Self-Organization in Artificial Intelligence

Paper- Self-Organization in Artificial Intelligence and the Brain by Ananth Ranganathan, Zsolt Kira

Suman(120050031)
Nitin(120050035)
Rakesh(120050047)

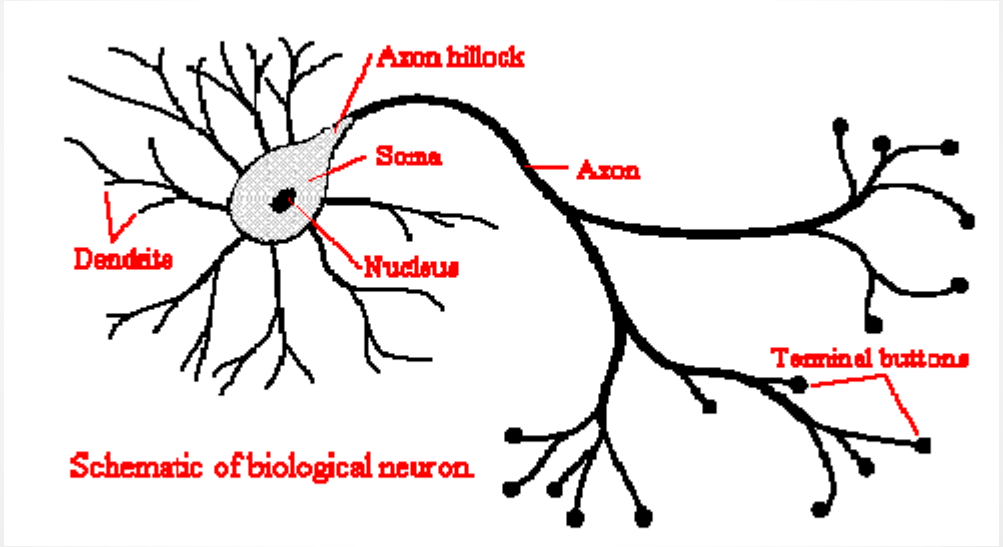
Introduction

- The main property of a neural network is an ability to learn from its environment, and to improve its performance through learning.
- Two types:
    1. **Supervised** or **active learning**
        - learning with an external teacher
    2. **Unsupervised** or **Self organised** learning
        - does not need an external teacher

Brain: A complex system

- The brain consists of approximately 10^{10} neurons, each of which can have up to 10^4 connections.
- Brain is not a static object, it changes through its interaction with the environment.
- Any form of central control cannot account for the incredible amount of processing that goes on in the brain.

Neuron


Axon hillock
Soma
Axon
Dendrite
Nucleus
Terminal buttons
Schematic of biological neuron

Neuron Structure

- Neurons contain long and short extensions called axons and dendrites, respectively.
- The neurons are connected to each other through these extensions.
- Dendrites carry electric potentials towards the cell and axons carry them away from the cell.
- The dendrite of one cell is connected to the axon of another, with a small gap in between called synapse.

Neural interaction

- Type of interaction is completely local.
- The connections among neurons are not static.
- Connections are strengthened and weakened constantly
- This type of interaction forms the basis of learning.

Self-organised learning

During the training session:

- the neural network receives a number of different input patterns
- discovers significant features in these patterns
- learns how to classify input data into appropriate categories

Unsupervised learning algorithms aim to learn rapidly and can be used in real-time.

Hebbian learning

- In 1949, Donald Hebb proposed one of the key ideas in biological learning, commonly known as **Hebb’s Law**.
- Hebbian learning is one of the most important concepts used for unsupervised learning in Neural Networks.
- Uses:
  1. Creating associative memory
  2. Pattern recognition.

Hebb's Law:

