Boltzmann Machines

Learning by Correlation
in a Hopfield-like Net

Boltzmann Machine

- Proposed by Ackley, Hinton, and Sejnowski, 1985.
- Extends Hopfield model with learning.
- Based on **probabilistic operation.**
- Learning is by **correlation** (sort of Hebbian in character).

COGNITIVE SCIENCE 9, 147–169 (1985)

A Learning Algorithm for Boltzmann Machines*

DAVID H. ACKLEY
GEOFFREY E. HINTON

Computer Science Department
Carnegie-Mellon University

TERRENCE J. SEJNOWSKI

Biophysics Department
The Johns Hopkins University

Spin Glasses and the Ising Model

(useful for understanding Boltzmann machine)

Spin Glass

(http://www.tutorgig.com/ed/Spin_glass)

A *'spin glass*' is a disordered material exhibiting high magnetic **frustration** (inability to remain in a single lowest energy state).

The origin of the behavior can be either a disordered structure (such as that of a conventional, chemical glass) or a disordered magnetic doping in an otherwise regular structure.

Spin Glass

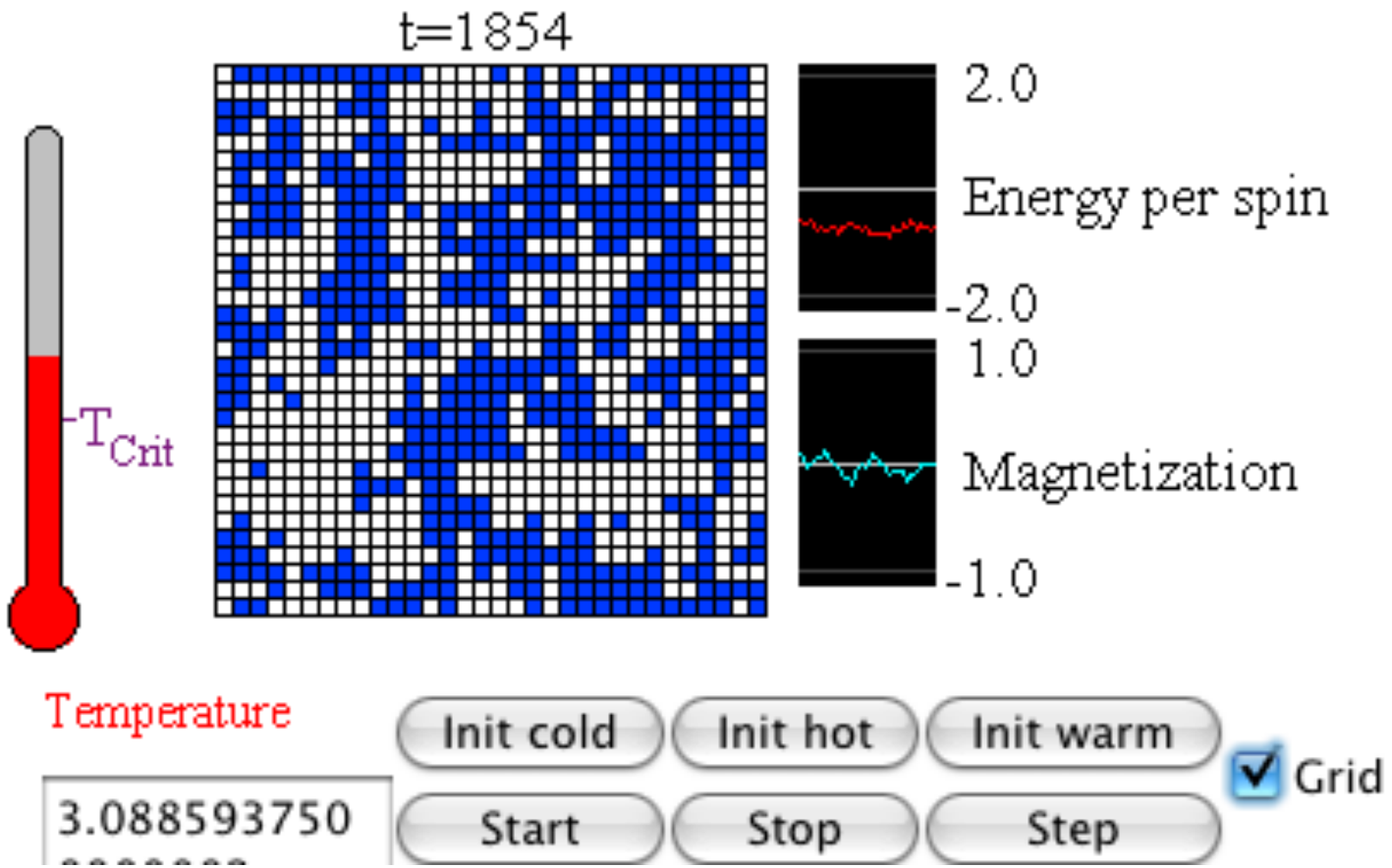
It is the time dependence which distinguishes spin glasses from other magnetic systems. Beginning **above** the spin glass transition temperature, **Tc**, where the spin glass exhibits more typical magnetic behavior, if an external magnetic field is applied and the magnetization is plotted versus temperature, it follows the typical Curie law (in which magnetization is inversely proportional to temperature) until Tc is reached, at which point the magnetization becomes virtually constant (this value is called the field cooled magnetization). This is the onset of the **spin glass phase**.

