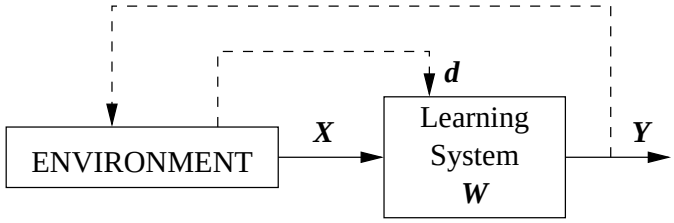
6 Learning and Self-Organization

Partly related to Lytton's Chapter 9.

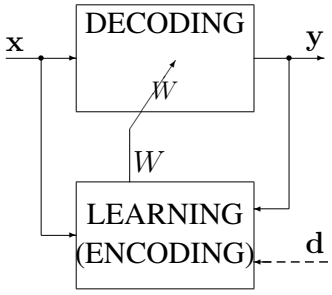
6.1 Introduction to learning

- In the previous sections we concentrated on the **decoding** part of a neural network assuming that the **weight matrix**, W, is given, or has been designed as in the Limulus vision model.
- If the weight matrix is satisfactory, during the decoding process the network performs some useful task it has been designed to do.
- In simple or specialised cases the weight matrix can be pre-computed, but more commonly it is obtained through the **learning** process.
- Learning can be thought of as a way of extracting information about the environment.
- The network (learning system) receives data or signals $\mathbf{X}$ from the environments and after processing stores them in the synaptic weight parameters



- In addition the network can produce signals $\mathbf{Y}$ that might influence the environment and receive additional signal reinforcing the process of learning.
- In general, learning is a dynamic process which modifies the weights of the network $\mathbf{W}$ in some desirable way.

- More specifically, a **neural network with learning** has the following structure consisting of the **decoding** part and **learning** (or encoding part:
- As any dynamic process learning can be described either in the continuous-time or in the discrete-time framework.



- In continuous-time we use the following **differential equation**:

$$\dot{W}(t) = L(\, W(t), \mathbf{x}(t), \mathbf{y}(t), \mathbf{d}(t)\,) \tag{6.1}$$

- In discrete-time we use the equivalent **difference equation**:

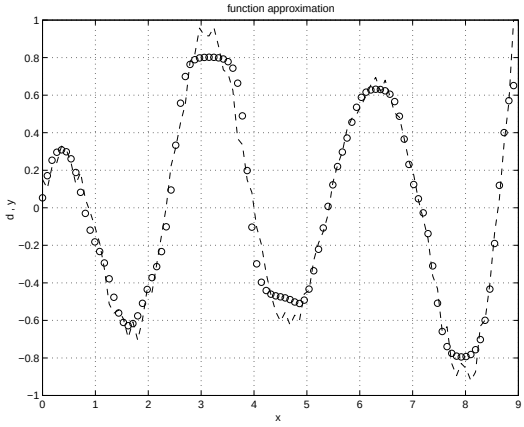
$$W(n+1) = L(\, W(n), \mathbf{x}(n), \mathbf{y}(n), \mathbf{d}(n)\,) \tag{6.2}$$

- $\mathbf{d}$ is an external teaching/supervising signal used in **supervised learning**.
- This signal in not present in networks employing the **unsupervised learning** paradigms.
- The discrete-time learning law is often used in a form of a **weight update** equation:

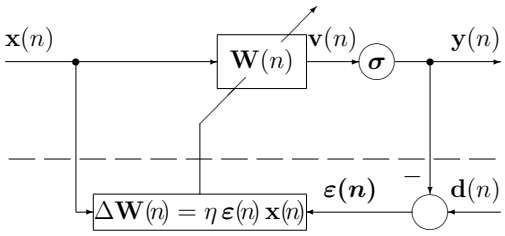
$$\begin{aligned} W(n+1) &= W(n) + \Delta W(n) \\ \Delta W(n) &= \mathcal{L}(\, W(n), \mathbf{x}(n), \mathbf{y}(n), \mathbf{d}(n)\,) \end{aligned} \tag{6.3}$$

6.2 Supervised and Unsupervised Learning

- Learning paradigms are typically classified into two big groups: **supervised** and **unsupervised** learning.
- We will concentrate on unsupervised algorithm as they seem to me more relevant to brain functions.
- **Supervised learning** algorithms are typically applied in a situation when the data is organized as a function $\mathbf{d} = f(\mathbf{x})$ and the network, typically a multilayer perceptron, approximates the unknown functional relationship.



