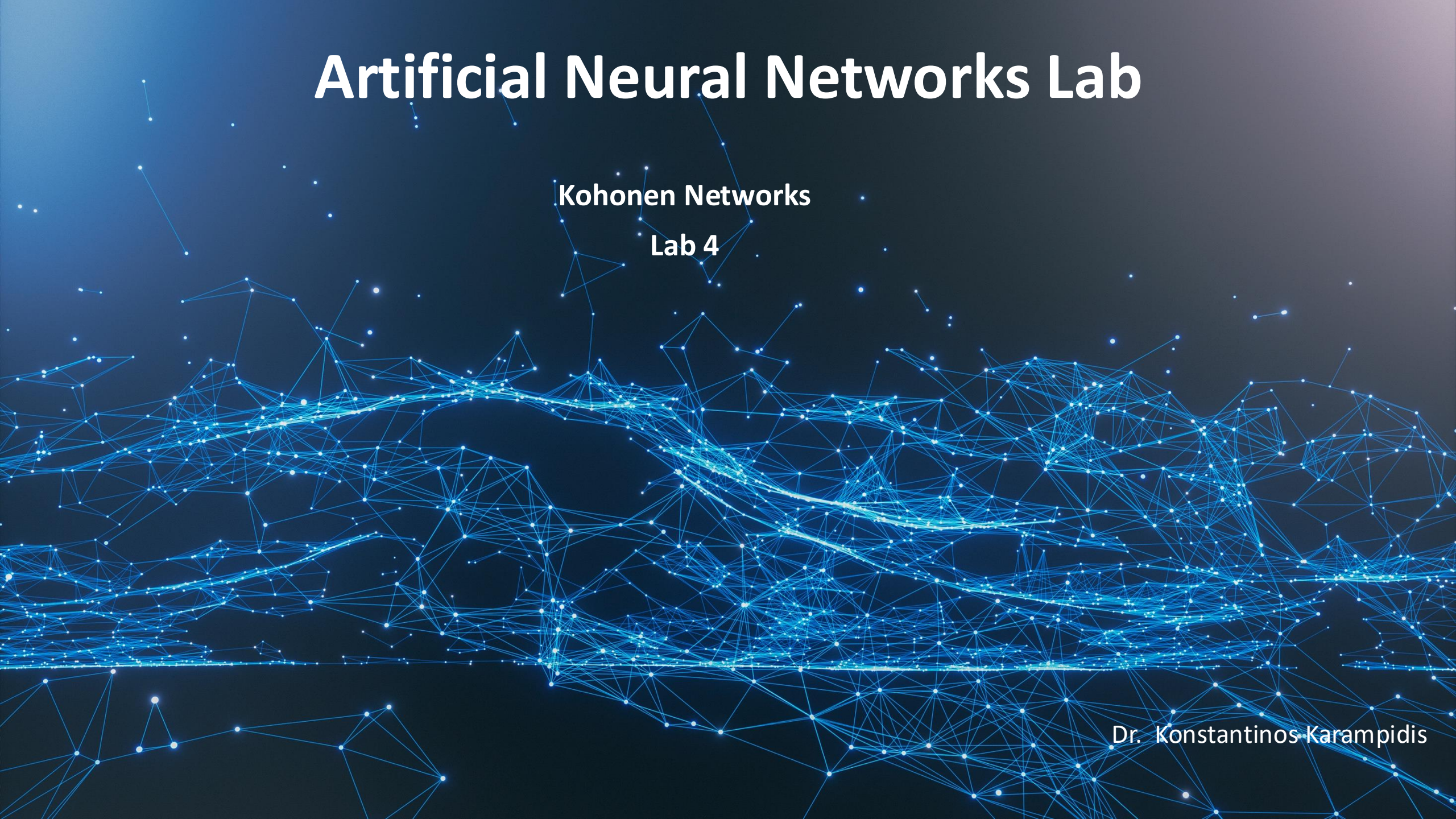
Artificial Neural Networks Lab

Kohonen Networks

Lab 4



Dr. Konstantinos Karampidis

Introduction

- The problems we have examined so far belonged to “supervised leraning”. That is, during the training process for each one of the input vectors we knew in advance the class they belonged to.
- These vectors were used in such a way that the neural network "learned" about these class characteristics.