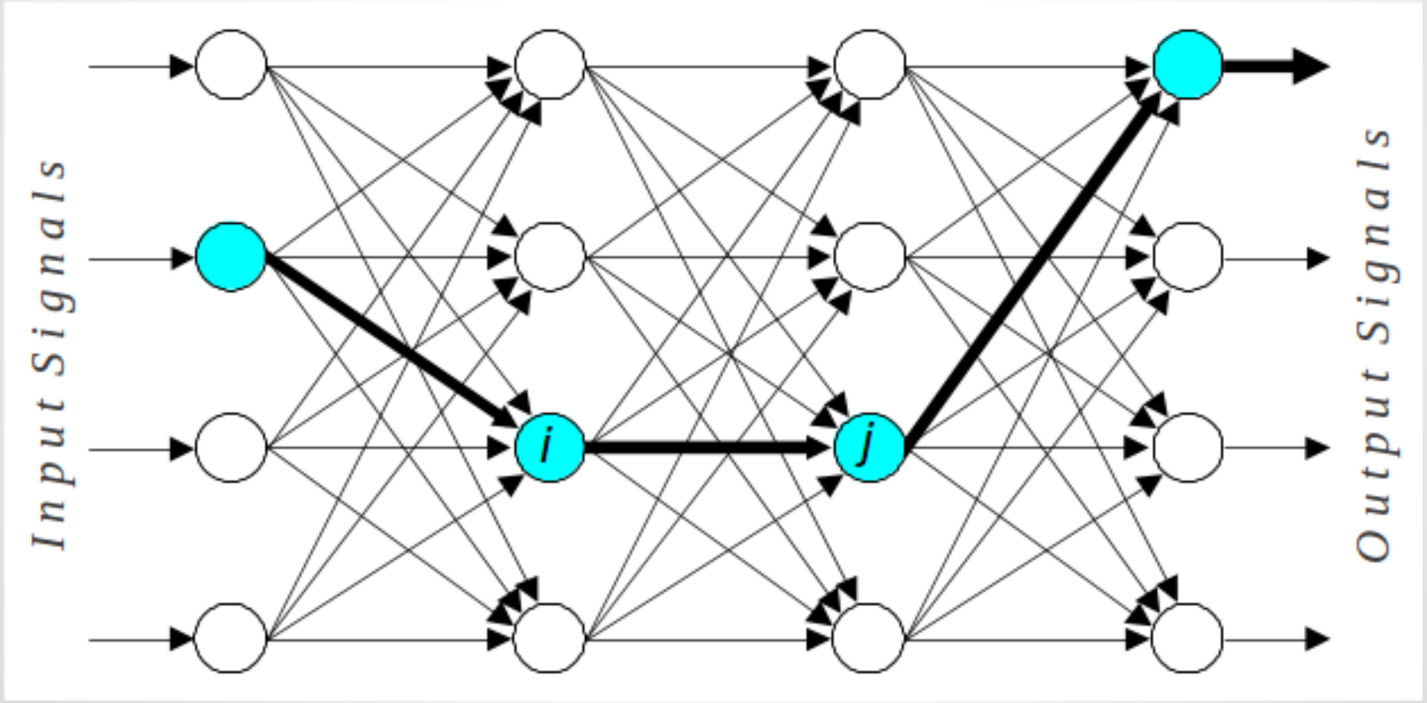
If neuron i is near enough to excite neuron j and repeatedly participates in its activation, the synaptic connection between these two neurons is strengthened and neuron j becomes more sensitive to stimuli from neuron i.

Two rules

- If two neurons on either side of a connection are activated synchronously, then the weight of that connection is increased.
- If two neurons on either side of a connection are activated asynchronously, then the weight of that connection is decreased.

These 2 rules ensures property that groups of neurons dealing with similar features are also located near each other in the brain

Hebbian learning in a neural network



Rule

The simple Hebbian rule can be written as:

$$y_i = \sum w_{ij} x_j \qquad (1)$$

y_i = the output from neuron i

x_j = the input from neuron j

w_{ij} = the weight of the connection between neuron i and neuron j.

Positive Feedback -

Hebbian Learning Rule -

Change in weight between neuron i and j is influenced by the learning rate η, the input received from neuron j, and the output of neuron i.

$$\Delta w_{ij} = \eta\ x_j\ y_i \qquad (2)$$

$$\Delta w_{ij} = \eta\ x_j \sum w_{ik} x_k = \eta \sum w_{ik} x_k\ x_j \qquad (3)$$