- In this case the training data is divided into input signals, $\mathbf{x}(n)$, and target signals, $\mathbf{d}(n)$.
- Example of one-dimensional function approximation can look as illustrated.
- In a more general case we can imagine that the data we approximate is located at some hyper-surface described by $\mathbf{d} = f(\mathbf{x})$



- A typical supervised learning algorithm, known as **error-correcting** learning is driven by error signals $\boldsymbol{\varepsilon}(n)$ which are the differences between the actual network output, $\mathbf{y}(n)$, and the desired (or target) output $\mathbf{d}(n)$, for a given input:

$$\boldsymbol{\varepsilon}(n) = \mathbf{d}(n) - \mathbf{y}(n)$$

- The weight update can be expressed in the following general form

$$\Delta\mathbf{w}(n) = \mathcal{L}(\mathbf{w}(n), \mathbf{x}(n), \boldsymbol{\varepsilon}(n))$$

where $\mathcal{L}$ represents a learning algorithm.

- If we say that a neural network can describe a model of data, then a multilayer perceptron describes the data in a form of a curve, or (hyper)surface which approximates a functional relationship between $\mathbf{x}(n)$, and $\mathbf{d}(n)$.

Self-organizing neural networks — Unsupervised Learning

- Self-organising neural networks employ unsupervised learning laws and discover **characteristic features** in input data without using a target or desired output.
- Information about the characteristic features of input data is created during the learning process and stored in the synaptic weights.
- Output signals describe relationship between the current input signals and the weight vectors.
- Two basic groups of unsupervised learning algorithms and related self-organizing neural networks, namely:
  - (Generalised) Hebbian Learning
  - Competitive Learning

  can be distinguished by the type of characteristic features that they "discover" from the input data, namely, "shape" of data and constellation of clusters of points.

Generalised Hebbian Learning

- Generalised Hebbian Learning extracts from data a set of **principal directions** along which data is organised in a p-dimensional space.
- Each direction is represent by a relevant weight vector. The number of those principal directions is, at most, equal to the dimensionality of the input space p.
- In an illustrative example presented in Figure 6–1 the two-dimensional data is organised along two principal directions, $\mathbf{w}_1$ and $\mathbf{w}_2$.

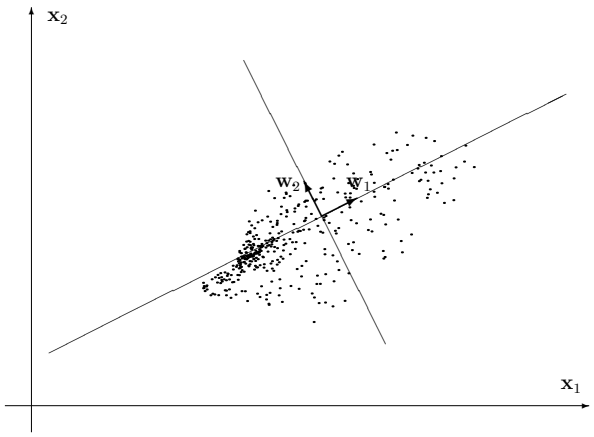
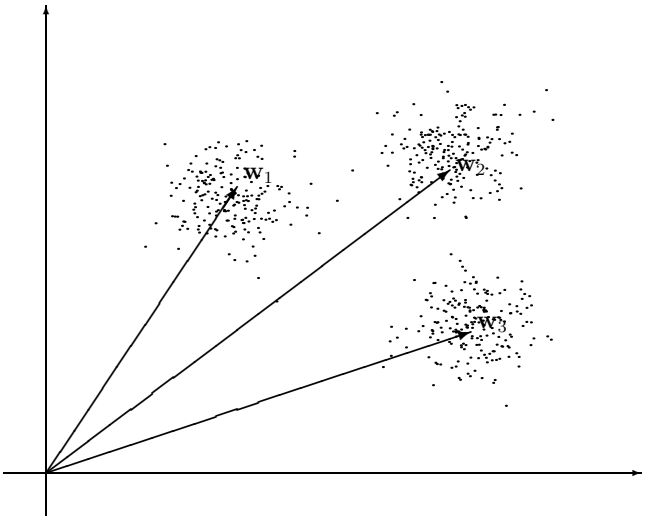


Figure 6–1: A 2-D pattern with principal directions

Competitive Learning

- Competitive Learning extracts from data a set of **centers of data clusters**.

