

9 Competitive Neural Networks

the basic competitive neural network consists of two layers of neurons:

- The distance-measure layer,
- The competitive layer, also known as a “Winner-Takes-All” (WTA) layer

The structure of the competitive neural network is shown if Figure 9–1

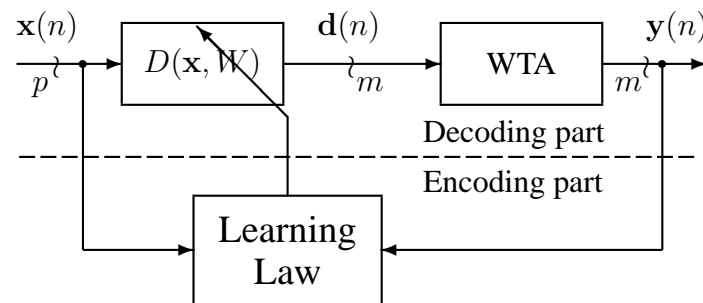


Figure 9–1: The structure of the competitive neural network

The **distance-measure** layer contains an $m \times p$ weight matrix, W , each row associated with one neuron.

This layer generates signals $\mathbf{d}(n)$ which indicate the distances between the current input vector $\mathbf{x}(n)$ and each synaptic vector $\mathbf{w}_j(n)$.

The competitive layer generates m binary signals y_j . This signal is asserted “1” for the neuron j -th winning the competition, which is the one for which the distance signal d_j attains minimum.

In other words, $y_j = 1$ indicates that the j -th weight vector, $\mathbf{w}_j(n)$, is most similar to the current input vector, $\mathbf{x}(n)$.

9.1 The Distance-Measure Layer

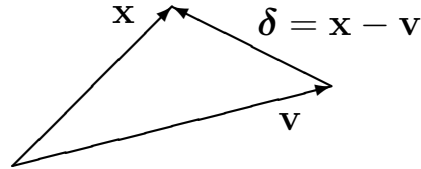
The detailed structure of the distance-measure layer depends on the specific measure employed.

Let

$$d = D(\mathbf{x}, \mathbf{v})$$

denote a distance, or similarity measure between two vectors, \mathbf{x} and \mathbf{v} . The following measures can be taken into considerations:

- The most obvious similarity measure is the Euclidean norm, that is, the magnitude of the difference vector, $\boldsymbol{\delta}$,



$$d = \|\mathbf{x} - \mathbf{v}\| = \|\boldsymbol{\delta}\| = \sqrt{\delta_1^2 + \dots + \delta_p^2} = \sqrt{\boldsymbol{\delta}^T \cdot \boldsymbol{\delta}}$$

Such a measure is relatively complex to calculate.

- The square of the magnitude of the difference vector

$$d = \|\mathbf{x} - \mathbf{v}\|^2 = \|\boldsymbol{\delta}\|^2 = \sum_{j=1}^p \delta_j^2 = \boldsymbol{\delta}^T \cdot \boldsymbol{\delta}$$

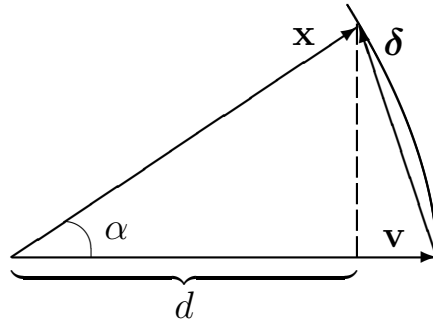
The square root has been eliminated, hence calculations of the similarity measure have been simplified.

- The Manhattan distance, that is, the sum of absolute values of the coordinates of the difference vector

$$d = \sum_{j=1}^p |\delta_j| = \text{sum}(\text{abs}(\boldsymbol{\delta}))$$

- The **projection** of \mathbf{x} on \mathbf{v} . This is the simplest measure of similarity of the **normalised** vectors:

$$d = \mathbf{v}^T \cdot \mathbf{x} = \|\mathbf{v}\| \cdot \|\mathbf{x}\| \cdot \cos \alpha$$



For normalised vectors, when $\|\mathbf{v}\| = \|\mathbf{x}\| = 1$ we have

$$d = \cos \alpha \in [-1, +1]$$

and also

- | | | |
|-------------|------------------------------|------------------------|
| if $d = +1$ | then $\ \delta\ = 0$ | vectors are identical |
| if $d = 0$ | then $\ \delta\ = \sqrt{2}$ | vectors are orthogonal |
| if $d = -1$ | then $\ \delta\ = 2$ | vectors are opposite |