Competitive Training

Usually, we do not have the "luxury" of training our neural network with patterns with known class.

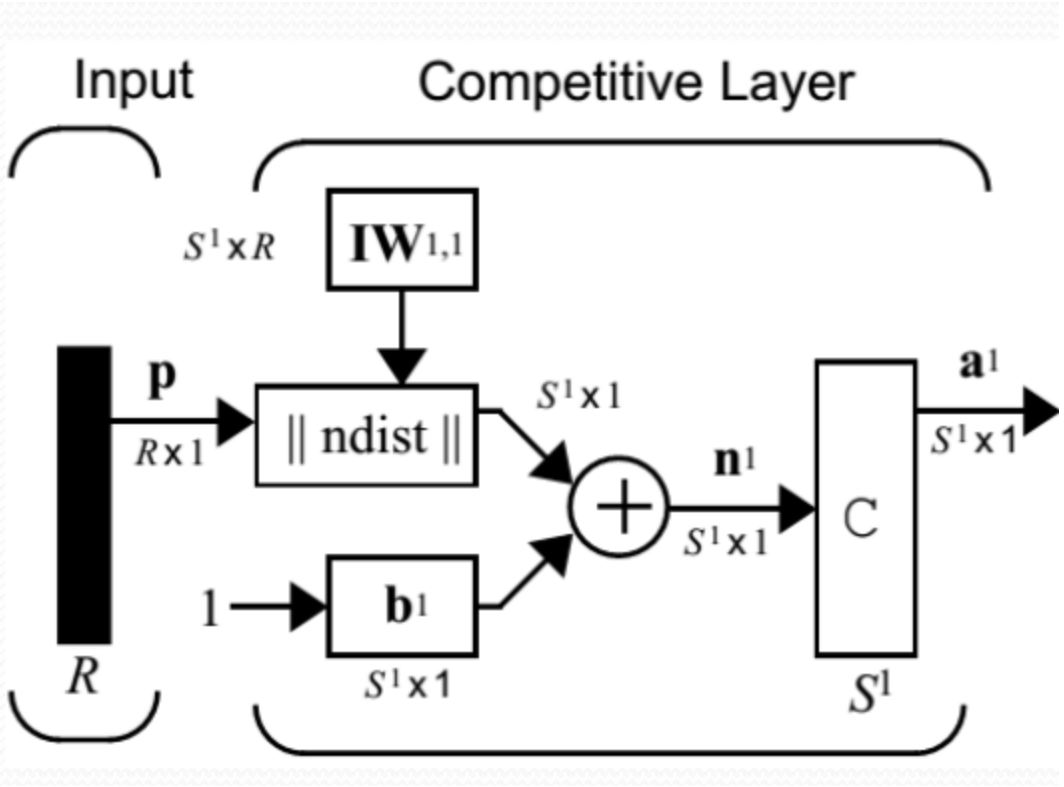
This can happen because:

- We do not have these vectors
- We do not even know the number of categories we have to divide.

In this case, the network should be trained in such a way that finds the similarities and disparities of the vectors in order to "build" the categories (unsupervised learning).

Kohonen Networks

- They consist of one layer.
- They are fully inter-connected.
- These networks have some basic differences in their operation with respect to the layers of the networks we have already studied:
  - It uses the competitive activation function (C)
  - It calculates the Euclidean distance for comparing the characteristic values of each pattern with the weights.



Kohonen Networks (Weights)

- Let us assume that a vector p (R dimensionsal) is presented as input.
- Let's also assume that the neural network has S neurons at its unique level.
- Since we have a complete connection between the features of the patterns and the neurons, the IW weight table that is formed is S x R dimensionsal.

