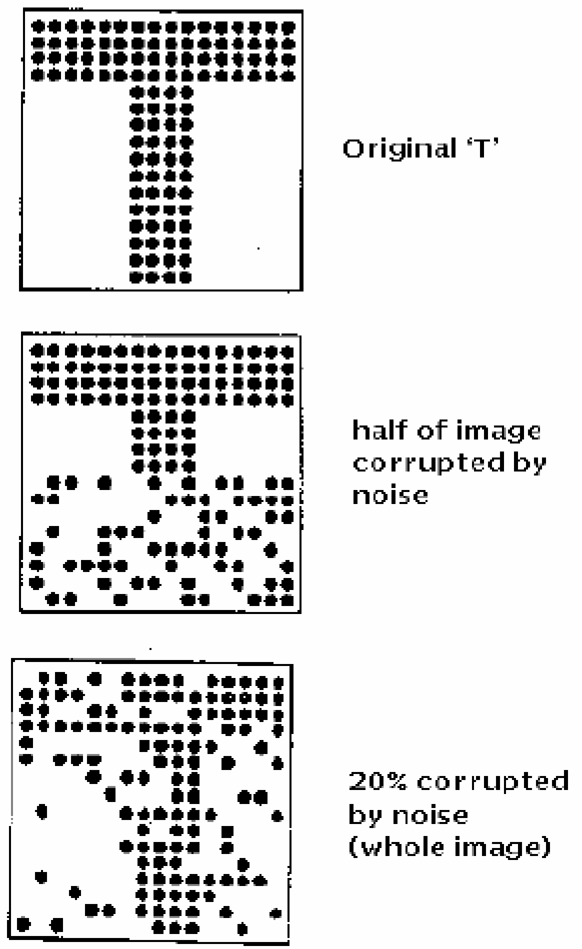
Machine Learning

Lecture 8

Associative memory based on neural networks. Hopfield model.

Tasks solved by associative memory:

1) restoration of noisy image



2) recall of associations

Input image

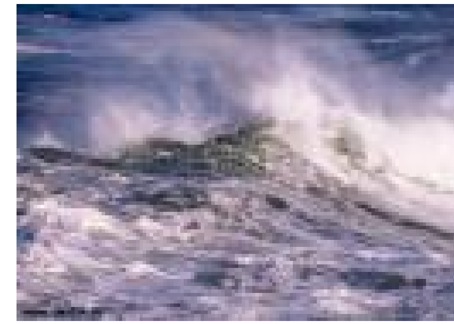


Image – result of association

3) Acquiring a habits
e.g. response on any situation

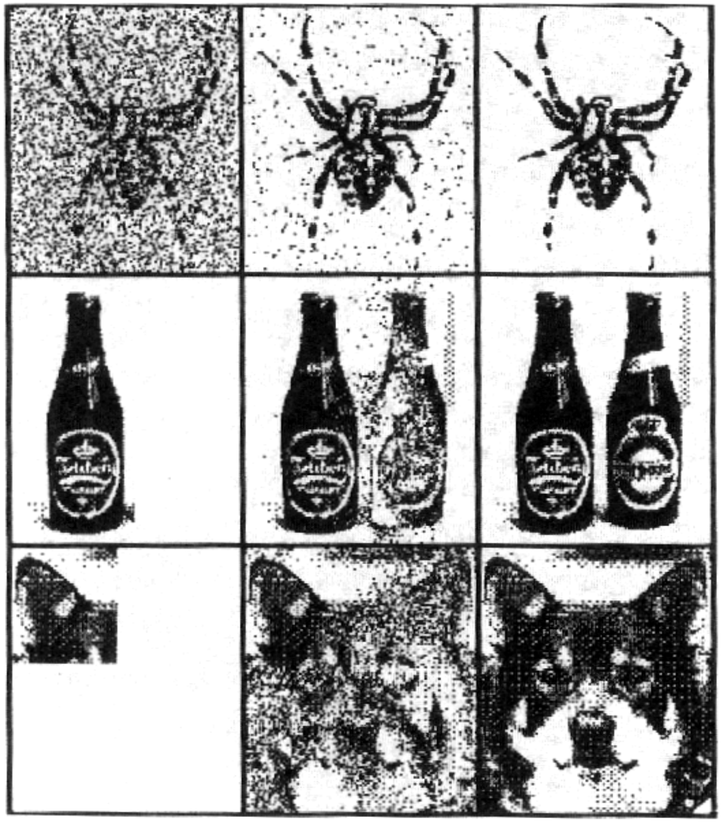
Hopfield model

Sub-type of recurrent neural nets

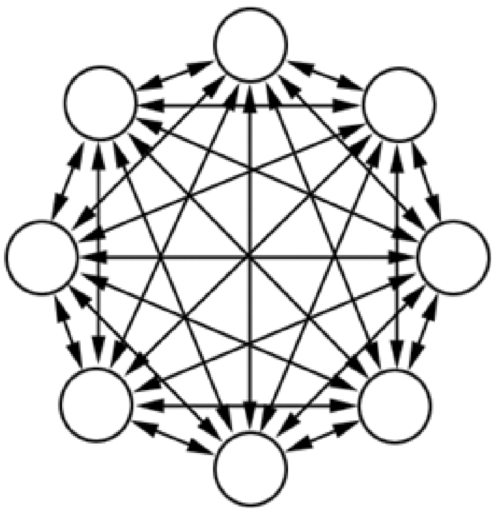
- Fully recurrent
- Weights are symmetric

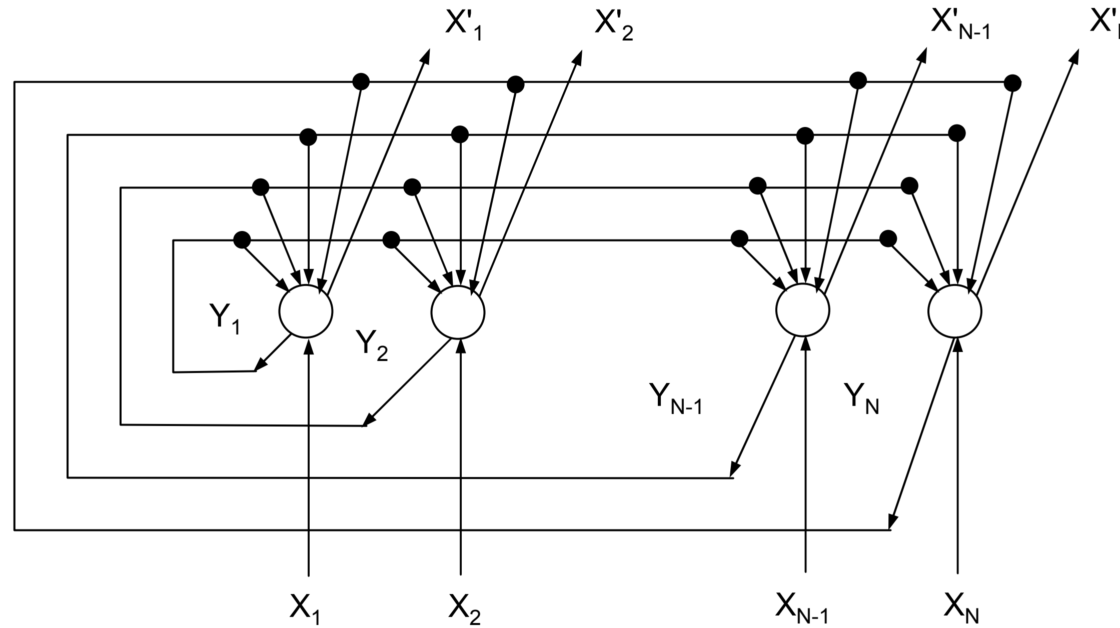
Learning: **Hebb rule** (cells that fire together wire together)
Can recall a memory, if presented with a corrupt or incomplete version

→ **auto-associative** or **content-addressable memory**



Hopfield Model (2)





Features of structure:

- Every neuron is connected with all others
- Connections are symmetric, i.e. for all *i* and *j* $w_{ij} - w_{ji}$
- Every neuron may be Input and output neuron
- Presentation of input is set of state of input neurons
- Getting of outputs is reading of states of output neurons

Neurons in Hopfield Network

- The neurons are binary units
  - They are either active (1) or passive
  - Alternatively + or –
  - May be two variants of performance: (-1,1) or (0,1)
- The network contains *N* neurons
- The state of the network is described as a vector from 0 and 1 (or -1 and 1):

$$U = (u_1, u_2, \ldots, u_N) = (0,1,0,1,\ldots,0,0,1)$$

Updating the Hopfield Network (during recall)