In general, we have

$$\|\delta\|^2 = (\mathbf{x} - \mathbf{v})^T (\mathbf{x} - \mathbf{v}) = \mathbf{x}^T \mathbf{x} - 2\mathbf{x}^T \mathbf{v} + \mathbf{v}^T \mathbf{v} = \|\mathbf{x}\|^2 + \|\mathbf{v}\|^2 - 2\mathbf{v}^T \mathbf{x}$$

Hence, for normalised vectors, the projection of the similarity measure, d , can be expressed as follows

$$d = \mathbf{v}^T \mathbf{x} = 1 - \frac{1}{2} \|\delta\|^2$$

Normalisation of the input vectors can be achieved without any loss of information by adding another dimension to the input space during preprocessing of the input data.

Assume that $(p - 1)$ -dimensional input vectors, $\hat{\mathbf{x}}(n)$, have been already scaled into the $[-1, +1]$ range, that is,

$$||\hat{\mathbf{x}}(n)|| \leq 1 \quad \forall(n = 1, \dots, N)$$

If we add the p th component, x_p to $\hat{\mathbf{x}}$, we obtain a p -dimensional vector, \mathbf{x} and we can write the following relationship

$$||\mathbf{x}(n)||^2 = ||\hat{\mathbf{x}}(n)||^2 + x_p^2(n)$$

Now, in order to normalise all p -dimensional vectors, \mathbf{x} , the p th components must be calculated as follows

$$x_p^2(n) = 1 - ||\hat{\mathbf{x}}(n)||^2 \quad \forall(n = 1, \dots, N)$$

This operation is equivalent to the projection of the input data from the $(p - 1)$ -dimensional hyper-plane up onto the p -dimensional unity hyper-sphere, as illustrated in Figures 9–2 and 9–3.

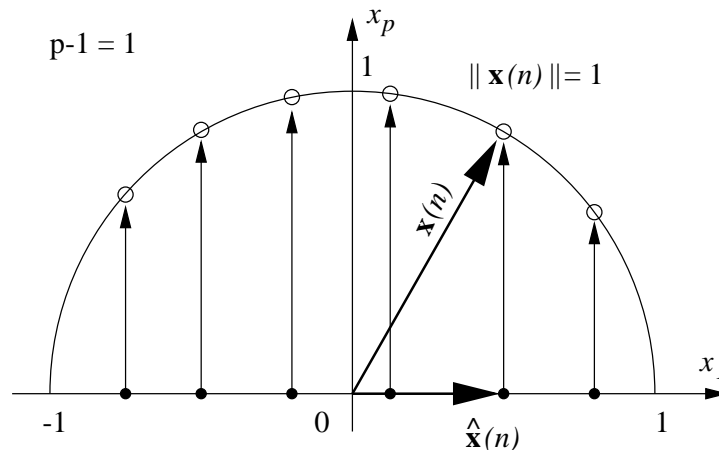


Figure 9-2: Normalisation of 1-D input data by projection onto a unity circle

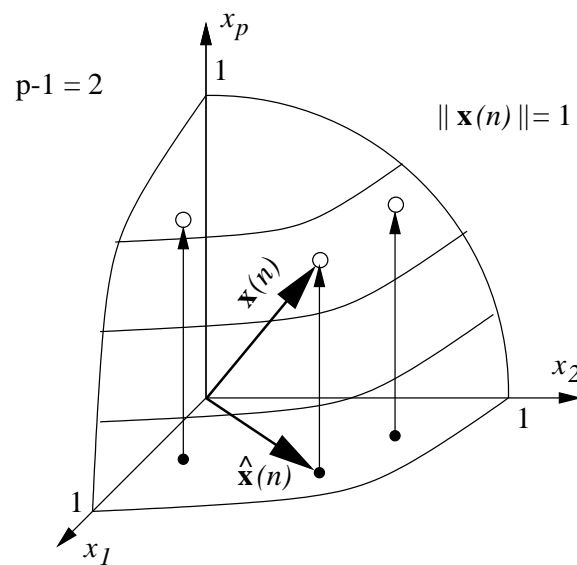


Figure 9-3: Normalisation of 2-D input data by projection onto a unity sphere

For the **normalised vectors** the distance-measure layer is linear, that is,

$$\mathbf{d} = W \cdot \mathbf{x}$$

The structure of the competitive neural network which employs projections as the distance measures is illustrated in Figure 9–4 in the form of a signal-flow block-diagram and the dendritic diagram.