W_{11}	W_{12}	W_{13}	...	W_{1R}
W_{21}	W_{22}	W23	...	W_{2R}
W_{31}	W_{32}	W_{33}	...	W_{3R}
W_{41}	W_{42}	W_{43}	...	W_{4R}
W_{51}	W_{52}	W_{53}	...	W_{5R}
...	...	...	...	...
W_{S1}	W_{S2}	W_{S3}	...	W_{SR}

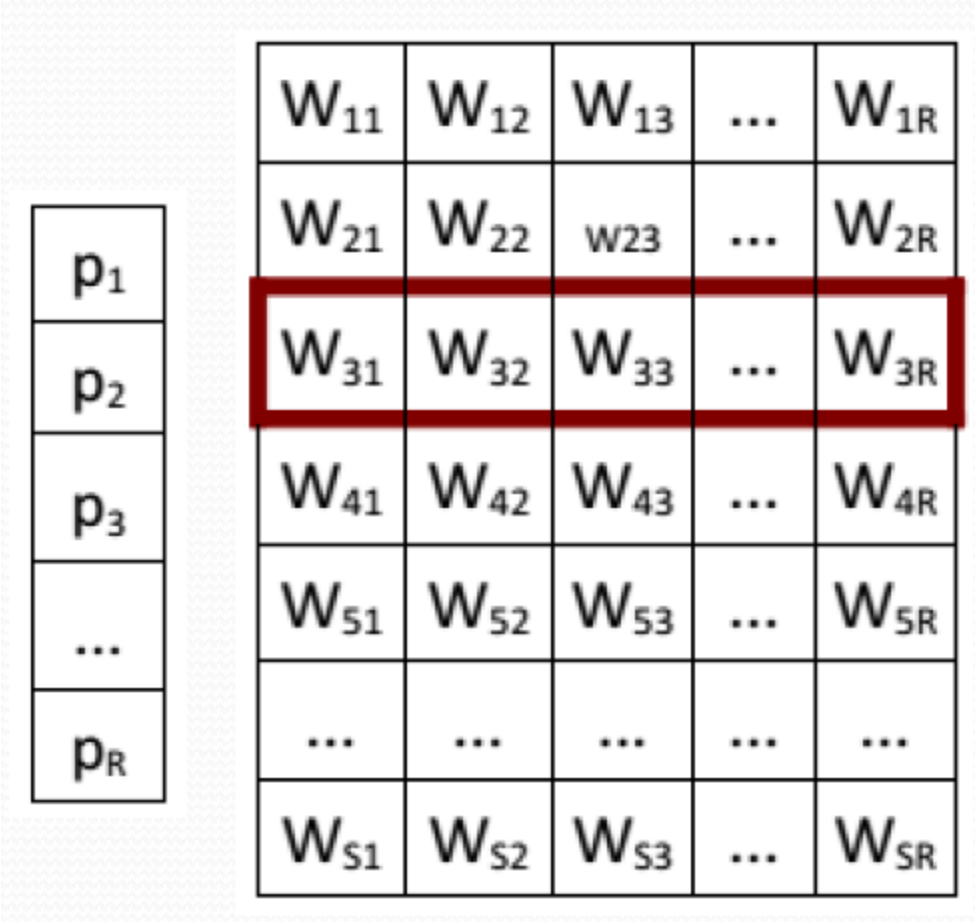
Kohonen Networks (Weights) cont

- Each row in this table represents the weights that start from a neuron and meet the features of the patterns.
- We can therefore assume that each row of the weights is itself a vector that R-dimensionsal (that is, the number of the features of the patterns introduced).

W_{11}	W_{12}	W_{13}	...	W_{1R}
W_{21}	W_{22}	W23	...	W_{2R}
W_{31}	W_{32}	W_{33}	...	W_{3R}
W_{41}	W_{42}	W_{43}	...	W_{4R}
W_{S1}	W_{S2}	W_{S3}	...	W_{5R}
...	...	...	...	...
W_{S1}	W_{S2}	W_{S3}	...	W_{SR}

Kohonen Networks (Weights) cont

- The input of a pattern on the network results in the comparison of its attributes with each row of the weight table.
- This comparison is made by calculating their Euclidean distance.
- The neuron representing this row is called a winner and has an output of 1 while all the others make an output of 0.