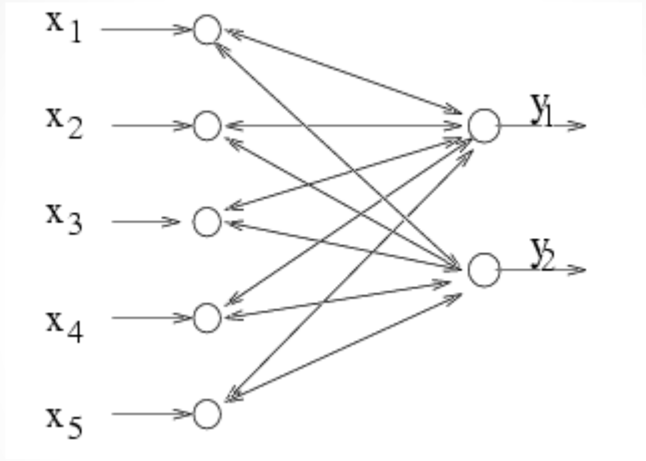
η = learning rate

Problems due to positive feedback

- Model is unstable, as repeated use can increase the weights of the connections without bounds, and the performance will degrade since all the neurons will be saturated to their maximum values.
- Larger weights will results in a larger output, which will result in a larger increase of weights.
- Model is also biologically implausible since there is a limit on the number and efficiency of synapses per neuron.

Oja in 1982 have come up with rules that have a decay term to implement negative feedback.

The negative feedback network



- Introducing decay term:

$$x_j\,(t + 1) \leftarrow x_j\,(t) - \sum w_{kj}\, y_k$$

Normalization

- Re-normalize the total synaptic weights of all the inputs, so that the total input weight is a constant.
- Such model results in competition, since an increase in weight from one neuron results in a decrease in the weights of connections to other neuron.
- Hard Competition - led to final activity of a single node
- Soft Competition - led to activity of many nodes

Hebbian learning algorithm

Step 1: Initialisation.

Set initial synaptic weights and thresholds to small random values, say in an interval [0, 1].

Step 2: Activation.

$$y_i[p]= \sum w_{ij}[p]\ x_j[p]$$

Compute the neuron output at iteration at pth iteration

Hebbian learning algorithm - Cont.

Step 3: Learning.

Update the weights in the network:

$$w_{ij}[p+1] = w_{ij}[p] + \Delta w_{ij}[p];$$

The weight correction is determined by the generalised activity product rule:

$$\Delta w_{ij} = \eta\ x_j\ y_i$$

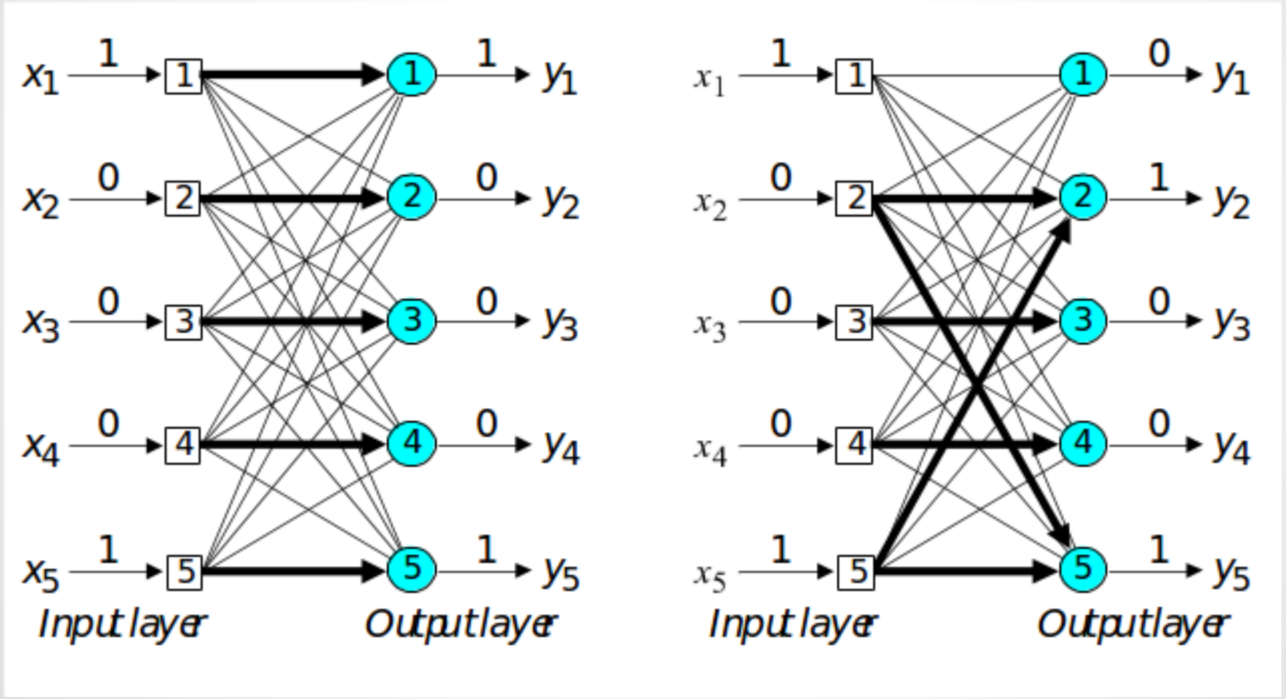
Step 4: Iteration.

