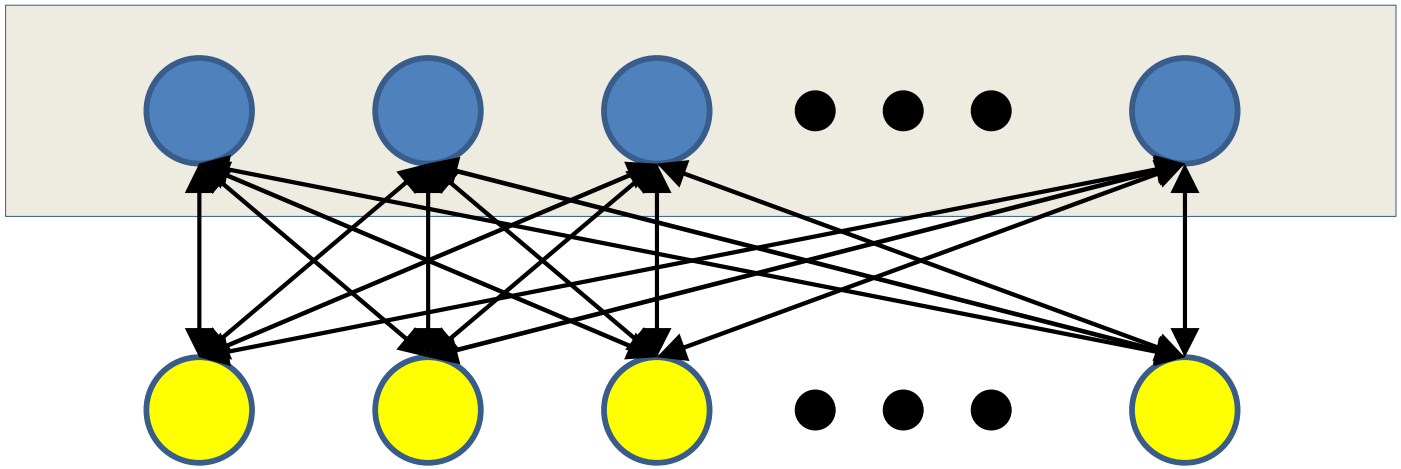
# Recap: Training full Boltzmann machines: Step 2