Network Training

Generally, for the training of the network, many patterns belonging to different clusters will be used.

- Network training consists of two steps:
  - The readjustment of weights.
  - The readjustment of biases.

Both have the ultimate goal of network training, but in theory each serves a different purpose.

The readjustment of weights

- After comparing the distances with the rows of the weight's matrix, the winner neuron is found.
- Then we will only modify the weights of the row that belongs to the particular pattern (weight row of the winning neuron), so that the winner neuron will look even more like the input vector.
- In this way, we assume that when the same pattern is presented again to the neural network, it will activate the same winner neuron, therefore it will recognize the pattern.

The readjustment of weights

- We can assume that when patterns that belong to the same group are presented to the network, the same neuron is always activated assuming that:
- Patterns belonging to the same category are "close" to each other and “away” from others belonging to different groups.
- When we present a large number of patterns to the network, the winner neuron is adjusted in such a way that it is as close to all patterns of this group as possible. In other words, it is close to the features of the group and does not memorize separate vectors.

The readjustment of biases

- It is possible that one of the neurons is far from the patterns of any class.
- Thus, it will never satisfy the criterion of the smallest distance from a pattern.
- Therefore, it will never be a winner neuron and its weights will never be adjusted to change this situation.

The readjustment of biases

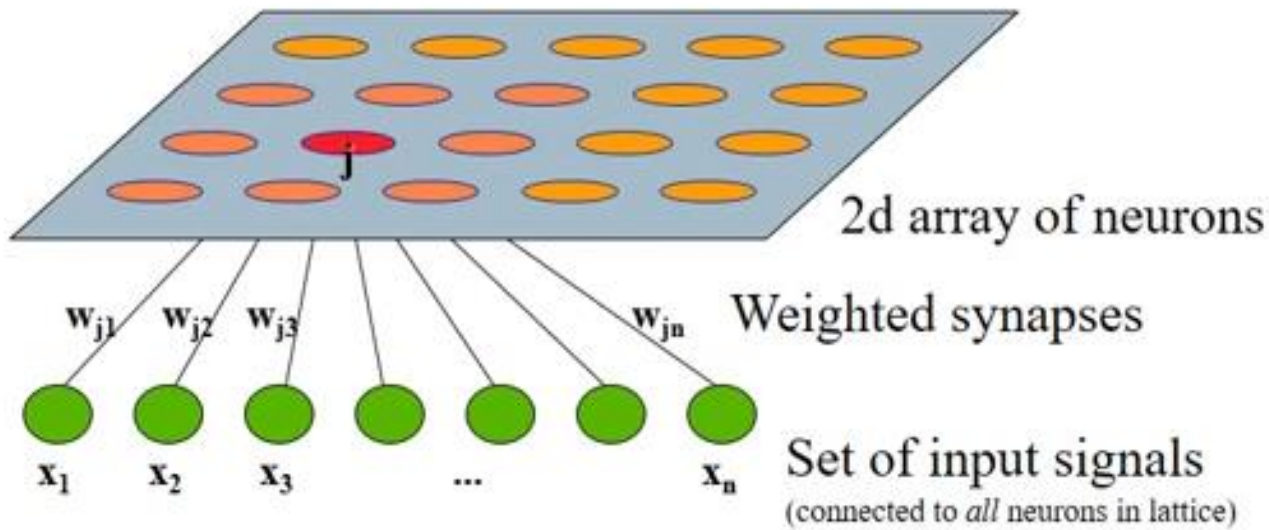
- If the network notices that one of the neurons remains inactive for a long time or never activates, then it will adjust the bias accordingly.
- The next time that its distance is calculated from an input vector, it subtracts from this distance a value equal to the bias (random value).
- This happens so that it can be a winner neuron and adjust its weights in order to look like one of the input vectors.

