T-61.3030 Principles of Neural Computing

Raivio, Koskela, Pöllä

Exercise 1

1. An odd sigmoid function is defined by

$$\varphi(v) = \frac{1 - \exp(-av)}{1 + \exp(-av)} = \tanh(av/2),$$

where tanh denotes a hyperbolic tangent. The limiting values of this second sigmoid function are -1 and $+1$. Show that the derivate of $\varphi(v)$ with respect to v is given by

$$\frac{d\varphi}{dv} = \frac{a}{2}(1 - \varphi^2(v)).$$

What is the value of this derivate at the origin? Suppose that the slope parameter a is made infinitely large. What is the resulting form of $\varphi(v)$?

2. (a) Show that the McCulloch-Pitts formal model of a neuron may be approximated by a sigmoidal neuron (i.e., neuron using a sigmoid activation function with large synaptic weights).

   (b) Show that a linear neuron may be approximated by a sigmoidal neuron with small synaptic weights.

3. Construct a fully recurrent network with 5 neurons, but with no self-feedback.

4. Consider a multilayer feedforward network, all the neurons of which operate in their linear regions. Justify the statement that such a network is equivalent to a single-layer feedforward network.

5. (a) Figure 1(a) shows the signal-flow graph of a recurrent network made up of two neurons. Write the nonlinear difference equation that defines the evolution of $x_1(n)$ or that of $x_2(n)$. These two variables define the outputs of the top and bottom neurons, respectively. What is the order of this equation?

   (b) Figure 1(b) shows the signal-flow graph of a recurrent network consisting of two neurons with self-feedback. Write the coupled system of two first-order nonlinear difference equations that describe the operation of the system.

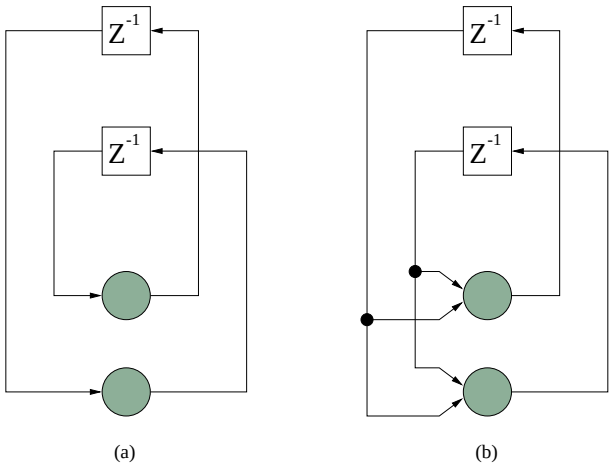


Figure 1: The signal-flow graphs of the two recurrent networks.

T-61.3030 Principles of Neural Computing

Answers to Exercise 1

1. An odd sigmoid function is defined as

$$\varphi(v) = \frac{1 - \exp(-av)}{1 + \exp(-av)} = \tanh(av/2). \tag{1}$$

The derivative:

$$\begin{aligned} \frac{d\varphi}{dv} &= \frac{a\exp(-av)(1+\exp(-av)) + a\exp(-av)(1-\exp(-av))}{(1+\exp(-av))^2} \\ &= \frac{a(\exp(-av) + \exp(-2av) + \exp(-av) - \exp(-2av))}{(1+\exp(-av))^2} \\ &= \frac{a}{2}\frac{(1+\exp(-av))^2 - (1-\exp(-av))^2}{(1+\exp(-av))^2} = \frac{a}{2}(1-\varphi^2(v)) \end{aligned} \tag{2}$$

At the origin:

$$\left.\frac{d\varphi}{dv}\right|_{v=0} = \frac{a}{2}(1 - \left(\frac{1-\exp(-a*0)}{1+\exp(-a*0)}\right)^2 = \frac{a}{2}. \tag{3}$$

$$\text{When } a \to \infty,\ \varphi(v) \to \begin{cases} +1, & v > 0 \\ -1, & v < 0 \\ 0, & v = 0 \end{cases} \tag{4}$$

Fig. 2 shows the values of the sigmoid function (1) with different values of a.

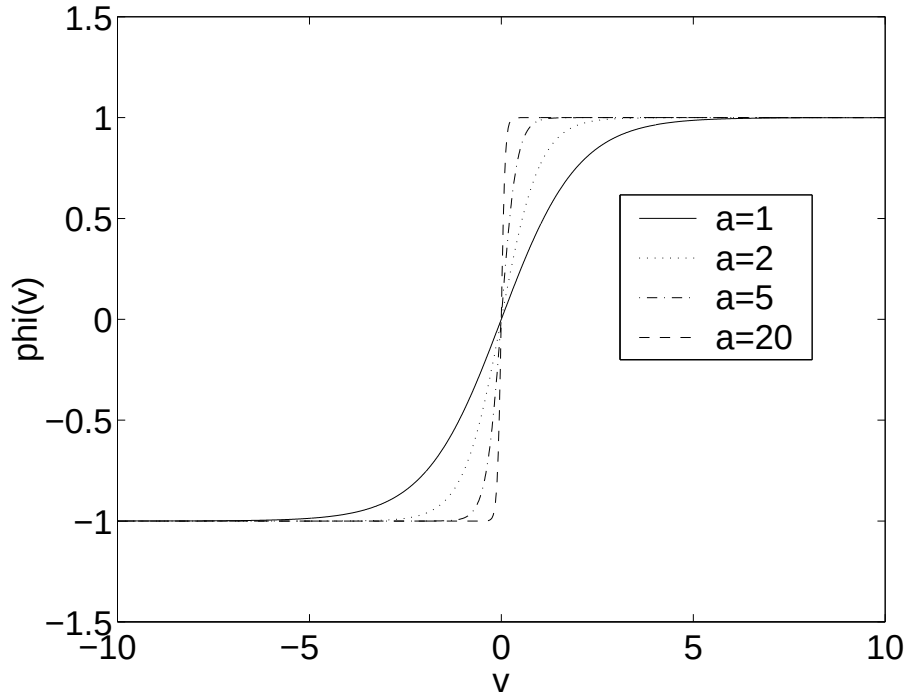


Figure 2: Sigmoid function $\varphi(v)$ with different values of a. The function behaves linearly near the origin.

2. (a) The McCulloch-Pitts formal of a neuron is defined as a threshold function:

$$\varphi(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0 \end{cases}, \tag{5}$$

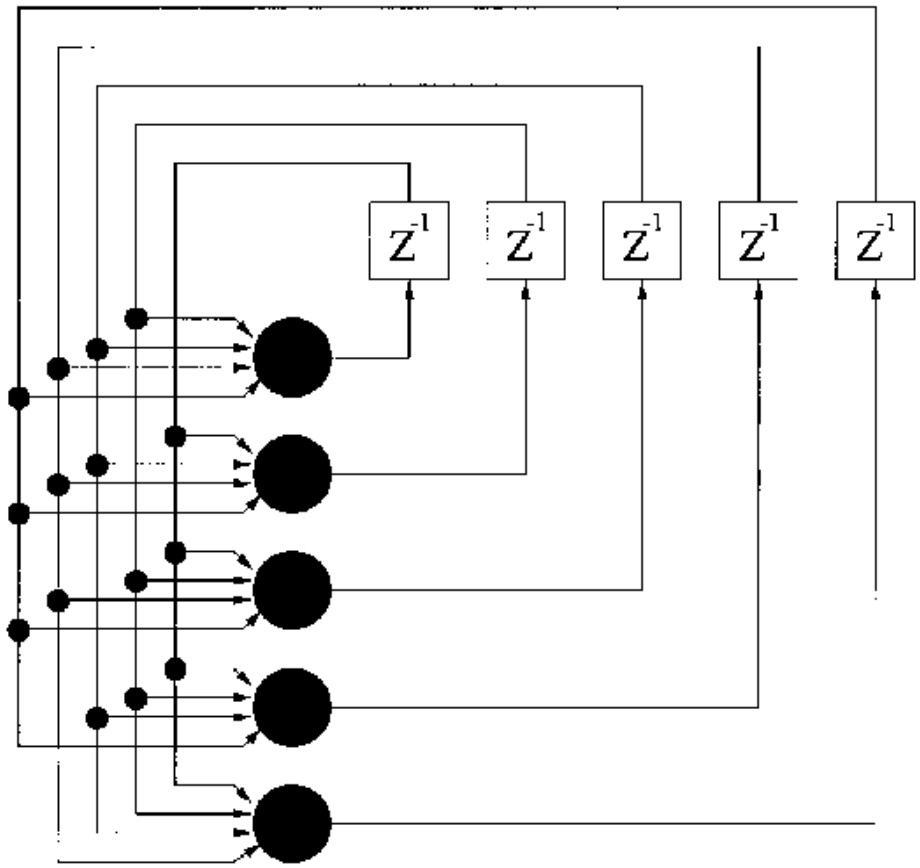
where $v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k$, where w_{kj}'s are the synaptic weights and b_k is the bias.
A sigmoid activation function on interval $[0, 1]$ is *e.g.* $\sigma(v) = 1/(1 + \exp(-av))$, where we assume $a > 0$ without loss of generality.
When synaptic weights have large values, also $|v|$ tends to have a large value:

$$\begin{aligned} \lim_{v \to \infty} \sigma(v) &= \left. \frac{1}{1 + \exp(-av)} \right|_{v \to \infty} = \frac{1}{1+0} = 1 \\ \lim_{v \to -\infty} \sigma(v) &= \left. \frac{1}{1 + \exp(-av)} \right|_{v \to -\infty} = \frac{1}{1-\infty} = 0 \end{aligned} \tag{6}$$

(b) We expand the $\exp(-av)$ in Taylor series around point $v = 0$:

$$\begin{aligned} \sigma(v) = \frac{1}{1+\exp(-av)} &= \frac{1}{1 + 1 - av + \underbrace{\frac{(av)^2}{2!} - \frac{(av)^3}{3!} + \cdots}_{\approx 0, \text{ for small values of } v}} \\ &\approx \frac{1}{2(1 - \frac{av}{2})} \\ &= \frac{1}{2} \frac{1 + \frac{av}{2}}{1 + \underbrace{\frac{(av)^2}{4}}_{\approx 0}} \approx \frac{1}{2}\left(1 + \frac{av}{2}\right) = L(v) \, \square \end{aligned} \tag{7}$$

3. The fully recurrent network of 5 neurons.



4. A single neuron is depicted in Fig. 3

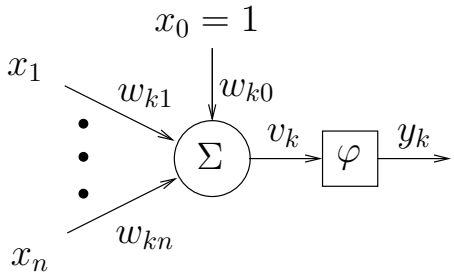


Figure 3: A single sigmoidal neuron.

There the input v_k to the nonlinearity φ is defined as $v_k = \sum_{j=0}^{n} w_{kj} x_j = \mathbf{w}_k^T \mathbf{x}$,
where $\mathbf{x} = \begin{pmatrix} 1 & x_1 & \cdots & x_n \end{pmatrix}^T$ and $\mathbf{w}_k = \begin{pmatrix} w_{k0} & \cdots & w_{kn} \end{pmatrix}^T$.
When the neuron operates in its linear region, $\varphi(v) \approx \alpha v$ and $y_k \approx \alpha \mathbf{w}_k^T \mathbf{x}$. For the whole layer, this gives:

$$\mathbf{y} = \begin{pmatrix} y_1 \\ \cdots \\ y_m \end{pmatrix} \approx \alpha \begin{pmatrix} \mathbf{w}_1^T \mathbf{x} \\ \cdots \\ \mathbf{w}_m^T \mathbf{x} \end{pmatrix} = \alpha \begin{pmatrix} \mathbf{w}_1^T \\ \cdots \\ \mathbf{w}_m^T \end{pmatrix} \mathbf{x} = \boldsymbol{W} \mathbf{x}, \tag{8}$$

where

$$\boldsymbol{W} = \alpha \begin{pmatrix} w_{10} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m0} & \cdots & w_{mn} \end{pmatrix} \tag{9}$$

The whole network is constructed from these single layers:

$$\mathbf{x} \longrightarrow \boxed{\boldsymbol{W_1}} \longrightarrow \mathbf{u}_1 \longrightarrow \boxed{\boldsymbol{W_2}} \longrightarrow \mathbf{u}_2 \longrightarrow \cdots \longrightarrow \boxed{\boldsymbol{W_N}} \longrightarrow \mathbf{y}$$

and the output of the whole network is

$$\mathbf{y} = \boldsymbol{W_N}\mathbf{u_{N-1}} = \boldsymbol{W_N}\boldsymbol{W_{N-1}}\mathbf{u_{N-2}} = \prod_{i=1}^{N} \boldsymbol{W_i}\mathbf{x}. \tag{10}$$

The product $\boldsymbol{T} = \prod_{i=1}^{N} \boldsymbol{W_i}$ is a matrix of size $m \times n$:

$$\mathbf{y} = \boldsymbol{T}\mathbf{x} = \begin{pmatrix} t_{10} & \cdots & t_{1n} \\ \vdots & \ddots & \vdots \\ t_{m0} & \cdots & t_{mn} \end{pmatrix} \mathbf{x} = \begin{pmatrix} \mathbf{t}_1^T \\ \cdots \\ \mathbf{t}_m^T \end{pmatrix} \mathbf{x}, \tag{11}$$

which is exactly the output of a single layer linear network having $\boldsymbol{T}$ as the weights □

5. (a) Fig. 4 shows the first signal-flow graph of a recurrent network made up of two neurons.

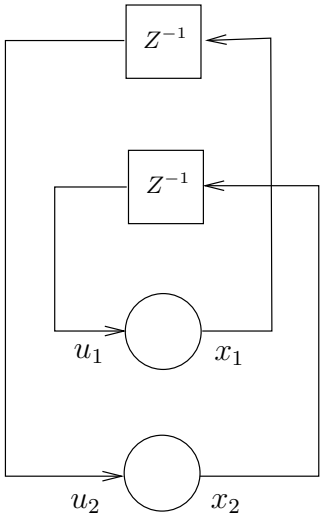


Figure 4: The signal-flow graphs of the first recurrent network.

From the figure it is evident that

$$\begin{cases} u_1(n) = x_2(n-1) \\ u_2(n) = x_1(n-1) \end{cases} \tag{12}$$

On the other hand:

$$\begin{cases} x_1(n) = \varphi(w_1 u_1(n)) \\ x_2(n) = \varphi(w_2 u_2(n)) \end{cases} \tag{13}$$

By eliminating the u_i's, we get

$$\begin{cases} x_1(n) = \varphi(w_1 x_2(n-1)) = \varphi(w_1 \varphi(w_2 x_1(n-2))) \\ x_2(n) = \varphi(w_2 x_1(n-1)) = \varphi(w_2 \varphi(w_1 x_2(n-2))) \end{cases} \tag{14}$$

These equations are 2nd order difference equations.

(b) Fig. 5 shows the second signal-flow graph of a recurrent network made up of two neurons.

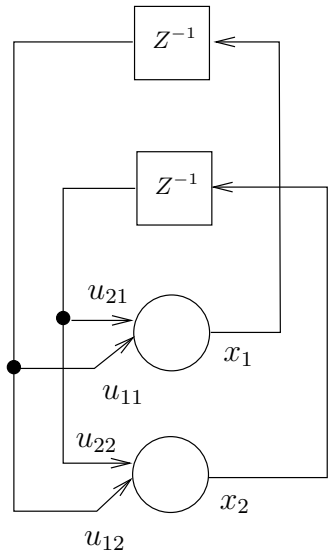


Figure 5: The signal-flow graphs of the second recurrent network.

Again from the figure:

$$\begin{cases} u_{11}(n) = x_1(n-1) \\ u_{21}(n) = x_2(n-1) \\ u_{12}(n) = x_1(n-1) \\ u_{22}(n) = x_2(n-1) \end{cases} \tag{15}$$

and

$$\begin{cases} x_1(n) = \varphi(w_{11} u_{11}(n) + w_{21} u_{21}(n)) \\ x_2(n) = \varphi(w_{12} u_{12}(n) + w_{22} u_{22}(n)) \end{cases} \tag{16}$$

Giving two coupled first order equations:

$$\begin{cases} x_1(n) = \varphi(w_{11} x_1(n-1) + w_{21} x_2(n-1)) \\ x_2(n) = \varphi(w_{12} x_1(n-1) + w_{22} x_2(n-1)) \end{cases} \tag{17}$$

T-61.3030 Principles of Neural Computing

Raivio, Koskela, Pöllä

Exercise 2

1. The error-correction learning rule may be implemented by using inhibition to subtract the desired response (target value) from the output, and then applying the anti-Hebbian rule. Discuss this interpretation of error-correction learning.