- 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

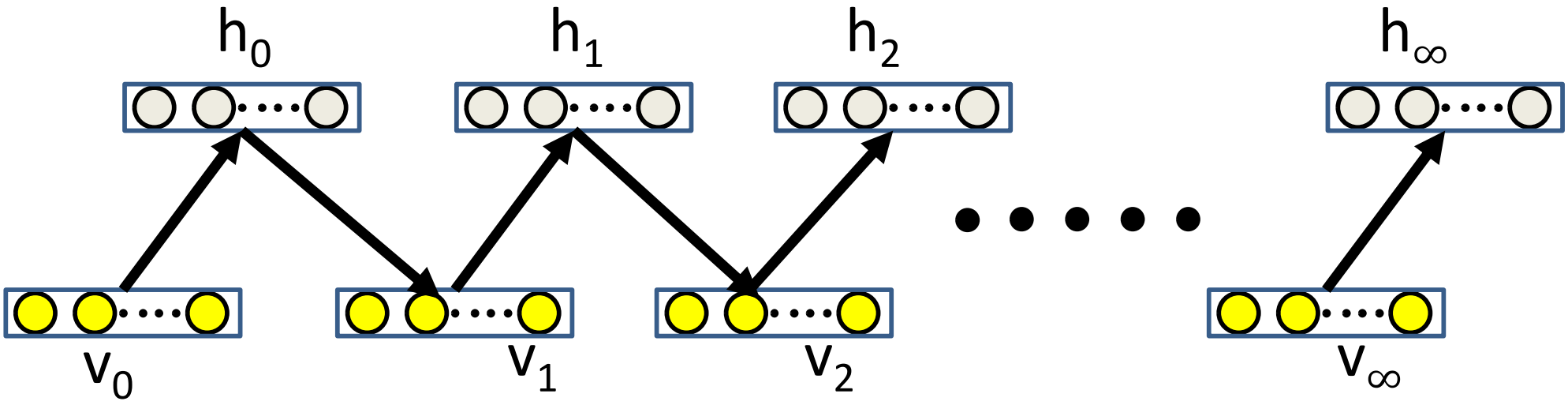



$$z_i = \sum_j w_{ji} v_i + b_i \qquad y_i = \sum_j w_{ji} h_i + b_i$$

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

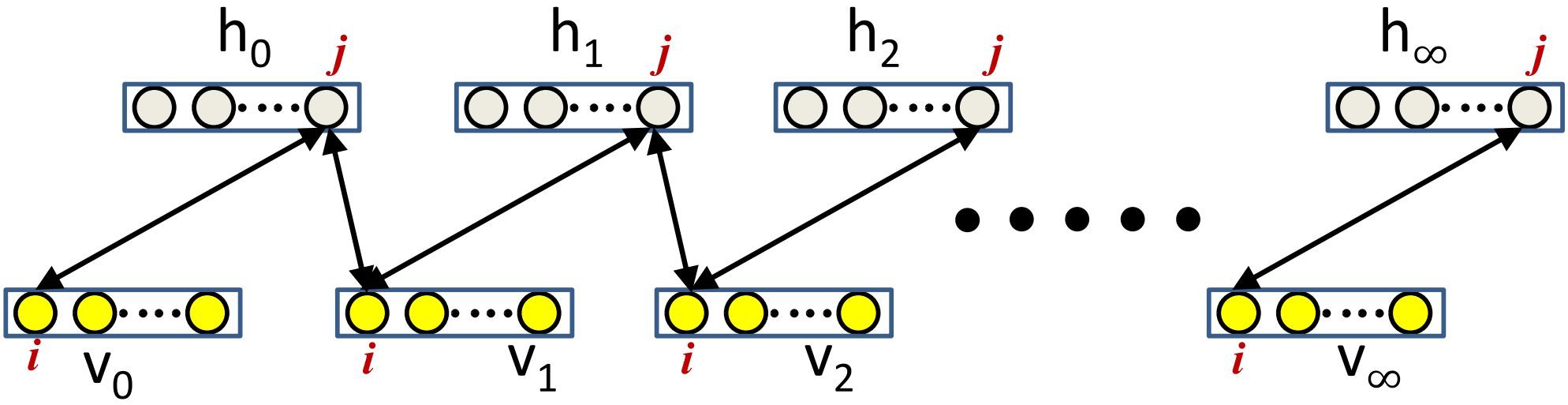
- For each sample:
  - Iteratively sample hidden and visible units for a long time
  - Draw final sample of both hidden and visible units

