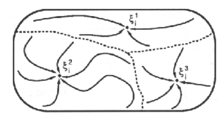
Hopfield Networks and Boltzmann Machines

15-486/782: Artificial Neural Networks
David S. Touretzky

Fall 2006

Properties of Hopfield Nets

- Special class of recurrent network.
- Fully connected; binary units (+1/–1 or 1/0.)
- The stable states are fixed point attractors.

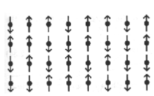


- Can act as a content-addressable memory.

John Hopfield

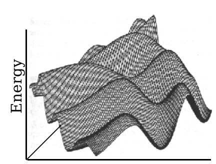
Properties of Hopfield Nets (cont.)

- Analogous to spin glass systems (Ising models) in physics, like magnetic bubble memories.



spin glass

  - Has an energy function.



  - We can use physics to analyze a neural net!

Definition of a Hopfield Net

1. Binary threshold units:

$$S_i = \begin{cases} +1 & \text{if } net_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Can also use 0/1 states.

2. Symmetric weight matrix:

$$W_{ij} = W_{ji}$$

$$W_{ii} = 0$$

Definition of a Hopfield Net (cont.)

3. No **systematic** communication delays between units.
In other words, updating must be asynchronous.

- Could update one at a time, in random order.
- Could update each unit at time t with probability p < 1.

'Update' means recompute S_i based on current net_i:

$$net_i = \sum_j S_j w_{ij}$$

5

Storing One Pattern

When is a pattern ξ stable?

$$S_i = \xi_i = sgn\left(\sum_j w_{ij}\xi_j\right) \quad \text{for all bits } i$$

Suppose $w_{ij} \propto \xi_i \xi_j$:

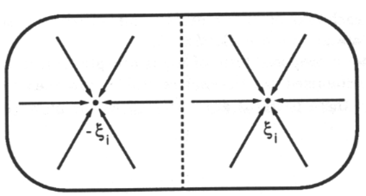
$$\begin{aligned} S_i &= sgn\left(\sum_j (\xi_i \xi_j)\cdot \xi_j\right) \\ &= sgn\left(\sum_j \xi_i \xi_j^2\right) \\ &= sgn\left(\sum_j \xi_i\right) \quad \text{since } \xi_j^2 = 1 \\ &= sgn(N\xi_i) \quad \text{where } N = \text{pattern size} \\ &= \xi_i \end{aligned}$$

For convenience set $w_{ij} = \frac{1}{N}\xi_i\xi_j$

6

Reversal States Are Also Stable

If ξ is a stable state, then so is $-\xi$.



7

Storing Multiple Patterns

$$w_{ij} = \frac{1}{N}\sum_{\mu=1}^{P} \xi_i^\mu \xi_j^\mu$$

Is ξ^ν stable?

$$\begin{aligned} \xi_i^\nu &= sgn\left(\sum_j w_{ij}\xi_j^\nu\right) \\ &= sgn\left(\frac{1}{N}\sum_j \sum_\mu \underbrace{\xi_i^\mu \xi_j^\mu \xi_j^\nu}_{\text{when } \mu=\nu \text{ this is just } \xi_i^\nu}\right) \\ &= sgn\left(\underbrace{\xi_i^\nu}_{\text{original pattern}} + \underbrace{\frac{1}{N}\sum_j \sum_{\mu\neq\nu} \xi_i^\mu \xi_j^\mu \xi_j^\nu}_{\text{noise or crosstalk term}}\right) \end{aligned}$$

ξ^ν is stable if |noise| < 1.

8

Stability

- Will units keep flipping state forever?
  - No: there are **stable states**.
- Are we guaranteed to reach a stable state from any starting point?
  - Yes, within a **finite number of flips**.
- Prove it!

9

Lyapunov Function

A **Lyapunov function** assigns a numerical value to each possible state of the system.

Also called an **energy function**.