- The state of the network changes at each time step. There are four updating modes:
  - Serial – Random:
    - The state of a randomly chosen single neuron will be updated at each time step
  - Serial-Sequential :
    - The state of a single neuron will be updated at each time step, in a fixed sequence
  - Parallel-Synchronous:
    - All the neurons will be updated at each time step synchronously
  - Parallel Asynchronous:
    - The neurons that are not in refractoriness will be updated at the same time

The updating Rule (1):

- Here we assume that updating is serial-Random
- Updating will be continued until a stable state is reached.
  - Each neuron receives a weighted sum of the inputs from other neurons:

$$h_j = \sum_{\substack{i=1 \\ i \neq j}}^{N} u_i . w_{j,i}$$

  - If the input h_j is positive the state of the neuron will be 1, otherwise -1:

$$u_j = \begin{cases} 1 & \text{if } h_j \geq 0 \\ -1 & \text{if } h_j < 0 \end{cases}$$

Convergence of the Hopfield Network (1)

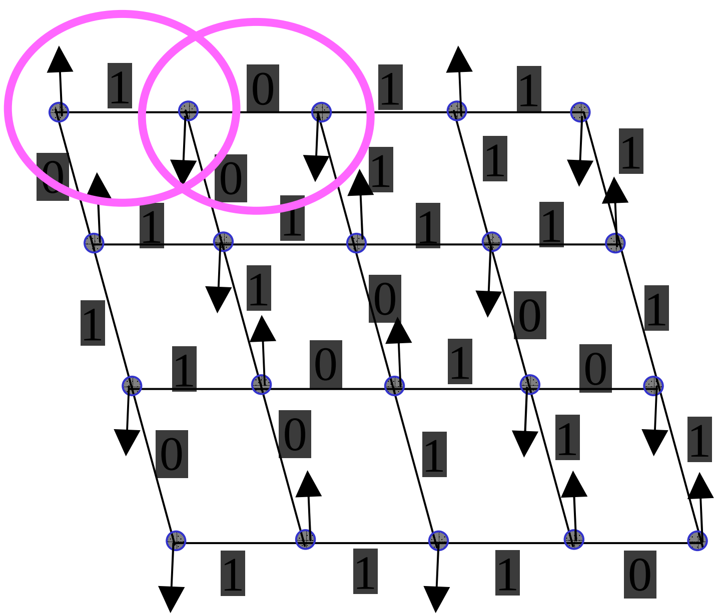
- Does the network eventually reach a stable state (convergence)?
- To evaluate this a ‘energy’ value will be associated to the network:

$$E = -\frac{1}{2} \sum_{j} \sum_{\substack{i=1 \\ i \neq j}}^{N} w_{j,i} u_i \, u_j$$

- The system will be converged if the energy is minimized

Convergence of the Hopfield Network (2)

- Why energy?
  - An analogy with spin-glass models of Ferro- magnetism (Ising model):



$$w_{i,j} = \frac{k}{d^2_{i,j}}, d_{i,j} = \text{distance}$$

$$u_j : \text{the spin of unit } j$$

$$h_j = \sum_{\substack{i=1 \\ i \neq j}}^{N} w_{j,i}\, u_i : \text{the local field exerted upon the unit } j$$

$$e_j = -\frac{1}{2} h_j\, u_j : \text{The potential energy of unit } j$$

$$E = \sum_j e_j : \text{The overal potential energy of the system}$$

$$E = -\frac{1}{2} \sum_j \sum_{\substack{i=1 \\ i \neq j}}^{N} w_{j,i} u_i\, u_j$$

  - The system is stable if the energy is minimized

Convergence of the Hopfield Network (3)

- Why convergence?

$$h_j = \sum_{\substack{i=1 \\ i \neq j}}^{N} u_i . w_{j,i} \qquad u_j = \begin{cases} 1 & \text{if } h_j \geq 0 \\ 0 & \text{if } h_j < 0 \end{cases}$$

$$E = -\frac{1}{2} \sum_j \sum_{\substack{i=1 \\ i \neq j}}^{N} w_{j,i} u_i \, u_j = -\frac{1}{2} \sum_j u_j \sum_{\substack{i=1 \\ i \neq j}}^{N} w_{j,i} u_i = -\frac{1}{2} \sum_j u_j h_j$$

$$\text{if } h_j > 0 \text{ and } u_j = 1 \text{ then } u_j \text{ will not change} \rightarrow u_j h_j = h_j > 0$$

$$\text{if } h_j > 0 \text{ and } u_j = 0 \text{ then } u_j \text{ will change} \rightarrow u_j h_j = 0$$

$$\text{if } h_j < 0 \text{ and } u_j = 0 \text{ then } u_j \text{ will not change} \rightarrow u_j h_j = 0$$

$$\text{if } h_j < 0 \text{ and } u_j = 1 \text{ then } u_j \text{ will change} \rightarrow u_j h_j = h_j < 0$$

in each case $u_j h_j$ is maximum when u_j does not change $\rightarrow$

$E = -\frac{1}{2} \sum_j u_j h_j$ is minimum if u_j values do not change

Convergence of the Hopfield Network (4)