- Each center point is stored as a weight vector.
- Note that the number of clusters is independent of dimensionality of the input space.
- Example of two-dimensional data organised in three clusters.
  Cluster centres are represented by three weight vectors.



- An important extension of a basic competitive learning is known as **feature maps**.
  A feature map is obtained by adding some form of topological organization to neurons.

6.3 Hebbian learning

6.3.1 Introductory concepts

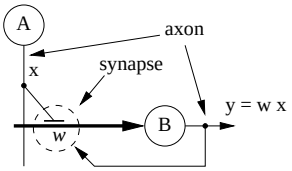
Donald Hebb, a Canadian psychologist, stated in 1949:

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

This fundamental statement is known as Hebb's law. When Hebb made his statement it was a hypothesis. It has since been verified in some parts of the brain so the term "law" is now legitimate.

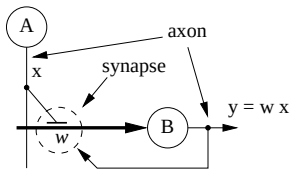
To incorporate Hebb's law in our computer simulations we must give it a mathematical formulation.

- Consider a simple network consisting of the two neurons A and B and their connecting synapse.
- The neuron on the presynaptic side, A, has the efferent signal x on its axon and the neuron on the postsynaptic side, B, has the efferent signal y on its axon.
- The synapse is assumed to have the strength w.



Formalizing the Hebb's law we say that **learning is based on modification of synaptic weights** depending both on the afferent and efferent signals and possibly on the current values of weights.

We typically start with an additive weight modification in the form:



$$w(n+1) = w(n) + \Delta w(n) \tag{6.4}$$

where n denotes the processing step, $w(n+1), w(n)$ are two subsequent values of the weight, and $\Delta w(n)$ is the weight change, or update.

This weight modification mechanism that takes place in the synapse is symbolised by the dashed circle.

- The obvious first mathematical formulation of Hebb's law is in the form of the **product of afferent and efferent signals** as follows:

$$\Delta w = \alpha\, x\, y \qquad \text{where } \alpha \text{ is the "learning rate".} \tag{6.5}$$

- When both x and y are large, i.e., both A and B are firing, w is increased.

  When one or both of x and y are small, w is changed very little.

  This may be seen as formalization of **Long Term Potentiation** (LTP).

- It is obvious that if A and B keep firing together in time during a lifetime, w would **grow very large**.

  This could not be biologically sustained and it would not make any sense anyway.

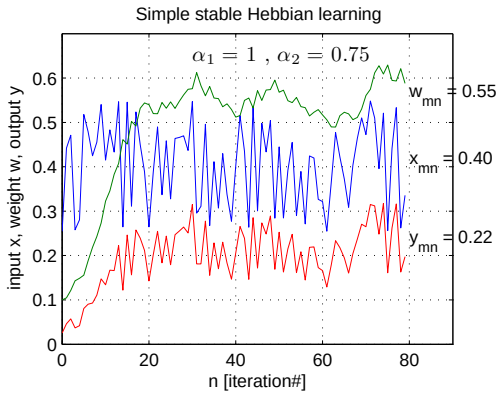
- One contribution of computational neuroscience is that it showed that Hebb's law is not sufficient for stable learning.

  We must introduce a mechanism for decreasing weight w, because "forgetting" is an integral part of learning.

- There are several ways of doing this, one choice is discussed in pracs. Another way of introducing forgetting is the following weight modification rule:

$$\Delta w = \alpha_1\, x\, y - \alpha_2\, y\, w \tag{6.6}$$

where α_1 and α_2 are two learning rates. Such a learning law should produce the following results:



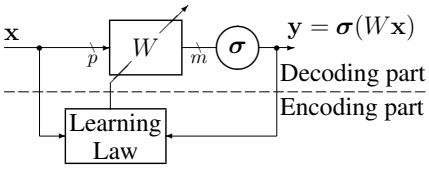
- In the example the afferent signal, $x(n)$ varies randomly around value $x_{mn} = 0.4$
- The initial value of the weight $w(0) = 0.1$.
- The mean value of the weight w_{mn} oscillates around the value x_{mn}/α_2 thus capturing the essential feature of the afferent signal.
- The efferent signal is simply equal to $y(n) = w(n) \cdot x(n)$

- This introductory example demonstrates a principle of **self-organization**:

  after the process of learning stabilizes, the synaptic weight has a value depending on the afferent signal(s).