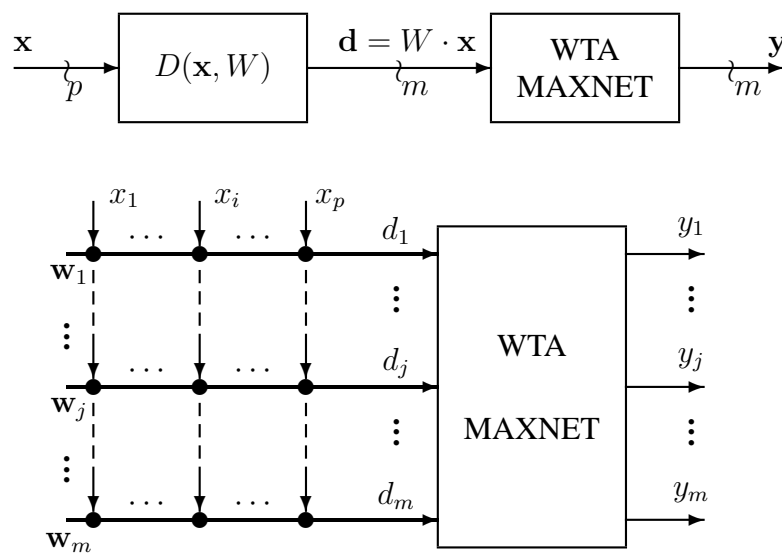


Figure 9–4: The structure of the distance-measure layer for the normalised vectors

The greater the signal $d_j(n)$ is, the more similar is the j th weight vector, w to the current input signal $\mathbf{x}(n)$.

9.2 The Competitive Layer

The competitive layer, also known as the MinNet (MaxNet), or the “Winner-Takes-All” (WTA) network, generates binary output signals, y_j , which, if asserted, point to the winning neuron, that is, the one with the weight vector being closest to the current input vector:

$$y_j = \begin{cases} 1 & \text{if } j = \arg \min_k D(\mathbf{x}, \mathbf{w}_k) \\ 0 & \text{otherwise} \end{cases}$$

In other words, the MaxNet (MinNet) determines the largest (smallest) input signal, $d_j = D(\mathbf{x}, \mathbf{w}_j)$.

The competitive layer is, in itself, a **recurrent** neural network with the predetermined and fixed feedback connection matrix, M . The matrix M has the following structure:

$$M = \begin{bmatrix} 1 & & & \\ & \ddots & -\alpha & \\ & -\alpha & \ddots & \\ & & & 1 \end{bmatrix}$$

where $\alpha < 1$ is a small positive constant. Such a matrix describes a network with a local unity feedback, and a feedback to other neurons with the connection strength $-\alpha$.

The MaxNet network

The MaxNet network is an implementation of the competitive layer described by the following block diagram

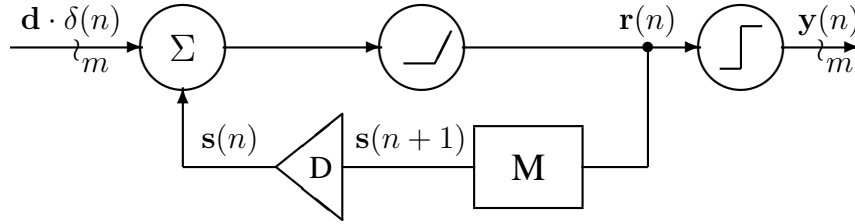


Figure 9–5: The block-diagram of the MaxNet network (competitive layer)

The input vector, \mathbf{d} , is active only at the initial time, $n = 0$, which is accomplished by means of the delta function

$$\delta(n) = \begin{cases} 1 & \text{for } n = 0 \text{ (initial condition)} \\ 0 & \text{for } n \neq 0 \end{cases}$$

The state equation, which describes how the state vector, $\mathbf{s}(n)$, evolves with time, can be written in the following form

$$\mathbf{s}(n+1) = M \cdot \mathbf{r}(n) + \mathbf{d} \cdot \delta(n)$$

where, the feedback signals $\mathbf{r}(n)$, are formed by clipping out the negative part of the state signals, that is,

$$r_j(n) = \max(0, s_j(n)) = \begin{cases} s_j(n) & \text{for } s_j(n) \geq 0 \\ 0 & \text{for } s_j(n) < 0 \end{cases} \quad \text{for } j = 1, \dots, m$$

Finally, the binary output signals $\mathbf{y}(n)$, are formed as follows

$$y_j(n) = \begin{cases} 1 & \text{for } r_j(n) > 0 \\ 0 & \text{for } r_j(n) = 0 \end{cases} \quad \text{for } j = 1, \dots, m$$

Alternatively, we can evaluate the state signals as:

$$\begin{aligned} \mathbf{s}(1) &= \mathbf{d} \\ \mathbf{r}(n) &= \max(0, \mathbf{s}(n)) \\ s_j(n+1) &= r_j(n) - \alpha \sum_{k \neq j} r_k, \text{ for } n > 1 \text{ and } j = 1, \dots, m \end{aligned}$$

At each step n , signals $s_j(n+1)$ consists of the self-excitatory contribution, $r_j(n)$, and a total lateral inhibitory contribution, $\alpha \sum_{k \neq j} r_k$.