- The changes of E with updating:

$$h_j = \sum_{\substack{i=1 \\ i \neq j}}^{N} u_i . w_{j,i} \qquad u_j = \begin{cases} 1 & \text{if } h_j \geq 0 \\ 0 & \text{if } h_j < 0 \end{cases} \qquad E = -\frac{1}{2} \sum_j \sum_{\substack{i=1 \\ i \neq j}}^{N} w_{j,i} u_i \, u_j = -\frac{1}{2} \sum_j u_j h_j$$

$$\Delta E = E_{new} - E_{old} = \left(-\frac{1}{2} \sum_{j \neq k} u_j h_j - \frac{1}{2} u_{k\,new} h_k\right) - \left(-\frac{1}{2} \sum_{j \neq k} u_j h_j - \frac{1}{2} u_{k\,old} h_k\right) = -\frac{1}{2}(u_{k\,new} - u_{k\,old}) h_k = -\frac{1}{2} \Delta u_k . h_k$$

$$\text{if } u_{k\,old} = 1 \text{ and } h_k > 0 \Rightarrow u_{k\,new} = 1 \Rightarrow \Delta u_k = 0 \Rightarrow -\frac{1}{2} \Delta u_k . h_k = 0$$

$$\text{if } u_{k\,old} = 1 \text{ and } h_k < 0 \Rightarrow u_{k\,new} = 0 \Rightarrow \Delta u_k = -1 \Rightarrow -\frac{1}{2} \Delta u_k . h_k < 0$$

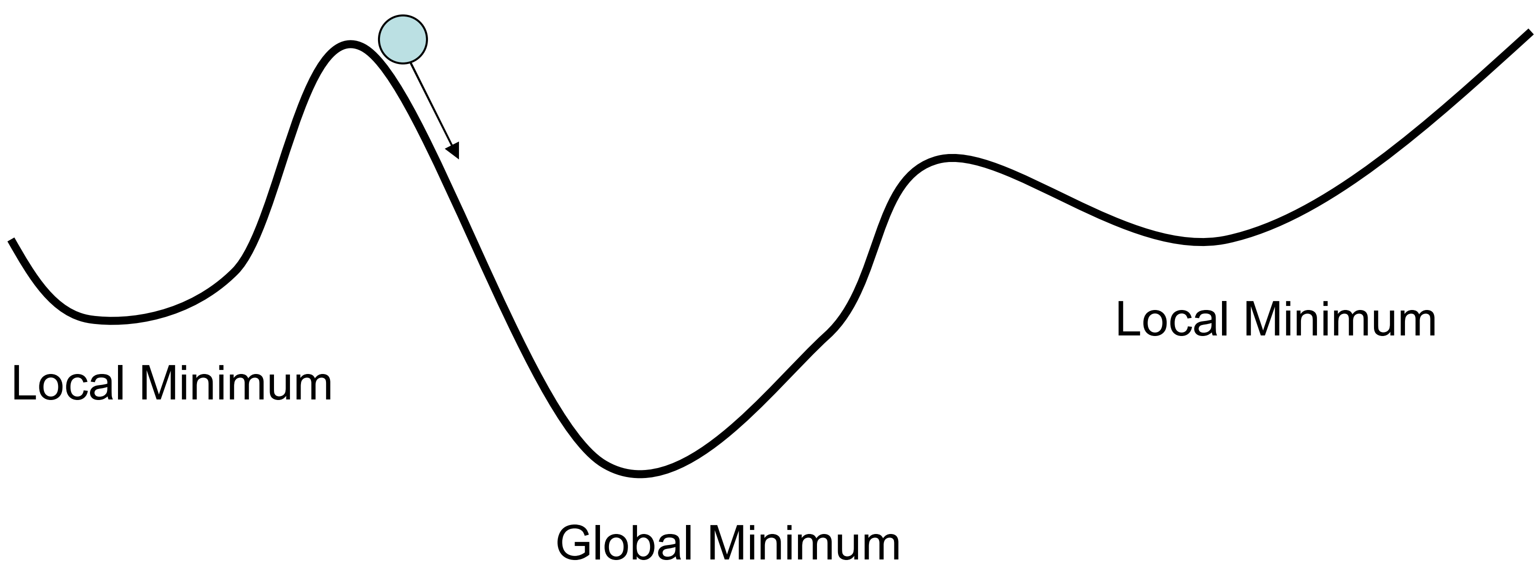
$$\text{if } u_{k\,old} = 0 \text{ and } h_k < 0 \Rightarrow u_{k\,new} = 0 \Rightarrow \Delta u_k = -1 \Rightarrow -\frac{1}{2} \Delta u_k . h_k = 0$$