# 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



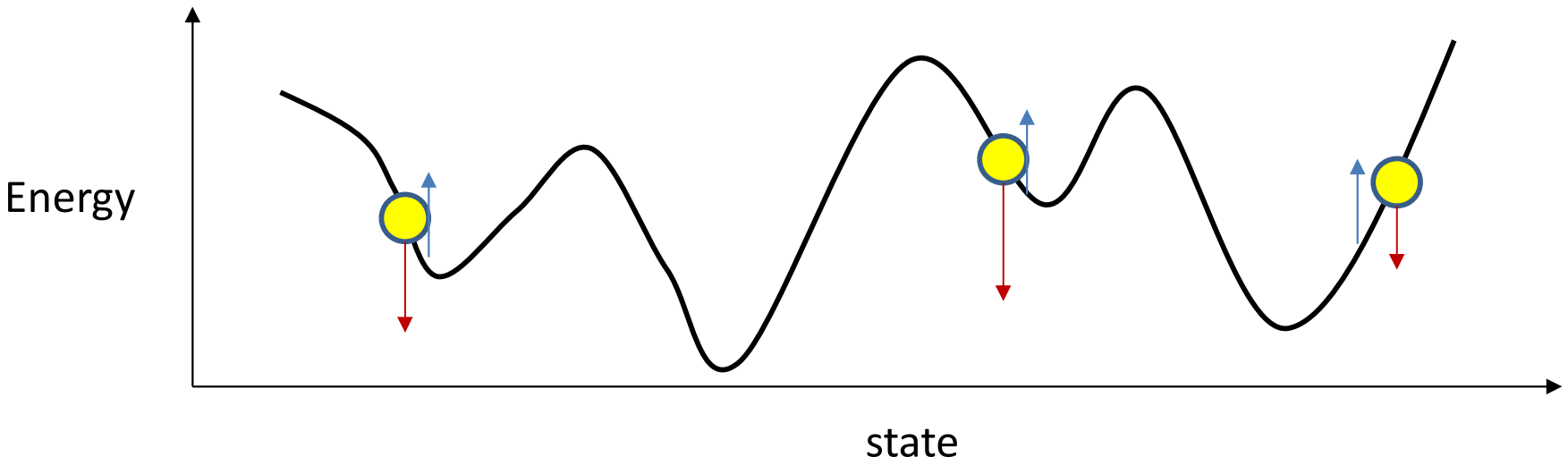

- Gradient (showing only one edge from visible node $i$ to hidden node $j$)

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

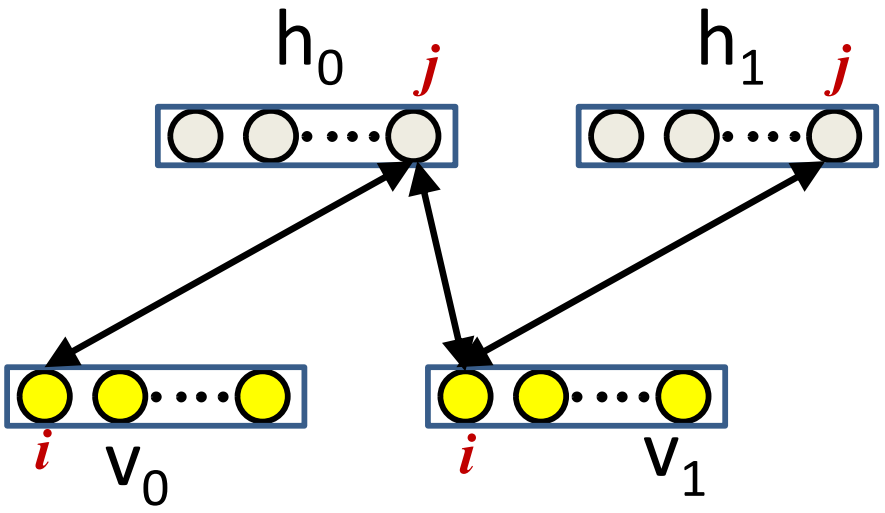
- $<v_i, h_j>$ represents average over many generated training samples