Ising Model

- The Ising model is a mathematical model that attempts to explain the behavior of spin glasses.
- An array of spin values is assumed, each +1 or -1.
- **An update consists of summing the neighboring spins and setting this spin to some function of that sum.**
- Generally this spin will tend to take on the value of the predominant spin of neighbors.

Ising Model Applet

(http://physics.ucsc.edu/~peter/ising/ising.html)



Ising Demo Information

- The energy is calculated from the formula $E= -\Sigma_{ij} Si\ Sj$ where i,j is over all pairs of nearest neighbors on the lattice.
- At infinite temperature the energy per spin (E/N, where $N = L^2$ is the number of spins) is zero.
- At zero temperature, most of the spins are parallel and the energy per spin is -2.
- The critical temperature of the two dimensional Ising model is T_{crit}=2/ln(1+sqrt 2) = 2.269 .
- The magnetization is simply the mean of all spins.

Ising Demo Information

- At temperatures well above the critical temperatures, the spin arrangement converges to a nearly random arrangement, independent of the starting state: "Init cold", "Init warm" or "Init hot", and fluctuates quickly. We say that, above the critical temperature, there is **a single thermodynamic state** and this has zero magnetization. The spin arrangement is truly random at infinite temperature.

- If you start below the critical temperature with "Init cold" (i.e. all the Si=-1) you will see that just a few small cluster of blue (i.e. Si=1) spins appear, and there is a non-zero (negative) magnetization. If we had started the simulation with all the Si=1 (blue) then there would have been a net positive magnetization. We see that, below the critical temperature, there are two thermodynamic states (the "up spin" state with positive magnetization and the "down spin" state negative magnetization) and the system stays in one or the other depending on how the spins are initialized.

Ising Demo Information

- If you start below the critical temperature with "Init hot" or "Init warm" then you see that the system initially cannot make up its mind whether to go into the "up spin" or "down spin" state. Large clusters of each spin form. Eventually, if you let the simulation run for a long time, one of the states will win. Which one wins depends on the random thermal fluctuations. There is equal probability for it to be the "up spin" or "down spin" state.

- For temperatures near the transition temperature, there are large clusters of spins with the same orientation, which fluctuate only very slowly. This is because the "correlation length" of an infinitely large system diverges at the critical point.

Smaller Ising Model Applet

(http://www.phy.syr.edu/courses/ijmp_c/Ising.html)

Inverse of temperature

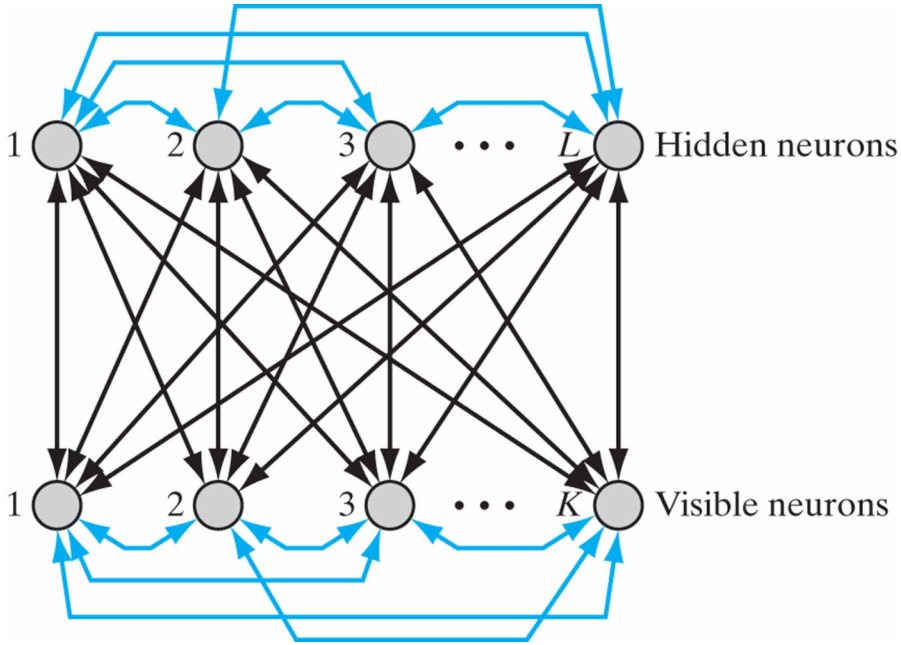
Boltzmann Machine Structure

- The Boltzmann Machine is like a Hopfield network, in which
- the neurons are divided into two subsets:
  - **visible**, which may be further divided into:
    - input
    - output
  - **hidden**
- As with the Hopfield model, the weights are symmetric.