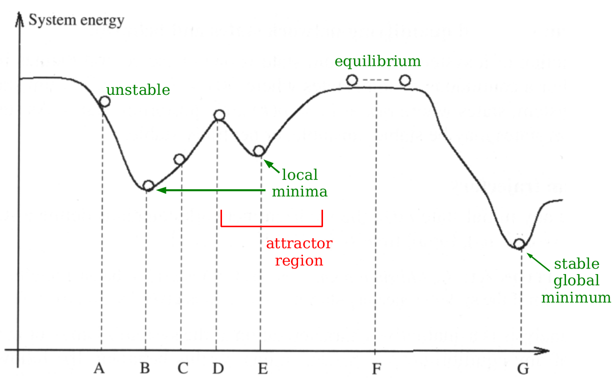
To prove stability, show that each state transition reduces the value of the Lyapunov function.

Result: stable states must exist.

- Minimum energy states are stable.
- But local minima may also exist.

10

Energy Landscape



11

Define an Energy Measure

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

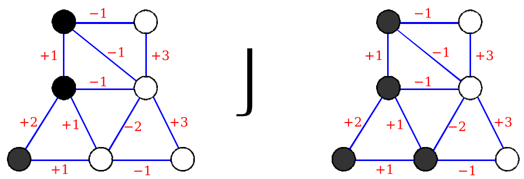
Update step: $S_i \leftarrow sgn\left(\sum_j S_j w_{ij}\right)$

$$\left.\begin{aligned} E(S_i{=}+1) &= -\frac{1}{2}\overset{net_i}{\boxed{\sum_j S_j w_{ij}}} \\ E(S_i{=}-1) &= -\frac{1}{2}\sum_j -S_j w_{ij} \end{aligned}\right\} + \left(-\frac{1}{2}\sum_{j,k\neq i} S_j S_k w_{jk}\right)$$

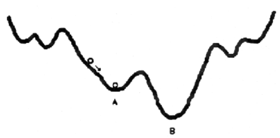
If $net_i > 0$*, then* $E(S_i{=}+1) < E(S_i{=}-1)$.
And... *If* $net_i \geq 0$*, state update rule sets* S_i *to* $+1$.
So with every update, the E goes down or stays the same.
Only 2^N *possible states, so a stable state must be reached.*

12

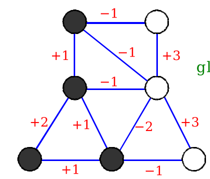
Settling Process



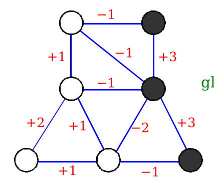
Energy E = −7   Energy E = −17



Stable States for +1/-1 Network



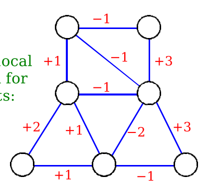
global minimum:
E = – 17

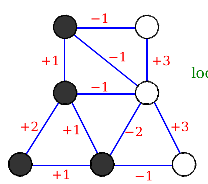


global minimum:
E = – 17

Stable States for 0/1 Network

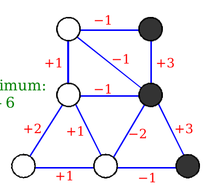
always a local minimum for 0/1 units:
E = 0

