

# Artificial Neural Networks

## Chapter 4: Unsupervised Learning Neural Networks

Prof. Dr. BARSEKH-ONJI Aboud

Facultad de Ingeniería  
Universidad Anáhuac México

January 22, 2026

## Agenda

- 1 Background**
  - 2 Hebbian Learning Rule**
  - 3 Principal Component Learning Rule**
  - 4 Learning Vector Quantizer-I**
  - 5 Self-Organizing Feature Maps**
    - Stochastic Training Rule
    - Batch Map
    - Growing SOM
    - Improving Convergence Speed
    - Using SOM
  - 6 Assignments**

# Introduction to Unsupervised Learning

## Definition

Unsupervised learning involves training a neural network on a dataset without explicit target values. The network must discover the underlying structure, patterns, or probability distribution of the data.

- No "teacher" signal to correct errors.
- Focuses on:
  - Clustering (grouping similar data)
  - Dimensionality Reduction (compressing information)
  - Feature Extraction

# Agenda

- 1** Background
- 2** Hebbian Learning Rule
- 3** Principal Component Learning Rule
- 4** Learning Vector Quantizer-I
- 5** Self-Organizing Feature Maps
  - Stochastic Training Rule
  - Batch Map
  - Growing SOM
  - Improving Convergence Speed
  - Using SOM
- 6** Assignments

# Hebbian Learning

## Hebb's Postulate

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

- Basic idea: "Cells that fire together, wire together."
- Mathematical formulation:

$$\Delta w_{ij} = \eta y_i x_j \quad (1)$$

$$w_{ij}(t) = w_{ij}(t - 1) + \Delta w_{ij}(t) \quad (2)$$

where  $y_i$  is the output of neuron  $i$  and  $x_j$  is the input from neuron  $j$ .

## Algorithm: Hebbian Learning

---

### Algorithm 1 Hebbian Learning Algorithm (Algorithm 4.1)

---

Initialize all weights such that  $u_{ki} = 0, \forall i = 1, \dots, l$  and  $\forall k = 1, \dots, K$ ;

**while** stopping condition(s) not true **do**

**for** each input pattern  $\mathbf{z}_p$  **do**

        Compute the corresponding output vector  $\mathbf{o}_p$ ;

**end for**

    Adjust the weights using equation (4.3):

$u_{ki}(t) = u_{ki}(t - 1) + \Delta u_{ki}(t)$ , where  $\Delta u_{ki}(t) = \eta o_{k,p} z_{i,p}$

**end while**

---

# Hebbian Learning

## A problem with Hebbian Learning

A problem with Hebbian learning is that repeated presentation of input patterns leads to an exponential growth in weight values, driving the weights into saturation.

- To prevent saturation, a limit is posed on the increase in weight values (nonlinear forgetting factor)
- Mathematical formulation:

$$\Delta w_{ij}(t) = \eta y_i x_j - \gamma y_i w_{ij}(t-1) \quad (3)$$

where  $y_i$  is the output of neuron  $i$  and  $x_j$  is the input from neuron  $j$ ;  $\gamma$  is a positive constant value.

# Agenda

- 1 Background
- 2 Hebbian Learning Rule
- 3 Principal Component Learning Rule
- 4 Learning Vector Quantizer-I
- 5 Self-Organizing Feature Maps
  - Stochastic Training Rule
  - Batch Map
  - Growing SOM
  - Improving Convergence Speed
  - Using SOM
- 6 Assignments

# Principal Component Analysis (PCA)

- PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.
- Neural networks can perform PCA using specific learning rules (e.g., Oja's Rule, Sanger's Rule).

## Goal

To find a direction (weight vector) that maximizes the variance of the output.

# Principal Component Analysis (PCA)

## Oja's Rule

Oja developed the first principal components learning rule, with the aim of extracting the principal components from the input data. Oja's principal components learning rule is an extension of the Hebbian learning rule, referred to as normalized Hebbian learning, to include a feedback term to constrain weights.

$$\begin{aligned}\Delta u_{ki}(t) &= u_{ki}(t) - u_{ki}(t-1) \\ &= \eta o_{k,p} [z_{i,p} - o_{k,p} u_{ki}(t-1)] \\ &= \underbrace{\eta o_{k,p} z_{i,p}}_{\text{Hebbian}} - \underbrace{\eta o_{k,p}^2 u_{ki}(t-1)}_{\text{forgetting factor}}\end{aligned}\tag{4}$$

# Algorithm: Principal Component Learning

---

## Algorithm 2 Principal Component Learning (Oja's Rule)

---

Initialize weights  $\mathbf{u}_k$  to small random values;  
**while** stopping condition not true **do**  
    **for** each input pattern  $\mathbf{z}_p$  **do**  
        Compute output  $o_{k,p} = \sum_i u_{ki} z_{i,p}$ ;  
        Update weights:  $\Delta u_{ki} = \eta o_{k,p} [z_{i,p} - o_{k,p} u_{ki}(t-1)]$ ;  
    **end for**  
**end while**