After a certain number of iterations all r_k signals but the one associated with the largest input signal, d_j , are zeros.

Example

Let $\alpha = 0.2$ and $m = 8$. The feedback signals $\mathbf{r}(n)$ can be calculated as follows:

$\mathbf{r}(1)$	—	$\mathbf{r}(2)$	—	$\mathbf{r}(3)$	—	$\mathbf{r}(4)$	—	$\mathbf{r}(5)$	—	$\mathbf{r}(6)$
7.3	6.98	.32	.99	0	*	0	*	0	*	0
4.2	7.60	0	*	0	*	0	*	0	*	0
9.6	6.52	3.08	.44	2.64	.24	2.4	.13	2.27	.04	2.23
.7	8.30	0	*	0	*	0	*	0	*	0
5.5	7.34	0	*	0	*	0	*	0	*	0
2.9	7.86	0	*	0	*	0	*	0	*	0
8.6	6.72	1.88	.68	1.20	.53	.67	.48	.19	.45	0
3.4	7.76	0	*	0	*	0	*	0	*	0

The columns marked '—' represent the total inhibitory contribution.

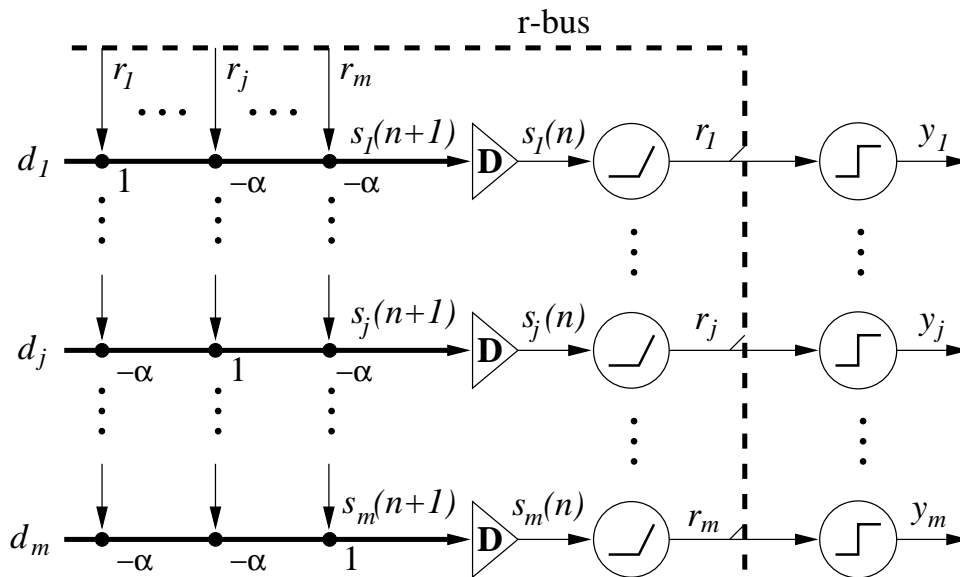
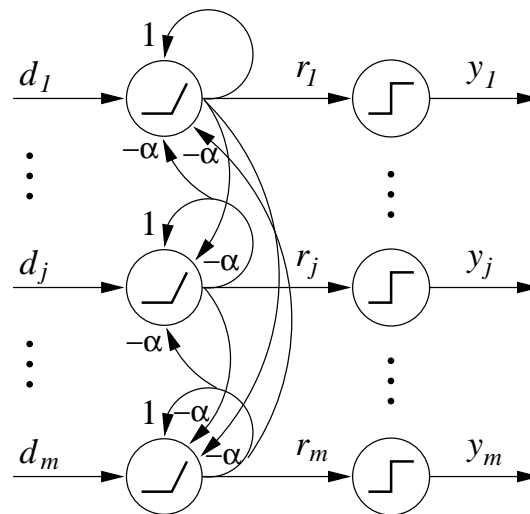
Dendritic view:**Signal-flow view:**

Figure 9–6: The dendritic and the signal-flow views of the competitive layer

Note the unity self-excitatory connections, and the lateral inhibitory connections.