2. Figure 6 shows a two-dimensional set of data points. Part of the data points belongs to class C_1 and the other part belongs to class C_2. Construct the decision boundary produced by the nearest neighbor rule applied to this data sample.

3. A generalized form of Hebb's rule is described by the relation

   $$\Delta w_{kj}(n) = \alpha F(y_k(n))G(x_j(n)) - \beta w_{kj}(n)F(y_k(n))$$

   where $x_j(n)$ and $y_k(n)$ are the presynaptic and postsynaptic signals, respectively; $F(\cdot)$ and $G(\cdot)$ are functions of their respective arguments; and $\Delta w_{kj}(n)$ is the change produced in the synaptic weight w_{kj} at time n in response to the signals $x_j(n)$ and $y_j(n)$. Find the balance point and the maximum depression that are defined by this rule.

4. An input signal of unit amplitude is applied repeatedly to a synaptic connection whose initial value is also unity. Calculate the variation in the synaptic weight with time using the following rules:

   (a) The simple form of Hebb's rule described by

   $$\Delta w_{kj}(n) = \eta y_k(n)x_j(n)$$

   assuming the learning rate $\eta = 0.1$.

   (b) The covariance rule described by

   $$\Delta w_{kj} = \eta(x_j - \overline{x})(y_k - \overline{y})$$

   assuming that the time-averaged values of the presynaptic signal and postsynaptic signal are $\overline{x} = 0$ and $\overline{y} = 1.0$, respectively.

5. Formulate the expression for the output y_j of neuron j in the network of Figure 7. You may use the following notations:

   $x_i = i$th input signal
   w_{ji} = synaptic weight from input i to neuron j
   c_{kj} = weight of lateral connection from neuron k to neuron j
   v_j = induced local field of neuron j
   $y_j = \varphi(v_j)$

   What is the condition that would have to be satisfied for neuron j to be the winning neuron?

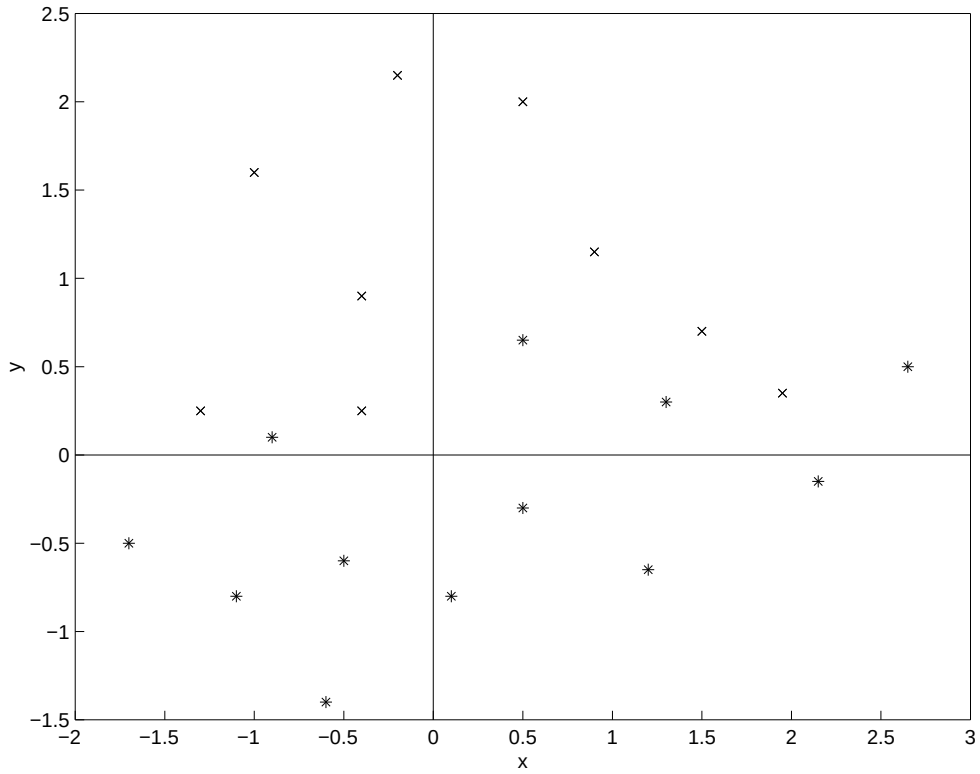


Figure 6: Data point belonging to class C_1 and C_2 are plotted with 'x' and '*', respectively.

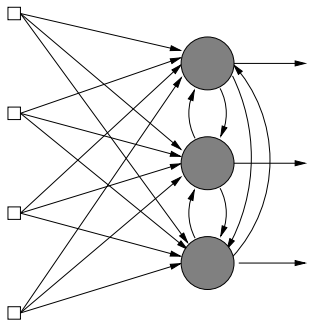


Figure 7: Simple competitive learning network with feedforward connections from the source nodes to the neurons, and lateral connections among the neurons.

T-61.3030 Principles of Neural Computing

Answers to Exercise 2

1. In the error correction learning, a desired output $d_k(n)$ is learned to mimick by the outputs $y_k(n)$ of the learning system. This is achieved by minimizing a cost function or index of performance, $E(n)$, which is defined in terms of the error signal

$$E(n) = \frac{1}{2}e_k^2(n), \tag{18}$$

where $e_k(n) = d_k(n) - y_k(n)$ is the difference between the desired output and the kth neuron output to input signal $\mathbf{x}(n)$.

$E(n)$ can be minimized using a gradient descent method:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n), \tag{19}$$

where $\Delta\mathbf{w} = -\eta\nabla E$. In ∇E the partial differentials are $\partial E/\partial w_{kj} = 2 \cdot \frac{1}{2}e_k e_k' = -e_k y_k' = -e_k x_j$, because $y_k = \sum_{j=1}^{N} w_{kj}x_j$.

Thus (19) gives for the particular w_{kj} an update:

$$\Delta w_{kj} = \eta e_k x_j = \eta(d_k - y_k)x_j \tag{20}$$

On the other hand, the Anti-Hebbian learning rule weakens positively correlated presynaptic and postsynaptic connections, and strengthens the negatively correlated signals:

$$\Delta w_{kj} = -\eta y_k x_j \tag{21}$$

The update rule (20) of error-correction learning can be interpreted as Anti-Hebbian learning, when the output of the Anti-Hebbian system is set to be the error $-e_k$:

$$\Delta w_{kj} = -\eta(\text{output})(\text{input}) = -\eta(y_k - d_k)x_j \tag{22}$$

2. Let $\mathbf{x} = (x_1, x_2)$ and $\mathbf{y} = (y_1, y_2)$ denote two points belonging to different classes. Then the decision boundary between the points is the line of points $\mathbf{z} = (z_1, z_2)$, to which the Euclidean distance from both these points is equal:

$$\begin{aligned} d(\mathbf{z},\mathbf{x}) &= d(\mathbf{z},\mathbf{y}) \\ (z_1 - x_1)^2 + (z_2 - x_2)^2 &= (z_1 - y_1)^2 + (z_2 - y_2)^2 \\ z_1^2 + z_2^2 + x_1^2 + x_2^2 - 2z_1x_1 - 2z_2x_2 &= z_1^2 + z_2^2 + y_1^2 + y_2^2 - 2z_1y_1 - 2z_2y_2 \\ \underbrace{2(y_2 - x_2)}_{\alpha} z_2 + \underbrace{2(y_1 - x_1)}_{\beta} z_1 + \underbrace{x_1^2 + x_2^2 - y_1^2 - y_2^2}_{\gamma} &= 0 \\ \alpha z_2 + \beta z_1 + \gamma = L &= 0, \end{aligned} \tag{23}$$

which is a line in z_1z_2 space.

The gradient ∇L is parallel to the line connecting the points $\mathbf{x}$ and $\mathbf{y}$ *i.e.* $\nabla L || \mathbf{y} - \mathbf{x}$. On the other hand, on the line there is a point $\mathbf{z}'$ lying exactly in the middle of $\mathbf{y} - \mathbf{x}$ that is $\alpha z_2' + \beta z_1' + \gamma = 0$, when $\mathbf{z}' = \left(\frac{x_1+y_1}{2}, \frac{x_2+y_2}{2}\right)$,

which is verified by inserting them in (23):

$$\begin{aligned} 2(y_2 - x_2)\frac{1}{2}(y_2 + x_2) + 2(y_1 - x_1)\frac{1}{2}(y_1 + x_1) + \gamma \\ \underbrace{y_2^2 - x_2^2 + y_1^2 - x_1^2}_{-\gamma} + \gamma = 0 \end{aligned} \tag{24}$$

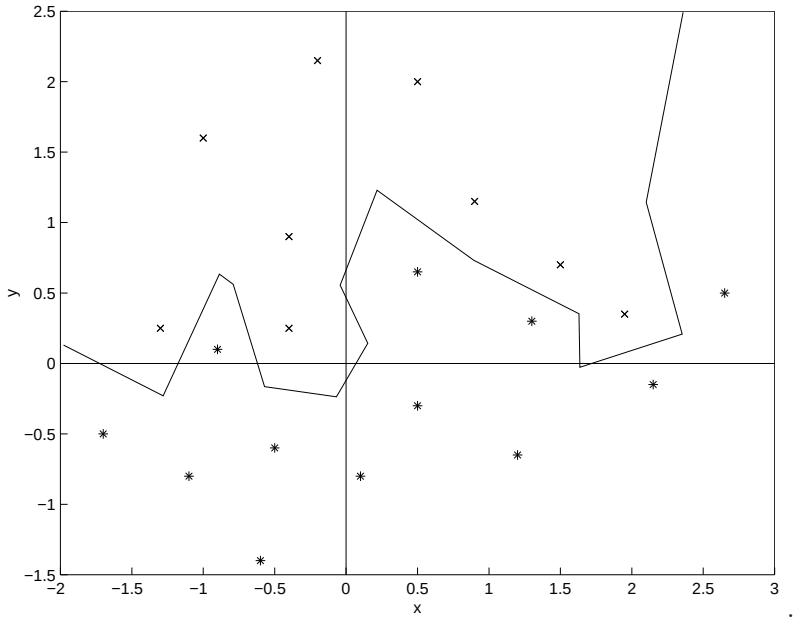


Figure 8: Data oint belonging to class C_1 and C_2 are plotted with 'x' and '*', respectively.

3. A generalized form of Hebb's rule is described by the relation

$$\Delta w_{kj}(n) = \alpha F(y_k(n))G(x_j(n)) - \beta w_{kj}(n)F(y_k(n)) \tag{25}$$
$$= F(y_k(n))[\alpha G(x_j(n)) - \beta w_k j(n)] \tag{26}$$

(a) At the balance point the weights do not change:

$$w_{kj}(n+1) = w_{kj}(n), \tag{27}$$

meaning that the difference $\Delta w_{kj} = w_{kj}(n+1) - w_{kj}(n) = 0$. Using (26), and neglecting the trivial solution $F(y_k(n)) = 0$, we get $\alpha G(x_j(n)) - \beta w_k j(n) = 0$ and the balance point

$$w_{kj}(\infty) = \frac{\alpha}{\beta} G(x_j(n)) \tag{28}$$

(b) When calculating the maximum depression, we assume that $\alpha, \beta, F(\cdot), G(\cdot)$ and $w \geq 0$. Then the maximum depression is achieved, when $\alpha = 0$ and

$$\Delta w_{kj}(n) = -\beta w_{kj}(n)F(y_k(n)) \tag{29}$$

4. $x(n) = 1$, for all $n > 0$. As well the initial weight $w(1) = 1$.

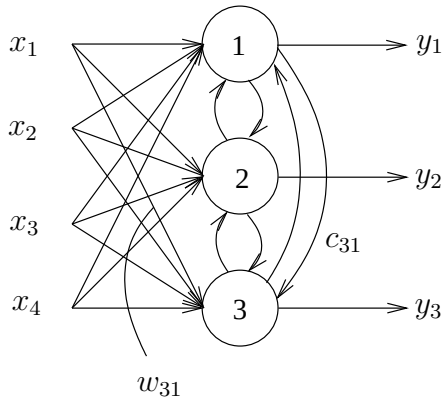
   (a) The update follows the simple Hebb rule: $\Delta w_{kj}(n) = \eta y_k(n) x_j(n)$, with learning rate $\eta = 0.1$:

$$\begin{aligned} w_{kj}(1) &= 1 \\ y_k(1) &= w_{kj}(1)x_j(1) = 1 \times 1 = 1 \\ \Delta w_{kj}(1) &= 0.1 \times 1 = 0.1 \\ w_{kj}(2) &= w_{kj}(1) + \Delta w_{kj}(1) = 1 + 0.1 = 1.1 \\ y_k(2) &= 1.1 \times 1 = 1.1 \\ \Delta w_{kj}(2) &= 0.11 \\ &\vdots \\ w_{kj}(n) &= (1+\eta)^{n-1} \\ &\vdots \\ w_{kj}(\infty) &= \infty \end{aligned} \tag{30}$$

   (b) The covariance rule $\Delta w_{kj}(n) = \eta(x_j - \bar{\mathbf{x}})(y_k - \bar{\mathbf{y}})$, where we assume the averages $\bar{\mathbf{x}} = 0$ and $\bar{\mathbf{y}} = 1$. Note that we do not update the averages during learning!
   Introducing the average values gives: $\Delta w_{kj}(n) = 0.1(w_{kj}(n) - 1)$. Iterating:

$$\begin{aligned} w_{kj}(1) &= 1 \\ \Delta w_{kj}(1) &= 0 \\ w_{kj}(2) &= 1 \\ \Delta w_{kj}(2) &= 0 \\ &\vdots \\ w_{kj}(n) &= 1, \text{ for all } n \end{aligned} \tag{31}$$

5. In the figure, w_{ji} means a forward connection from the source node i to the neuron j, and c_{jk} is the lateral feedback connection from the neuron k to neuron j.



Output of the neuron j is

$$\begin{aligned} &y_j = \varphi(v_j), \mathrm{where} \\ &v_j = \sum_{i=1}^{4} w_{ji}x_i + \sum_{k=1}^{3} c_{jk}y_k,\ j = 1, \cdots 3. \end{aligned} \tag{32}$$

Since there are no self-feedbacks, $c_{jj} = 0, j = 1, \cdots 3$ *i.e*

$$v_j = \sum_{i=1}^{4} w_{ji}x_i + \sum_{k=1, k\neq j}^{3} c_{jk}y_k,\ j = 1, \cdots 3. \tag{33}$$

The winning neuron is defined to be the neuron having the largest output. Thus, the neuron j is a winning neuron iff $y_j > y_k$, for all $k \neq j$.

T-61.3030 Principles of Neural Computing

Raivio, Koskela, Pöllä

Exercise 3

1. To which of the two paradigms, learning with a teacher and learning without a teacher, do the following algorithms belong? Justify your answers.

   (a) nearest neighbor rule

   (b) k-nearest neighbor rule

   (c) Hebbian learning

   (d) error-correction learning

2. Consider the difficulties that a learning machine faces in assigning credit for the outcome (win, loss, or draw) of a game of chess. Discuss the notations of temporal credit assignment and structural credit assignment in the context of this game.

3. A supervised learning task may be viewed as a reinforcement learning task by using as the reinforcement signal some measure of the closeness of the actual response of the system to the desired response. Discuss this relationship between supervised learning and reinforcement learning.

4. Heteroassociative memory $\mathbf{M}$, a matrix of size $c \times d$, is a solution to the following group of equation systems:

   $$\mathbf{M}\mathbf{x}_j = \mathbf{y}_j, \quad j = 1, \ldots, N,$$

   where $\mathbf{x}_j$ is the jth input vector of size $d \times 1$ and $\mathbf{y}_j$ is the corresponding desired output vector of size $c \times 1$. The ith equation of the jth equation system can be written as follows:

   $$\mathbf{m}_i^T \mathbf{x}_j = y_{ij},$$

   where $\mathbf{m}_i^T = [m_{i1}, m_{i2}, \ldots, m_{id}]$. Derive a gradient method which minimizes the following sum of squared errors:

   $$\sum_{j=1}^{N} (\mathbf{m}_i^T \mathbf{x}_j - y_{ij})^2.$$

   How it is related to the LMS-algorithm (Widrow-Hoff rule)?

5. Show that $\mathbf{M} = \mathbf{Y}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is a solution to the following group of equation systems:

   $$\mathbf{M}\mathbf{x}_j = \mathbf{y}_j, \quad j = 1, \ldots, N.$$

   Vectors $\mathbf{x}_j$ and $\mathbf{y}_j$ are the jth columns of matrixes $\mathbf{X}$ and $\mathbf{Y}$, respectively.

