3.3 Discrete Hopfield Net

An iterative autoassociative net similar to the nets described in the previous sections has been developed by Hopfield (1982, 1984).

- The net is a fully interconnected neural net, in the sense that each unit is connected to every other unit.
- The net has symmetric weights with no self-connections, i.e.,

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

and $$w_{ii} = 0$$

The two small differences between this net and the iterative autoassociative net can have a significant effect in terms of whether or not the nets converge for a particular input pattern. The differences are that in the Hopfield net presented here,

1. ***only one unit updates its activation at a time*** (based on the signal it receives from each other unit).
2. ***each unit continues to receive an external signal in addition to the signal from the other units in the net.***

The asynchronous updating of the units allows a function, known as an ***energy or Lyapunov function*** to be found for the net. The existence of such a function enables the net to converge to a stable set of activations, rather than oscillating.

Architecture

An expanded form of a common representation of the Hopfield net is shown in the Figure

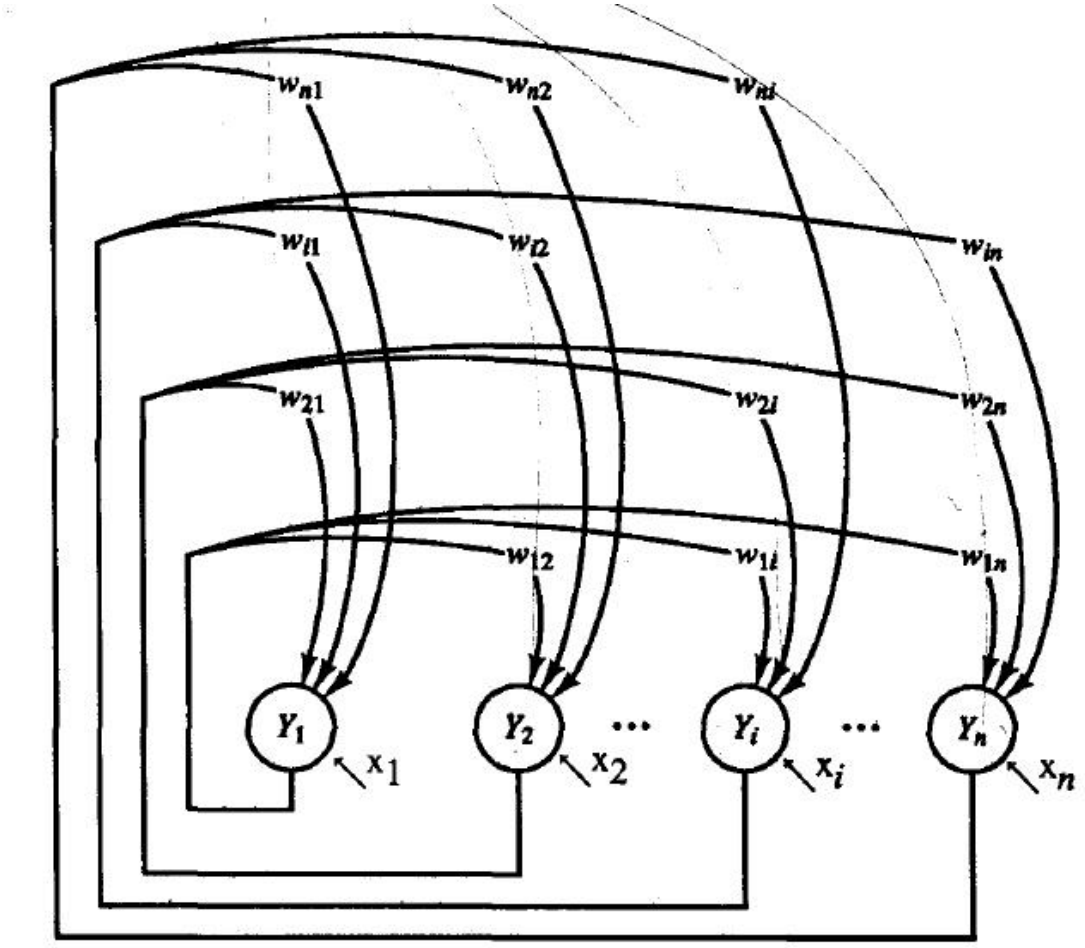

Figure illustrates the expanded form of the discrete Hopfield net

Algorithm

There are several versions of the discrete Hopfield net. Hopfield's first description **[1982]** used binary input vectors.