Increase iteration p by one, go back to Step 2.

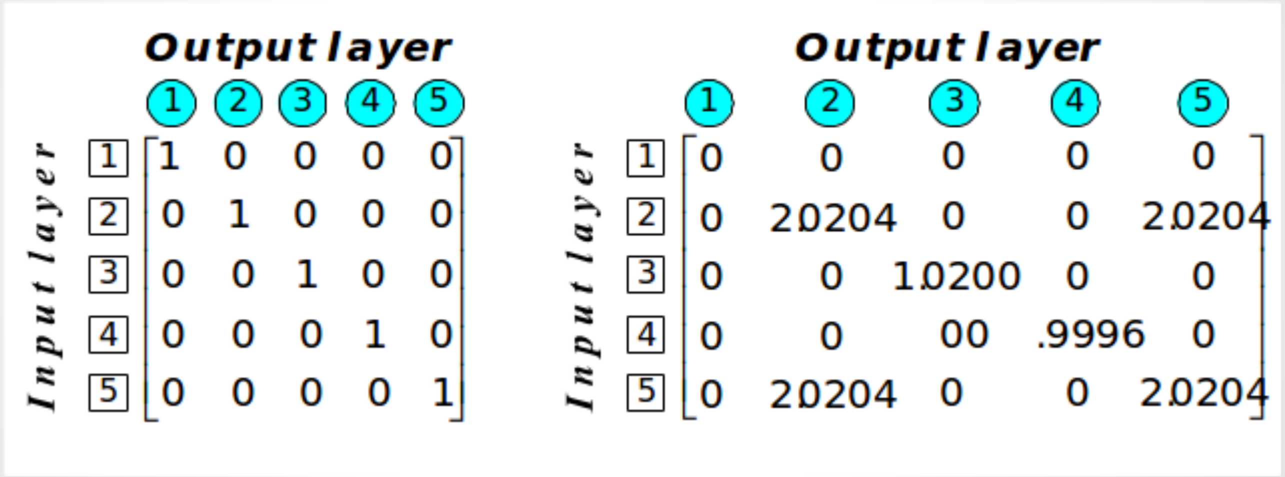
Example

$$\mathbf{x}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{x}_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x}_5 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Example Cont.



Example Cont. - Final Weights



$$\begin{array}{c|ccccc} \text{Input layer} \backslash \text{Output layer} & 1 & 2 & 3 & 4 & 5 \\ \hline \boxed{1} & 1 & 0 & 0 & 0 & 0 \\ \boxed{2} & 0 & 1 & 0 & 0 & 0 \\ \boxed{3} & 0 & 0 & 1 & 0 & 0 \\ \boxed{4} & 0 & 0 & 0 & 1 & 0 \\ \boxed{5} & 0 & 0 & 0 & 0 & 1 \end{array}$$

$$\begin{array}{c|ccccc} \text{Input layer} \backslash \text{Output layer} & 1 & 2 & 3 & 4 & 5 \\ \hline \boxed{1} & 0 & 0 & 0 & 0 & 0 \\ \boxed{2} & 0 & 2.0204 & 0 & 0 & 2.0204 \\ \boxed{3} & 0 & 0 & 1.0200 & 0 & 0 \\ \boxed{4} & 0 & 0 & 00 & .9996 & 0 \\ \boxed{5} & 0 & 2.0204 & 0 & 0 & 2.0204 \end{array}$$