T-61.3030 Principles of Neural Computing

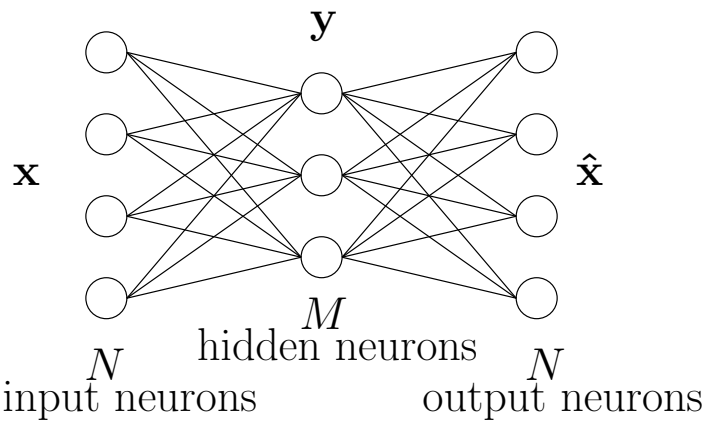
Answers to Exercise 3

1. Learning with a teacher (supervised learning) requires the teaching data (the prototypes) to be labeled somehow. In unsupervised learning (learning without a teacher) labeling is not needed, but the system tries to model the data the best it can.

(a) and (b) Nearest neighbour and k-nearest neighbour learning rules belong clearly to the "learning with a teacher" paradigm as a teacher is required to label the prototypes.

(c) Error-correction learning is usually used in a supervised manner, *i.e.* it usually belongs to the "learning with a teacher" paradigm: typically the desired response is provided with the prototype inputs.

It is also possible to use the error-correction learning in an unsupervised manner, without a teacher using the input as the desired output of the system. An example of this kind is the bottle-neck coding (less hidden neurons than input / output neurons, $M < n$) of the input vectors (see the figure below).



(d) Hebbian learning belongs to the "learning without a teacher" as no desired response is required for updating the weights.

2. The outcome of the game depends on the moves selected; the moves themselves typically depend on a multitude of internal decisions made by the learning system.

   The temporal credit assignment problem is to assign credit to moves based on how they changed the expected outcome of the game (win, draw, loose).

   On the other hand, the structural credit assignment concerns the credit assigment of internal decisions: which internal decision rules / processes were responsible for the good moves and which of the worse moves. The credit assignment to moves is thereby converted into credit assignments to internal decisions.

3. Let $f(n)$ denote a reinforcement signal providing a measure of closeness of the actual response $y(n)$ of a ninlinear system to the desired response $d(n)$. For example, $f(n)$ might have three possible values:

   (a) $f(n) = 0$, if $y(n) = d(n)$,

   (b) $f(n) = a$, if $|y(n) - d(n)|$ is small,

   (c) $f(n) = b$, if $|y(n) - d(n)|$ is large.

   This discards some of the information making the learning task actually harder. A longer learning time is thus expected, when supervised learning is done using reinforcement learning (RL).

The reason for studying renforcement learning algorithms is not that they offer the promise of fastest learning in supervised learning tasks, but rather it is because there are certain tasks that are naturally viewed as RL tasks, but not as supervised learning tasks. It may also happen that a teacher is not available to provide a desired response.

4. The error function to be minimised is

$$E(\mathbf{m_i}) = \frac{1}{2}\sum_{j=1}^{N}\left(\mathbf{m_i}^T\mathbf{x_j} - y_{ij}\right)^2. \tag{34}$$

The partial differentials:

$$\begin{aligned}\frac{\partial E(\mathbf{m_i})}{\partial m_{ik}} &= \frac{1}{2}\sum_{j=1}^{N} 2\left(\mathbf{m_i}^T\mathbf{x_j} - y_{ij}\right)\underbrace{\frac{\partial}{\partial m_{ik}}\sum_{l=1}^{d} m_{il}x_{jl}}_{\neq 0,\text{ if } l=k} \\ &= \sum_{j=1}^{N}\left(\mathbf{m_i}^T\mathbf{x_j} - y_{ij}\right)x_{jk}.\end{aligned} \tag{35}$$

$$\nabla E(\mathbf{m_i}) = \begin{pmatrix} \vdots \\ \frac{\partial E(\mathbf{m_i})}{\partial m_{ik}} \\ \vdots \end{pmatrix}. \tag{36}$$

A gradient method minimising (34) is

$$\mathbf{m_i}^{new} = \mathbf{m_i}^{old} - \alpha\nabla E(\mathbf{m_i}^{old}) = \mathbf{m_i}^{old} - \alpha\sum_{j=1}^{N}\left(\mathbf{m_i}^{old^T}\mathbf{x_j} - y_{ij}\right)x_j. \tag{37}$$

The Widrow-Hoff learning rule also minimises (34) in the following way:

$$\mathbf{m_i}^{new} = \mathbf{m_i}^{old} - \alpha\left(\mathbf{m_i}^{old^T}\mathbf{x_j} - y_{ij}\right)x_j. \tag{38}$$

The Widrow-Hoff learning rule (38) can be considered to be an on-line version of the Heteroassociative memory rule (37): the weights are updated after each single output pair. The Heteroassociative rule may propvide faster convergence, but the Widrow-Hoff rule is simplerand the learning system can adapt to some extent to slow changes (non-stationarities) in the data.

5. A group of equation systems $\boldsymbol{M}\mathbf{x}_j = \mathbf{y}_j$, $j = 1, \ldots, N$ can be written in a matrix form in the following way:

$$\begin{pmatrix}\boldsymbol{M}\mathbf{x}_1 & \boldsymbol{M}\mathbf{x}_2 & \ldots & \boldsymbol{M}\mathbf{x}_N\end{pmatrix} = \begin{pmatrix}\mathbf{y}_1 & \mathbf{y}_2 & \ldots & \mathbf{y}_N\end{pmatrix} \tag{39}$$

$$\boldsymbol{M}\boldsymbol{X} = \boldsymbol{Y} \tag{40}$$

Introducing $\boldsymbol{M} = \boldsymbol{Y}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T$ to (39), gives

$$\boldsymbol{M}\boldsymbol{X} = \boldsymbol{Y}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{Y}_{\square} \tag{41}$$

Matrix $\widehat{\boldsymbol{M}} = \boldsymbol{Y}\boldsymbol{X}^{+}$, where $\boldsymbol{X}^{+}$ is the pseudoinverse of $\boldsymbol{X}$, minimises the error of linear associator $\boldsymbol{M}$, $E(\boldsymbol{M}) = \frac{1}{2}\sum_{j=1}^{N} ||\boldsymbol{M}\mathbf{x}_j - \mathbf{y}_j||^2$.

Every matrix has a pseudoinverse and if the columns of the matrix are linearly independent (uncorrelated) it is exactly $\boldsymbol{X}^{+} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T$. As it was shown above, the association is perfect, when $\boldsymbol{M} = \boldsymbol{Y}\boldsymbol{X}^{+}$.

T-61.3030 Principles of Neural Computing

Raivio, Koskela, Pöllä

Exercise 4

1. Let the error function be

   $$\mathcal{E}(\mathbf{w}) = w_1^2 + 10w_2^2,$$

   where w_1 and w_2 are the components of the two-dimensional parameter vector $\mathbf{w}$. Find the minimum value of $\mathcal{E}(\mathbf{w})$ by applying the steepest descent method. Use $\mathbf{w}(0) = [1, 1]^T$ as an initial value for the parameter vector and the following constant values for the learning rate:

   (a) $\alpha = 0.04$

   (b) $\alpha = 0.1$

   (c) $\alpha = 0.2$

   (d) What is the condition for the convergence of this method?

2. Show that the application of the Gauss-Newton method to the error function

   $$\mathcal{E}(\mathbf{w}) = \frac{1}{2}\left[\delta\|\mathbf{w} - \mathbf{w}(n)\|^2 + \sum_{i=1}^{n} e_i^2(\mathbf{w})\right]$$

   yields the the following update rule for the weights:

   $$\Delta\mathbf{w} = -\left[\mathbf{J}^T(\mathbf{w})\mathbf{J}(\mathbf{w}) + \delta\mathbf{I}\right]^{-1}\mathbf{J}^T(\mathbf{w})\mathbf{e}(\mathbf{w}).$$

   All quantities are evaluated at iteration step n. (Haykin 3.3)

3. The normalized LMS algorithm is described by the following recursion for the weight vector:

   $$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \frac{\eta e(n)\mathbf{x}(n)}{\|\mathbf{x}(n)\|^2},$$

   where η is a positive constant and $\|\mathbf{x}(n)\|$ is the Euclidean norm of the input vector $\mathbf{x}(n)$. The error signal $e(n)$ is defined by

   $$e(n) = d(n) - \hat{\mathbf{w}}(n)^T\mathbf{x}(n),$$

   where $d(n)$ is the desired response. For the normalized LMS algorithm to be convergent in the mean square, show that $0 < \eta < 2$. (Haykin 3.5)

4. The ensemble-averaged counterpart to the sum of error squares viewed as a cost function is the mean-square value of the error signal:

   $$J(\mathbf{w}) = \frac{1}{2}E[e^2(n)] = \frac{1}{2}E[(d(n) - \mathbf{x}^T(n)\mathbf{w})^2].$$

   (a) Assuming that the input vector $\mathbf{x}(n)$ and desired response $d(n)$ are drawn from a stationary environment, show that

   $$J(\mathbf{w}) = \frac{1}{2}\sigma_d^2 - \mathbf{r}_{\mathbf{x}d}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{R}_{\mathbf{x}}\mathbf{w},$$

   where $\sigma_d^2 = E[d^2(n)]$, $\mathbf{r}_{\mathbf{x}d} = E[\mathbf{x}(n)d(n)]$, and $\mathbf{R}_{\mathbf{x}} = E[\mathbf{x}(n)\mathbf{x}^T(n)]$.

(b) For this cost function, show that the gradient vector and Hessian matrix of $J(\mathbf{w})$ are as follows, respectively:

$$\mathbf{g} = -\mathbf{r}_{\mathbf{x}d} + \mathbf{R}_{\mathbf{x}}\mathbf{w} \text{ and}$$
$$\mathbf{H} = \mathbf{R}_{\mathbf{x}}.$$

(c) In the LMS/Newton algorithm, the gradient vector $\mathbf{g}$ is replaced by its instantaneous value. Show that this algorithm, incorporating a learning rate parameter η, is described by

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{R}_{\mathbf{x}}^{-1}\mathbf{x}(n)\left[d(n) - \mathbf{x}^T(n)\hat{\mathbf{w}}(n)\right].$$

The inverse of the correlation matrix $\mathbf{R}_{\mathbf{x}}$, assumed to be positive definite, is calculated ahead of time. (Haykin 3.8)

5. A linear classifier separates n-dimensional space into two classes using a $(n-1)$-dimensional hyperplane. Points are classified into two classes, ω_1 or ω_2, depending on which side of the hyperplane they are located.

(a) Construct a linear classifier which is able to separate the following two-dimensional samples correctly:

$$\omega_1 : \{[2,1]^T\},$$
$$\omega_2 : \{[0,1]^T, [-1,1]^T\}.$$

(b) Is it possible to construct a linear classifier which is able to separate the following samples correctly?

$$\omega_1 : \{[2,1]^T, [3,2]^T\},$$
$$\omega_2 : \{[3,1]^T, [2,2]^T\}$$

Justify your answer.

T-61.3030 Principles of Neural Computing

Answers to Exercise 4

1. The error function is

$$\mathcal{E}(\mathbf{w}) = w_1^2 + 10w_2^2,$$

where w_1 and w_2 are the components of the two-dimensional parameter vector $\mathbf{w}$.

The gradient is

$$\nabla_{\mathbf{w}}\mathcal{E}(\mathbf{w}) = \begin{pmatrix} \frac{\partial \mathcal{E}(\mathbf{w})}{\partial w_1} \\ \frac{\partial \mathcal{E}(\mathbf{w})}{\partial w_2} \end{pmatrix} = \begin{pmatrix} 2w_1 \\ 20w_2 \end{pmatrix}.$$

The steepest descent method has a learning rule:

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) - \alpha \nabla_{\mathbf{w}(n)}\mathcal{E}(\mathbf{w}(n)) \\ &= \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \alpha \begin{pmatrix} 2w_1 \\ 20w_2 \end{pmatrix} = \begin{pmatrix} (1-2\alpha)w_1 \\ (1-20\alpha)w_2 \end{pmatrix}. \end{aligned}$$

It was given $\mathbf{w}(0) = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$:

$$\mathbf{w}(n+1) = \begin{pmatrix} (1-2\alpha)^{n+1} w_1(0) \\ (1-20\alpha)^{n+1} w_2(0) \end{pmatrix} = \begin{pmatrix} (1-2\alpha)^{n+1} \\ (1-20\alpha)^{n+1} \end{pmatrix}.$$

(a) $\alpha = 0.04$:

$$\mathbf{w}(n+1) = \begin{pmatrix} 0.92^{n+1} \\ 0.20^{n+1} \end{pmatrix} \xrightarrow[n\to\infty]{} \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

(b) $\alpha = 0.1$:

$$\mathbf{w}(n+1) = \begin{pmatrix} 0.8^{n+1} \\ (-1)^{n+1} \end{pmatrix} \xrightarrow[n\to\infty]{} \begin{cases} \begin{pmatrix} 0 \\ -1 \end{pmatrix} & \text{,when } n \text{ is even} \\ \begin{pmatrix} 0 \\ 1 \end{pmatrix} & \text{,when } n \text{ is odd} \end{cases}$$

(c) $\alpha = 0.2$:

$$\mathbf{w}(n+1) = \begin{pmatrix} 0.6^{n+1} \\ (-3)^{n+1} \end{pmatrix} \xrightarrow[n\to\infty]{} \begin{cases} \begin{pmatrix} 0 \\ -\infty \end{pmatrix} & \text{,when } n \text{ is even} \\ \begin{pmatrix} 0 \\ \infty \end{pmatrix} & \text{,when } n \text{ is odd} \end{cases}$$

(d) Iteration converges if $|1-2\alpha| < 1$ and $|1-20\alpha| < 1 \Rightarrow 0 < \alpha < 0.1$. No oscillations occur if $0 < 1-2\alpha < 1$ and $0 < 1-20\alpha < 1 \Rightarrow 0 < \alpha < 0.05$

2.

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2}\left[\delta\|\mathbf{w} - \mathbf{w}(n)\|^2 + \underbrace{\sum_{i=1}^{n} e_i^2(\mathbf{w})}_{\mathbf{e}(\mathbf{w})^T\mathbf{e}(\mathbf{w})}\right]$$

The linear approximation of $\mathbf{e}(\mathbf{w})$ at point $\mathbf{w} = \mathbf{w}(n)$:

$$\hat{\mathbf{e}}(\mathbf{w}; \mathbf{w}(n)) = \mathbf{e}(\mathbf{w}(n)) + J(\mathbf{w}(n))[\mathbf{w} - \mathbf{w}(n)]$$

where

$$J(\mathbf{w}(n)) = \begin{pmatrix} \frac{\partial e_1(\mathbf{w}(n))}{\partial w_1} & \cdots & \frac{\partial e_1(\mathbf{w}(n))}{\partial w_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial e_n(\mathbf{w}(n))}{\partial w_1} & \cdots & \frac{\partial e_n(\mathbf{w}(n))}{\partial w_m} \end{pmatrix}$$

An approximation of $\mathcal{E}(\mathbf{w})$ at the point $\mathbf{w} = \mathbf{w}(n)$:

$$\begin{aligned} \hat{\mathcal{E}}(\mathbf{w}; \mathbf{w}(n)) = \frac{1}{2} \big[& \mathbf{e}^T(\mathbf{w}(n))\mathbf{e}(\mathbf{w}(n)) + 2\mathbf{e}^T(\mathbf{w}(n))J(\mathbf{w}(n))[\mathbf{w} - \mathbf{w}(n)] + \\ & + [\mathbf{w} - \mathbf{w}(n)]^T J^T(\mathbf{w}(n))J(\mathbf{w}(n))[\mathbf{w} - \mathbf{w}(n)] + \delta \|\mathbf{w} - \mathbf{w}(n)\|^2 \big] \end{aligned}$$

According to Gauss-Newton method:

$$\mathbf{w}(n+1) = \underset{\mathbf{w}}{\operatorname{argmin}} \hat{\mathcal{E}}(\mathbf{w}; \mathbf{w}(n))$$

$$\begin{aligned} \nabla_{\mathbf{w}} \hat{\mathcal{E}}(\mathbf{w}; \mathbf{w}(n)) &= J^T(\mathbf{w}(n))\mathbf{e}(\mathbf{w}(n)) + J^T(\mathbf{w}(n))J(\mathbf{w}(n))[\mathbf{w} - \mathbf{w}(n)] + \delta[\mathbf{w} - \mathbf{w}(n)] = 0 \\ &\Rightarrow J^T(\mathbf{w}(n))e(\mathbf{w}(n)) + (J^T(\mathbf{w}(n))J(\mathbf{w}(n)) + \delta \boldsymbol{I})[\mathbf{w} - \mathbf{w}(n)] = 0 \\ \Rightarrow \mathbf{w}(n+1) &= \mathbf{w}(n) - (J^T(\mathbf{w}(n))J(\mathbf{w}(n)) + \delta \boldsymbol{I})^{-1} J^T(\mathbf{w}(n))\mathbf{e}(\mathbf{w}(n)) \end{aligned}$$

3. Convergence in the mean square means that $E\left[e^2(n)\right] \xrightarrow[n \to \infty]{} \text{constant}$.

   In the conventional LMS algorithm we have

   $$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta e(n)\mathbf{x}(n)$$

   which is convergent in the mean square if

   $$0 < \eta < \frac{2}{\text{sum of mean-square values of the inputs}}$$

   or

   $$0 < \eta < \frac{2}{\|\mathbf{x}(n)\|^2} \forall n.$$

   (This is a stricter condition than the first one!)