9.3 Unsupervised Competitive Learning

The objective of the competitive learning is to adaptively quantize the input space, that is, to perform **vector quantization** of the input space

It is assumed that the input data is organised in, possibly overlapping, **clusters**. Each synaptic vector, \mathbf{w}_j , should converge to a **centroid** of a cluster of the input data. An example from a 2-D space is given in Figure 9–7.

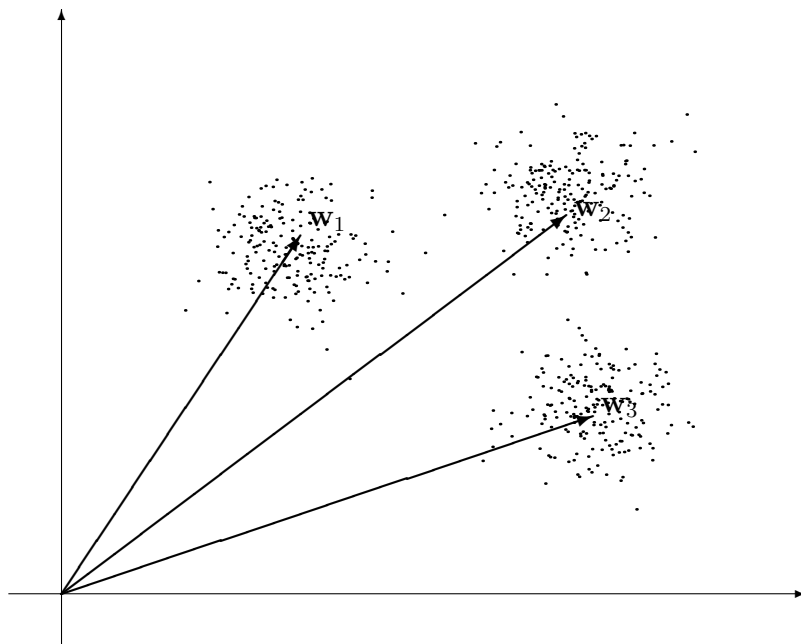


Figure 9–7: A 2-D pattern with three clusters of data

In other words, the input vectors are categorized into m classes (clusters), each weight vector representing the center of a cluster.

It is said that such a set of weight vectors describes **vector quantization** also known as **Voronoi** (or **Dirichlet**) **tessellation** of the input space.

Example of the Voronoi tessellation is given in Figure 9–8.

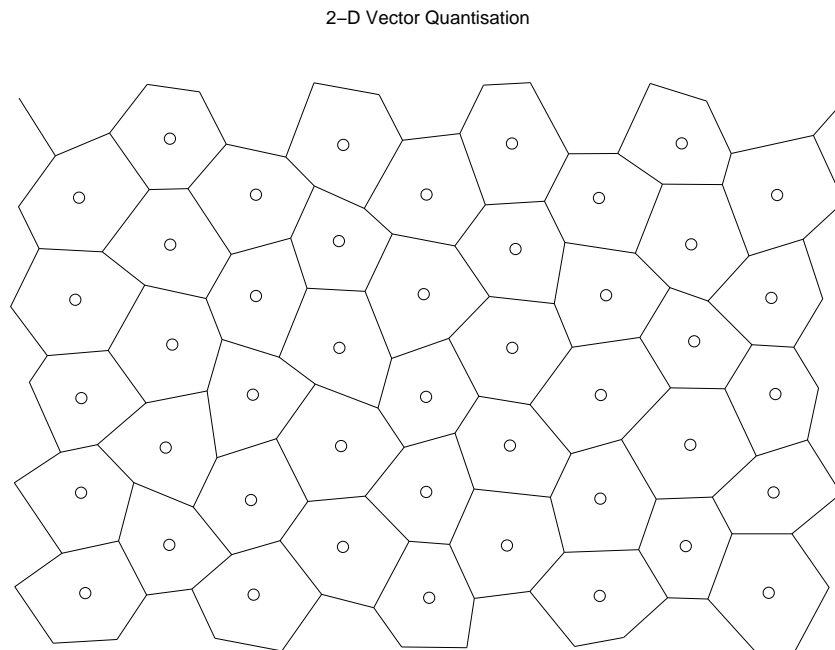


Figure 9–8: Voronoi tessellation (vector quantization) of a 2-D space.

The space is partitioned into polyhedral regions with centres represented by weight vectors (dots in Figure 9–8). The boundaries of the regions are planes perpendicularly bisecting lines joining pairs of centres (prototype vectors) of the neighbouring regions.