6.3.2 Basic structure of Hebbian learning neural networks

The following block diagram illustrates a general concept of Hebbian learning:



- The upper section of the network, called sometimes the **decoding part**, is a single layer network specified by an $m \times p$ weight matrix, W.
- As usual each row of the weight matrix is associated with one neuron.
- The activation function σ keeps values of the signals between 0 and 1. For simplicity, we consider first a linear activation function.

- The learning law is implemented by the **encoding part** of the network and the block-diagram illustrates that the **modification of weights during learning** is a function of afferent and efferent signals.

  In general we can write

$$w_{ji}(n+1) = f(w_{ji}(n), y_j(n), x_i(n)) \tag{6.7}$$

  which means that the next value of the ji weight is a function of the current value of weight and afferent and efferent signals.

$$w_{ji}(n+1) = f(w_{ji}(n), y_j(n), x_i(n))$$

- Note that two signals, y_j and x_i and weight w_{ji} are locally available at the ji synapse, therefore, we often say that Hebbian learning law is an example of a **local learning law**.
- The concept of the **local learning** law is further illustrated in Figure 6–2.

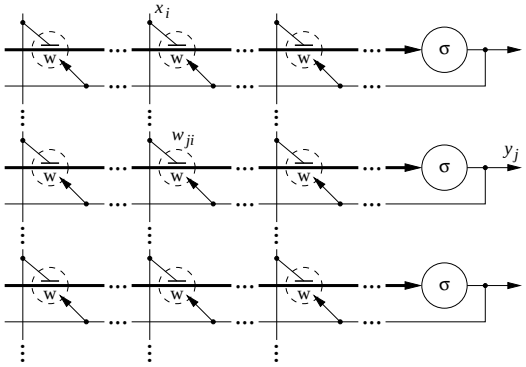


Figure 6–2: A neural network with a local learning law

- Note that each synapse, represented by a dashed circle, receives locally available afferent and efferent signals, say x_i, y_j, so that its weight w_{ji} can be modified according to a specified learning law.
- The local feedback provides the efferent signal, y_j to all synapses along the neuron's dendrite.
- This feedback is essential for a learning process to occur.

6.3.3 Stable Hebbian learning. Oja's rule

One of the important forms of a stable Hebbian learning law, know as Oja's rule uses the **augmented afferent signals**, $\tilde{x}_i$ and can be written in the following form:

$$\Delta w_i = \alpha \cdot y \cdot \tilde{x}_i \,, \quad \text{where} \quad \tilde{x}_i = x_i - y \cdot w_i \tag{6.8}$$

Hence the input signal is augmented by a product of the output signal and the synaptic weight.
The Oja's rule is primarily formulated for a single neuron with p synapses and can be written in a vectorised form in the following way:

$$\Delta \mathbf{w} = \alpha \cdot y \cdot \tilde{\mathbf{x}}^T = \alpha \cdot y \cdot (\mathbf{x}^T - y \cdot \mathbf{w}) \,, \quad \text{where} \quad \tilde{\mathbf{x}}^T = \mathbf{x}^T - y \cdot \mathbf{w} \,, \quad \text{and} \quad y = \mathbf{w} \cdot \mathbf{x} \tag{6.9}$$

where

$\Delta \mathbf{w} = \mathbf{w}(n+1) - \mathbf{w}(n)$ is an update of the weight vector,

$\mathbf{w} = [w_1 \dots w_p]$ is a p-component weight row vector,

$\mathbf{x} = [x_1 \dots x_p]^T$ is a p-component column vector of afferent signals

$\tilde{\mathbf{x}} = [\tilde{x}_1 \dots \tilde{x}_p]^T$ is a p-component column vector of augmented afferent signals.

y is the efferent signal created as an inner product of weights and afferent signals.

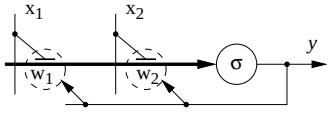
The important property of the learning law as in eqn (6.9) is the fact that the **length (Euclidian norm) of the weight vector tends to unity**, that is,

$$||\mathbf{w}|| \longrightarrow 1 \tag{6.10}$$

This important and useful condition means that Oja's rule is a **stable learning law**.