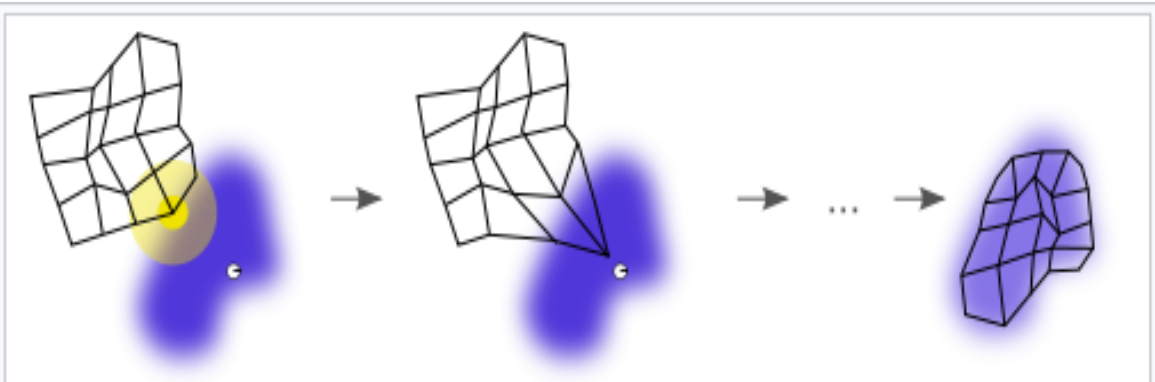
The readjustment of biases

The advantages of this method are:

- All neurons are activated to capture a class of patterns.
- If we have a group consisting of many patterns that can form distant sub-groups, two or more neurons are devoted to better capture the characteristics of this group.