A very important application of the vector quantization is in data coding/compression. In this context the set of weights (prototype vectors) is referred to as a **codebook**.

We can find a set of prototype vectors with competitive learning algorithms.

A simple competitive learning

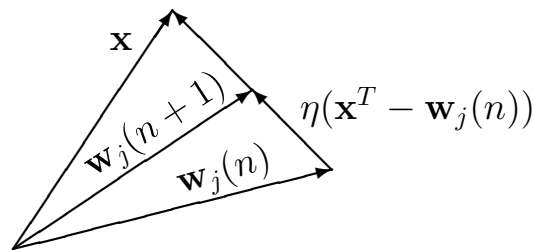
A simple competitive learning can be describe as follow:

- Weight vectors are usually initialise with n randomly selected input vectors:

$$\mathbf{w}_j(0) = \mathbf{x}^T(\text{rand}(j))$$

- For each input vector, $\mathbf{x}(n)$, determine the winning neuron, j for which its weight vector, $\mathbf{w}_j(n)$, is closest to the input vector. For this neuron, $y_j(n) = 1$.
- Adjust the weight vector of the winning neuron, $\mathbf{w}_j(n)$, in the direction of the input vector, $\mathbf{x}(n)$; do not modify weight vectors of the loosing neurons, that is,

$$\Delta \mathbf{w}_j(n) = \eta(n)y_j(n)(\mathbf{x}^T(n) - \mathbf{w}_j(n))$$



- In order to arrive at a static solution, the learning rate is gradually linearly reduced, for example

$$\eta(n) = 0.1\left(1 - \frac{n}{N}\right)$$

Example — scripts Cmpti.m, Cmpts.m

In this example we consider vector quantization of the 2-D input space into $m = 5$ regions specified by m weight vectors arranged in a 5×2 weight matrix W .

We start with generation of a 2-D pattern consisting of m clusters of normally distributed points. Weights are initialised with points from the data matrix X .

```
% Cmpti.m
% Initialisation of competitive learning
p = 2; m = 5 ;          % p inputs, m outputs
clst = randn(p, m); % cluster centroids
Nk = 100;               % points per cluster
N = m*Nk ;             % total number of points
sprd = 0.2 ;           % a relative spread of the Gaussian "blobs"
X = zeros(p,N+m); % X is p by m+N input data matrix
wNk = ones(1, Nk);
for k = 1:m % generation of m Gaussian "blobs"
    xc = clst(:,k) ;
    X(:,(1+(k-1)*Nk):(k*Nk))=sprd*randn(p,Nk)+xc(:,wNk) ;
end
[xc k] = sort(rand(1,N+m));
X = X(:, k) ;          % input data is shuffled randomly
winit = X(:,1:m)'; % Initial values of weights
X = X(:,m+1:N+m); % Data matrix is p by N
```

In the script implementing a simple competitive learning algorithm we pass once over the data matrix, aiming at weights to converge to the centres of Gaussian “blobs”.

```

% Cmpts.m
W = winit ;
V = zeros(N, m, p); % to store all weights
V(1, :, :) = W ;
wnm = ones(m, 1) ;
eta = 0.08 ; % learning gain
deta = 1-1/N ; % learning gain decaying factor
for k = 1:N % main training loop
    xn = X(:, k)' ;
% the current vector is compared to all weight vectors
    xmw = xn(wnm, :) - W ;
    [win jwin] = min(sum((xmw.^2), 2));
% the weights of the winning neurons are update
    W(jwin, :) = W(jwin, :) + eta*xmw(jwin, :);
    V(k, :, :) = W;
% eta = eta*deta ;
end

plot(X(1,:), X(2,:), 'g.', clst(1,:), clst(2,:), 'ro', ...
     winit(:, 1), winit(:, 2), 'bx' , ...
     V(:, :, 1), V(:, :, 2), 'b', W(:, 1), W(:, 2), 'r*'), grid

```

In Figure 9–9 there are four examples of data organised in five overlapping clusters. After one training epoch, the weights converged to centroids of the clusters with varying degree of success.