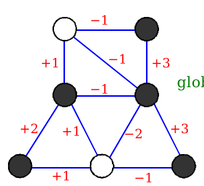




local minimum:
E = – 5

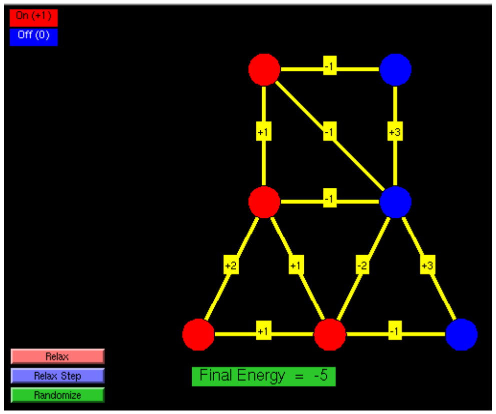
local minimum:
E = – 6



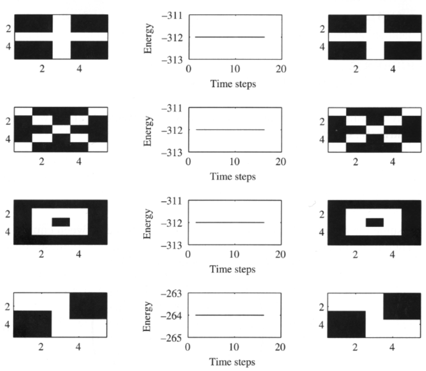


global minimum:
E = – 7

Hopfield with 0/1 Units



Associative Retrieval: Learned Patterns



5x5 = 25 units

4 patterns

Associative Retrieval: Noisy Cues

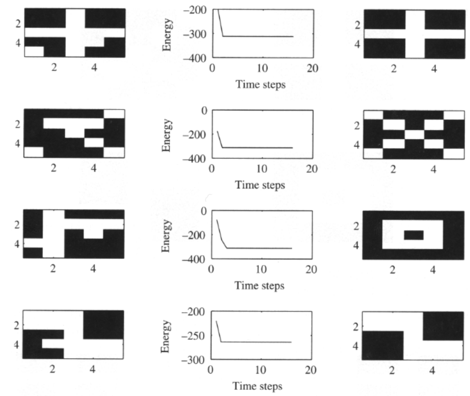
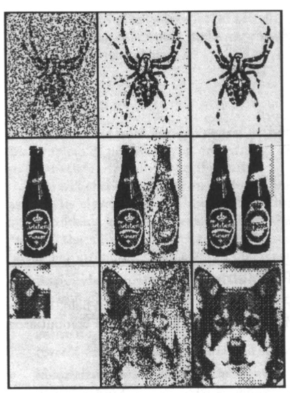


Image Retrieval From Partial Cues

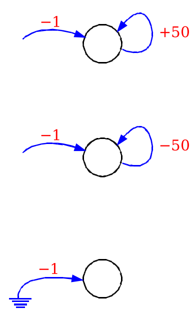


130 x 180 binary pixels =
23,400 bit patterns

sparsely connected network

7 stored patterns

Why No Self-Links?



$w_{ii} > 0$ causes spurious stable states

$w_{ii} < 0$ oscillates; no Lyapunov fn.

bias term is okay

Setting the Weights: A Heuristic

$$w_{ij} = \sum_{\mu} \xi_i^{\mu} \xi_j^{\mu} \quad \text{for } i \neq j$$

Note: this is just an outer product Hebbian learning rule.

$w_{ii} = 0$ *simplifies analysis; gives better performance*
$w_{ii} > 0$ *allowed, but may cause spurious stable states*
$w_{ii} < 0$ *no Lyapunov function; can cause oscillations*

Stored Patterns Are Energy Minima

Consider the case of one stored memory ξ.
Show that $S_i = \xi_i$ *(for all i) is an energy minimum.*

$$w_{ij} = \xi_i \xi_j \quad \text{for } i \neq j$$

$$\begin{aligned} E &= -\frac{1}{2} \sum_{i,j} S_i S_j w_{ij} \\ &= -\frac{1}{2} \sum_{i \neq j} S_i S_j (\xi_i \xi_j) \end{aligned}$$

When $S_i = \xi_i$ *and* $S_j = \xi_j$*, all terms are positive, so E is minimal. Any state change would increase E.*

Memory Capacity

How many patterns can we store in a net of N units?

- Each pattern is a vector of length N.
- Assume vectors are random (uncorrelated).

Hopfield: capacity C is ~ 0.15 N.

Tighter bound:

$$\frac{N}{4 \ln N} < C < \frac{N}{2 \ln N}$$

100 neurons can reliably store about 8 patterns.

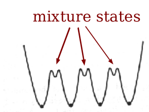
Types of Stable States

1. Retrieval states: ξ^{μ}

2. Reversed states: $-\xi^{\mu}$

3. Mixture states: any linear combination of an odd number of patterns.

$$\xi^{mix} = sgn\left(\pm \xi^1 \pm \xi^2 \pm \xi^3\right)$$



4. 'Spinglass' states: local minima not derivable from finite mixtures of patterns ξ.