$$\text{if } u_{k\,old} = 0 \text{ and } h_k > 0 \Rightarrow u_{k\,new} = 0 \Rightarrow \Delta u_k = 1 \Rightarrow -\frac{1}{2} \Delta u_k . h_k < 0$$

In each case the energy will decrease or remains constant thus the system tends to Stabilize.

The Energy Function:

- The energy function is similar to a multidimensional (N) terrain



Associative memory based on Hopfield model

- Two processes
  - Learning
  - Testing (using, recalling)

Learning

- Each pattern can be denoted by a vector from -1 and 1:

$$S_p = (-1,1,-1,1,\ldots,-1,-1,1) = (s_1^p, s_2^p, s_3^p, \ldots s_N^p)$$

- If the number of patterns is m then:

$$w_{i,j} = \sum_{p=1}^{m} s_i^{\,p} s^{p}{}_j$$

  – May be calculated without presentation of examples
- Hebbian Learning:
  – The neurons that fire together , wire together
  – For Hopfield model: Weight of link increases for neurons which fire together (with same states) and decreases if otherwise

Example of implementation of learning

```
procedure TNN.Learn;
var
  i,j,k:integer;
begin
For k:=0 to Form1.Memo1.Lines.Count-1 do
  begin
  SetUnits(k);
For i:=1 to N-1 do
    for j:=i+1 to N do
       begin
         W[i,j]:=W[i,j]+(2*S1[i]-1)*(2*S1[j]-1);
         W[j,i]:=W[i,j];
       end;
  end;
  ShowW;
end;
```

Recalling

- Iteration process of calculation of states of neurons until convergence will be achieved
- Input neurons may be freeze (can not change its state), if input pattern has not noise and may be changed otherwise
- To obtain right pattern (one from stored during learning) it is needed to present on inputs enough large vector and model must have enough large information capacity

Simulation of neuron

```
procedure TNN.Neuron(i:integer);
var
  Sum,k:integer;
begin
       Sum:=0;
       for k:=1 to N do
          Sum:=Sum + S1[k]*W[k,i];
       if Sum>H then
          S2[i]:=1
         else
          if Sum<H then
            S2[i]:=0
           else
             S2[i]:=S1[i];
       if S1[i]<>S2[i] then
          begin
            S1[i]:=S2[i];
            Net.Change:=True;
          end;
end;
```

Brief algorithm of working (part)

```
Net.Change:=False;
repeat
        input_vector;
        for j:=1 to Net.N do
          if Froz[j]=0 then
              Neuron(j);
 Until Not Net.Change or (k>NSim);
```

S1 – current states of neurons
S2 – new stats of neurons
W – weights
Froz – frozen or not neuron
Nsim – maximal number of iterations

Example of preparing of data for learning working (task – estimation of prize of flat). Length of vector (N) - 29

Destrict:	
Name 1	000
Name 2	001
Name 3	010
Name 4	011
Name 5	100
Name 6	101
Type of flat	
no	00
Panel	01
Large size	10
Monolith	11
Floor	
1	0000
2	0001
3	0010
4	0011
5	0100
6	0101
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110

Number of storeys:	
1	0000
2	0001
3	0010
4	0011
5	0100
6	0101
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
Material:	
panels	00
bricks	01
concrete	10
Square all	
20-30	00
31-40	01
41-50	10
51-63	11
Square of rooms	
10-15	000
16-20	001
21-25	010
26-30	011
31-35	100
36-40	101

Square of kitchen	
4-6	00
7-8	01
9-10	10
11-12	11
Balcony	
no	00
balcony	01
loggia	10
Balcony + loggia	11
Phone	
yes	0
no	1
Prize	
71-90	0000
91-110	0001
111-130	0010
131-150	0011
151-170	0100
171-190	0101
191-210	0110
211-230	0111
231-250	1000
251-270	1001
271-290	1010
291-310	1011
311-330	1100
331-350	1101
351-370	1110
371-390	1111