   In the normalized LMS algorithm we have

   $$\mathbf{w}(n+1) = \mathbf{w}(n) + \tilde{\eta} \frac{e(n)\mathbf{x}(n)}{\|\mathbf{x}(n)\|^2}$$

   Comparing this to the conventional LMS algorithm we have:

   $$\tilde{\eta} = \eta \|\mathbf{x}(n)\|^2$$

   Using this result in the convergence condition yields the condition for convergence of the normalized LMS algorithm in the mean square as $0 < \tilde{\eta} < 2$.

4. (a) We are given the cost function

$$J(\mathbf{w}) = \frac{1}{2}E[(d(n) - \mathbf{x}^T(n)\mathbf{w})^2].$$

Expanding terms, we get

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{2}E[d^2(n)] - E[d(n)\mathbf{x}^T(n)]\mathbf{w} + \frac{1}{2}\mathbf{w}^T E[\mathbf{x}(n)\mathbf{x}^T(n)]\mathbf{w} \\ &= \frac{1}{2}\sigma_d^2 - \mathbf{r}_{xd}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\boldsymbol{R}_x\mathbf{w} \end{aligned}$$

(b) The gradient vector is defined by

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \mathbf{g} = -\mathbf{r}_{xd} + \boldsymbol{R}_x\mathbf{w}$$

The Hessian matrix is defined by

$$\frac{\partial^2 J(\mathbf{w})}{\partial \mathbf{w}^2} = \boldsymbol{H} = \boldsymbol{R}_x^T = \boldsymbol{R}_x$$

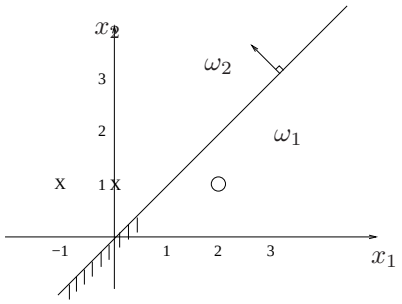
For the quadratic cost function $J(\mathbf{w})$ the Hessian $\boldsymbol{H}$ is exactly the same as the correlation matrix $\boldsymbol{R}_x$ of the input $\mathbf{x}$.

(c) According to the conventional form of Newtons method, we have (see equation 3.16 on page 124 in Haykin) $\Delta\mathbf{w} = -\boldsymbol{H}^{-1}\mathbf{g}$, where $\boldsymbol{H}^{-1}$ is the inverse of the Hessian and $\mathbf{g}$ is the gradient vector. The instantaneous value of the gradient vector is

$$\begin{aligned} \hat{\mathbf{g}}(n) &= -\mathbf{x}(n)d(n) + \mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w} \\ &= -\mathbf{x}(n)(d(n) - \mathbf{x}^T(n)\mathbf{w}) \\ \Rightarrow \hat{\mathbf{w}}(n+1) &= \hat{\mathbf{w}}(n) + \eta\boldsymbol{R}_x^{-1}\mathbf{x}(n)(d(n) - \mathbf{x}^T(n)\hat{\mathbf{w}}) \end{aligned}$$

5. (a)

$$\omega_1 : \{[2,1]^T\},$$
$$\omega_2 : \{[0,1]^T, [-1,1]^T\}.$$

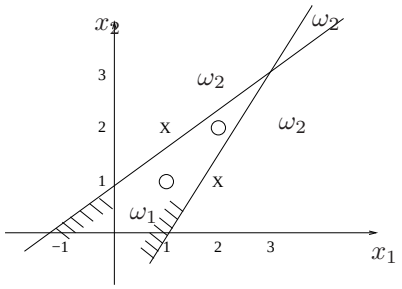


Classifications are carried out as follows:

$$\begin{cases} \mathbf{w}^T\mathbf{x} < 0 \Rightarrow \mathbf{x} \in \omega_1 \\ \mathbf{w}^T\mathbf{x} \geq 0 \Rightarrow \mathbf{x} \in \omega_2 \end{cases}$$

(b)

$\omega_1 : \{[2,1]^T, [3,2]^T\}$,
$\omega_2 : \{[3,1]^T, [2,2]^T\}$



It is not possible to separate the classes with a single hyperplane. At least two hyperplanes are required to separated the classes correctly.

T-61.3030 Principles of Neural Computing

Raivio, Koskela, Pöllä

Exercise 5,

1. The McCulloch-Pitts perceptrons can be used to perform numerous logical tasks. Neurons are assumed to have two binary input signals, x_1 and x_2, and a constant bias signal which are combined into an input vector as follows: $\mathbf{x} = [x_1, x_2, -1]^T,\ x_1, x_2 \in \{0, 1\}$. The output of the neuron is given by

$$y = \begin{cases} 1, & \text{if } \mathbf{w}^T\mathbf{x} > 0 \\ 0, & \text{if } \mathbf{w}^T\mathbf{x} \leq 0 \end{cases}$$

   where $\mathbf{w}$ is an adjustable weight vector. Demonstrate the implementation of the following binary logic functions with a single neuron:

   (a) A
   (b) not B
   (c) A or B
   (d) A and B
   (e) A nor B
   (f) A nand B
   (g) A xor B.

   What is the value of weight vector in each case?

2. A single perceptron is used for a classification task, and its weight vector $\mathbf{w}$ is updated iteratively in the following way:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \alpha(y - y')\mathbf{x}$$

   where $\mathbf{x}$ is the input signal, $y' = \text{sgn}(\mathbf{w}^T\mathbf{x}) = \pm 1$ is the output of the neuron, and $y = \pm 1$ is the correct class. Parameter α is a positive learning rate. How does the weight vector $\mathbf{w}$ evolve from its initial value $\mathbf{w}(0) = [1, 1]^T$, when the above updating rule is applied with $\alpha = 0.4$, and we have the following samples from classes $\mathcal{C}_1$ and $\mathcal{C}_2$:

$$\mathcal{C}_1 : \{[2, 1]^T\},$$
$$\mathcal{C}_2 : \{[0, 1]^T, [-1, 1]^T\}$$

3. Suppose that in the signal-flow graph of the perceptron illustrated in Figure 9 the hard limiter is replaced by the sigmoidal linearity:

$$\varphi(v) = \tanh(\frac{v}{2})$$

   where v is the induced local field. The classification decisions made by the perceptron are defined as follows:

   Observation vector $\mathbf{x}$ *belongs to class* $\mathcal{C}_1$ *if the output* $y > \theta$ *where* θ *is a threshold; otherwise,* $\mathbf{x}$ *belongs to class* $\mathcal{C}_2$

   Show that the decision boundary so constructed is a hyperplane.

4. Two pattern classes, $\mathcal{C}_1$ and $\mathcal{C}_2$, are assumed to have Gaussian distributions which are centered around points $\mu_1 = [-2, -2]^T$ and $\mu_2 = [2, 2]^T$ and have the following covariance matrixes:

$$\Sigma_1 = \begin{bmatrix} \alpha & 0 \\ 0 & 1 \end{bmatrix} \text{ and } \Sigma_2 = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}.$$

Plot the distributions and determine the optimal Bayesian decision surface for $\alpha = 3$ and $\alpha = 1$. In both cases, assume that the prior probabilities of the classes are equal, the costs associated with correct classifications are zero, and the costs associated with misclassifications are equal.

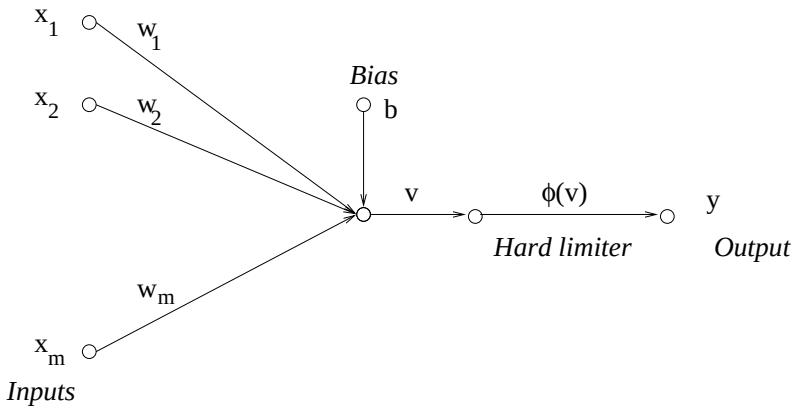


Figure 9: The signal-flow graph of the perceptron.

T-61.3030 Principles of Neural Computing

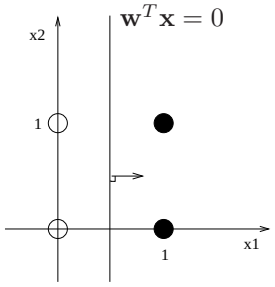
Answers to Exercise 5

1. Let input signals x_1 and x_2 represent events A and B, respectively, so that

$$\begin{cases} \text{A is true} & \Leftrightarrow \quad x_1 = 1 \\ \text{B is true} & \Leftrightarrow \quad x_2 = 1 \end{cases}$$

(a)

$$y = \text{A} \rightarrow \begin{cases} w_1 1 + w_2 0 - \theta & > 0 \\ w_1 1 + w_2 1 - \theta & > 0 \\ w_1 0 + w_2 0 - \theta & \leq 0 \\ w_1 0 + w_2 1 - \theta & \leq 0 \end{cases}$$



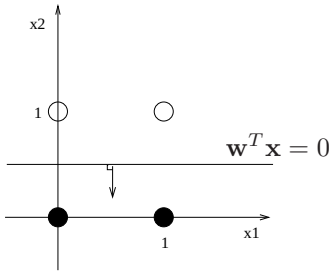
From the figure

$$x_1 = \frac{1}{2} \Leftrightarrow \begin{pmatrix} 1 & 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ -1 \end{pmatrix} = 0$$

$$\mathbf{w} = \begin{pmatrix} 1 & 0 & \frac{1}{2} \end{pmatrix}^T$$

(b)

$$y = \text{not B} \rightarrow \begin{cases} w_1 1 + w_2 0 - \theta & > 0 \\ w_1 1 + w_2 1 - \theta & \leq 0 \\ w_1 0 + w_2 0 - \theta & > 0 \\ w_1 0 + w_2 1 - \theta & \leq 0 \end{cases}$$



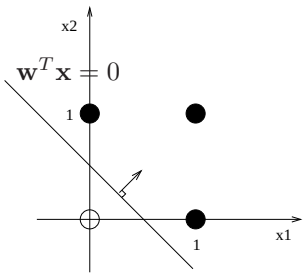
From the figure

$$x_2 = \frac{1}{2} \Leftrightarrow \begin{pmatrix} 0 & 1 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ -1 \end{pmatrix} = 0$$

$$\mathbf{w} = \begin{pmatrix} 0 & -1 & -\frac{1}{2} \end{pmatrix}^T$$

(c)

$$y = \text{A or B} \rightarrow \begin{cases} w_1 1 + w_2 0 - \theta & > 0 \\ w_1 1 + w_2 1 - \theta & > 0 \\ w_1 0 + w_2 0 - \theta & \leq 0 \\ w_1 0 + w_2 1 - \theta & > 0 \end{cases}$$

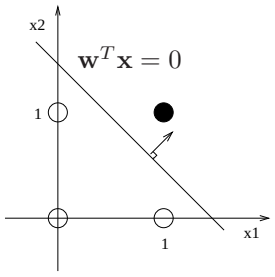


From the figure

$$\begin{aligned} x_2 &= -x_1 + \frac{1}{2} \Leftrightarrow \begin{pmatrix} 1 & 1 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ -1 \end{pmatrix} = 0 \\ \mathbf{w} &= \begin{pmatrix} 1 & 1 & \frac{1}{2} \end{pmatrix}^T \end{aligned}$$

(d)

$$y = \text{A and B} \rightarrow \begin{cases} w_1 1 + w_2 0 - \theta & \leq 0 \\ w_1 1 + w_2 1 - \theta & > 0 \\ w_1 0 + w_2 0 - \theta & \leq 0 \\ w_1 0 + w_2 1 - \theta & \leq 0 \end{cases}$$



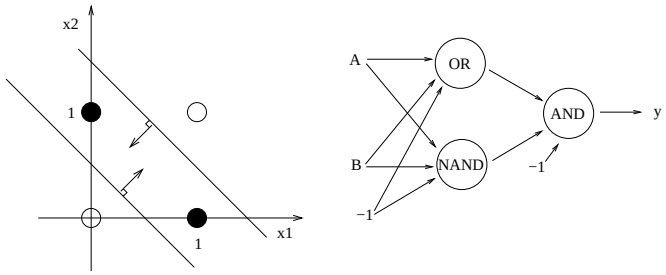
From the figure

$$\begin{aligned} x_2 &= -x_1 + \frac{3}{2} \Leftrightarrow \begin{pmatrix} 1 & 1 & \frac{3}{2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ -1 \end{pmatrix} = 0 \\ \mathbf{w} &= \begin{pmatrix} 1 & 1 & \frac{3}{2} \end{pmatrix}^T \end{aligned}$$

(e) $y = \text{A nor B} = \text{not (A or B)} \Leftarrow \mathbf{w} = -\begin{pmatrix} 1 & 1 & \frac{1}{2} \end{pmatrix}^T$

(f) $y = \text{A nand B} = \text{not (A and B)} \Leftarrow \mathbf{w} = -\begin{pmatrix} 1 & 1 & \frac{3}{2} \end{pmatrix}^T$

(g) y = A xor B = (A or B) and (not(A and B))



This can not be solved with a single neuron. Two layers are needed.

2. The updating rule for the weight vector is $\mathbf{w}(n+1) = \mathbf{w}(n) + \alpha(y - y')\mathbf{x}$, $\mathbf{w}(0) = (\ 1 \quad 1\ )^T$ and $\alpha = 0.4$.

n	$\mathbf{w}(n)^T$	$\mathbf{x}(n)^T$	$\mathbf{w}(n)^T\mathbf{x}(n)$	$y(n)$	$y'(n)$	$\alpha(y(n) - y'(n))\mathbf{x}(n)$
0	(1 1)	(2 1)	3	+1	+1	(0 0)
1	(1 1)	(0 1)	1	-1	+1	(0 −0.8)
2	(1 0.2)	(−1 1)	-0.8	-1	-1	(0 0)
3	(1 0.2)	(2 1)	2.2	+1	+1	(0 0)
4	(1 0.2)	(0 1)	0.2	-1	+1	(0 −0.8)
5	(1 −0.6)	(−1 1)	-1.6	-1	-1	(0 0)
6	(1 −0.6)	(2 1)	1.4	+1	+1	(0 0)
7	(1 −0.6)	(0 1)	-0.6	-1	-1	(0 0)
	⋮					⋮

3. The output signal is defined by

$$y = \tanh\left(\frac{\upsilon}{2}\right) = \tanh\left(\frac{b}{2} + \frac{1}{2}\sum_i w_i x_i\right).$$

Equivalently, we may write

$$y' = b + \sum_i w_i x_i = 2\tanh^{-1}(y).$$

This is the equation of a hyperplane.