To store a set of binary patterns ***s (p) , p = 1 ,*** . . . , P, where,

$$S(P) = (s_1(p),\ldots, s_i(p), \ldots, s_n(p))$$

the weight matrix W is given by

$$w_{ij} = \sum_p [2s_i(p)-1][2s_j(p)-1] \qquad \text{for i} \neq \text{j}$$

and

$$w_{ii} = 0$$

Other descriptions [Hopfield, 1984] allow for bipolar inputs. The weight matrix is found as follows:

$$w_{ij} = \sum_{p} s_i(p) s_j(p) \qquad \text{for } i \neq j$$

and

$$w_{ii} = 0$$

Application Algorithm for the Discrete Hopfield Net

1. Initialize weights to store patterns. (Use Hebb rule.)

2. For each input vector x, do Steps **3-7.**

3. Set initial activations of net equal to the external input vector x:

$$y_i = x_i, \ (i = 1,2, \ldots, n)$$

4. Do Steps **5-7** for each unit Y_i (Units should be updated in random order.)

5. Compute net input:

$$y\_in_i = x_i + \sum_j y_j w_{ji}.$$

6. Determine activation (output signal):

$$y_i = \begin{cases} 1 & \text{if } y\_in_i > \theta_i \\ y_i & \text{if } y\_in_i = \theta_i \\ 0 & \text{if } y\_in_i < \theta_i. \end{cases}$$

7. Broadcast the value of y_i to all other units. (This updates the activation vector.)

8. Test for convergence.

The threshold θ, is usually taken to be zero. ***The order of update of the units is random***, but each unit must be updated at the same average rate.

There are a ***number of variations*** on the discrete Hopfield net presented in this algorithm. Originally, Hopfield ***used binary activations, with no external input after the first time step*** [Hopfield, 1982]. ***Later, the external input was allowed to continue during processing*** [Hopfield, 1984]**.**

Application

A binary Hopfield net can be used to determine whether an input vector is a *"**known**"* vector (i.e., one that was stored in the net) or an "***unknown***" vector.

The net recognizes a "known" vector by producing a pattern of activation on the units of the net that is the same as the vector stored in the net.

Example 10 Testing a discrete Hopfield net: mistakes in the first and second components of the stored vector

In this example consider the vector (1, 1, 1,0) (or its bipolar equivalent (1, 1, 1, - 1)) was stored in a net. The binary input vector corresponding to the input vector used (with mistakes in the first and second components) is (0, 0, 1, 0). Although the Hopfield net uses binary vectors, the weight matrix is bipolar, the same as was used in Example 9. The units update their activations in a random order. For this example the update order is Y_1, Y_4, Y_3, Y_2

Sol:

Initialize weights to store patterns:

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

The input vector is **x** = (0, 0, 1, 0). For this vector,

y = x = (0, 0, 1, 0).

Choose unit Y_1 to update its activation:

$$y\_in_i = x_i + \sum_j y_j w_{ji}.$$

$$y\_in_1 = x_1 + y_1*w_{11} + y_2*w_{21} + y_3*w_{31} + y_4*w_{41}$$

$$= 0 + 0*0 + 0*1 + 1*1 + 0*(-1) = 1$$

$y\_in_1 > 0 \rightarrow y_1 = 1$

$\mathbf{y = (1,0,1,0).}$

Choose unit Y_4 to update its activation:

$$y\_in_4 = x_4 + y_1 * w_{14} + y_2 * w_{24} + y_3 * w_{34} + y_4 * w_{44}$$

$$= 0 + 1*(-1) + 0 * (-1) + 1*(-1) + 0*0 = -2$$

$y\_in_4 < 0 \rightarrow y_4 = 0$

$\mathbf{y = (1,0,1,0).}$

Choose unit Y_3 to update its activation:

$$y\_in_3 = x_3 + y_1 * w_{13} + y_2 * w_{23} + y_3 * w_{33} + y_4 * w_{43}$$

$$= 1 + 1*1 + 0 * 1 + 1*0 + 0*(-1) = 2$$

$y\_in_3 > 0 \rightarrow y_3 = 1$

$\mathbf{y = (1,0,1,0).}$

Choose unit Y_2 to update its activation:

$$y\_in_2 = x_2 + y_1 * w_{12} + y_2 * w_{22} + y_3 * w_{32} + y_4 * w_{42}$$

$$= 0 + 1*1 + 0 * 0 + 1*1 + 0*(-1) = 2$$

$y\_in_2 > 0 \rightarrow y_2 = 1$

$\mathbf{y = (1,1,1,0).}$

Since some activations have changed during this update cycle, at least one more pass through all of the input vectors should be made. The reader can confirm that further iterations do not change the activation of any unit. The net has converged to the stored vector.