Boltzmann Machine Structure

Haykin Figure 11.5 Architectural graph of Botzmann machine; K is the number of visible neurons, and L is the number of hidden neurons. The distinguishing features of the machine are:

1. The connections between the visible and hidden neurons are symmetric.
2. The symmetric connections are extended to the visible and hidden neurons.



Boltzmann Machine Operation

- There are two modes of operation:
  - clamped mode (used only for learning)
  - free mode
- In **clamped** mode, the input and output of **visible neurons are held fixed**, while the hidden neurons are allowed to vary.
- In **free-running** mode, **only the inputs are held fixed** and all other neurons are allowed to vary.

Weights & Energy

$$E = -\sum_{i<j} w_{ij}\, s_i\, s_j + \sum_i \theta_i\, s_i$$

Where:

- w_{ij} is the connection strength between unit j and unit i.
- s_i is the state, $s_i \in \{0, 1\}$, of unit i.
- θ_i is the threshold of unit i.

The connections in a Boltzmann machine have two restrictions:

- $w_{ii} = 0 \qquad \forall i$. (No unit has a connection with itself.)
- $w_{ij} = w_{ji} \qquad \forall i, j$. (All connections are symmetric.)

Probabilistic Firing

When unit i is given the opportunity to update its binary state, it first computes its total input, z_i, which is the sum of its own bias, b_i, and the weights on connections coming from other active units:

(Equation 1)

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

where w_{ij} is the weight on the connection between i and j, and s_j is 1 if unit j is on and 0 otherwise. Unit i then turns on with a probability given by the logistic function:

(Equation 2)

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

(Equations from Scholarpedia article)

Probability Distribution of States Depends on Energy

If the units are updated sequentially in any order that does not depend on their total inputs, the network will eventually reach a Boltzmann distribution (also called its equilibrium or stationary distribution) in which the probability of a state vector, v , is determined solely by the "energy" of that state vector relative to the energies of all possible binary state vectors:

$$P(\mathbf{v}) = e^{-E(\mathbf{v})} / \sum_{\mathbf{u}} e^{-E(\mathbf{u})} \qquad \text{(Equation 3)}$$

As in Hopfield nets, the energy of state vector $\mathbf{v}$ is defined as

$$E(\mathbf{v}) = -\sum_{i} s_i^{\mathbf{v}} b_i - \sum_{i<j} s_i^{\mathbf{v}} s_j^{\mathbf{v}} w_{ij} \qquad \text{(Equation 4)}$$

where $s_i^{\mathbf{v}}$ is the binary state assigned to unit i by state vector $\mathbf{v}$.

(Equations from Scholarpedia article)

Escaping from Local Minima

If the weights on the connections are chosen so that the energies of state vectors represent the cost of those state vectors, then the stochastic dynamics of a Boltzmann machine can be viewed as a way of escaping from poor local optima while searching for low-cost solutions. The total input to unit i, z_i, represents the difference in energy depending on whether that unit is off or on, and the fact that unit i occasionally turns on even if z_i is negative means that the energy can occasionally increase during the search, thus allowing the search to jump over energy barriers.

The search can be improved by using simulated annealing. This scales down all of the weights and energies by a factor, T, which is analogous to the temperature of a physical system. By reducing T from a large initial value to a small final value, it is possible to benefit from the fast equilibration at high temperatures and still have a final equilibrium distribution that makes low-cost solutions much more probable than high-cost ones. At a temperature of 0 the update rule becomes deterministic and a Boltzmann machine turns into a Hopfield Network

Controlling T: Simulated Annealing

- We can think of the decrease in T as “cooling” the network.
- The exact values of T can be controlled by a schedule or a function of “time”.

Annealing Schedule

- The annealing schedule determines the temperature T as a function of the step of the algorithm.

- Example:

  $$T = T_0/(1 + \log k)$$

  where k is the step number and T_0 is an initial temperature.