The neurons in the output layer are arranged on a map

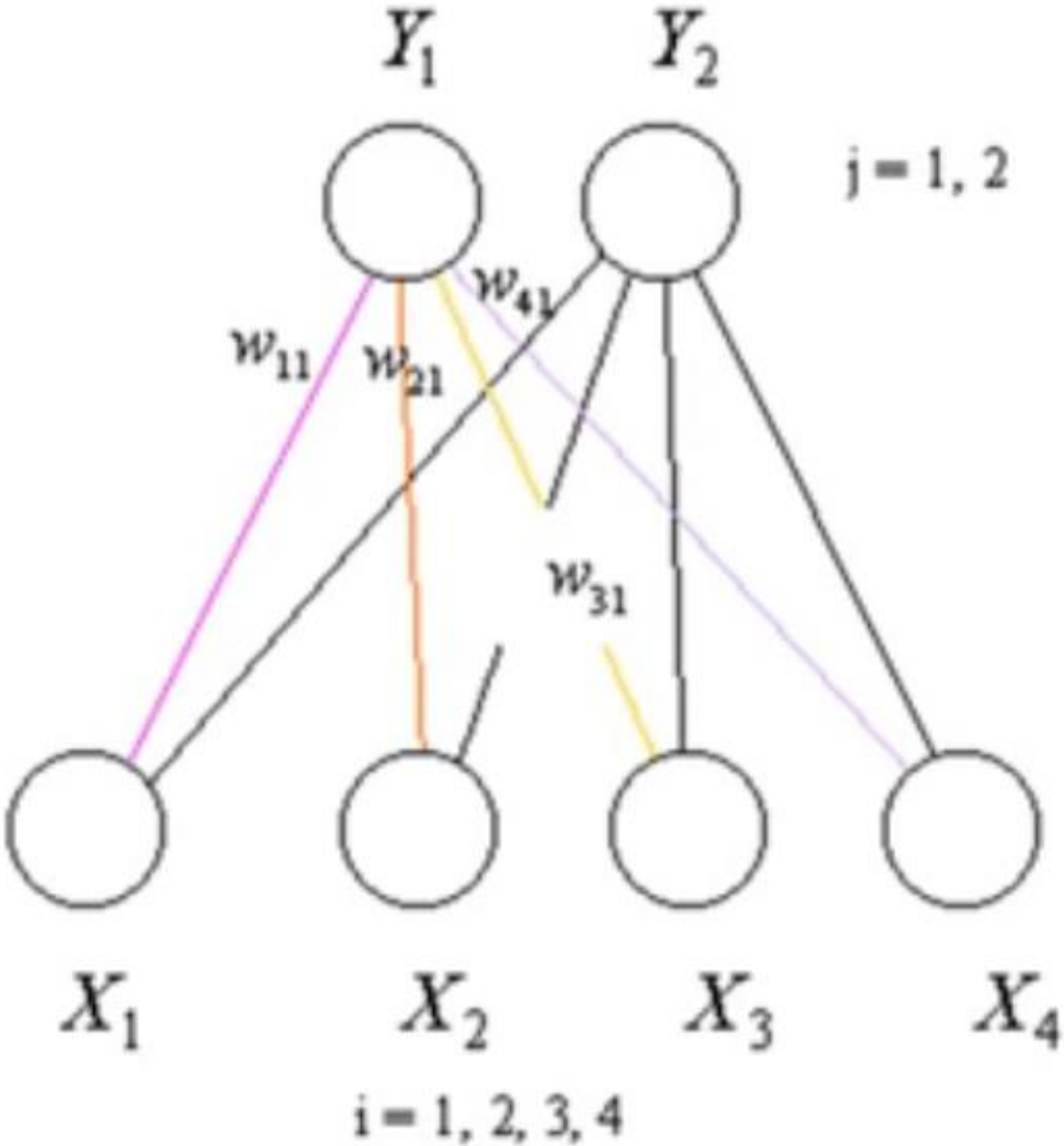




An illustration of the training of a self-organizing map. The blue blob is the distribution of the training data, and the small white disc is the current training datum drawn from that distribution. At first (left) the SOM nodes are arbitrarily positioned in the data space. The node (highlighted in yellow) which is nearest to the training datum is selected. It is moved towards the training datum, as (to a lesser extent) are its neighbors on the grid. After many iterations the grid tends to approximate the data distribution (right).

Source: https://en.wikipedia.org/wiki/Self-organizing_map

Step	Action
0	Initialize weights. Set max value for R, set learning rate α.
1	While stopping condition false do steps 2 to 8
2	For each input vector $\underline{\mathbf{x}}$ do steps 3 to 5
3	For each j neuron, compute the **Euclidean** distance $D(j) = \sqrt{\sum_{i=1}^{n} (x_i - w_{ij})^2}$
4	Find the index J such that $D(J)$ is a minimum
5	For all neurons j within a specified neighbourhood of J and for all i $w_{ij}(new) = w_{ij}(old) + \alpha(x_i - w_{ij}(old))$
6	Update learning rate α. It is a decreasing function of the number of epochs.
7	Reduce radius of topological neighbourhood at specified times
8	Test stopping condition. Typically this is a small value of the learning rate with which the weight updates are insignificant.



Let the initial weight matrix be

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \\ 0.5 & 0.7 \\ 0.9 & 0.3 \end{bmatrix}$$

x_1	x_2	x_3	x_4
1	1	0	0
0	0	0	1
1	0	0	0
0	0	1	1

Consider a simple example in which there are only **4 input training patterns**.

Let the learning rate at time $t + 1$ be given by $\alpha(t+1) = \frac{\alpha(t)}{2}$,

and suppose $\alpha(t=0) = 0.6$

Let topological radius $R = 0$.

Following the algorithm presented in the previous algorithm:

For vector 1100 *(We are using the Euclidean distance squared for convenience)*

$D(1) = (1-0.2)^2+(1-0.6)^2+(0-0.5)^2+(0-0.9)^2 = 1.86$

$D(1) = 1.86$, $D(2) = 0.98$

Hence J = 2. Note that $R = 0$, so we need not update the weights of any neighboring neurons.

Using $w_{ij}(new) = w_{ij}(old) + \alpha(x_i - w_{ij}(old))$, the new weight matrix is

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0.2 & 0.92 \\ 0.6 & 0.76 \\ 0.5 & 0.28 \\ 0.9 & 0.12 \end{bmatrix}$$

$$w_{12}(new) = w_{12}(old) + \alpha(x_1 - w_{12}(old)$$
$$= 0.8 + 0.6(1 - 0.8) = 0.92$$

Likewise for remains training patterns

For vector 0001

D(1) = 0.66, D(2) = 2.2768

Hence J = 1.