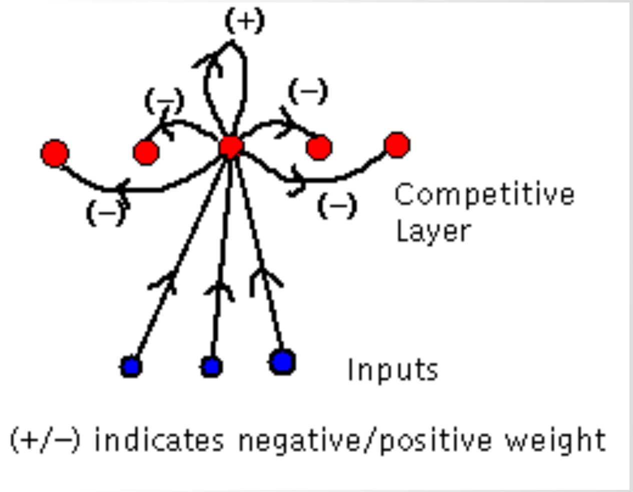
Competitive learning

Competitive learning

- Competitive learning is an algorithmic implementation of Hebbian learning in artificial neural networks.
- Neurons compete among themselves to be activated
- While in Hebbian learning, several output neurons can be activated simultaneously, in competitive learning, only a single output neuron is active at any time.
- The output neuron that wins the “competition” is called the **winner-takes-all** neuron.

Nodes in CL

Lateral connections in addition to in-layer connections which cause competition.



Feedbacks

- Lateral inhibitory connections and self-excitation to pick up the winner of the competition.
- winner’s weight vector is changed to minimize the error between the input and the weights.

Types of competitions

- Hard- Final activity of a single node; the strongest one

- Soft- drive down more than one nodes to activation.

Problems with hard Competition

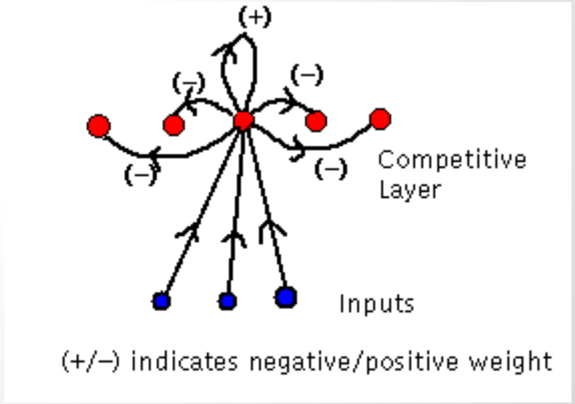
- Possible existence of “dead units”.These are units which, perhaps, are never winners for any input signal due to inappropriate initialization and thus keep their position indefinitely.
- Different random initializations of weights may lead to widely differing results.
- The purely local adaptations may not be able to get the system out of the local minimum .

Biological plausibility

- The nodes in competitive network develop into feature-sensitive detector, similar in brain.
- inhibition plays a similar role in various brain functions

An illustration

- The feed-forward network usually implements an excitatory Hebbian learning rule.
- The lateral competitive network is inhibitory in nature.
- The network serves the important role of selecting the winner, often via ”winner-take-all” schema.



An illustration

- Each unit computes a weighted sum s of the inputs provided by this vector.

  $s = \sum w_i x_i$
- If the node activation a, is allowed to evolve by making use of the lateral connections, then node k will develop a maximal value while the others get reduced.

An illustration

- Rate of change of the activation.

$$\frac{da}{dt} = \beta_s s + \beta_l l - \gamma a$$

where β_s and γ are positive constants, β_l is negative and the γ term denotes activation decay.

The Learning Technique

- The goal for the network is to learn a weight vector configuration that minimizes the total error between the weight vectors and the training set.