Limitations of Hopfield associative memory

- The evoked pattern is sometimes not necessarily the most similar pattern to the input because local minima
- Some patterns will be recalled more than others
- Spurious states: non-original patterns because symmetry of weight matrix
- Information capacity: $\leq 0.15\ N$

- One of method to overcome local minima of E is to introduce in model of random process of updating of weights, i.e. to append to Hopfield model of Boltzmann machine

Boltzmann machine. Definition of wikipedia

A **Boltzmann machine** is a type of stochastic recurrent neural network originally invented by Geoffrey Hinton and Terry Sejnowski.
Boltzmann machines can be seen as the stochastic, generative counterpart of Hopfield nets.
They were an early example of neural networks capable of forming internal representations. Because they are very slow to simulate they are not very useful for most practical purposes.
However, they are theoretically intriguing due to the biological plausibility of their training algorithm.

Definition of BM (2)

Boltzmann machine, like a Hopfield net is a network of binary units with an "energy" defined for the network. Unlike Hopfield nets though, Boltzmann machines only ever have units that take values of 1 or 0. The global energy, *E*, in a Boltzmann machine is identical to that of a Hopfield network, that is:

$$E = -\frac{1}{2}\sum_{j}\sum_{\substack{i=1 \\ i\neq j}}^{N} w_{ij} s_i \, s_j + \sum_{i} \theta_i s_i$$

Where:

- w_{ij} is the connection weight from unit j to unit i.
- s_i is the state (1 or 0) of unit i.
- θ_i is the threshold of unit i.

Definition of BM (3)

Thus, the difference in the global energy that results from a single unit *i* being 0 or 1, written ΔE_i, is given by:

$$\Delta E_i = \sum_j w_{ij} s_j - \theta_i$$

A Boltzmann machine is made up of stochastic units. The probability, p_i of the i^{th} unit being on is given by:

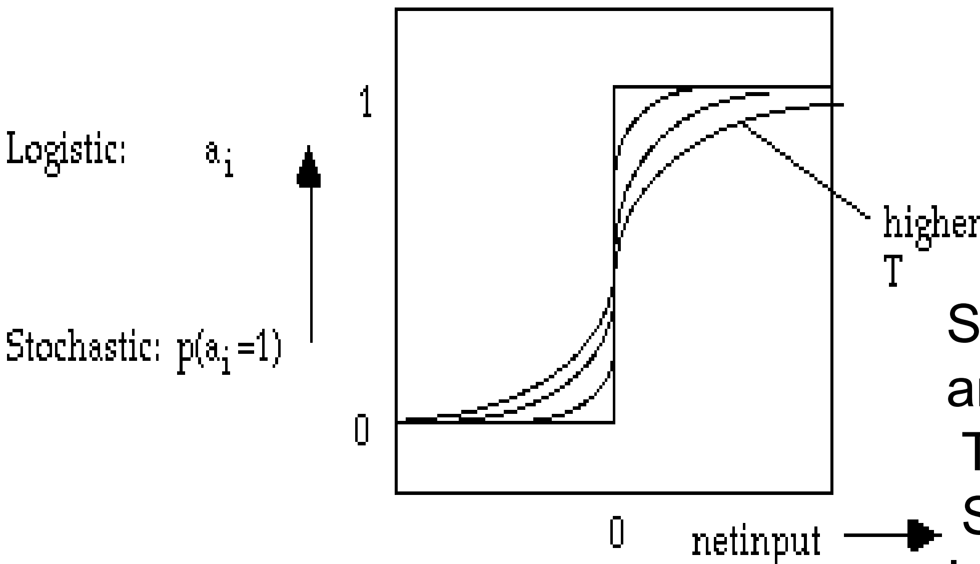
$$p_i = \frac{1}{\left(1 + e^{-(\Delta E_i / T)}\right)}$$

(The scalar *T* is referred to as the "temperature" of the system.)

Definition of BM (4)

Notice that temperature T plays a crucial role in the equation, and that in the course of running the network, the value of T will start high, and gradually 'cool down' to a lower value.

This is a **continuous function** that transforms any inputs - from –infinity to +infinity - into real numbers in the interval [0, 1]. This is the logistic function, & has a characteristic sigmoid shape:



So when Net = 0, e-Net = 1, because any number raised to the power 0 is 1
This true for all temperatures.
So always prob(A = 1) = 1/2.
I.e. if the netinput is 0, it's as likely to fire as not.