The decision boundary: $y = \theta \Rightarrow 2\tanh^{-1}(\theta) = b + \sum_i w_i x_i = \theta'$

4. The optimal Bayes decision surface is found by minimizing the expected risk (expected cost)

$$R = \int_{H_1} c_{11}p(x|c_1)P(c_1)dx + \int_{H_2} c_{22}p(x|c_2)P(c_2)dx + \int_{H_2} c_{21}p(x|c_1)P(c_1)dx + \int_{H_1} c_{12}p(x|c_2)P(c_2)dx,$$

where c_{ij} is the cost associated with the classification of x into class i when it belongs to class j, H_i is a classification region for class i, $p(x|c_i)$ is the probability of x given that its classification is ci and $P(c_i)$ is the prior probability of class c_i.
Now $H = H_1 + H_2$ and $\int_{H-H_1} p(x|c_i)dx = 1 - \int_{H_1} p(x|c_i)dx$.

$$\Rightarrow R = c_{21}P(c_1) + c_{22}P(c_2) + \int_{H_1} (c_{12} - c_{22})p(x|c_2)P(c_2) - (c_{21} - c_{11})p(x|c_1)P(c_1)dx$$

The first two terms are constant and thus don't affect optimization. The risk is minimized if we divide the pattern space in the following manner:
if $(c_{21} - c_{11})p(x|c_1)P(c_1) > (c_{12} - c_{22})p(x|c_2)P(c_2)$,$x \in H_1$; otherwise $x \in H_2$
The decision surface is

$$(c_{21} - c_{11})p(x|c_1)P(c_1) = (c_{12} - c_{22})p(x|c_2)P(c_2)$$

Now:

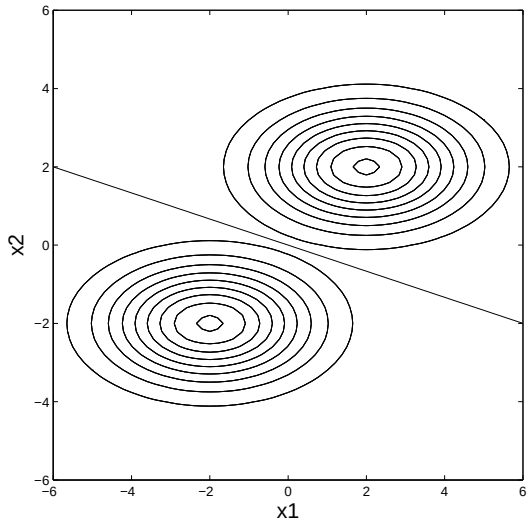
$$\begin{cases} P(c_1) = P(c_2) = \frac{1}{2} \\ c_{11} = c_{22} = 0 \\ c_{21} = c_{12} = c \\ p(x|c_i) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_i|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}-\mu_i)} \end{cases}$$

and the decision surface becomes

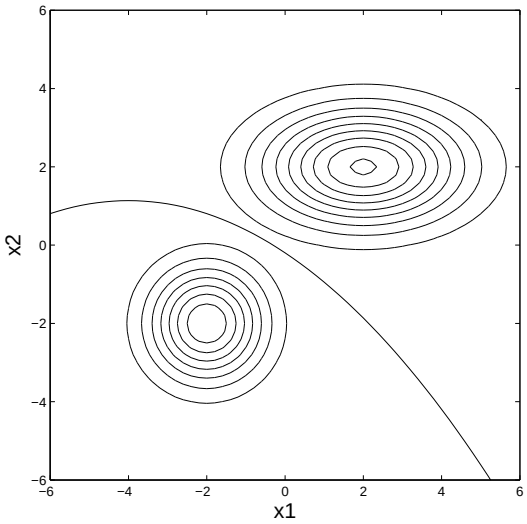
$$\ln\left(\frac{1}{2\pi|\Sigma_1|^{\frac{1}{2}}}\right) - \frac{1}{2}(\mathbf{x}-\mu_1)^T\Sigma_1^{-1}(\mathbf{x}-\mu_1) = \ln\left(\frac{1}{2\pi|\Sigma_2|^{\frac{1}{2}}}\right) - \frac{1}{2}(\mathbf{x}-\mu_2)^T\Sigma_2^{-1}(\mathbf{x}-\mu_2).$$

$$\begin{aligned} \Rightarrow \quad & \ln\left(\frac{1}{2\pi\alpha^{\frac{1}{2}}}\right) - \ln\left(\frac{1}{2\pi 3^{\frac{1}{2}}}\right) = \\ & \frac{1}{2}\left[(\mathbf{x} + \begin{pmatrix} 2 \\ 2 \end{pmatrix})^T \begin{pmatrix} \frac{1}{\alpha} & 0 \\ 0 & 1 \end{pmatrix} (\mathbf{x} + \begin{pmatrix} 2 \\ 2 \end{pmatrix}) - (\mathbf{x} - \begin{pmatrix} 2 \\ 2 \end{pmatrix})^T \begin{pmatrix} \frac{1}{3} & 0 \\ 0 & 1 \end{pmatrix} (\mathbf{x} - \begin{pmatrix} 2 \\ 2 \end{pmatrix})\right] \\ \Leftrightarrow \quad & \ln\left(\frac{3}{\alpha}\right) = \frac{1}{\alpha}(x_1+2)^2 + (x_2+2)^2 - \frac{1}{3}(x_1-2)^2 - (x_2-2)^2 \\ & = \left(\frac{1}{\alpha} - \frac{1}{3}\right)x_1^2 + 4\left(\frac{1}{\alpha} + \frac{1}{3}\right)x_1 + 8x_2 + 4\left(\frac{1}{\alpha} - \frac{1}{3}\right) \end{aligned}$$

$\alpha = 3 \Rightarrow 0 = 4\frac{2}{3}x_1 + 8x_2 \Leftrightarrow x_1 + 3x_2 = 0$
$\alpha = 1 \Rightarrow \ln(3) = \frac{2}{3}x_1^2 + 4\frac{4}{3}x_1 + 8x_2 + 4\frac{2}{3} \Leftrightarrow 2x_1^2 + 16x_1 + 32x_2 + 8 - 3\ln 3 = 0$



$\alpha = 3$



$\alpha = 1$

T-61.3030 Principles of Neural Computing

Raivio, Koskela, Pöllä

Exercise 6

1. Construct a MLP network which is able to separate the two classes illustrated in Figure 10. Use two neurons both in the input and output layer and an arbitrary number of hidden layer neurons. The output of the network should be vector $[1, 0]^T$ if the input vector belongs to class $\mathcal{C}_1$ and $[0, 1]^T$ if it belongs to class $\mathcal{C}_2$. Use nonlinear activation functions, namely McCulloch-Pitts model, for all the neurons and determine their weights by hand without using any specific learning algorithm.

   (a) What is the minimum amount of neurons in the hidden layer required for a perfect separation of the classes?

   (b) What is the maximum amount of neurons in the hidden layer?

2. The function

   $$t(x) = x^2, \quad x \in [1, 2]$$

   is approximated with a neural network. The activation functions of all the neurons are linear functions of the input signals and a constant bias term. The number neurons and the network architecture can be chosen freely. The approximation performance of the network is measured with the following error function:

   $$\mathcal{E} = \int_1^2 [t(\mathbf{x}) - y(\mathbf{x})]^2 d\mathbf{x},$$

   where $\mathbf{x}$ is the input vector of the network and $y(\mathbf{x})$ is the corresponding response.

   (a) Construct a single-layer network which minimizes the error function.

   (b) Does the approximation performance of the network improve if additional hidden layers are included?

3. The MLP network of Figure 11 is trained for classifying two-dimensional input vectors into two separate classes. Draw the corresponding class boundaries in the (x_1, x_2)-plane assuming that the activation function of the neurons is (a) sign, and (b) tanh.

4. Show that (a)

   $$\Delta w_{ij}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) y_i(t)$$

   is the solution of the following difference equation:

   $$\Delta w_{ij}(n) = \alpha \Delta w_{ij}(n-1) + \eta \delta_j(n) y_i(n),$$

   where α is a positive momentum constant. (b) Justify the claims 1-3 made on the effects of the momentum term in Haykin pp. 170-171.

5. Consider the simple example of a network involving a single weight, for which the cost function is

   $$\mathcal{E}(w) = k_1(w - w_0)^2 + k_2,$$

   where w_0, k_1, and k_2 are constants. A back-propagation algorithm with momentum is used to minimize $\mathcal{E}(w)$. Explore the way in which the inclusion of the momentum constant α influences the learning process, with particular reference to the number of epochs required for convergence versus α.

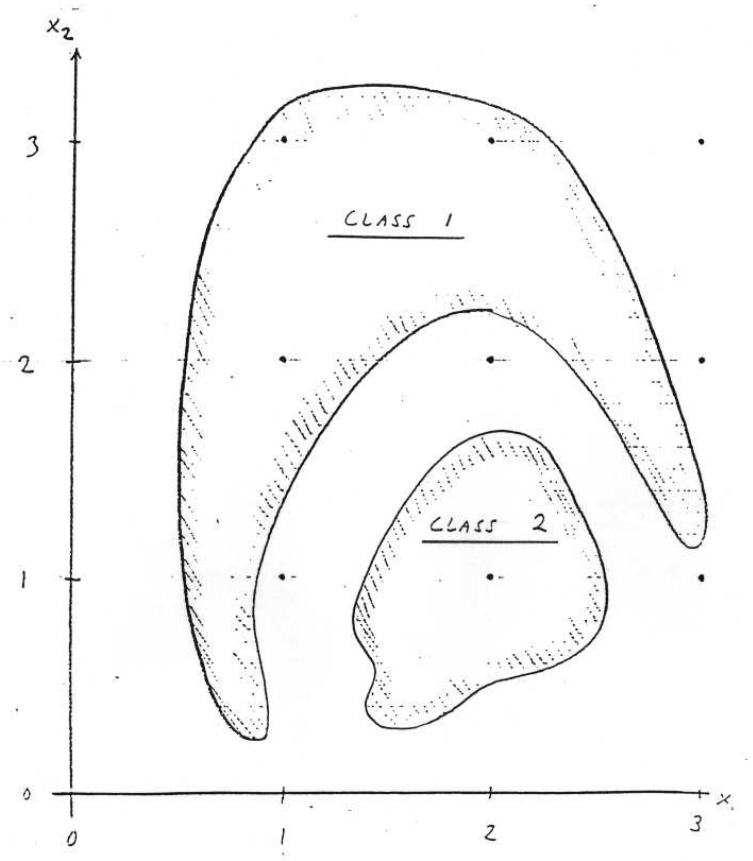


Figure 10: Classes $\mathcal{C}_1$ and $\mathcal{C}_2$.

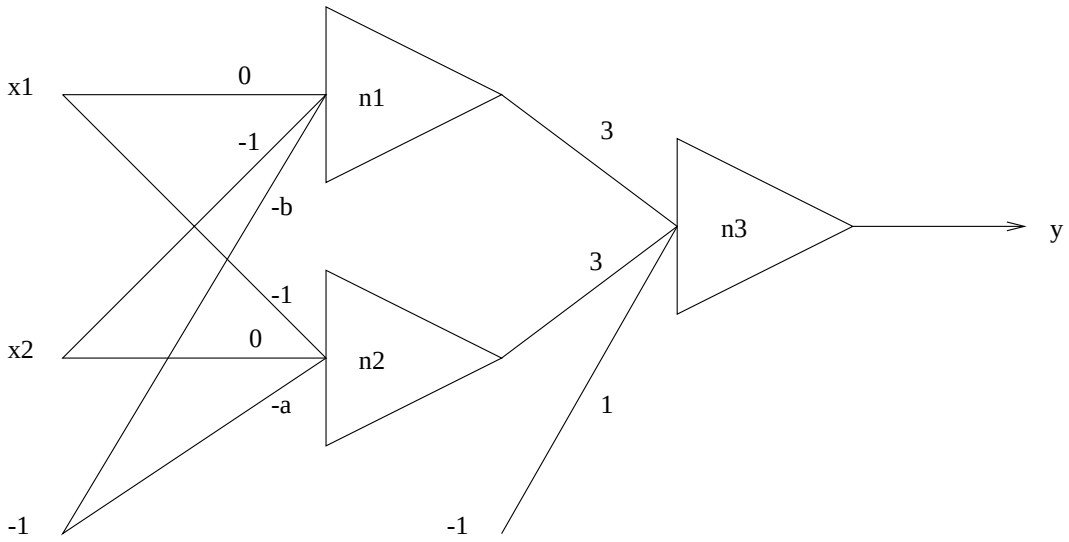
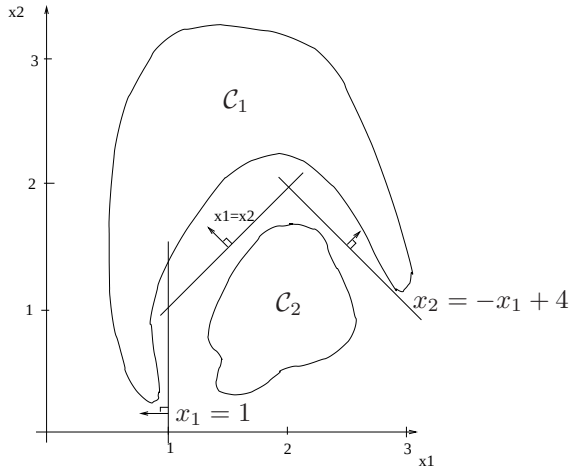


Figure 11: The MLP network.

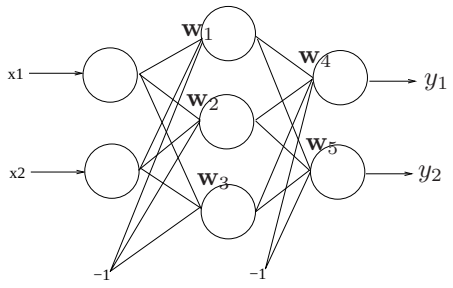
T-61.3030 Principles of Neural Computing

Answers to Exercise 6

1. A possible solution:



The network:



From the above figure:

$$\left.\begin{array}{l} \mathbf{w}_1 = (\ -1 \quad 0 \quad 1\ )^T \\ \mathbf{w}_2 = (\ -1 \quad 1 \quad 0\ )^T \\ \mathbf{w}_3 = (\ 1 \quad 1 \quad -4\ )^T \end{array}\right\} \text{hidden layer}$$

Let $z_i \in \{0, 1\}$ be the output of the i:th hidden layer neuron and $\mathbf{z} = (\ z_1 \quad z_2 \quad z_3 \quad -1\ )^T$.
The outputs of McCulloch-Pitts perceptrons are determined as follows:

$$z_i = \begin{cases} 1, \text{ if } \mathbf{w}_i^T \mathbf{x} > 0 \\ 0, \text{ otherwise} \end{cases}$$

The weights of the output layer neurons, $\mathbf{w}_4$ and $\mathbf{w}_5$ are set so that

$\mathbf{w}_4 = -\mathbf{w}_5$ and

$$\begin{cases} \mathbf{w}_4^T \mathbf{z} > 0, \text{ if } z_1 = 1 \text{ or } z_2 = 1 \text{ or } z_3 = 1 \\ \mathbf{w}_4^T \mathbf{z} \leq 0, \text{ otherwise} \end{cases}$$

$\mathbf{w}_4 = \left(\ 1 \quad 1 \quad 1 \quad \frac{1}{2}\ \right)$ is a feasible solution. (The above problem has infinitely many solutions!)

(a) The minimum amount of neurons in the hidden layer is two as the classes can be separated with two lines.

(b) There is no upper limit for the number of hidden layer neurons. However, the network might over learn the training set and loose its generalization capability. When the number of hidden layer neurons is increased, the boundary between the classes can be estimated more precisely.

2. Function $t(x) = x^2$, $x \in [1, 2]$ is approximated with a neural network. All neurons have linear activation functions and constant bias terms.

(a) Output of a single-layer network is

$$y(x) = \sum_i w_i x + \theta = Wx + \theta$$

and the approximation performance measure is

$$\begin{aligned}
\mathcal{E} &= \int_1^2 (t(x) - y(x))^2 dx = \int_1^2 (x^2 - Wx - \theta)^2 dx \\
&= \int_1^2 (x^4 + \theta^2 + W^2x^2 - 2Wx^3 + 2W\theta x - 2\theta x^2) dx \\
&= \Big/_1^2 \frac{x^5}{5} + \theta^2 x + \frac{W^2x^3}{3} - \frac{Wx^4}{2} + \frac{W\theta x^2}{-} \frac{2}{3}\theta x^3 \\
&= \frac{31}{5} + \theta^2 + \frac{7}{3}W^2 - \frac{15}{2}W + 3W\theta - \frac{14}{3}\theta
\end{aligned}$$

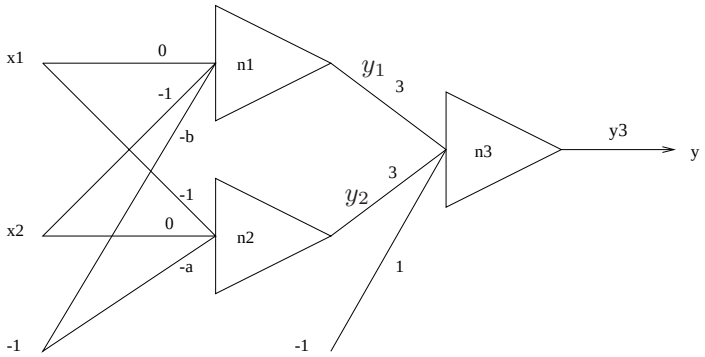
Lets find W and θ which minimize $\mathcal{E}$:

$$\begin{cases} \frac{\partial \mathcal{E}}{\partial W} = \frac{14}{3}W - \frac{15}{2} + 3\theta = 0 \\ \frac{\partial \mathcal{E}}{\partial \theta} = 2\theta + 3W - \frac{14}{3} = 0 \end{cases} \Rightarrow \begin{cases} W^* = 3 \\ \theta^* = -2\frac{1}{6} \end{cases}$$

As $\left(\begin{array}{cc} \frac{\partial^2 \mathcal{E}}{\partial W^2} & \frac{\partial^2 \mathcal{E}}{\partial W \partial \theta} \\ \frac{\partial^2 \mathcal{E}}{\partial W \partial \theta} & \frac{\partial^2 \mathcal{E}}{\partial \theta^2} \end{array} \right) \Big|_{\substack{W=W^* \\ \theta=\theta^*}}$ is positive definite, $\mathcal{E}$ gets its minimum value when $W = W^*$ and $\theta = \theta^*$.