We will now demonstrate that the network implementing Oja's rule is a **principal direction detector**.

Consider a neuron with two synapse that implements Oja,s learning law described in the following way:



- for each time step n we have:

$$y(n) = \begin{bmatrix} w_1(n) & w_2(n) \end{bmatrix} \begin{bmatrix} x_1(n) \\ x_2(n) \end{bmatrix} = \mathbf{w}(n) \cdot \mathbf{x}(n)$$

and the **weight update** is calculated as:

$$\begin{aligned} \Delta w_1(n) &= \alpha\, y(n)\, (x_1(n) - y(n)\, w_1(n)) \\ \Delta w_2(n) &= \alpha\, y(n)\, (x_2(n) - y(n)\, w_2(n)) \end{aligned} \quad \text{or simply} \quad \Delta \mathbf{w} = \alpha\, y\, (\mathbf{x}^T - y\, \mathbf{w})$$

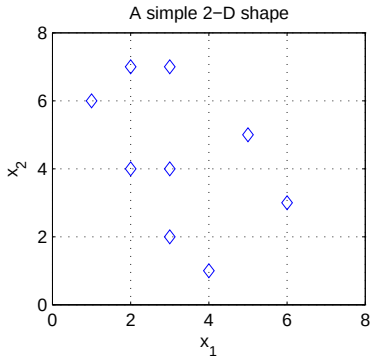
Assume that we have a collection of N points $\mathbf{x}(n)$ which are arranged into a $p \times N$ matrix X

$$X = [\mathbf{x}(1) \;\; \mathbf{x}(2) \;\; \dots \;\; \mathbf{x}(N)]$$

where

$$\mathbf{x}(n) = \begin{bmatrix} x_1(n) \\ x_2(n) \end{bmatrix}$$

Such a collection of points represented by afferent signals can be thought of as a **shape**.



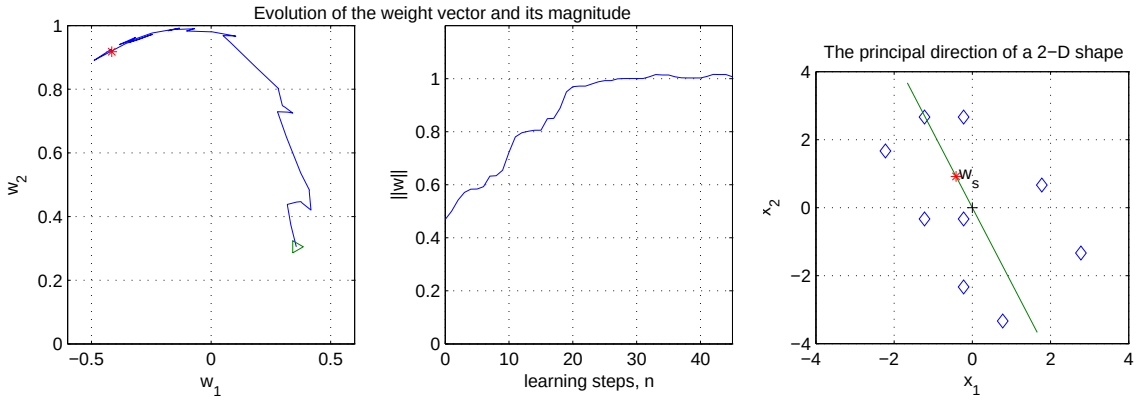
Now we will try to analyze the behaviour of a neuron implementing **Oja's rule based learning** when presented with afferent signals representing the shape.

- We start with pre-processing of our data **removing its mean**:

$$X = X - \text{mean}(X)$$

- This operation moves the coordinate frame to the center of the shape so that we can evaluate the shape itself **independently of its position in space**.
- Interestingly enough, removal of the mean can be performed using the familiar lateral inhibition with a Mexican hat mask.
- To initialize the **learning process** we can start with a randomly selected weight vector $\mathbf{w}(0)$
- Next we applied the efferent vectors $\mathbf{x}(n)$ one by one calculating the resulting output signal $y(n)$, the weight update $\Delta\mathbf{w}(n)$ and the next weight vector $\mathbf{w}(n+1)$
- One pass through all input vectors is called an epoch.
- The learning procedure is repeated for a number of epochs until the convergence of the weight vector is achieved.
- It can be shown that in Oja's learning algorithm the **convergence** is achieved when the **length of the weight vector approaches unity**.
- The speed of convergence depends on the learning rate α