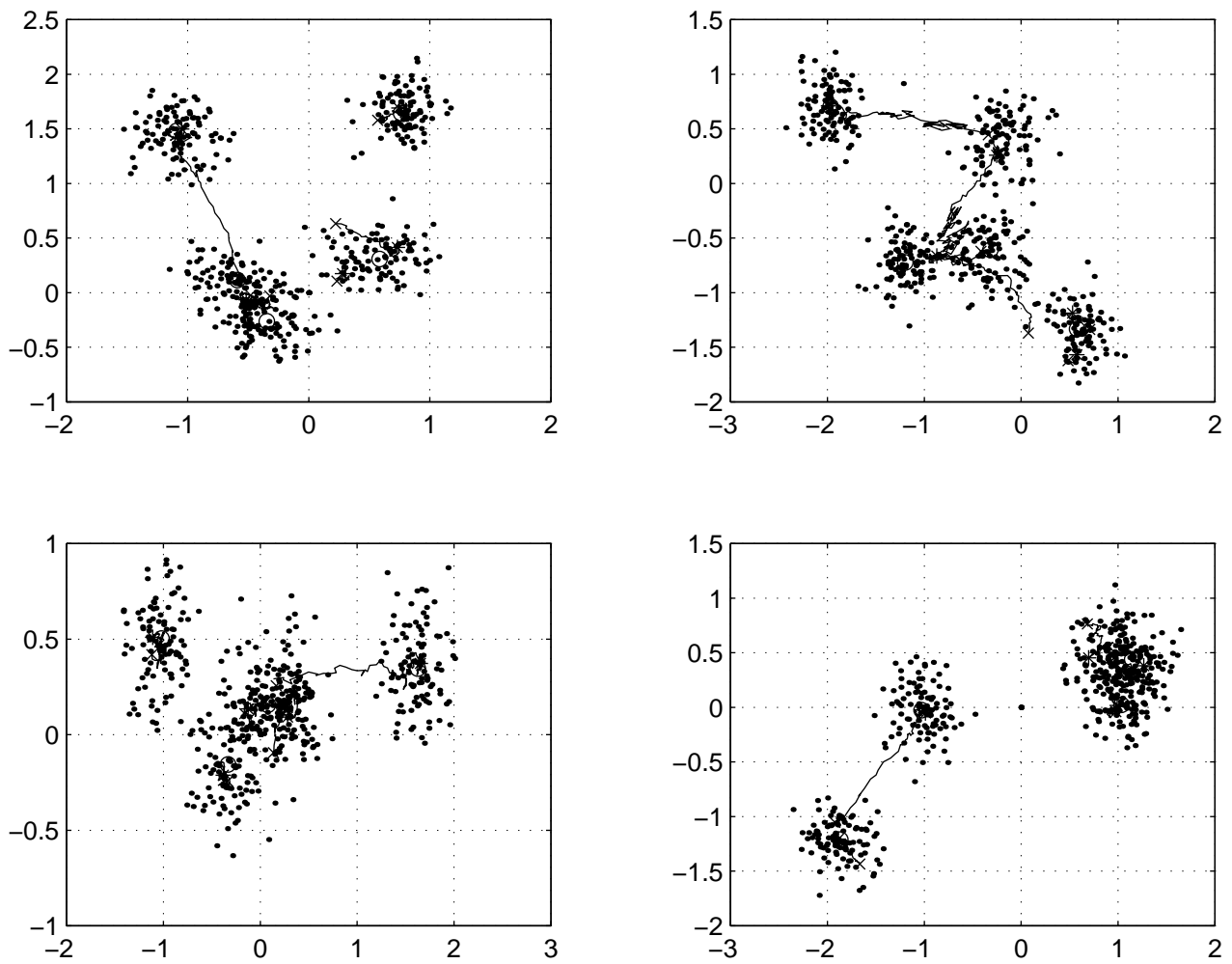


Figure 9–9: Simple competitive learning: ‘o’ – centroids of generated clusters, ‘x’ – initial weights, ‘*’ – Final weights

9.4 Competitive Learning and Vector Quantization

- Competitive learning is used to create a **codebook**, that is a weight matrix, W , which stores the centers of data clusters.

For a p -dimensional input space and m -neurons (size of the codebook) the structure of the codebook (weight matrix, W) is as follows

cluster #	W	1	...	p
1	\mathbf{w}_1	w_{11}	...	w_{1p}
2	\mathbf{w}_2	w_{21}	...	w_{2p}
\vdots	\vdots	\vdots	W	\vdots
m	\mathbf{w}_m	w_{m1}	...	w_{mp}

- The codebook (weight matrix) describes the tessellation of the input space known as **vector quantization** (VQ). For a 2-D input space the above concepts can be illustrated as in Figure 9–10.