(b) No because $y(x) = \sum_z w_{1z}^{(n)} \left(\sum_y w_{zy}^{(n-1)} \left(\cdots \sum_a w_{a1}^{(1)} x \right) \cdots \right) = Wx$

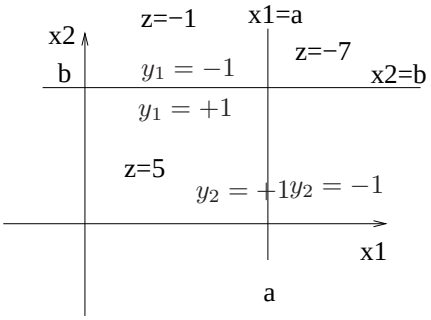
3. Network

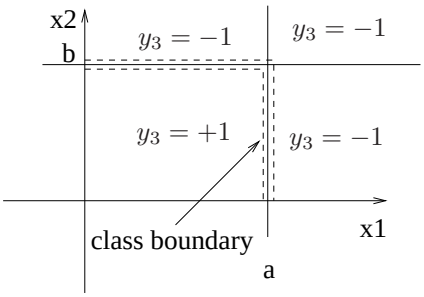


(a)
$$\begin{cases} y_1 = \text{sign}(-x_2 + b) \\ y_2 = \text{sign}(-x_1 + a) \\ y_3 = \text{sign}(\underbrace{3y_1 + 3_y 2 - 1}_{z}) \end{cases}$$

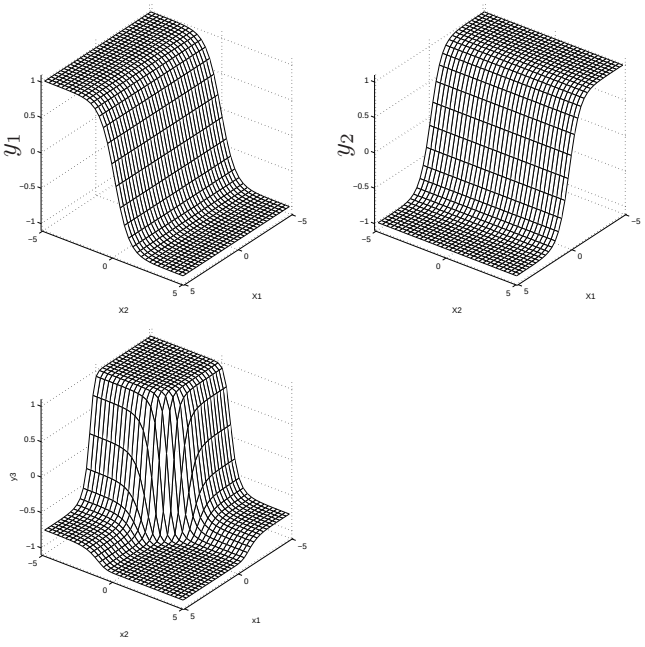
Hidden layer

Output layer





(b) Activation function is tanh.



4. (a)

$$\begin{aligned} \Delta w_{ij}(n) &= \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) y_i(t) \ (1) \\ \Rightarrow \Delta w_{ij}(0) &= \eta \delta_j(0) y_i(0) \ (2) \end{aligned}$$

On the other hand

$$\begin{aligned} \Delta w_{ij}(n) &= \alpha \Delta w_{ij}(n-1) + \eta \delta_j(n) y_i(n) \ (3) \\ (2)\ \&\ (3) \Rightarrow \Delta w_{ij}(1) &= \alpha \Delta w_{ij}(0) + \eta \delta_j(1) y_i(1) \\ &= \alpha \eta \delta_j(0) y_i(0) + \eta \delta_j(1) y_i(1) \\ &= \eta \sum_{t=0}^{1} \alpha^{1-t} \delta_j(t) y_i(t) \ (4) \\ (2)\ \&\ (4) \Rightarrow \Delta w_{ij}(2) &= \alpha \Delta w_{ij}(1) + \eta \delta_j(2) y_i(2) \\ &= \alpha \eta \sum_{t=0}^{1} \alpha^{1-t} \delta_j(t) y_i(t) + \eta \delta_j(2) y_i(2) \\ &= \eta \sum_{t=0}^{2} \alpha^{2-t} \delta_j(t) y_i(t) \ (4) \\ &\vdots \\ \Delta w_{ij}(n) &= \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) y_i(t) \Rightarrow \ (1) \text{ is solution to } (3) \end{aligned}$$

(b) Claims on the effect of the momentum term.

Claim 1 The current adjustment $\Delta w_{ij}(n)$ represents the sum of an exponentially weighted time series. For the time series to be convergent, the momentum constant must be restricted to the range $0 \leq |\alpha| < 1$.
if $|\alpha| > 1$, clearly then $|\alpha|^{n-t} \xrightarrow[n\to\infty]{} \infty$, and the solution$\Delta w_{ij}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) y_i(t)$ "explodes" (becomes unstable). In the limiting case $|\alpha| = 1$, the terms $\delta_j(t) y_i(t)$ may add up so that the sum becomes very large, depending on their signs. On the other hand, if $0 \leq |\alpha| < 1, |\alpha|^{n-t} \xrightarrow[n\to\infty]{} 0$, and the sum converges.

Claim 2 When the partial derivative $\frac{\partial \mathcal{E}(t)}{\partial w_{ij}(t)} = -\delta_j(t) y_i(t)$ has the same algebraic sign on the consecutive iterations, the exponentially weighted sum $\Delta w_{ij}(n)$ grows in magnitude, and so the weight $w_{ij}(n)$ is adjusted by a large amount.
Assume for example that $\frac{\partial \mathcal{E}(t)}{\partial w_{ij}(t)}$ is positive on consecutive iterations. Then for positive α the subsequent terms $\alpha^{n-t} \frac{\partial \mathcal{E}(t)}{\partial w_{ij}(t)}$ are all positive, and the sum $\Delta w_{ij}(n)$ grows in magnitude. The same holds for negative $\frac{\partial \mathcal{E}(t)}{\partial w_{ij}(t)}$: the terms in summation are now all negative, and reinforce each other.

Claim 3 When the partial derivative $\frac{\partial \mathcal{E}(t)}{\partial w_{ij}(t)}$ has opposite signs on consecutive iterations, the exponentially weighted sum $\Delta_{ij}(n)$ shrinks in magnitude, so that weight $w_{ij}(n)$ is adjusted by a small amount.
The terms $\alpha^{n-t} \frac{\partial \mathcal{E}(t)}{\partial w_{ij}(t)}$ have now different signs for subsequent iterations $t-2, t-1, \cdots$, and tend to cancel each other.

5. From Eq. (4.41 Haykin) we have

$$\Delta w_{ij}(n) = -\eta \sum_{t=1}^{n} \alpha^{n-t} \frac{\partial \mathcal{E}(t)}{\partial w_{ij}(t)} \quad (1)$$

For the case of a single weight the cost function is defined by

$$\mathcal{E} = k_1(w - w_0)^2 + k_2.$$

Hence, the application of (1) to this case yields

$$\Delta w(n) = -2k_1\eta \sum_{t=1}^{n} \alpha^{n-t}(w - w_0)$$

In the case the partial derivative $\frac{\partial \mathcal{E}(t)}{\partial w(t)}$ has the same algebraic sign on the consecutive iterations and $0 \leq \alpha < 1$, the exponentially weighted adjustment $\Delta w(n)$ to the weight w at time n grows in magnitude. That is, the weight w is adjusted by a large amount. The inclusion of the momentum constant α in the algorithm for computing the optimum wight $w^* = w_0$ tends to accelerate the downhill descent toward the optimum point.

T-61.3030 Principles of Neural Computing

Raivio, Koskela, Pöllä

Exercise 7

1. In section 4.6 (part 5, Haykin pp. 181) it is mentioned that the inputs should be normalized to accelerate the convergence of the back-propagation learning process by preprocessing them as follows: 1) their mean should be close to zero, 2) the input variables should be uncorrelated, and 3) the covariances of the decorrelated inputs should be approximately equal.

   (a) Devise a method based on principal component analysis performing these steps.

   (b) Is the proposed method unique?

2. A continuous function $h(x)$ can be approximated with a step function in the closed interval $x \in [a, b]$ as illustrated in Figure 12.

   (a) Show how a single column, that is of height $h(x_i)$ in the interval $x \in (x_i - \Delta x/2, x_i + \Delta x/2)$ and zero elsewhere, can be constructed with a two-layer MLP. Use two hidden units and the sign function as the activation function. The activation function of the output unit is taken to be linear.

   (b) Design a two-layer MLP consisting of such simple sub-networks which approximates function $h(x)$ with a precision determined by the width and the number of the columns.

   (c) How does the approximation change if tanh is used instead of sign as an activation function in the hidden layer?

3. A MLP is used for a classification task. The number of classes is C and the classes are denoted with $\omega_1, \ldots, \omega_C$. Both the input vector $\mathbf{x}$ and the corresponding class are random variables, and they are assumed to have a joint probability distribution $p(\mathbf{x}, \omega)$. Assume that we have so many training samples that the back-propagation algorithm minimizes the following expectation value:

$$E\left(\sum_{i=1}^{C}[y_i(\mathbf{x}) - t_i]^2\right),$$

   where $y_i(\mathbf{x})$ is the actual response of the ith output neuron and t_i is the desired response.

   (a) Show that the theoretical solution of the minimization problem is

$$y_i(\mathbf{x}) = E(t_i|x).$$

   (b) Show that if $t_i = 1$ when $\mathbf{x}$ belongs to class ω_i and $t_i = 0$ otherwise, the theoretical solution can be written

$$y_i(\mathbf{x}) = P(\omega_i|\mathbf{x})$$

   which is the optimal solution in a Bayesian sense.

   (c) Sometimes the number of the output neurons is chosen to be less than the number of classes. The classes can be then coded with a binary code. For example in the case of 8 classes and 3 output neurons, the desired output for class ω_1 is $[0, 0, 0]^T$, for class ω_2 it is $[0, 0, 1]$ and so on. What is the theoretical solution in such a case?

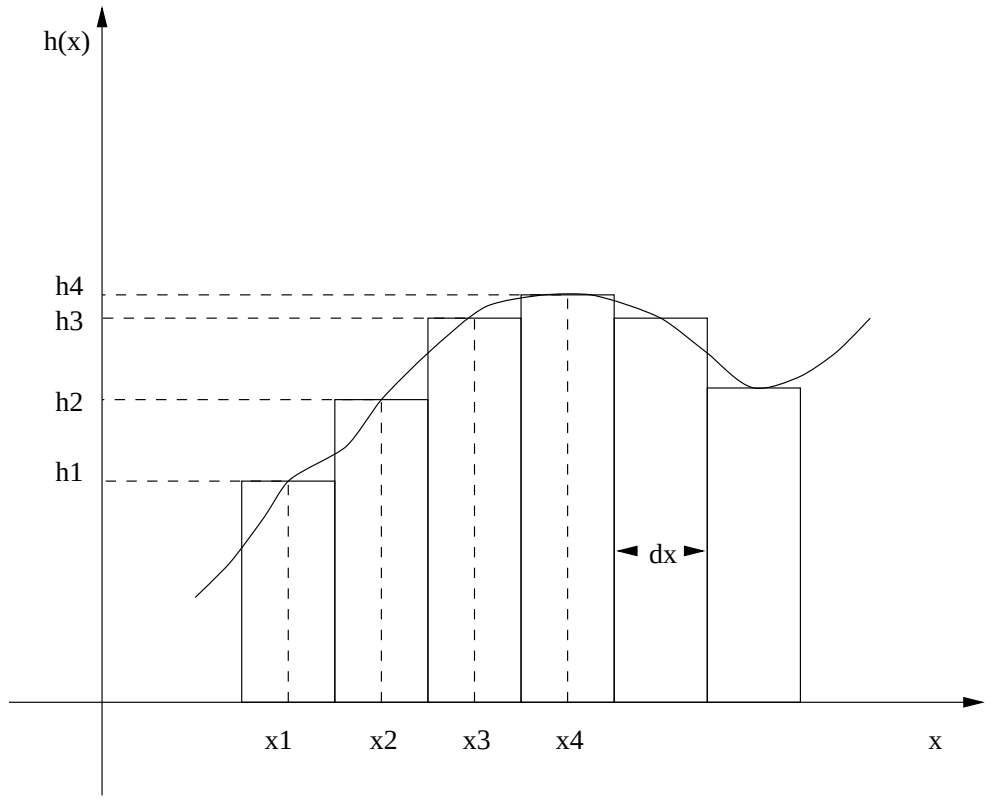


Figure 12: Function approximation with a step function.

T-61.3030 Principles of Neural Computing

Answers to Exercise 7

1. (a) Making the means to zero for the given set $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ of training (input) vectors is easy. Estimate fist the mean vector of the training set:

$$\hat{\mathbf{m}} = \frac{1}{N}\sum_{j=1}^{N}\mathbf{x}_j$$

Then, the modified vectors

$$\mathbf{z}_j = \mathbf{x}_j - \hat{\mathbf{m}}$$

have zero means. $(E(\mathbf{z}_j) = (0, 0, \ldots, 0)^T)$

2) and 3) The estimated covariance matrix of the zero mean vectors $\mathbf{z}_j$ is

$$\hat{\mathbf{C}}_{zz} = \frac{1}{N}\sum_{j=1}^{N}\mathbf{z}_j\mathbf{z}_j^T = \frac{1}{N}\sum_{j=1}^{N}(\mathbf{x}_j - \hat{\mathbf{m}})(\mathbf{x}_j - \hat{\mathbf{m}})^T$$

Theoretically, the conditions of uncorreltatedness and equal variance can be satisfied by lookin for a transformation $\mathbf{V}$ such that

$$\mathbf{y} = \mathbf{V}\mathbf{z}$$

and

$$\mathbf{C}_{yy} = E(\mathbf{y}\mathbf{y}^T) = E(\mathbf{V}\mathbf{z}\mathbf{z}\mathbf{V}^T) = \mathbf{V}E(\mathbf{z}\mathbf{z}^T)\mathbf{V}^T = \mathbf{V}\mathbf{C}_{zz}\mathbf{V}^T = \mathbf{I},$$

where $\mathbf{I}$ is the identity matrix.