$$\Delta w = \alpha(x - w)y$$

Algorithm

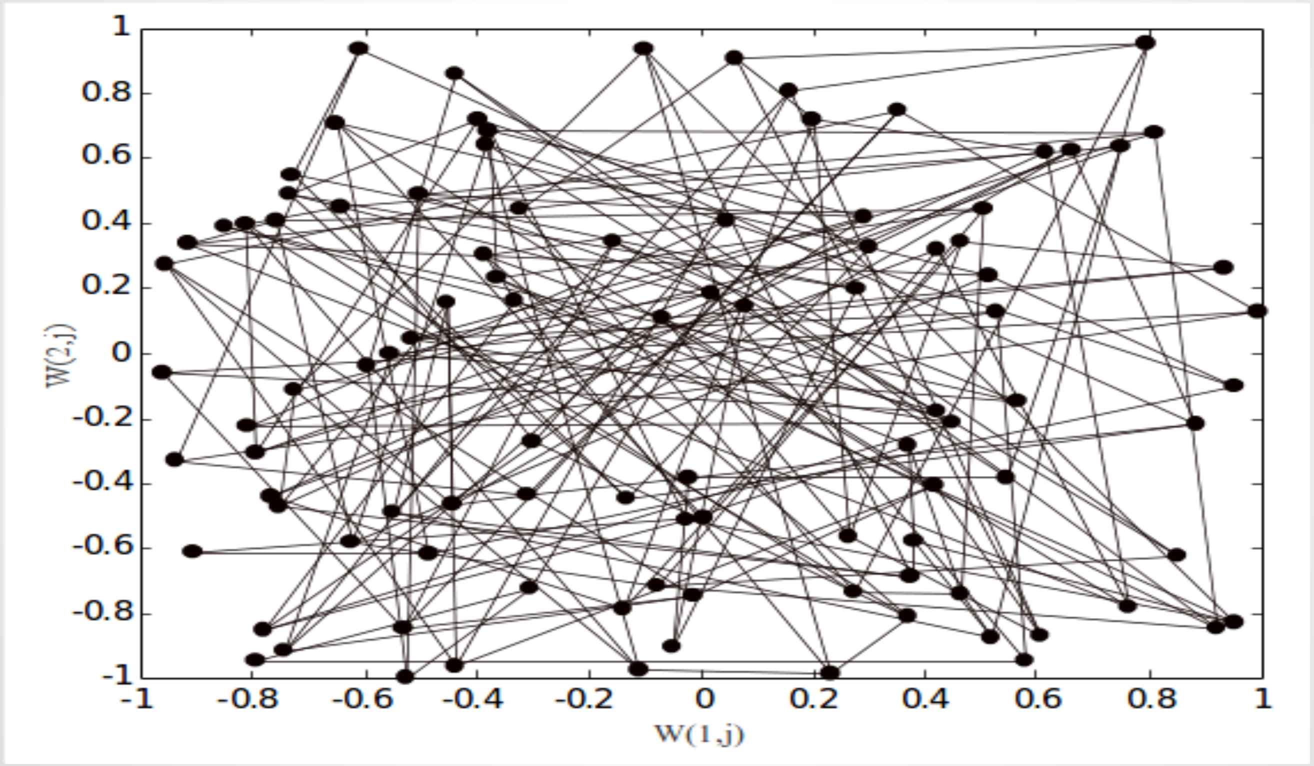
The stages in learning (for a single vector presentation) are:

1. Apply vector at input to net and evaluate s for each node
2. Update the net (in practice, in discrete steps) according to $\frac{da}{dt} = \beta_s s + \beta_I l - \gamma a$ for a finite time or until it reaches equilibrium.
3. Train all nodes according to $\Delta w = \alpha(x - w)y$.