---

# Agenda

- 1 Background
- 2 Hebbian Learning Rule
- 3 Principal Component Learning Rule
- 4 Learning Vector Quantizer-I
- 5 Self-Organizing Feature Maps
  - Stochastic Training Rule
  - Batch Map
  - Growing SOM
  - Improving Convergence Speed
  - Using SOM
- 6 Assignments

## Learning Vector Quantizer-I (LVQ-I)

- Supervised version of Vector Quantization? (Note: LVQ is typically supervised, but grouped here. Check context).
- Wait, standard LVQ is supervised, but closely related to SOM/VQ which are unsupervised. In Engelbrecht's book, LVQ-I might be introduced in the context of prototype-based learning.
- Basic concept: Move prototype vectors closest to the input data closer (if same class) or further away (if different class).

### Algorithm: LVQ-I

**Algorithm 3** Learning Vector Quantizer-I Training Algorithm (Algorithm 4.2)

Initialize the number of output clusters,  $K$ :

Initialize weight vectors  $\mathbf{u}_k$ :

Initialize the learning rate  $\eta(0)$ ;

**while** stopping condition(s) not true **do**

**for each input pattern  $z_p$  do**

Find the winning unit  $k^* = \arg \min_k \{||\mathbf{z}_p - \mathbf{u}_k||\}$ ;

Update the winning unit:

$$\mathbf{u}_{k^*}(t+1) = \mathbf{u}_{k^*}(t) + \eta(t)[\mathbf{z}_p - \mathbf{u}_{k^*}(t)];$$

end for

Adjust the learning rate;

**end while**

# Agenda

- 1 Background
- 2 Hebbian Learning Rule
- 3 Principal Component Learning Rule
- 4 Learning Vector Quantizer-I
- 5 Self-Organizing Feature Maps
  - Stochastic Training Rule
  - Batch Map
  - Growing SOM
  - Improving Convergence Speed
  - Using SOM
- 6 Assignments

# Self-Organizing Maps (SOM)

## Overview

Introduced by Teuvo Kohonen. A SOM is a type of artificial neural network that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map.

- Preserves topological properties of the input space.
- Uses competitive learning: neurons compete to be activated.

## Stochastic Training Rule

# SOM Training Algorithm (Stochastic)

---

## Algorithm 4 Stochastic Self-Organizing Map Training

---

Initialize weight vectors  $\mathbf{w}_{mn}$  to small random values;  
Initialize learning rate  $\eta(0)$  and neighborhood size  $\sigma(0)$ ;  
**while** stopping condition not true **do**  
    **for** each input pattern  $\mathbf{z}_p$  **do**  
        Find the winning neuron  $\mathbf{c} = \arg \min_{mn} \{||\mathbf{z}_p - \mathbf{w}_{mn}||\}$ ;  
        Update weights for all neurons  $ij$  in the neighborhood of  $\mathbf{c}$ :  
             $\mathbf{w}_{ij}(t + 1) = \mathbf{w}_{ij}(t) + \eta(t)h_{cij}(t)[\mathbf{z}_p - \mathbf{w}_{ij}(t)]$ ;  
    **end for**  
    Update learning rate  $\eta(t)$  and neighborhood size  $\sigma(t)$ ;  
**end while**

---

[Batch Map](#)

# Batch SOM

---

## Algorithm 5 Batch Self-Organizing Map (Algorithm 4.3)

---

Initialize codebook vectors by assigning the first  $K_J$  training patterns to them;

**while** stopping condition(s) not true **do**

**for** each neuron  $k_j$  **do**

        Collect a list of copies of all patterns  $\mathbf{z}_p$  whose nearest codebook vector belongs to the topological neighborhood of that neuron;

**end for**

**for** each codebook vector **do**

        Compute the codebook vector as the mean over the corresponding list of patterns;

**end for**

**end while**

---

## Growing SOM

# Growing SOM

- Standard SOM has a fixed grid size.
- Growing SOMs (e.g., Growing Grid, Growing Cell Structures) add neurons dynamically during training to better represent data distribution.

[Improving Convergence Speed](#)

# Improving Convergence

- Initialization methods (e.g., using PCA).
- Adaptive learning rates.
- Time-varying neighborhood functions.

## Using SOM

## Applications of SOM

- **Clustering:** Grouping similar data.
- **Visualization:** Visualizing high-dimensional data in 2D.
- **Feature Extraction:** Pre-processing for other models.

# Agenda

- 1 Background
- 2 Hebbian Learning Rule
- 3 Principal Component Learning Rule
- 4 Learning Vector Quantizer-I
- 5 Self-Organizing Feature Maps
  - Stochastic Training Rule
  - Batch Map
  - Growing SOM
  - Improving Convergence Speed
  - Using SOM
- 6 Assignments

## Assignments

- 1 Implement a basic Hebbian learning rule for a simple pattern association task.
  - 2 Use a library (e.g., MiniSom or sklearn) to apply SOM to a dataset (e.g., Iris or handwritten digits). Visualize the result.
  - 3 Compare PCA and SOM on a dimensionality reduction task.