Then for the components of $\mathbf{y}$ it hods that

$$\mathbf{C}_{yy}(j,k) = E(\mathbf{y}_j\mathbf{y}_k) = \begin{cases} 1, & \text{for } i = k \\ 0, & \text{for } i \neq k \end{cases}$$

For convenience, the variances are here set to unity. In practice, $\mathbf{C}_{zz}$ is replaced by its sample estimate $\hat{\mathbf{C}}_{zz}$. Now recall that for a symmetric matrix $\mathbf{C}$ its eigenvalue/eigenvector decomposition satisfies $\mathbf{U}\mathbf{C}\mathbf{U}^T = \mathbf{D}$ (*), where $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_M]$ is a matrix having the eigenvectors of $\mathbf{u}_i$ of $\mathbf{C}$ as its columns and the elements of the diagonal matrix $\mathbf{D} = [\lambda_1, \lambda_2, \ldots, \lambda_M]$ are the corresponging eigenvalues. These satisfy the condition

$$\mathbf{C}\mathbf{u}_i = \lambda_i\mathbf{u}_i \Leftrightarrow \mathbf{C}\mathbf{U} = \mathbf{D}\mathbf{U},$$

where

$$\mathbf{u}_i^T\mathbf{u}_j = \delta_{ij} \Leftrightarrow \mathbf{U}^T\mathbf{U} = \mathbf{I}$$

($\mathbf{U}$ is orthogonal)

When $\mathbf{C}$ is the covariance matric, this kind of decomposition is just the principal component ananlysis:

$$\mathbf{C}\mathbf{D}\mathbf{U}^T = \sum_{i=1}^{M}\lambda_i\mathbf{u}_i\mathbf{u}_i^T$$

Now multiplying (*) from the left and right by $\mathbf{D}^{-\frac{1}{2}} = [\lambda_1^{-\frac{1}{2}}, \lambda_2^{-\frac{1}{2}}, \ldots, \lambda_M^{-\frac{1}{2}}]$ we get

$$\mathbf{D}^{-\frac{1}{2}}\mathbf{U}\mathbf{C}\mathbf{U}^T\mathbf{D}^{-\frac{1}{2}} = \mathbf{I}$$

Hence the desired whitening (decorrelation - covariance equalitzation) transformation matrix $\mathbf{V}$ can be chosen as

$$\mathbf{V} = \mathbf{D}^{-\frac{1}{2}}\mathbf{U},\ \mathbf{V}^T = \mathbf{U}^T\mathbf{D}^{-\frac{1}{2}}$$

due to the symmetricity on $\mathbf{D}$. Furthermore $\mathbf{D}^{-\frac{1}{2}}$ always exists since the eigenvalues of a full-rank covariance matrix $\mathbf{C}$ are always positive.

(b) The removal of the mean is unique. However the whitening transformation matrix $\mathbf{V}$ is not unique. For example, we can change the order of the eigenvectors of $\mathbf{C}_{zz}$ in $\mathbf{U}$ (and respectively the order of the eigenvalues in $\mathbf{D}$!), and get a different transformation matrix.

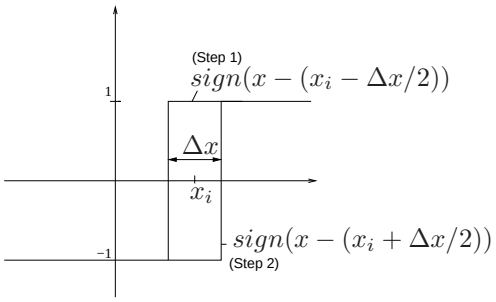
More generally, due to the symmetricity of $\mathbf{C}_{zz}$ the whitening condition $\mathbf{V}\mathbf{C}_{zz}\mathbf{V}^T = \mathcal{I}$ represents only $\frac{M(M+1)}{2}$ different conditions for the M^2 elements of $\mathbf{V}$ (assuming that $\mathbf{V}$ and $\mathbf{C}$ are $M * M$ matrices). Thus

$$M^2 - \frac{M(M+1)}{2} = \frac{M^2}{2} - \frac{M}{2} = \frac{M(M-1)}{2}$$

elements of $\mathbf{V}$ could be chosen freely.

There exist several other methods for performing whitening. For example , one can easily generalize the well-known Gram-Schimdt orthonormalization procedure for producing a whitening transformation matrix.

2. (a) Pillar



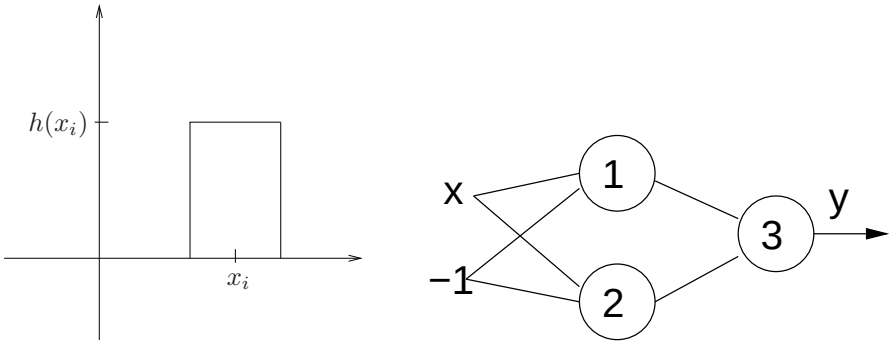
$$\begin{aligned} y &= \frac{h(x_i)}{2}(\text{step 1 - step 2}) \\ &= \frac{h(x_i)}{2}\left[\text{sign}\left(x - \left(x_i - \frac{\Delta x}{2}\right)\right) - \text{sign}\left(x - \left(x_i + \frac{\Delta x}{2}\right)\right)\right] \\ &= \mathbf{w}_3^T \begin{pmatrix} \text{sign}(\mathbf{w}_1^T\mathbf{x}) \\ \text{sign}(\mathbf{w}_2^T\mathbf{x}) \end{pmatrix} \end{aligned}$$

where

$$\mathbf{x} = \begin{pmatrix} x \\ -1 \end{pmatrix}$$

and

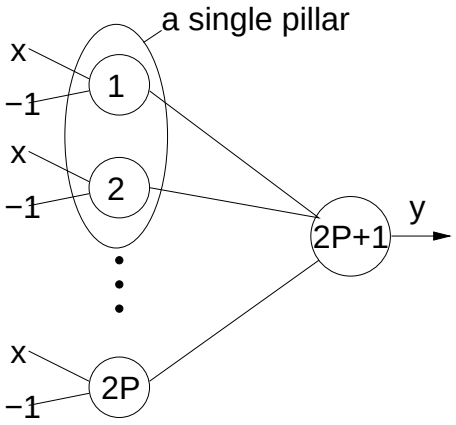
$$
\begin{aligned}
\mathbf{w}_1 &= (1, x_i - \Delta x/2)^T \\
\mathbf{w}_2 &= (1, x_i - \Delta x/2)^T \\
\mathbf{w}_3 &= (h(x_i)/2, -h(x_i)/2)^T
\end{aligned}
$$



(b) The pillars are combined by summing them up:

$$y = \sum_{i=1}^{P} \frac{h(x_i)}{2} \left[sign(x - (x_i - \Delta x/2)) - sign(x - (x_i + \Delta x/2)) \right]. \tag{42}$$

If x_i and Δx are correctly chosen ($x_{i+1} = x_i + \Delta x$, $\forall i = 1, \ldots, P-1$), only one of the terms in the sum is nonzero.



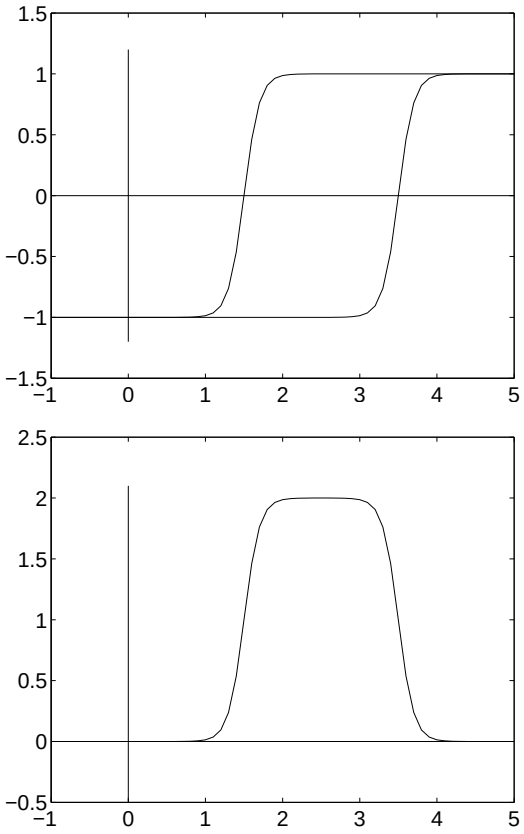
The weights of the hidden layer neurons are

$$
\begin{aligned}
\mathbf{w}_1 &= (1, x_1 - \Delta x/2)^T \\
\mathbf{w}_2 &= (1, x_1 - \Delta x/2)^T \\
&\vdots \\
\mathbf{w}_{2P-1} &= (1, x_P - \Delta x/2)^T \\
\mathbf{w}_{2P} &= (1, x_P - \Delta x/2)^T
\end{aligned}
$$

and the weight of the output neuron is

$$\mathbf{w}_{2P+1} = (h(x_1)/2, -h(x_1)/2, \ldots, h(x_P)/2, -h(x_P)/2)^T.$$

(c) If $tanh(\alpha x)$ is used as an activation function instead of $sign(x)$, the corners of the pillars are less "sharp".



Note! $\lim_{\alpha\to\infty} tanh(\alpha x) = sign(x)$

3. (a) let $y_i = y_i(\mathbf{x})$, $E(t_i) = E(t_i|\mathbf{x})$. Because y_i is a deterministic function of $\mathbf{x}$, $E(y_i) = y_i$.

$$\begin{aligned}
\mathcal{E} &= E(\sum_{i=1}^{C}(y_i - t_i)^2) \\
&= E(\sum_{i=1}^{C}(y_i^2 - 2y_i t_i + t_i^2)) \\
&= \sum_{i=1}^{C} y_i^2 - 2y_i E(t_i) + E(t_i^2) + \underbrace{E^2(t_i) - E^2(t_i)}_{=0} \\
&= \sum_{i=1}^{C}(y_i - E(t_i))^2 + \underbrace{\sum_{i=1}^{C} var(t_i)}_{\text{Does not depend on } y_i}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \mathcal{E}}{\partial y_i} &= 2(y_i - E(t_i)) = 0 \\
&\Rightarrow \quad y_i = E(t_i) \\
&\Rightarrow \quad y_i = E(t_i|\mathbf{x})
\end{aligned}$$

(b) Let $t_i(w_j) = \begin{cases} 1, \text{ if } i = j \\ 0, \text{ otherwise} \end{cases}$.

$$y_i(\mathbf{x}) = E(t_i|\mathbf{x}) = \sum_{j=1}^{C} t_i(w_j)p(w_j|\mathbf{x}) = p(w_i|\mathbf{x})$$

(c) Now $t_i(w_j) = 1$ or 0 depending on the coding scheme

$$y_i(\mathbf{x}) = E(t_i|\mathbf{x}) = \sum_{j=1}^{C} t_i(w_j)p(w_j|\mathbf{x}) = p(t_i = 1|\mathbf{x})$$

T-61.3030 Principles of Neural Computing

Raivio, Koskela, Pöllä

Exercise 8 (Demonstration)

```
% Create data
Ndat=100;
x=[0:8/(Ndat-1):8];
a=0.1;

y=sin(x)./(x+a);

t=y+0.1*randn(size(y));

plot(x,y,'-b',x,t,'-r');

% divide data to a training set and testing set
rind=randperm(Ndat);

x_train=x(rind(1:Ndat/2));
t_train=t(rind(1:Ndat/2));

x_test=x(rind(Ndat/2+1:Ndat));
t_test=t(rind(Ndat/2+1:Ndat));

plot(x_train,t_train,'bo',x_test,t_test,'r+');


% create a new MLP (fead forward) network

% net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)
% PR - R x 2 matrix of min and max values for R input elements.
% Si - Size of ith layer, for Nl layers.
% TFi - Transfer function of ith layer, default = 'tansig'.
% BTF - Backpropagation network training function, default = 'traingdx'.
% BLF - Backpropagation weight/bias learning function, default = 'learngdm'.
% PF - Performance function, default = 'mse'.

%number of hidden neurons N

N=8;

net=newff([min(x_train),max(x_train)],[N 1],{'tansig' 'purelin'},'traingdm','learngd','mse');

net


%Initialize weights to a small value
% Hidden layer (input) weights
net.IW{1,1}
net.IW{1,1}=0.001*randn([N 1]);
net.IW{1,1}

% Output layer weights
```

```
net.LW{2,1}
net.LW{2,1}=0.001*randn([1 N]);
net.LW{2,1}

% Biases

net.b{1,1}=0.001*randn([N 1]);
net.b{2,1}=0.001*randn;

%train(NET,P,T,Pi,Ai,VV,TV) takes,
%     net - Neural Network.
%     P - Network inputs.
%     T - Network targets, default = zeros.
%     Pi - Initial input delay conditions, default = zeros.
%     Ai - Initial layer delay conditions, default = zeros.
%     VV - Structure of validation vectors, default = [].
%     TV - Structure of test vectors, default = [].


net.trainParam.epochs = 100000;
%net.trainParam.epochs = 10000;
net.trainParam.goal = 0.01;

[net,tr,Y,E]=train(net,x_train,t_train)
[xtr, tr_ind]=sort(x_train);
xts=sort(x_test);

%simulate the output of test set
Yts=sim(net,xts);

%plot the results for training set and test set
plot(x,y,'-k',xtr,Y(tr_ind),'-r',xts,Yts,'-g',x_train,t_train,'b+')
```

T-61.3030 Principles of Neural Computing

Raivio, Koskela, Pöllä

Exercise 9

1. One of the important matters to be considered in the design of a MLP network is its capability to generalize. Generalization means that the network does not give a good response only to the learning samples but also to more general samples. Good generalization can be obtained only if the number of the free parameters of the network is kept reasonable. As a thumb rule, the number of training samples should be at least five times the number of parameters. If there are less training samples than parameters, the network easily overlearns – it handles perfectly the training samples but gives arbitrary responses to all the other samples.

   A MLP is used for a classification task in which the samples are divided into five classes. The input vectors of the network consist of ten features and the size of the training set is 800. How many hidden units there can be at most according to the rule given above?

2. Consider the steepest descent method, $\Delta\mathbf{w}(n) = -\eta\mathbf{g}(n)$, reproduced in formula (4.120) and earlier in Chapter 3 (Haykin). How could you determine the learning-rate parameter η so that it minimizes the cost function $\mathcal{E}_{av}(\mathbf{w})$ as much as possible?

3. Suppose that we have in the interpolation problem described in Section 5.3 (Haykin) more observation points than RBF basis functions. Derive now the best approximative solution to the interpolation problem in the least-squares error sense.

T-61.3030 Principles of Neural Computing

Answers to Exercise 9

1. Let x be the number of hidden neurons. Each hidden neuron has 10 weights for the inputs and one bias term. Ech output neuron has one weight for each hidden neuron and one bias term. Thus the total number of weights in the network is:

   $$N = (10+1)x + (x+1)5 = 16x + 5$$

   Thumb rule:
   The number of training samples should be at least five times the number of parameters.

   $$5N \leq 800 \Rightarrow 5(16x+5) \leq 800 \Rightarrow 80x + 25 \leq 800 \Rightarrow x \leq 9.6875$$

   A reasonable number of hidden units is less than 10.

2. In the steepest descent method the adjustment $\Delta\mathbf{w}(n)$ applied to the parameter vector $\mathbf{w}(n)$ is defined by $\Delta\mathbf{w}(n) = -\eta\mathbf{g}(n)$, where η is the learning-rate parameter and

   $$\mathbf{g}(n) = \left.\frac{\partial\mathcal{E}_{av}(\mathbf{w})}{\partial\mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}(n)}$$

   is the local gradient vector of cost function $\mathcal{E}_{av}(\mathbf{w})$ averaged over the learning samples.

   The direction of the gradient $\mathbf{g}(n)$ defines a search line in multi-dimensional parameter space. Since we know, or can compute, $\mathcal{E}_{av}(\mathbf{w})$, we may optimize it on the one-dimensional search line. This is carried out by finding the optimal step size:

   $$\eta^* = \arg\min_{\eta} \mathcal{E}_{av}(\mathbf{w}(n) - \eta\mathbf{g}(n)).$$

   Depending on the nature of $\mathcal{E}_{av}(\mathbf{w})$ the above problem can be solved iteratively or analytically.

   Optimal η can be found, for example, as follows:

   Take first some suitable guess for η, say η_0. Then compute $\mathcal{E}_{av}(\mathbf{w}(n)-\eta_0\mathbf{g}(n))$. If this is greater than $\mathcal{E}_{av}(\mathbf{w}(n))$, take a new point η_1 from the search line between $\mathbf{w}(n)$ and $\mathbf{w}(n) - \eta_0\mathbf{g}(n)$. Otherwise, choose $\eta_1 > \eta_0$ to see if a larger value of η yields an even smaller value $\mathcal{E}_{av}(\mathbf{w}(n)-\eta_1\mathbf{g}(n))$. Continue this procedure until the optimal η, or a given precicion, is reached.

   The simplest way to implement such a iterative search would be to divide the search line into equal intervals so that

   $$\begin{aligned} \eta_1 &= \eta_0/2, \text{ if } \mathcal{E}_{av}(\mathbf{w}(n) - \eta_0\mathbf{g}(n)) > \mathcal{E}_{av}(\mathbf{w}(n)) \\ \eta_1 &= 2\eta_0, \text{otherwise} \end{aligned}$$

   when we have found the interval where the minimum lies, we can divide it into two parts and ge an better estimate of η^*.

   Dividing the search interval in two equal parts is not optimal with respect to the number of iterations needed to find the optimal learning rate parameter. A better way is to use Fibonacci numbers that can be used for optimal division of a search interval.