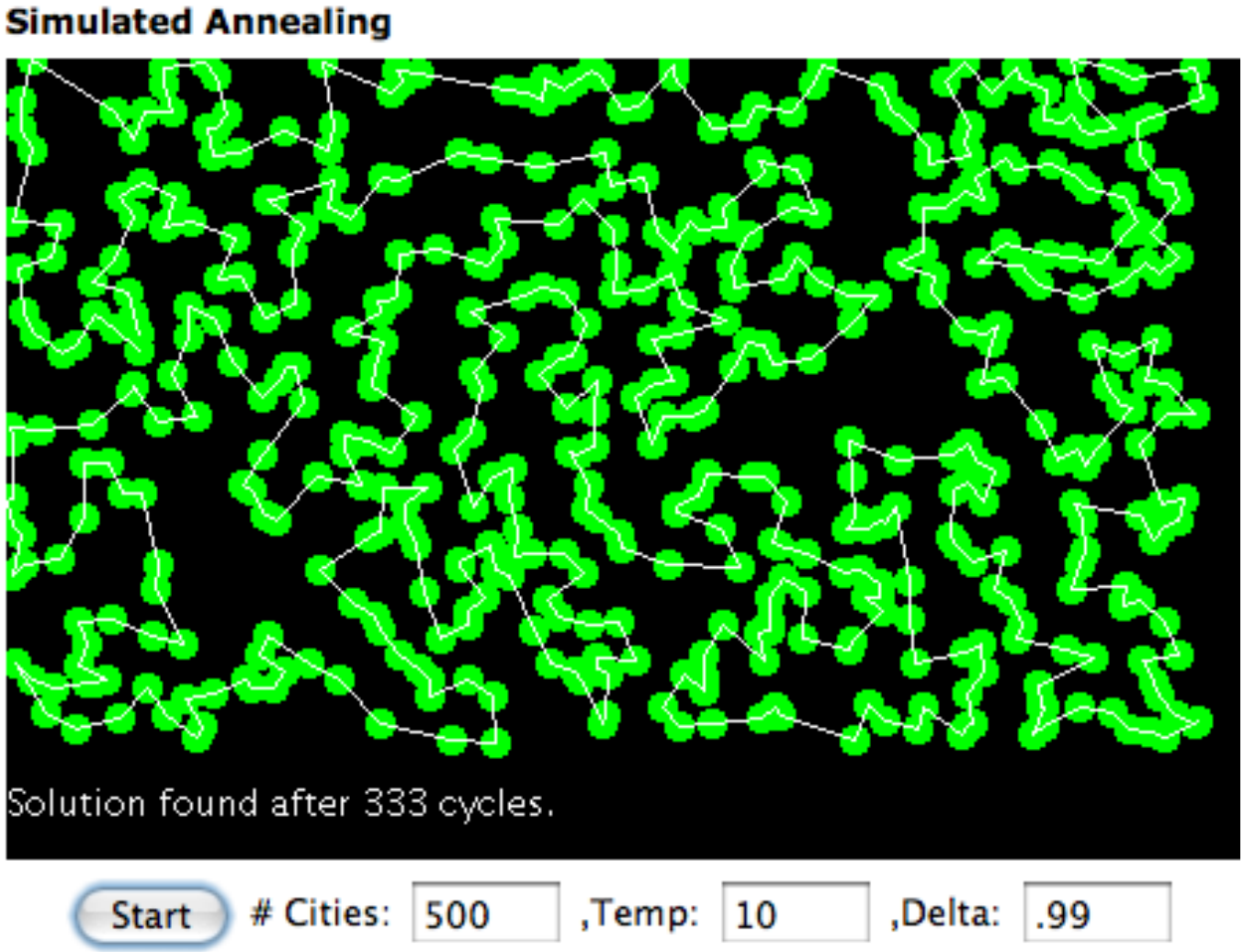
History of Simulated Annealing

- SA was first proposed in 1983 as a method for optimizing wire-routing on VLSI chips (an NP-hard problem)
  in a widely-celbrated paper by
  Kirkpatrick, Gelatt, and Vecchi.

- SA is now used as a general way to avoid local minima in optimization problems.

TSP via Simulated Annealing

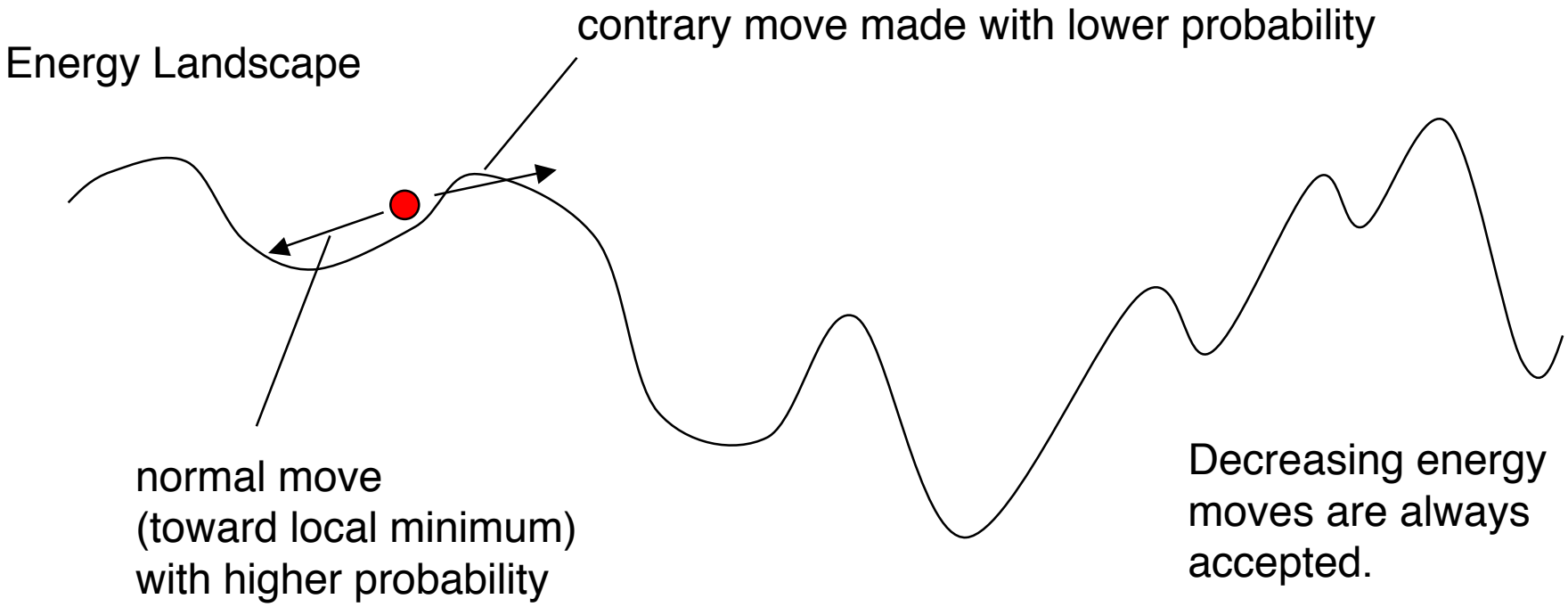
http://www.heatonresearch.com/articles/64/page1.html