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0.08 & 0.92 \\ 0.24 & 0.76 \\ 0.20 & 0.28 \\ 0.96 & 0.12 \end{bmatrix}$$

For vector 0011

D(1) = 0.7056, D(2) = 2.724

Hence J = 1

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0.032 & 0.968 \\ 0.096 & 0.304 \\ 0.680 & 0.112 \\ 0.984 & 0.048 \end{bmatrix}$$

For vector 1000

D(1) = 1.8656, D(2) = 0.6768

Hence J = 2

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0.08 & 0.968 \\ 0.24 & 0.304 \\ 0.20 & 0.112 \\ 0.96 & 0.048 \end{bmatrix}$$

Now reduce learning rate (step 6):

$$\alpha(1) = \frac{\alpha(0)}{2} = \frac{0.6}{2} = 0.3$$

It can be shown that after **100 presentations** of all the input vector, the final weight matrix is

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 6.7x10^{-17} & 1 \\ 2x10^{-16} & 0.49 \\ 0.51 & 2.3x10^{-16} \\ 1 & 1x10^{-16} \end{bmatrix}$$

This matrix seems to converge to

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0.5 \\ 0.5 & 0 \\ 1 & 0 \end{bmatrix}$$

Cluster 1 Cluster 2

TEST NETWORK

Suppose the input pattern is 1100. This matrix seems to c

Then

$$D(j) = (w_{1j} - x_1)^2 + (w_{2j} - x_2)^2 + (w_{3j} - x_3)^2 + (w_{4j} - x_4)^2$$

$$D(1) = (0-1)^2 + (0-1)^2 + (0.5-0)^2 + (1-0)^2 = 3.25$$

$$D(2) = (1-1)^2 + (0.5-1)^2 + (0-0)^2 + (0-0)^2 = 0.25$$

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0.5 \\ 0.5 & 0 \\ 1 & 0 \end{bmatrix}$$

Cluster 1

Thus neuron 2 is the "**winner**", and is the localized active region of the SOM. Notice that we may label this input pattern to belong to cluster 2.

For all the other patterns, we find the clusters are as listed below.

x_1	x_2	x_3	x_4	Cluster
1	1	0	0	2
0	0	0	1	1
1	0	0	0	2
0	0	1	1	1

Function newc

To create a Kohonen network we use the function **newc** from neurolab library.

Syntax:

```
neurolab.net.newc(minmax, cn)
```

Parameters:

- **minmax:** list of list, the outer list is the number of input neurons, inner lists must contain 2 elements: min and maxRange of each input value
- **cn:** int, number of output neurons

Returns:

- **net:** Net

Example 1

Create a Kohonen network with 2 inputs and 10 neurons

```
import neurolab as nl
net = nl.net.newc([[-1, 1], [-1, 1]], 10)
```

Train Functions

The train functions for Kohonen networks are:

Function	Algorithm
train_cwta	Consience Winner Take All algorithm
train_wta	Winner Take All algorithm

Network Training

nl.train.train_cwta(net, input, epochs=a, show=b, goal=c)

where,

net: the Kohonen network we have created with newc function
input: array like, train input patterns
a: are the max number of epochs that we want to train the network
b: is the number of epochs' shows
c: is the minimum number of error we want to achieve