Definition of BM (5)

- With very low temperatures, e.g. 0.001, if you get a little bit of **positive** activation, the probability it will fire goes to 1.
- conversely, with if it goes **negative**, i.e. at very low temperatures - as it approaches 0 - the Boltzmann machine becomes deterministic. Otherwise, the higher the temperature, the more it diverges from this.

Definition of BM (6)

Units are divided into "visible" units, *V*, and "hidden" units, *H*.
The visible units are those which receive information from the "environment", i.e. those units that receive binary state vectors for training.

The connections in a Boltzmann machine have three restrictions on them:

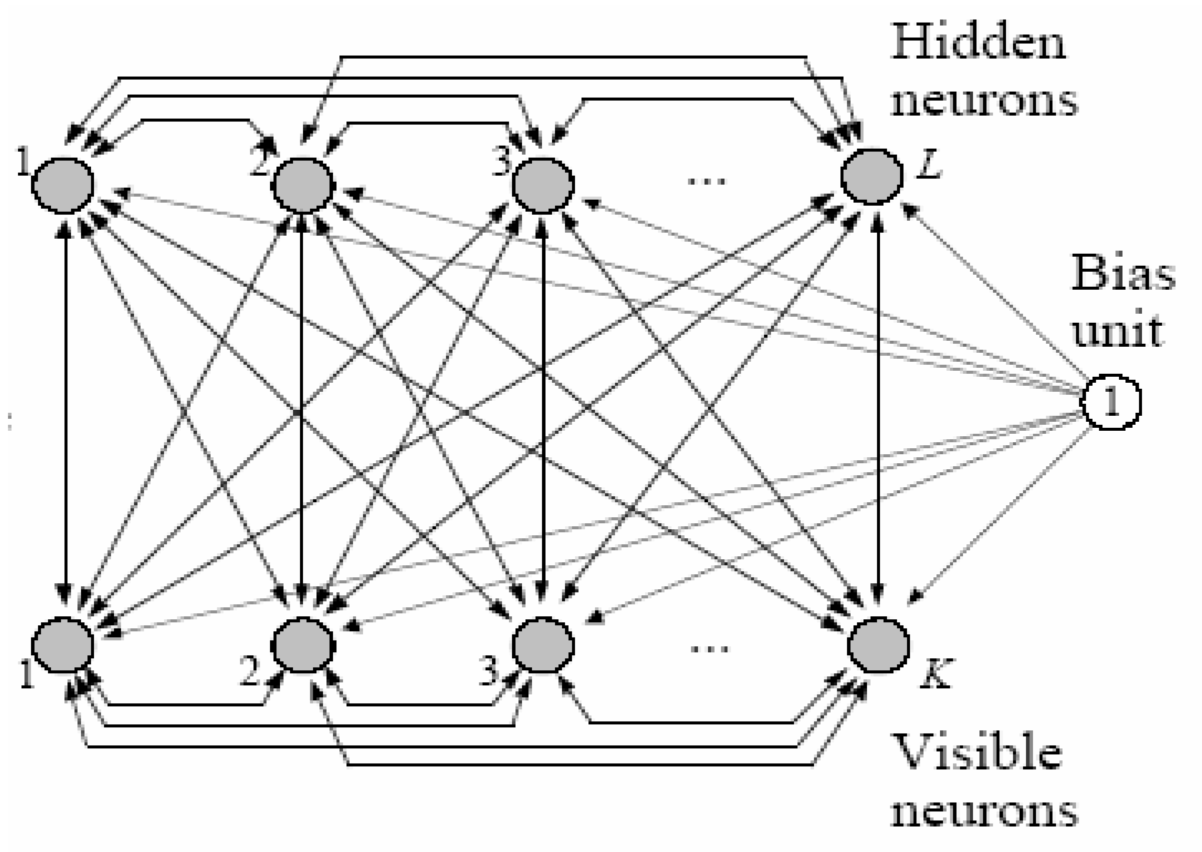
$w_{ii} = 0, \forall i$ (No unit has a connection with itself)

$w_{ij} = w_{ji}, \forall i, j$ (All connections are symmetric)

$w_{ij} = 0, \forall i, j : i \in V, j \in V$

(Visible units have no connections between them)

Structure of BM



Training of BM

Boltzmann machines can be viewed as a type of maximum likelihood model, i.e. training involves modifying the parameters (weights) in the network to maximize the probability of the network producing the data as it was seen in the training set. In other words, the network must successfully model the probabilities of the data in the environment.