The results of the learning process, that is, self-organization are presented below:



- The first two plots present the evolution of the weight vector and its length during learning process.
- Note that the length of the weight vector (send plot) is close to unity after approx. 20 learning steps
- Note also from the first figure that when the length is close to unity, the weight vector moves on a segment of a **unity circle**.
- Note from the third plot that the weight vector bisects the shape in the direction where it is the **most spread**, that is, where there is most variation of data. This is called the **principal direction** of the shape.
- Hence, a neuron, which implements Hebbian learning according to the Oja's rule, is a principal direction detector.

- It is also possible to extend the Oja's rule in such a way that the network is able to discover the **next principal direction**, orthogonal to the main principal direction.

In summary:

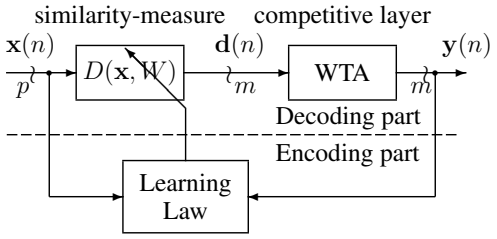
- Hebbian learning considered in this section aims at organising the synaptic weights in such a way that they approximate the shape of the data.
- The shape of data can be approximated by its principal direction (components),
  the first gives the direction of the highest variability of data,
  the second is orthogonal to the first direction and indicates the highest variability in that direction, and so on.
- Mathematically the procedure is known as the principal component analysis.

6.4 Competitive learning

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

- The similarity-measure layer,
- The competitive layer, also known as a "Winner-Takes-All" (WTA) layer

The block-structure of the competitive neural network

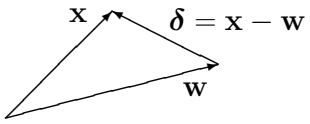


- The **similarity-measure** layer contains an $m \times p$ weight matrix, W, each row associated with one neuron.
- This layer generates signals $\mathbf{d}(n)$ which indicate the distances between the current input vector $\mathbf{x}(n)$ and each synaptic vector $\mathbf{w}_j(n)$.
- The competitive 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 similarity 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, $\mathbf{x}(n)$.

6.4.1 The Similarity-Measure Layer

The similarity-measure layer calculate the **similarity** between each stimulus and every weight vector. Such similarity can be measure in a number of possible ways.

- The most obvious way of measuring similarity or the distance between two vectors is the Euclidean norm (length) of the difference vector, $\boldsymbol{\delta}$,



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

  The problem is that such a measure of similarity is relatively complex to calculate.

- A simpler way of measuring the similarity between two vectors is just the square of the length of the difference vector

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

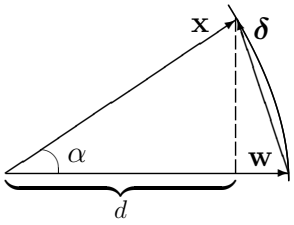
  The square root has been eliminated, hence calculations of the similarity measure have been simplified.

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

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

- The **projection** of $\mathbf{x}$ on $\mathbf{w}$. This is the simplest measure of similarity that works particularly well for the **normalised** vectors. The projection is calculated using the inner product of two vectors, namely (assuming both are column vectors):

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



  For normalised vectors, when $||\mathbf{w}|| = ||\mathbf{x}|| = 1$

  we have

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

  and also

$$\begin{aligned} &\text{if} \quad d = +1 \quad \text{then} \quad ||\boldsymbol{\delta}|| = 0 \quad \text{vectors are identical} \\ &\text{if} \quad d = 0 \quad \text{then} \quad ||\boldsymbol{\delta}|| = \sqrt{2} \quad \text{vectors are orthogonal} \\ &\text{if} \quad d = -1 \quad \text{then} \quad ||\boldsymbol{\delta}|| = 2 \quad \text{vectors are opposite} \end{aligned}$$

- In general, the square of the norm of the difference vector can be calculated as

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

- Hence, for normalised vectors, the projection as the similarity measure, d, can be expressed in terms of the norm of the difference vector as follows

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

- Note that the **greater** the signal d is, the **more similar** is the weight vector, $\mathbf{w}$ to the input signal $\mathbf{x}$.
- In summary we say that for the **normalised vectors** the similarity-measure performs the "typical" operation of signal aggregation, that is:

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

The structure of the similarity-measure layer is illustrated in Figure 6–3 in the form of a typical dendritic diagram and related signal-flow block-diagram.