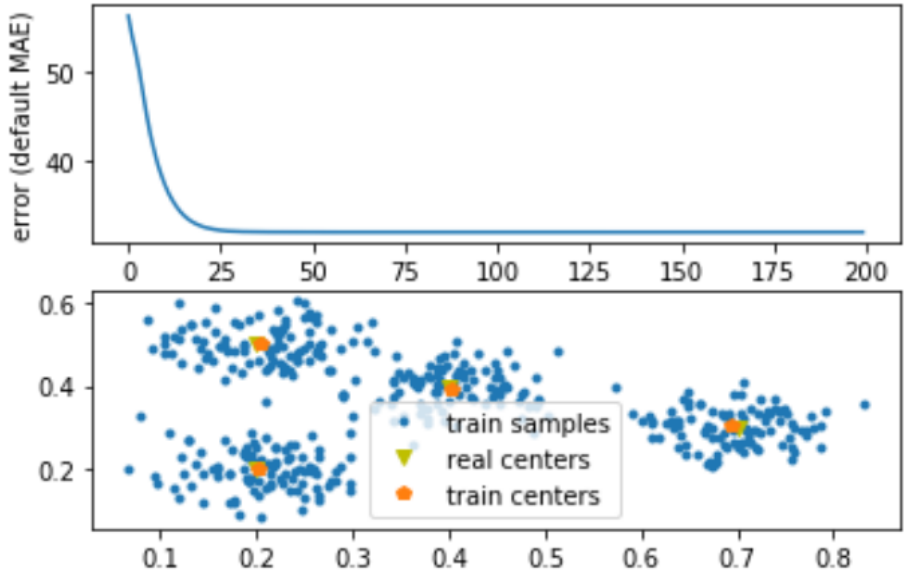
Example 2

```
import numpy as np
import neurolab as nl
import matplotlib.pyplot as plt
import numpy.random as rand

centr = np.array([[0.2, 0.2], [0.4, 0.4], [0.7, 0.3], [0.2, 0.5]])
rand_norm = 0.05 * rand.randn(100, 4, 2)
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * 4, 2)
rand.shuffle(inp)

# Create net with 2 inputs and 4 neurons
net = nl.net.newc([[0.0, 1.0],[0.0, 1.0]], 4)

# Train the network
error = net.trainf = nl.train.train_cwta(net, inp, epochs = 1000)
```

Example 2

```
# Plot results:
plt.title('Example 2')
plt.subplot(2,1,1)
plt.plot(error)
plt.xlabel('Epoch number')
plt.ylabel('error (default MAE)')
w = net.layers[0].np['w']

plt.subplot(2,1,2)
plt.plot(inp[:,0], inp[:,1], '.', \
      centr[:,0], centr[:, 1] , 'yv', \
      w[:,0], w[:,1], 'p')
plt.legend(['train samples', 'real centers', 'train centers'])
plt.show();
```

```
Epoch: 100; Error: 32.078282322610605;
Epoch: 200; Error: 32.07842169387743;
Epoch: 300; Error: 32.07842170339633;
Epoch: 400; Error: 32.078421703397055;
Epoch: 500; Error: 32.078421703397055;
Epoch: 600; Error: 32.078421703397055;
Epoch: 700; Error: 32.078421703397055;
Epoch: 800; Error: 32.078421703397055;
Epoch: 900; Error: 32.078421703397055;
Epoch: 1000; Error: 32.078421703397055;
The maximum number of train epochs is reached
```



Example 3

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl
import pylab as pl
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data

# Scale the data [0, 1]
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
X = min_max_scaler.fit_transform(X)
```

Example 3

Train the network

```
net   = nl.net.newc(nl.tool.minmax(X), 3)
# we choose 3 because we know that there are 3 categories.
However, we should use the Elbow method to find it.

error = net.trainf = nl.train.train_wta(net, X, epochs = 1000,
show = 200)

# Plot results:
plt.title('Example 3')
plt.subplot(2,1,1)
plt.plot(error)
plt.xlabel('Epoch number')
plt.ylabel('error (default MAE)')
w = net.layers[0].np['w']

plt.subplot(2,1,2)
plt.plot(X[:,2], X[:,3], '.', \
      w[:,2], w[:,3], 'p')
plt.legend(['train samples', 'train centers'])
plt.show();
```

```
Epoch: 200; Error: 49.70558453000503;
Epoch: 400; Error: 49.705584536073445;
Epoch: 600; Error: 49.705584536073445;
Epoch: 800; Error: 49.705584536073445;
Epoch: 1000; Error: 49.705584536073445;
The maximum number of train epochs is reached
```