Role of Annealing in Stabilization

- Generally, move in direction of decreasing energy.
- **Occasionally, accept a move that increases energy.**
- This will be done with high probability at first, but **lower probability** as annealing progresses.

Role of Annealing in Stabilization



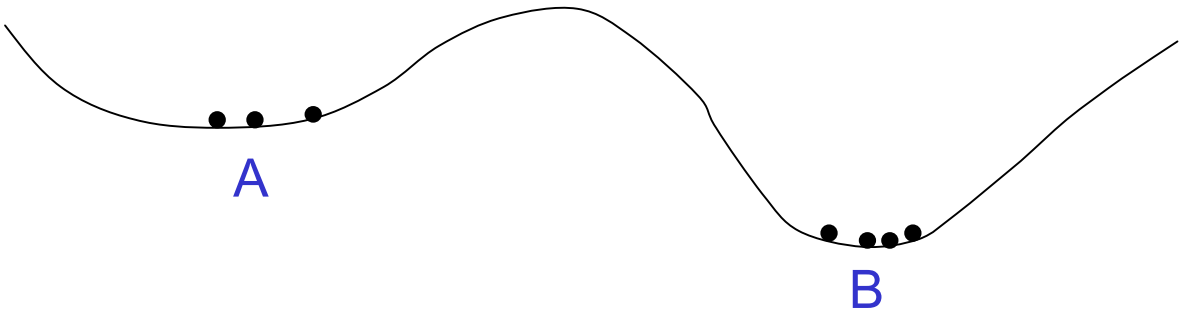
The probability of making a contrary move is inversely proportional to the energy increase and to the temperature (higher probability earlier in the annealing schedule).

How temperature affects transition probabilities (slide from Hinton)

High temperature transition probabilities

$$p(A \rightarrow B) = 0.2$$

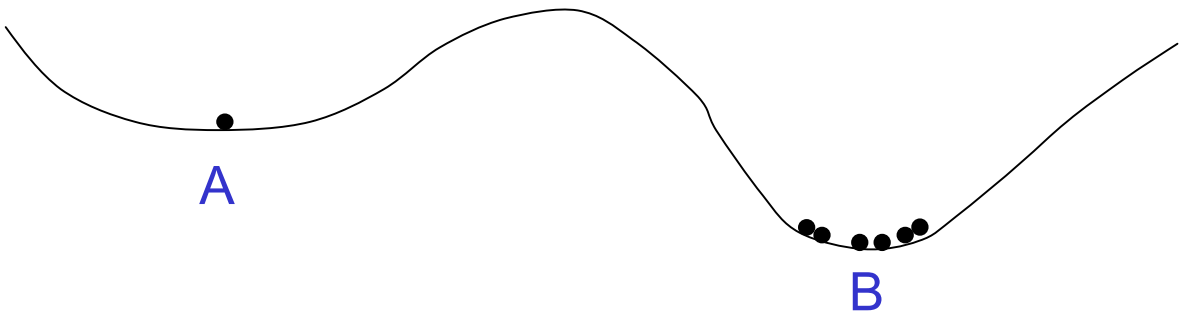
$$p(A \leftarrow B) = 0.1$$



Low temperature transition probabilities

$$p(A \rightarrow B) = 0.001$$

$$p(A \leftarrow B) = 0.000001$$



Annealing Advantage

- At high temperature the ***transition*** probabilities for uphill jumps are much greater.
- At low temperature the ***equilibrium*** probabilities of low energy states are much higher than the equilibrium probabilities of high energy ones.

Energy-Based Simulation

- As we know from Hopfield theory, making a single transition according to the activation function will decrease the energy.
- So we can simply decide to "flip" a neuron based on whether the flip lowers the energy (defined as $-\Sigma \Sigma w_{ij} y_i y_j$).
- The decrease in energy for flipping the i^{th} neuron is equivalent to:

$$\Delta E_i = E_i(-1) - E_i(+1) = \Sigma_j w_{ij} y_j$$

Energy-Based Simulation

- To include the annealing temperature:
  - If a flip **lowers** the energy, accept it.
  - If a flip **raises** the energy by ΔE, accept it probabilistically.

Energy-Based Simulation

- A similar technique was originally used in the famous Metropolis, Rosenbluth, Teller equation of state calculations in statistical mechanics ("spin-glass" model).