# 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!

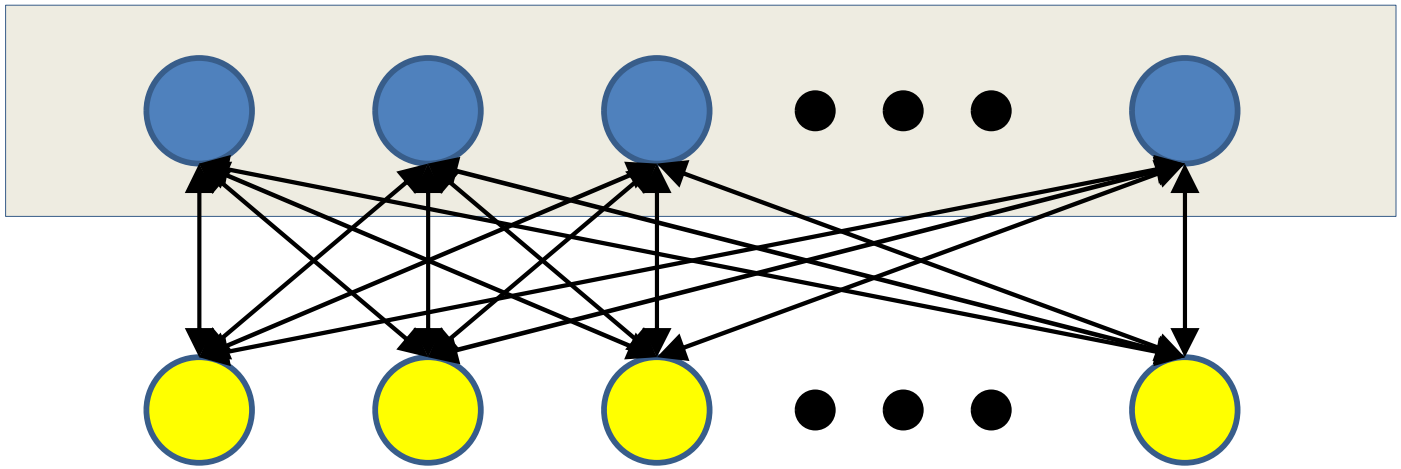
$$\frac{\partial \log p(v)}{\partial w_{ij}} = <v_i h_j>^0 - <v_i h_j>^1$$

- 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

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

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

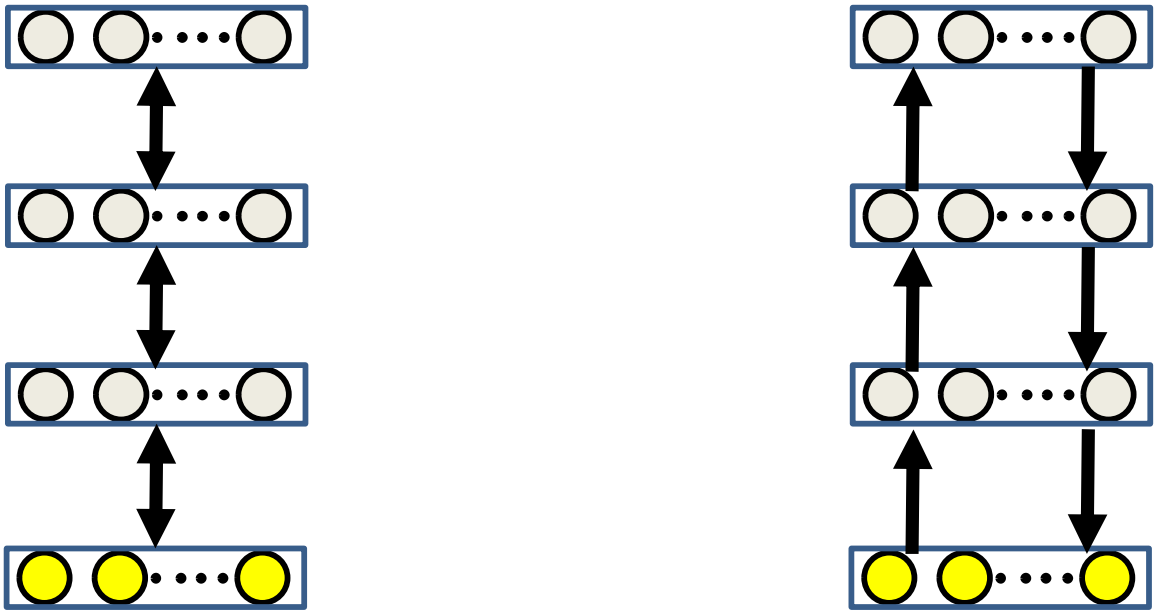
VISIBLE

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

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

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
- …