Types 3 & 4 are spurious states. Spinglass states occur when too many patterns are stored.

An Aside: Optimization by Simulated Annealing

Simulated annealing is a stochastic search technique introduced by Kirkpatrick, Gelatt, & Vecchi in 1983.

Define some cost function C we want to minimize.

Try to make moves that lower C.

Scott Kirkpatrick

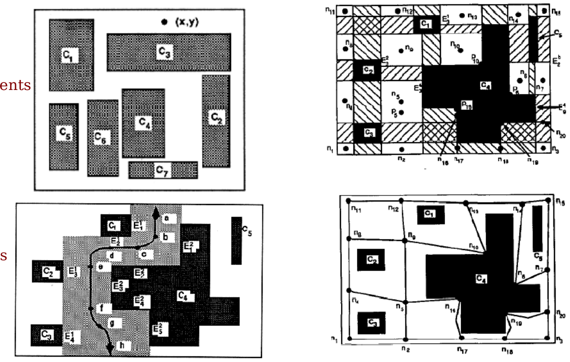
But accept moves that raise C with some probability that depends on a "temperature" parameter T.

Can escape from local minima!

Start out at high T; "anneal" by slowly lowering T.

Chip Layout by Simulated Annealing

Illustrations from Sechen (1988), inspired by Kirkpatrick, Gelatt, & Vecchi's work:



Back to Neural Networks

$$\textit{Energy gap } \Delta E_i = E(S_i{=}{+}1) - E(S_i{=}{-}1)$$

$$= -\sum_j S_j w_{ij} = -net_j$$

$$= \textit{change in } E \textit{ when } S_i \textit{ turns on.}$$

Hopfield: $S_i \leftarrow sgn(net_i)$ *always decreases E.*

What if we were to allow E to increase occasionally?

The Boltzmann Machine

Hinton and Sejnowski combined two great ideas:

Spin glass neural net models (Hopfield)

Simulated annealing search (Kirkpatrick et al.)

Geoff Hinton

Terry Sejnowski

The Boltzmann Machine

The **Boltzmann Machine is** a stochastic Hopfield net that avoids local minima through simulated annealing.

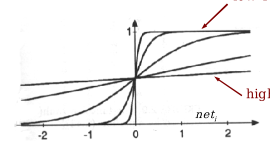
$$P[S_i{=}{+}1] \;=\; \frac{1}{1+e^{\Delta E_i/T}} \;=\; \frac{1}{1+e^{-net_i/T}}$$

where T is the temperature.

Ludwig Boltzmann, pioneer of statistical mechanics

Stochastic Units

$$P[S_i{=}{+}1] \;=\; \frac{1}{1+e^{-net_i/T}}$$



If $net_i = 0$, unit fluctuates randomly.

For large $|net_i|$, unit is mostly on (or mostly off).

We can use this randomness to jump out of local minima!

How to Make a Stochastic Unit

Calculate the net input net_i

Calculate the probability that the unit is on:

$$P[S_i{=}{+}1] \;=\; \frac{1}{1+e^{-net_i/T}}$$

Pick a random number r.

Turn unit on if $P \geq r$

Boltzmann Distribution of Energy States

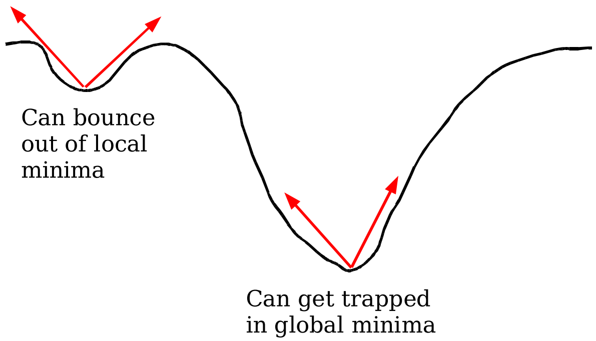
Given states $\boldsymbol{x_a}, \boldsymbol{x_b}$ with energies $E(\boldsymbol{x_a}), E(\boldsymbol{x_b})$, the ratio of their probabilities ***at equilibrium*** *at temperature T is given by the Boltzmann distribution:*

$$\frac{P(\boldsymbol{x_a})}{P(\boldsymbol{x_b})} \;=\; \frac{\exp(-E(\boldsymbol{x_a})/T)}{\exp(-E(\boldsymbol{x_b})/T)}$$