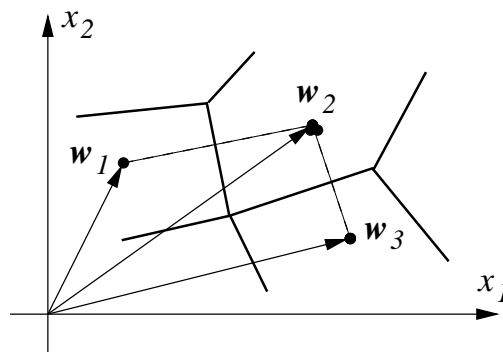


Figure 9–10: Voronoi tessellation (vector quantization) of a 2-D space.

- The codebook is created from a representative set of data using a competitive learning.

Vector quantization is used in **data compression**.

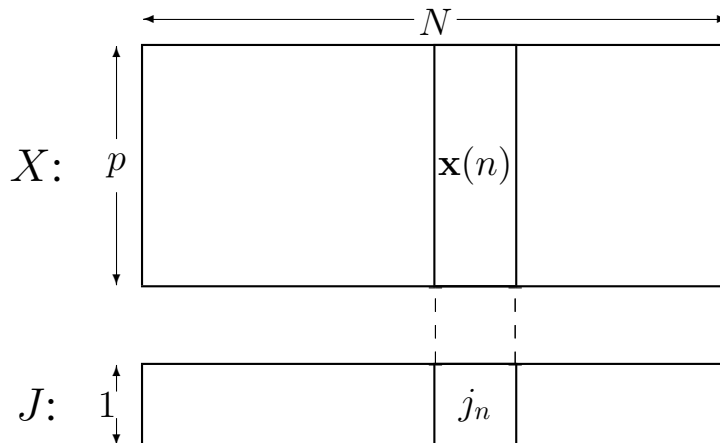
Given a codebook, $W = [\mathbf{w}_1; \dots; \mathbf{w}_1]$ the process of data compression can be described as follows:

1. For every input vector, $\mathbf{x}(n)$ find the closest codebook entry, that is the weight vector \mathbf{w}_{j_n} , for which the distance

$$|\mathbf{w}_{j_n} - \mathbf{x}(n)|$$

attains minimum. Then we identify that the input vector, $\mathbf{x}(n)$ belongs to the j -th cluster.

2. Replace all input data $X = \{\mathbf{x}(n)\}_{1:N}$ by their corresponding indices $J = \{j_n\}_{1:N}$.



3. Transmit J instead of X . The codebook, W must be made known/transmitted to the receiver.
4. At the receiver, using a codebook, replace every j_n with the corresponding \mathbf{w}_{j_n} which will now represent $\mathbf{x}(n)$. The representation error is:

$$\varepsilon_n = |\mathbf{w}_{j_n} - \mathbf{x}(n)|$$

The total squared error is

$$F = \sum_{n=1}^N \varepsilon_n^2$$

In order to calculate the **compression ratio**, C , let us assume that

- Weights and input data are represented by B -bit numbers,
- The length of the codebook

$$m \leq 2^b$$

that is, all cluster indices are b -bit numbers

Then,

- the total number of bits to represent the input data, that is, the size of X is

$$B \times p \times N$$

- The size of the compressed data, J , is

$$b \times N$$

- The size of the codebook, W , is

$$B \times p \times m$$

- The compression ratio is:

$$C = \frac{BpN}{bN + Bpm} \approx \frac{Bp}{b}$$

For example, for $B = 16$, $p = 12$, $b = 8$

$$C \approx 24$$

References

- [DB98] H. Demuth and M. Beale. *Neural Network TOOLBOX User's Guide. For use with MATLAB*. The MathWorks Inc., 1998.
- [FS91] J.A. Freeman and D.M. Skapura. *Neural Networks. Algorithms, Applications, and Programming Techniques*. Addison-Wesley, 1991. ISBN 0-201-51376-5.
- [Has95] Mohamad H. Hassoun. *Fundamentals of Artificial Neural Networks*. The MIT Press, 1995. ISBN 0-262-08239-X.
- [Hay99] Simon Haykin. *Neural Networks – a Comprehensive Foundation*. Prentice Hall, New Jersey, 2nd edition, 1999. ISBN 0-13-273350-1.
- [HDB96] Martin T. Hagan, H Demuth, and M. Beale. *Neural Network Design*. PWS Publishing, 1996.
- [HKP91] Hertz, Krogh, and Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991. ISBN 0-201-51560-1.
- [Kos92] Bart Kosko. *Neural Networks for Signal Processing*. Prentice-Hall, 1992. ISBN 0-13-617390-X.
- [Sar98] W.S. Sarle, editor. *Neural Network FAQ*. Newsgroup: comp.ai.neural-nets, 1998. URL: <ftp://ftp.sas.com/pub/neural/FAQ.html>.