- This is generally called the "**Metropolis algorithm**".

Boltzmann Distribution

- The name of the machine derives from the fact that, at steady state, if s and t are two states with energies E_s and E_t respectively, then the probabilities of being in those states P[s] vs. P[t] satisfy

$$P[s]/P[t] = \exp((E_t - E_s)/T)$$

where T is the temperature. This is known as the “Boltzmann distribution” or “Boltzmann-Gibbs distribution”.

Learning in the Boltzmann Machine

Multiple Simulations Per Sample

- Suppose we set the input and output neurons according to a specific **sample**.
- We then anneal the network.
  The final state reached is not necessarily unique, due to the probabilistic moves are made along the way.
- We can observe, over **several** simulation steps, which neurons' outputs are **correlated** at the ends, represented as a **correlation** $\rho_{ij} = E[y_i\, y_j]$, the expected (average) value of the ***product*** of the outputs of neurons i and j.

Learning Rule

- For a given I/O sample, let ρ_{ij}^{+} be the correlation value when the network is run in **clamped** mode, and ρ_{ij}^{-} be the correlation value when the network is run in **free** mode.

- The Boltzmann learning rule is

$$\Delta w_{ij} = \eta(\rho_{ij}^{+} - \rho_{ij}^{-})$$

  where η is the learning rate.

- In other words, whether weights are changed depends on the difference between the correlations in clamped vs. free mode.

Derivation of Learning Rule

Boltzmann distribution (also called its equilibrium or stationary distribution) in which the probability of a state vector, v , is determined solely by the "energy" of that state vector relative to the energies of all possible binary state vectors:

$$P(\mathbf{v}) = e^{-E(\mathbf{v})} / \sum_{\mathbf{u}} e^{-E(\mathbf{u})} \qquad \text{(Equation 3)}$$

By differentiating Equation 3 and using the fact that $\partial E(\mathbf{v}) / \partial w_{ij} = -s_i^{\mathbf{v}} s_j^{\mathbf{v}}$ it can be shown that

$$\left\langle \frac{\partial \log P(\mathbf{v})}{\partial w_{ij}} \right\rangle_{\text{data}} = \langle s_i s_j \rangle_{\text{data}} - \langle s_i s_j \rangle_{\text{model}} \qquad \text{(Equation 5)}$$

where $\langle \cdot \rangle_{\text{data}}$ is an expected value in the data distribution and $\langle \cdot \rangle_{\text{model}}$ is an expected value when the Boltzmann machine is sampling state vectors from its equilibrium distribution at a temperature of 1.

(Equations from Scholarpedia article)

On the Learning Rule (wikipedia)

Remarkably, this learning rule is fairly **biologically plausible** because the only information needed to change the weights is provided by "local" information.

That is, the connection (or synapse biologically speaking) does not need information about anything other than the two neurons it connects.

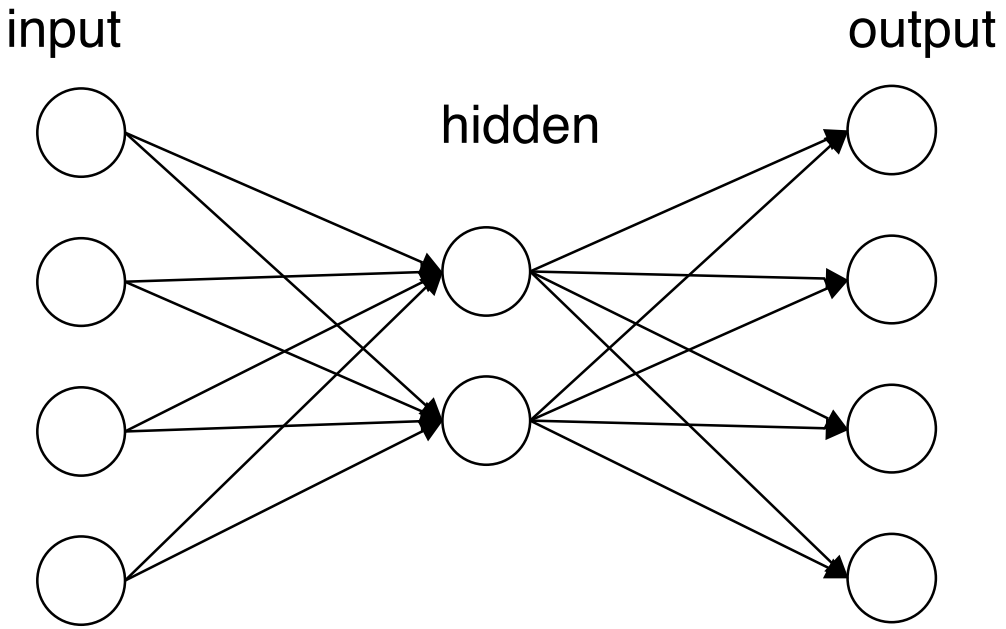
This is far more biologically realistic than the information needed by a connection in many other neural network training algorithms, such as backpropagation.