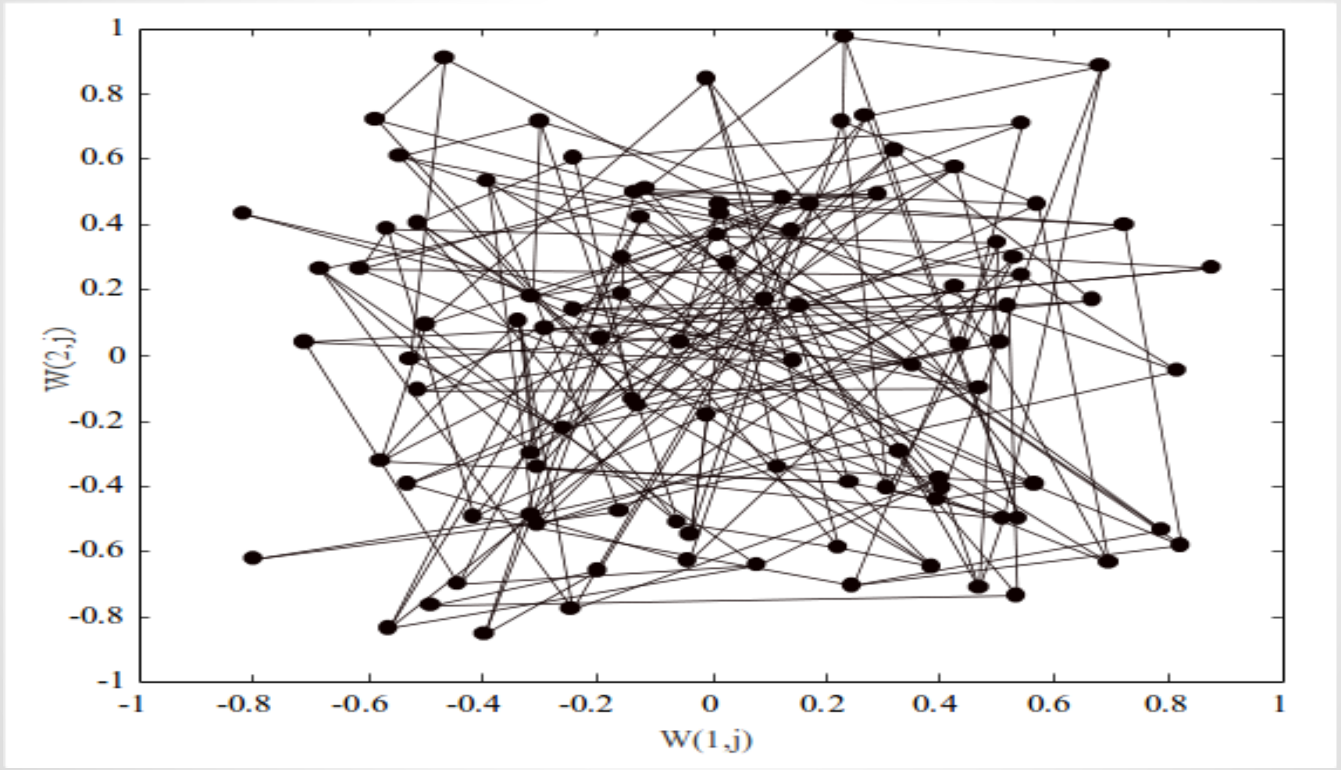
Competitive learning example

- Network with 100 neurons arranged in the form of a two-dimensional lattice with 10 rows and 10 columns.
- The network is required to classify two dimensional input vectors.
- The network is trained with 1000 two dimensional random input vectors.