There are two phases to Boltzmann machine training. One is the "positive“ phase where the visible units' states are clamped to a particular binary state vector from the training set. The other is the "negative" phase where the network is allowed to run freely, i.e. no units have their state determined by external data.

A vector over the visible units is denoted V_α and a vector over the hidden units is denoted as H_β. The probabilities $P^+(S)$ *and* $P^-(S)$ represent the probability for a given state, *S*, in the positive and negative phases respectively.

Note that this means that $P^+(V_\alpha)$ is determined by the environment for every V_α, because the visible units are set by the environment in the positive phase.

Training of BM (2)

Boltzmann machines are trained using a gradient descent algorithm, so a given weight, w_{ij} is changed by subtracting the partial derivative of a cost function with respect to the weight. The cost function used for Boltzmann machines, G, is given as:

$$G = \sum_{\alpha} P^{+}(V_{\alpha}) \ln \frac{P^{+}(V_{\alpha})}{P^{-}(V_{\alpha})}$$

This means that the cost function is lowest when the probability of a vector in the negative phase is equivalent to the probability of the same vector in the positive phase. As well, it ensures that the most probable vectors in the data have the greatest effect on the cost

Training of BM (3)

This cost function would seem to be complicated to perform gradient descent with. Suprisingly though, the gradient with respect to a given weight, w_{ij}, at thermal equilibrium is given by the very simple equation:

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T}[p_{ij}^{+} - p_{ij}^{-}]$$

Where:

- p_{ij}^{+} is the probability of units *i* and *j* both being on in the positive phase.
- p_{ij}^{-} is the probability of units *i* and *j* both being on in the negative phase.

Training of BM (4)

Simulated annealing (SA) is a generic probabilistic meta-algorithm for the global optimization problem, namely locating a good approximation to the global optimum of a given function in a large search space. It was independently invented by S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi in 1983, and by V. Cerny in 1985.
The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

Summary of the Boltzmann Machine Learning Procedure

1. *Initialization*: set weights to random numbers in [−1,1]
2. *Clamping Phase*: Present the net with the mapping it is supposed to learn by clamping input and output units to patterns. For each pattern, perform simulated annealing on the hidden units at a sequence $T_0, T_1, \ldots, T_{final}$ of temperatures. At the final temperature, collect statistics to estimate the correlations

$$\rho_{ji}^{+} = \langle s_j s_i \rangle^{+} \ (j \neq i)$$

3. *Free-Running Phase*: Repeat the calculations performed in step 2, but this time clamp only the input units. Hence, at the final temperature, estimate the correlations

$$\rho_{ji}^{-} = \langle s_j s_i \rangle^{-} \ (j \neq i)$$

4. *Updating of Weights*: update them using the learning rule

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

   where η is a learning rate parameter.

5. *Iterate until Convergence*: Iterate steps 2 to 4 until the learning procedure converges with no more changes taking place in the synaptic weights w_{ji} for all j, i.

Similarity and difference of BM and Hopfield model

The Boltzmann machine (named in honour of a 19th-century scientist by its inventors) has similarities to and differences from the Hopfield net.

Similarities:

1. Processing units have binary states (±1)
2. Connections between units are symmetric
3. Units are picked at random and one at a time for updating
4. Units have no self-feedback.

Differences:

1. Boltzmann machine permits the use of *hidden neurons*.
2. Boltzmann machine uses *stochastic neurons* with a probabilistic firing mechanism, whereas the standard Hopfield net uses neurons based on the McCulloch-Pitts model with a *deterministic* firing mechanism.
3. Boltzmann machine may also be trained by a probabilistic form of supervision.

Sometimes machine boltzmann are used in combination with Hopfield model and perceptron as device for find global minimum of energy function or error function correspondingly.

In first case during of working (recall) state of any neuron is changed according of T (temperature) and if energy function decreases then this changing is accepted and process continues.

In perceptron during of learning any weight is changed and this changing is accepted or not in according with estimation of error function.

This process of changing may be executed in mixture with usual process of working or learning or after it to improve result.

Boltzmann Machine Applications

Try `http://attrasoft.com/Boltzmann.html`

Boltzmann Machine in stock market trend prediction
Boltzmann Machine in character recognition
Boltzmann Machine in Face recognition
Boltzmann Machine in Internet Application
Boltzmann Machine in Cancer Detection
Boltzmann Machine in Loan Application
Boltzmann Machine in Decision making