Learning Algorithm

- For each I/O sample:
  - Clamped phase:
    - Clamp the data vector on the visible units.
    - Let the hidden units reach thermal equilibrium at a temperature of 1 (may use annealing to speed this up).
    - Accumulate ρ_{ij}^{+} by sampling $y_i\ y_j$ for all pairs of units.
  - Free phase:
    - Do not clamp any of the units.
    - Let the whole network reach thermal equilibrium at a temperature of 1.
    - Accumulate ρ_{ij}^{-} by sampling $y_i\ y_j$ for all pairs of units.
  - Update each weight by adding $\eta(\rho_{ij}^{+} - \rho_{ij}^{-})$.

Boltzman Example

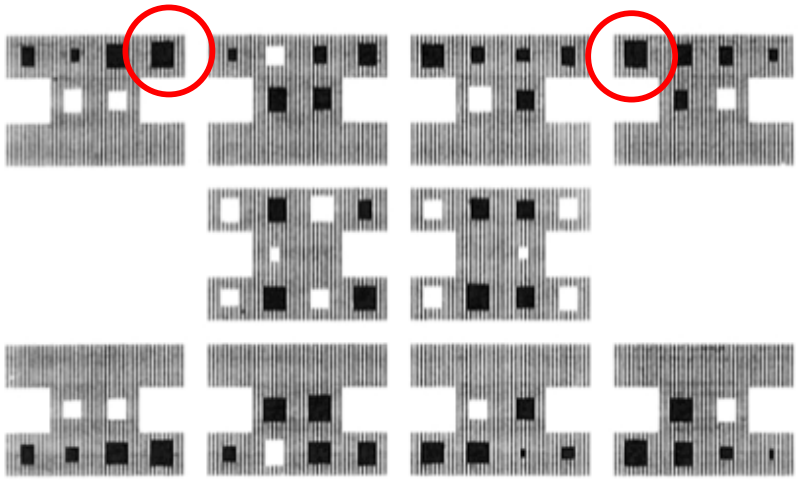
- Ackley, Hinton, and Sejnowski, 1985 presented the following example:
- 4 line one-hot encoder-decoder, 2 hidden units



Boltzman Example

- "To prevent weights from growing too large", used a "noisy" **clamping** technique: each *on* bit of a clamped vector is set to *off* with prob. 0.15 and each *off* bit set to *on* with prob. 0.05.
- Network was **unclamped** and allowed to reach equilibrium. Statistics were gathered for the same number of annealings as in the clamped case.
- Annealing schedule: (time units @ temperature) 2@20, 2@15, 2@12, 4@10.
- 1 time unit = interval giving each neuron a chance to flip.

Hinton Diagram for Weights



example of symmetric weights

Figure 2. A solution to an encoder problem. The link weights are displayed using a recursive notation. Each unit is represented by a shaded 1-shaped box; from top to bottom the rows of boxes represent groups V_1, H, and V_2. Each shaded box is a map of the entire network, showing the strengths of that unit's connections to other units. At each position in a box, the size of the white (positive) or black (negative) rectangle indicates the magnitude of the weight. In the position that would correspond to a unit connecting to itself (the second position in the top row of the second unit in the top row, for example), the bias is displayed. All connections between units appear twice in the diagram, once in the box for each of the two units being connected. For example, the black square in the top right corner of the leftmost unit of V_1 represents the same connection as the black square in the top left corner of the rightmost unit of V_1. This connection has a weight of -30.

Additional Examples

- 4-**3**-4 encoder/decoder converged quickly
- 8-3-8: more difficult
- 40-10-40: converged in 98.6% of runs.

Boltzmann Simulators

- /cs/cs152/boltzmann
  - Simulates only the distribution, not learning.
  - It is analogous to a spin-glass simulation.

- /cs/cs152/boltz
  - Learning, weight-saving, etc.
  - Uses mean-field annealing (to be explained)
  - An example, bxor, provides a demo of training a Boltzmann machine to implement xor. Run the shell script bxor.run (may have to run more than once for convergence).

Speedup Possibility

- Training of a Boltzmann machine is extremely slow.
- A possible speedup is to use the “mean-field” approximation (see next) to get the correlation values.
- This approach is due to Peterson and Anderson, 1987.

Mean-Field Theory Machine

- If f is a function of two variables, then the expectation

  $E[f(x, y)]$

  can be *approximated* by

  $f(E[x], E[y])$

- This idea can be applied to the weight change rule of the Boltzmann machine, which entails computing

  $\rho_{ij} = E[y_i\, y_j] \approx E[y_i]\, E[y_j]$

Mean-Field Theory

- For the Boltzmann distribution, the probability that node i takes value 1 at temperature T can be shown to be:

$$p_1 = 1/(1+\exp(-\Sigma\, w_{ij}\, E[y_j]/T))$$

- So the *expected* output of node i is

$$E[y_i] = 1*p_1 + (-1)*(1- p_1)$$

$$= \tanh(\Sigma\, w_{ij}\, E[y_j] / 2T)$$

Mean-Field Theory

- We now have n non-linear equations in n unknowns $E[y_j]$ which can be solved ***deterministically*** by using successive approximations (**without simulation**!, but we still have to anneal).