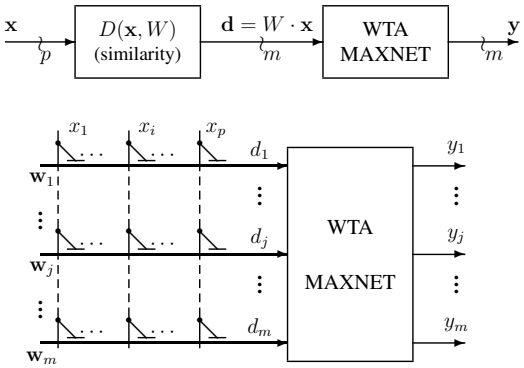


Figure 6–3: The structure of the similarity-measure layer for normalised vectors

6.4.2 The Competitive Layer

- The competitive layer, also known as the "Winner-Takes-All" (WTA) network, identifies the neuron that have the **weight vector most similar to the current input vector**.
- For the **winning neuron**, say jth the competitive layer generates binary output signals, $y_j = 1$

$$y_j = \begin{cases} 1 & \text{if } \; d_j \text{ is the largest of all } d_k \\ 0 & \text{otherwise} \end{cases}$$

that is, the jth weight vector $\mathbf{w}_{j:}$ is the **most similar** to the current afferent vector $\mathbf{x}$.

- 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.

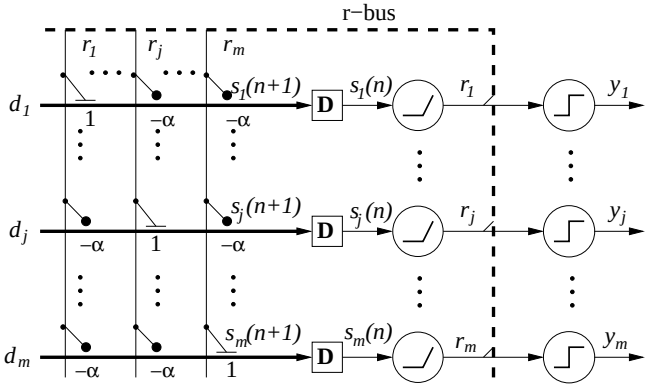
- For example, for $\alpha = 0.1$

$$M = \begin{bmatrix} 1 & -0.1 & -0.1 & -0.1 \\ -0.1 & 1 & -0.1 & -0.1 \\ -0.1 & -0.1 & 1 & -0.1 \\ -0.1 & -0.1 & -0.1 & 1 \end{bmatrix}$$

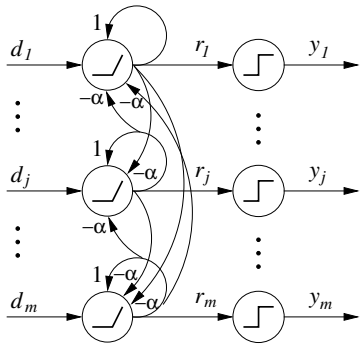
- Such a matrix M describes a network with a local unity feedback, and a feedback to other neurons with the connection strength $-\alpha$.

- The structure of the competitive layer (dendritic and the signal-flow views):

Dendritic view:



Signal-flow view:



- In the dendritic view note that each signal r_j is fed back (through the "r-bus") and connected to an excitatory synapse belonging to the same neuron, and laterally to all inhibitive synapses from other neurons.
- In the signal-flow view we can equivalently observe the **unity self-excitatory** connections, and the **lateral inhibitory** connections.
- Signals $s_j(n+1)$ are delayed by one unit and form respective $s_j(n)$ signals.
- Signals r_j are formed from s_j by removing their negative part before they are fed back to synapses.
- Finally signals r_j are converted into binary signals y_j.

Mathematically, we can evaluate the signals in the following way:

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

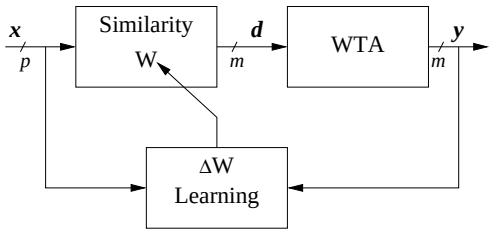
At each step n,

- signals $s_j(n+1)$ consist of the self-excitatory contribution, $r_j(n)$, and
- a total lateral inhibitory contribution equal to: $\alpha \sum_{k \neq j} r_k$.