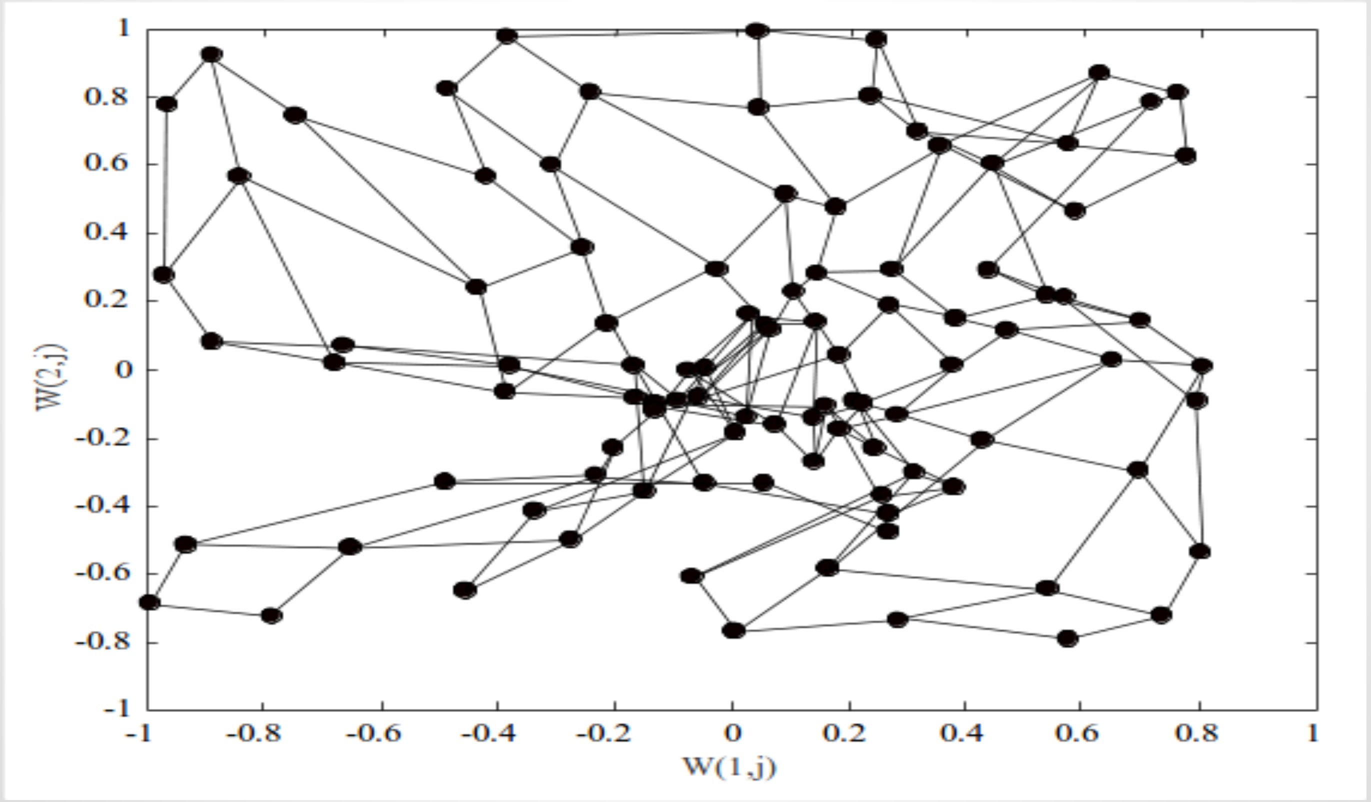
Initial random weights



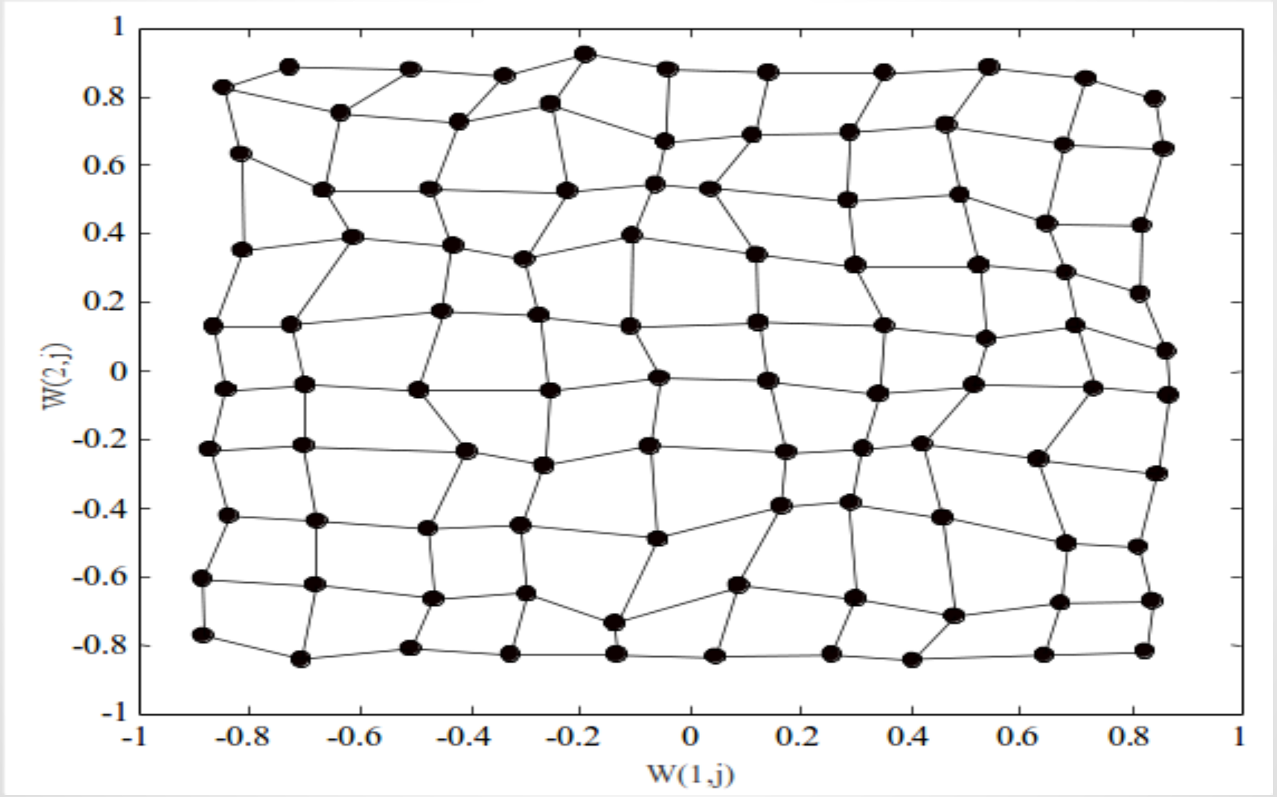
Network after 100 iterations



Network after 1000 iterations



Network after 10,000 iterations



References:

- Self-Organization in Artificial Intelligence and the Brain by Ananth Ranganathan, Zsolt Kira

  College of Computing Georgia Institute of Technology
- Kelso, J. A. S. 1995. “Dynamic Patterns: The Self-Organization of Brain and Behavior”. MIT Press, Cambridge, Mass.