- We can then use these approximations to update the weights:

$$\Delta w_{ij} = \eta(E^+[y_i]\, E^+[y_j] - E^-[y_i]\, E^-[y_j])$$

where + and - designate clamped vs. free as before.

Cauchy Machine (Szu, 1986)

- Same topology as Boltzmann machine
- Learns arbitrary spatial patterns by Hebbian encoding and **fast simulated annealing**.
- Claimed to find min. energy with probability 1.
- Probability of neuron being 1 is
  $p_1 = T/(T+(\Delta E)^2)$ vs.
  $p_1 = 1/(1+\exp(-\Delta E /T))$ for Boltzmann.
- Annealing schedule is
  $T = T_0/(1 + k)$ vs.
  $T = T_0/(1 + \log k)$ (typical) for Boltzmann.

For Fun and Profit!
Solves any problem for only $99.99

Attrasoft Boltzmann Machine (ABM) for Windows ...

Attrasoft Boltzmann Machine (ABM) Description:
Attrasoft Boltzmann Machine (ABM) for Windows 95/98 simulates two types of neural networks:
Hopfield Model
Boltzmann Machine
ABM 2.7:
Nominated by PC Magazine for the 1996 Ziff-Davis Shareware Award;
Simulates two powerful types of neural networks, the Boltzmann Machine and the Hopfield Model;
If it fails to recognize a pattern, one retraining will fix it;
If you train with the wrong information by accident or by design, retraining will correct it;
Can handle conflicting training data;
Supports 1D and 2D translation, scaling, and rotation- symmetries;
Works for any pattern recognition problem;
Supports up to 10,000 external neurons;
Includes more than ten examples to walk you through, including one with more than 4,000 classes;
Learns more than 4,000 characters in 20 seconds and recognizes one of the 4,000 characters in 0.5 seconds, a speed unachievable by humans;
ABM does not lose focus or clarity as the net gets larger.
ABM 2.7 will allow a maximum number of 10,000 external neurons to be used. There are two basic types of tasks we expect the software to perform:
1. Classification: Given a pattern, find its class;
2. Determine a Pattern: Given a classification and part of a pattern, complete the pattern.
There are several reasons you should choose ABM:
ABM is large: ABM supports up to 10,000 neurons. Customized software can increase the number of neurons to 1,000,000.

wow!

ABM is fast: ABM can learn more than 4,000 characters in 20 seconds and recognize one of the 4,000 characters in 0.5 seconds.
ABM is accurate: ABM can solve any problem (for example, image recognition, prediction of stock market and Internet image search engine).
ABM is neural-net-size invariant, i.e., you can encode a problem with a smaller net and a larger net; the results for the larger net is equally good as the smaller net.
Customized versions of the ABM are also available upon request

Download Now!

Version:

Date: 08.01.2002

License: Shareware

Cost: Free to try, 99.99 $ - to buy.

OS:
Win95
Win98
WinMe
WinNT4.x
WinNT3.x
Windows 2000
Windows XP

Interface languages:

Buy Now! (99.99 $) | **Home page**

Restricted Boltzmann Machines (RBMs)

A restricted Boltzmann machine (Smolensky, 1986) consists of a layer of visible units and a layer of hidden units with no visible-visible or hidden-hidden connections. With these restrictions, the hidden units are conditionally independent given a visible vector, so unbiased samples from $\langle s_i s_j \rangle_{\text{data}}$ can be obtained in one parallel step. To sample from $\langle s_i s_j \rangle_{\text{model}}$ still requires multiple iterations that alternate between updating all the hidden units in parallel and updating all of the visible units in parallel. However, learning still works well if $\langle s_i s_j \rangle_{\text{model}}$ is replaced by $\langle s_i s_j \rangle_{\text{reconstruction}}$ which is obtained as follows:

1. Starting with a data vector on the visible units, update all of the hidden units in parallel.
2. Update all of the visible units in parallel to get a "reconstruction".
3. Update all of the hidden units again.

This efficient learning procedure does approximate gradient descent in a quantity called "contrastive divergence" and works well in practice (Hinton, 2002).

Deep-Belief Networks

After learning one hidden layer, the activity vectors of the hidden units, when they are being driven by the real data, can be treated as "data" for training another restricted Boltzmann machine. This can be repeated to learn as many hidden layers as desired. After learning multiple hidden layers in this way, the whole network can be viewed as a single, multilayer generative model and each additional hidden layer improves a lower bound on the probability that the multilayer model would generate the training data (Hinton et. al., 2006).

Learning one hidden layer at a time is a very effective way to learn deep neural networks with many hidden layers and millions of weights. Even though the learning is unsupervised, the highest level features are typically much more useful for classification than the raw data vectors. These deep networks can be fine-tuned to be better at classification or dimensionality reduction using the backpropagation algorithm (Hinton & Salakhutdinov, 2006). Alternatively, they can be fine-tuned to be better generative models using a version of the "wake-sleep" algorithm (Hinton et. al., 2006).

View: "The Next Generation of Neural Networks"

http://www.youtube.com/watch?v=AyzOUbkUf3M

(59 mins.)