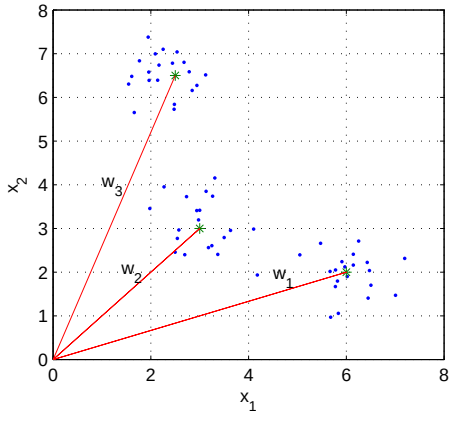
After a certain number of iterations all r_k signals but the one associated with the largest input signal, d_j, are zeros. For example

```
          n =  6 ,  kwin = 3
rHist =
      7.3    0.3200         0         0         0         0
      4.2         0         0         0         0         0
      9.6    3.0800    2.6400    2.4000    2.2656    2.2272
      0.7         0         0         0         0         0
      5.5         0         0         0         0         0
      2.9         0         0         0         0         0
      8.6    1.8800    1.2000    0.6720    0.1920         0
      3.4         0         0         0         0         0
```


6.4.3 Simple Competitive Learning



- The block-diagram of competitive learning re-visited:
- The learning (or encoding) part is responsible for self-organization of data.
- The weight update is a function of input and output signals.
- **Hebbian learning** considered in sec. 6.3 has organised the synaptic weights in such a way that they approximate the shape of the data by their principal directions (components).
- The **competitive learning** aims at approximation of data organised in **clusters**.
- If we assume that the input data is organised in, possibly overlapping, **clusters**, then each synaptic vector, $\mathbf{w}_j$, should converge to a **centroid** of a cluster of the input data
- Example of approximation of three 2-D clusters of data with weights:
- In other words, the input vectors are categorized into m classes (clusters), each weight vector representing the center of a cluster.



- A **simple competitive learning** works in such a way that for each afferent signals (stimulus) presented to the network, its weight are pulled towards this stimulus.
- This should result in weight vectors being eventually located close to the centres of data (stimuli) clusters.

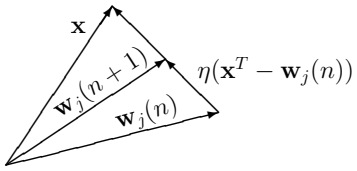
More formally, simple competitive learning can be describe as follow:

- Weight vectors are usually initialise with m randomly selected input vectors (stimuli):

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

- 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:

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

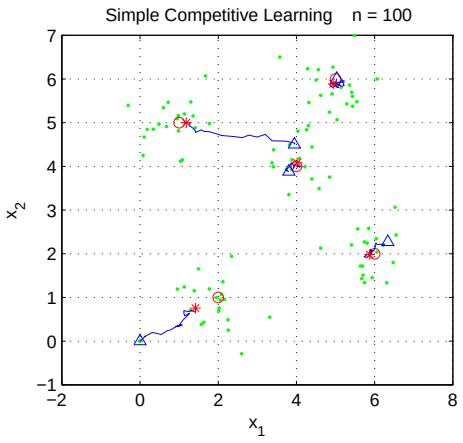


- In order to arrive at a stable solution, the learning rate is gradually linearly reduced, for example

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

6.4.4 Example of simple competitive learning

- Consider 2-D data organized in $m = 5$ clusters.
- We need $m = 5$ neurons each with $p = 2$ synapses.
- Number of synapses p matches the dimensionality of afferent signals (stimuli)
- The weight matrix W is 5×2, that is, it stores $m = 5$ 2-D vectors, one per neuron.



- Each cluster has been generated as a Gaussian "blob" of 20 points.
- The stimuli (or data points) are marked with green dots.
  Centres of clusters are marked with read circles '○',
  Initial weights are marked with the blue triangles '△'
  Final weights are marked with stars '*'
- Note that after **one training epoch**, the weights are relatively close to the centroids of the clusters.
- This is an indication that synaptical weights approximate the input data in a satisfactory way.