States with equal energy are equally probable.
From the above equation we can derive $P(\boldsymbol{x_a})$:

$$P(\boldsymbol{x_a}) \;=\; \frac{\exp(-E(\boldsymbol{x_a})/T)}{\sum_{\boldsymbol{x}} \exp(-E(\boldsymbol{x})/T)}$$

Stochastic Search at Moderate Temperature



33

Boltzmann Machine Stochastic Search

Start at high temperature.

- $P[S_i=1]$ is close to 0.5. Units fluctuate a lot.

Gradually cool to lower temperatures.

- Units fluctuate less as P moves closer to 1 or 0.
- Hope to get trapped in the global minimum.

At zero temperature, we have a Hopfield net.

Annealing schedule:

$$T_{i+1} \leftarrow 0.9\,T_i$$

34

Variations on Hopfield/Boltzmann

$$\textit{Hopfield:}\quad S_i \leftarrow \begin{cases} +1 & \textit{if } net_i > 0 \\ \textit{unchanged} & \textit{if } net_i = 0 \\ -1 & \textit{if } net_i < 0 \end{cases}$$

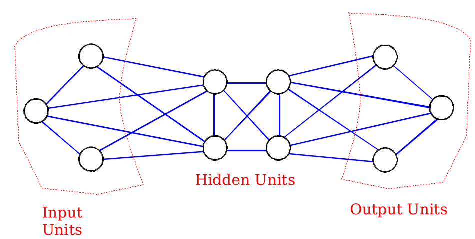
Can also choose randomly if $net_i = 0$

$$\textit{Boltzmann:}\quad P(\textit{flip}) = \begin{cases} 1 & \textit{if } \Delta E(\textit{flip}) < 0 \\ f(net_i) & \textit{if } \Delta E(\textit{flip}) > 0 \end{cases}$$

Settles to local minima more rapidly: always flips state if a flip would move downhill in energy.

35

Boltzmann Machines Can Have Hidden Units



Hidden units add extra computational power.

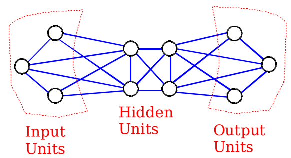
36

Boltzmann Machines With Hidden Units Are Universal

1) Clamp the input units to an input pattern.

2) Perform simulated annealing on the whole network.

3) Read the “answer” on the output units



A Boltzmann machine with enough hidden units can mimic any distribution of output states and compute any computable function.

But annealing may have to be very slow.

Mean Field Approximation

Mean field approximation to Boltzmann machine:

Replace S_i by $<S_i>$, which is proportional to $P(S_i = 1)$

Settling is faster than with a regular Boltzmann machine since we don't have to wait a long time to reach **equilibrium state**.

But not as good at avoiding local minima.

Learning in Boltzmann Machines: The Wake/Sleep Algorithm

1. *Clamp, anneal, measure* $\langle S_i S_j \rangle^{+}$ — *'wake' state*
2. *Unclamp, anneal, measure* $\langle S_i S_j \rangle^{-}$ — *'sleep' state*
3. $\Delta w_{ij} = \eta \left[\langle S_i S_j \rangle^{+} - \langle S_i S_j \rangle^{-} \right]$ — *weight update*

Hebbian learning in wake state; antihebbian in sleep state.
Unlike backprop, this is a completely local learning rule!

Very, very slow, because each learning step requires many annealings to estimate $\langle S_i S_j \rangle$*, and each must reach equilibrium.*