3. We can still write the interpolation conditions in the matrix form:

$$\mathbf{\Phi w} = \mathbf{d}, \tag{43}$$

where $\mathbf{\Phi}$ is a NxN matrix, $[\mathbf{\Phi}]_{ij} = \varphi(\|\mathbf{t}_j - \mathbf{x}_i\|)$. φ is a radial-basis function and $\mathbf{t}_j,\ j = 1, \ldots, M$, are the centers. The centers can be chosen, for example, with the k-means clustering algorithm (sec. 5.13, Haykin), and they are assumed to be given. $\mathbf{x}_i,\ i = 1, \ldots, N$, are the input vectors and $\mathbf{d} = (d_1, \ldots, d_N)^T$ is the vector consisting of the desired outputs. $\mathbf{w}$ is the weight vector.

Matrix $\mathbf{\Phi}$ has more rows than columns, $N > M$, and thus the equation (43) usually has no exact solution (More equations than unknowns!). However, we may look for the best least-square error solution.

Let $\mathbf{\Phi w} = \mathbf{d} + \mathbf{e}$, where $\mathbf{e}$ is the error vector. Now we want $\|\mathbf{e}\|^2 = \sum_{i=1}^{N} e_i^2 = \|\mathbf{\Phi w} - \mathbf{d}\|^2$ to be minimized.

$$\begin{aligned}
\frac{\partial \|\mathbf{e}\|^2}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}}(\mathbf{\Phi w} - \mathbf{d})^T(\mathbf{\Phi w} - \mathbf{d}) = 0 \\
&\Rightarrow \frac{\partial}{\partial \mathbf{w}}(\mathbf{w}^T\mathbf{\Phi}^T\mathbf{\Phi w} - \mathbf{w}^T\mathbf{\Phi}^T\mathbf{d} - \mathbf{d}^T\mathbf{\Phi w} + \mathbf{d}^T\mathbf{d}) = 0 \\
&\Rightarrow (\mathbf{\Phi}^T\mathbf{\Phi} + (\mathbf{\Phi}^T\mathbf{\Phi})^T)\mathbf{w} - \mathbf{\Phi}^T\mathbf{d} - (\mathbf{d}^T\mathbf{\Phi})^T = 0 \\
&\Rightarrow 2\mathbf{\Phi}^T\mathbf{\Phi w} - 2\mathbf{\Phi}^T\mathbf{d} = 0 \\
&\Rightarrow \mathbf{w} = \underbrace{(\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T}_{=\text{pseudoinverse of } \mathbf{\Phi}}\ \mathbf{d}
\end{aligned}$$

T-61.3030 Principles of Neural Computing

Raivio, Koskela, Pöllä

Exercise 10

1. The thin-plate-spline function is described by

$$\varphi(r) = \left(\frac{r}{\sigma}\right)^2 \log\left(\frac{r}{\sigma}\right) \quad \text{for some } \sigma > 0 \text{ and } r \in \mathbb{R}^+$$

   Justify the use of this function as a translationally and rotationally invariant Green's function. Plot the function graphically. (Haykin, Problem 5.1)

2. The set of values given in Section 5.8 for the weight vector $\mathbf{w}$ of the RBF network in Figure 5.6. presents one possible solution for the XOR problem. Solve the same problem by setting the centers of the radial-basis functions to

$$\mathbf{t}_1 = [-1, 1]^T \text{ and } \mathbf{t}_2 = [1, -1]^T.$$

   (Haykin, Problem 5.2)

3. Consider the cost functional

$$\mathcal{E}(F^*) = \sum_{i=1}^{N} \left[d_i - \sum_{j=1}^{m_1} w_j a(\|\mathbf{x}_j - \mathbf{t}_i\|) \right]^2 + \lambda \|\mathbf{D}F^*\|^2$$

   which refers to the approximating function

$$F^*(\mathbf{x}) = \sum_{i=1}^{m_1} w_i G(\|\mathbf{x} - \mathbf{t}_i\|).$$

   Show that the cost functional $\mathcal{E}(F^*)$ is minimized when

$$(\mathbf{G}^T\mathbf{G} + \lambda\mathbf{G}_0)\mathbf{w} = \mathbf{G}^T\mathbf{d}$$

   where the N-by-m_1 matrix $\mathbf{G}$, the m_1-by-m_1 matrix $\mathbf{G}_0$, the m_1-by-1 vector $\mathbf{w}$, and the N-by-1 vector $\mathbf{d}$ are defined by Equations (5.72), (5.75), (5.73), and (5.46), respectively. (Haykin, Problem 5.5)

4. Consider more closely the properties of the singular-value decomposition (SVD) discussed very briefly in Haykin, p. 300.

   (a) Express the matrix $\mathbf{G}$ in terms of its singular values and vectors.

   (b) Show that the pseudoinverse $\mathbf{G}^+$ of $\mathbf{G}$ can be computed from Equation (5.152):

$$\mathbf{G}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^T.$$

   (c) Show that the left and right singular vector $\mathbf{u}_i$ and $\mathbf{v}_j$ are obtained as eigenvectors of the matrices $\mathbf{G}\mathbf{G}^T$ and $\mathbf{G}^T\mathbf{G}$, respectively, and the squared singular values are the corresponding nonzero eigenvalues.

T-61.3030 Principles of Neural Computing

Answers to Exercise 10

1. Starting with the definition of the thin-plate function

$$\varphi(r) = \left(\frac{r}{\sigma}\right) log\left(\frac{r}{\sigma}\right), \text{ for } \sigma > 0 \text{ and } r \in \mathbb{R}^+$$

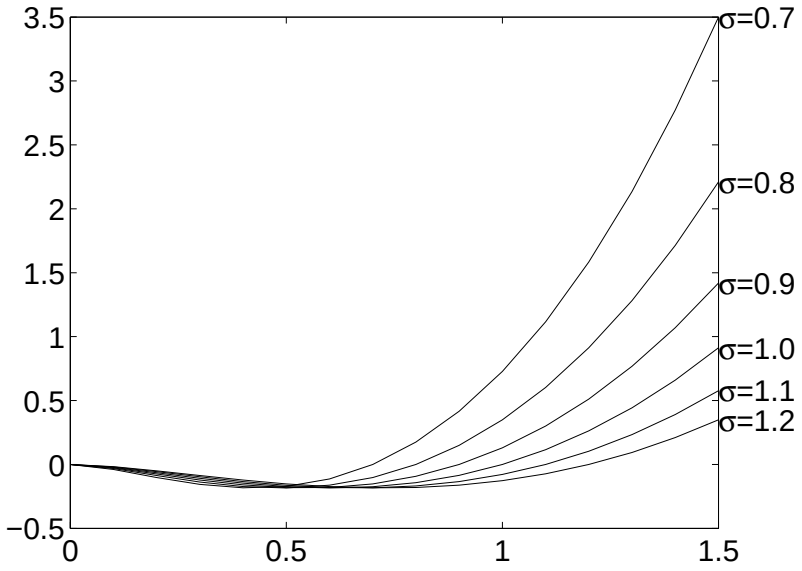
We may write

$$\varphi(\|\mathbf{x}; \mathbf{x}_i\|) = \left(\frac{\|\mathbf{x} - \mathbf{x}_i\|}{\sigma}\right) log\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|}{\sigma}\right).$$

Hence

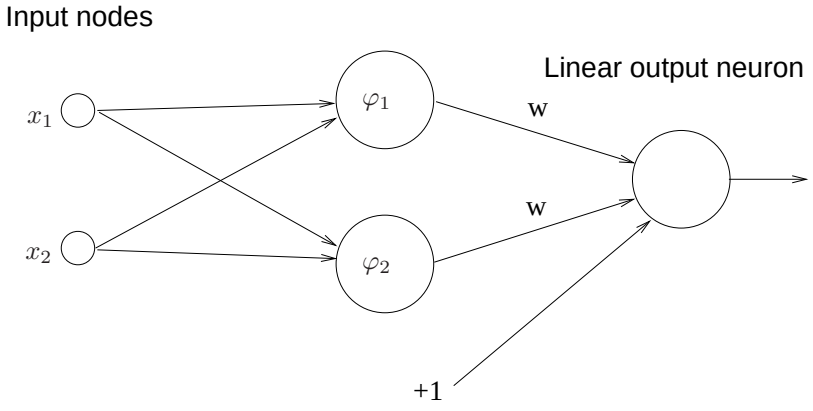
$$\varphi(\|\mathbf{x}; \mathbf{x}_i\|) = \varphi(\|\mathbf{x} - \mathbf{x}_i\|),$$

which means that $\varphi(\|\mathbf{x}; \mathbf{x}_i\|)$ is both translationally and rotationally invariant.



2. RBF network for solving the XOR problem

$$\varphi_i(\mathbf{x}) = G(\|\mathbf{x} - \mathbf{t}_i\|) = e^{-\|\mathbf{x}-\mathbf{t}_i\|^2}, \; i = 1, 2$$



Some notes on the above network:

- weight sharing is justified by the symmetry of the problem
- the desired output values of the problem have nonzero mean and thus the output unit includes a bias

Now the cneters of the radial-basis functions are

$$\begin{aligned}\mathbf{t}_1 &= (-1,1)^T \\ \mathbf{t}_2 &= (1,-1)^T.\end{aligned}$$

Let it be assuved that the logical symbol 0 is represented by level -1 and symbol 1 is represented by level +1.

(a) For the input pattern (0,0):
$\mathbf{x} = (-1,-1)^T$ and

$$\begin{aligned}(1)\ G(\|\mathbf{x}-\mathbf{t}_1\|) &= e^{-\|(-1,-1)^T-(-1,1)^T\|^2} \\ &= e^{-\|(0,-2)^T\|^2} = e^{-4} = 0.01832 \\ (2)\ G(\|\mathbf{x}-\mathbf{t}_2\|) &= e^{-\|(-1,-1)^T-(1,-1)^T\|^2} \\ &= e^{-\|(-2,0)^T\|^2} = e^{-4} = 0.01832\end{aligned}$$

(b) For the input pattern (0,1):
$\mathbf{x} = (-1,1)^T$ and

$$\begin{aligned}(3)\ G(\|\mathbf{x}-\mathbf{t}_1\|) &= e^{-\|(-1,1)^T-(-1,1)^T\|^2} \\ &= e^{-\|(0,0)^T\|^2} = e^{-0} = 1 \\ (4)\ G(\|\mathbf{x}-\mathbf{t}_2\|) &= e^{-\|(-1,1)^T-(1,-1)^T\|^2} \\ &= e^{-\|(-2,2)^T\|^2} = e^{-8} = 0.000336\end{aligned}$$

(c) For the input pattern (1,1):
$\mathbf{x} = (1,1)^T$ and

$$\begin{aligned}(5)\ G(\|\mathbf{x}-\mathbf{t}_1\|) &= e^{-\|(1,1)^T-(-1,1)^T\|^2} \\ &= e^{-\|(2,0)^T\|^2} = e^{-4} = 0.01832 \\ (6)\ G(\|\mathbf{x}-\mathbf{t}_2\|) &= e^{-\|(1,1)^T-(1,-1)^T\|^2} \\ &= e^{-\|(0,2)^T\|^2} = e^{-4} = 0.01832\end{aligned}$$

(d) For the input pattern (1,0):
$\mathbf{x} = (1,-1)^T$ and

$$\begin{aligned}(7)\ G(\|\mathbf{x}-\mathbf{t}_1\|) &= e^{-\|(1,-1)^T-(-1,1)^T\|^2} \\ &= e^{-\|(2,-2)^T\|^2} = e^{-8} = 0.000336 \\ (8)\ G(\|\mathbf{x}-\mathbf{t}_2\|) &= e^{-\|(1,-1)^T-(1,-1)^T\|^2} \\ &= e^{-\|(-0,0)^T\|^2} = e^{-0} = 1\end{aligned}$$

Results (1)-(8) are combinded as a matrix

$$\mathbf{G} = \begin{array}{c} \begin{array}{ccc} \varphi_1(\mathbf{x}) & \varphi_2(\mathbf{x}) & \text{bias} \end{array} \\ \begin{bmatrix} e^{-4} & e^{-4} & 1 \\ 1 & e^{-8} & 1 \\ e^{-4} & e^{-4} & 1 \\ e^{-8} & 1 & 1 \end{bmatrix} \end{array} \qquad \begin{array}{r} \text{pattern } (0,0) \\ (0,1) \\ (1,1) \\ (1,0) \end{array}$$

and the desired responses as vector $d = (-1, 1, -1, 1)^T$.

The linear parameters of the network are defined by

$$\mathbf{G}\mathbf{w} = \mathbf{d}, \text{ where } \mathbf{w} = (w,\ w,\ b)^T.$$

The solution to the equation is $\mathbf{w} = \mathbf{G}^\dagger \mathbf{d}$, where $\mathbf{G}^\dagger = (\mathbf{G}^T\mathbf{G})^{-1}\mathbf{G}^T$ is the pseudo inverse of $\mathbf{G}$

$$\mathbf{G}^\dagger = \begin{bmatrix} -0.5188 & 1.0190 & -0.5188 & 0.0187 \\ -0.5188 & 0.0187 & -0.5188 & 1.0190 \\ -0.5190 & -0.0190 & -0.5190 & -0.0190 \end{bmatrix}$$

$$\mathbf{w} = (2.0753,\ 2.0753,\ -1.076)^T$$

3. In equation (5.74) it is shown that $\|DF^*\|^2 = \mathbf{w}^T\mathbf{G}_0\mathbf{w}$ where $\mathbf{w}$ is the weight vector of the ouptu layer and $\mathbf{G}_0$ is defined by

$$\mathbf{G}_0 = \begin{bmatrix} G(\mathbf{t}_1;\mathbf{t}_1) & G(\mathbf{t}_1;\mathbf{t}_2) & \dots & G(\mathbf{t}_1;\mathbf{t}_n) \\ -G(\mathbf{t}_2;\mathbf{t}_1) & G(\mathbf{t}_2;\mathbf{t}_2) & \dots & G(\mathbf{t}_2;\mathbf{t}_n) \\ \vdots & \vdots & \ddots & \vdots \\ G(\mathbf{t}_n;\mathbf{t}_1) & G(\mathbf{t}_n;\mathbf{t}_2) & \dots & G(\mathbf{t}_n;\mathbf{t}_n) \end{bmatrix}.$$

We may thus express the cost functional as

$$\mathcal{E}(F^*) = \sum_{i=1}^{N}[d_i - \sum_{j=1}^{m_1} w_j G(\|\mathbf{x}_i - \mathbf{t}_j\|)]^2 + \lambda \mathbf{w}^T\mathbf{G}_0\mathbf{w} = \underbrace{\|\mathbf{d} - \mathbf{G}\mathbf{w}\|^2}_{\mathcal{E}_{fs}(\mathbf{w})} + \lambda \underbrace{\mathbf{w}^T\mathbf{G}_0\mathbf{w}}_{\mathcal{E}_{fc}(\mathbf{w})},$$

where $\mathbf{d} = (d_1,\ d_2\ \dots,\ d_N)^T$ and $[\mathbf{G}]_{ij} = G(\|\mathbf{x}_i - \mathbf{t}_j\|)$. Finding the function F^* that minimizes the cost functional $\mathcal{E}(F^*)$ is equivalent to finding the weight vector $\mathbf{w} \in \mathbb{R}^{m_1}$ that minimizes the function $\mathcal{E}_f(\mathbf{w}) = \mathcal{E}_{fs}(\mathbf{w}) + \lambda\mathcal{E}_{fc}(\mathbf{w})$. A necessary condition for an extremum at $\mathbf{w}$ is that

$$\begin{aligned} \nabla_{\mathbf{w}}\mathcal{E}_f(\mathbf{w}) &= 0 \\ \nabla_{\mathbf{w}}\mathcal{E}_{fs}(\mathbf{w}) &= -2\mathbf{G}^T(\mathbf{d} - \mathbf{G}\mathbf{w}) \\ \nabla_{\mathbf{w}}\mathcal{E}_{fc}(\mathbf{w}) &= 2\mathbf{G}_0\mathbf{w} \end{aligned}$$

$$\begin{aligned} &\Rightarrow \quad \mathbf{G}^T(\mathbf{d} - \mathbf{G}\mathbf{w}) + \lambda\mathbf{G}_0\mathbf{w} = 0 \\ &\Rightarrow \quad \mathbf{w} = (\mathbf{G}^T\mathbf{G} + \lambda\mathbf{G}_0)^{-1}\mathbf{G}^T\mathbf{d}. \end{aligned}$$

4. (a) Singular value decomposition of $\mathbf{G}$

$$\underset{L\times L}{\mathbf{U}^T}\ \underset{L\times M}{\mathbf{G}}\ \underset{M\times M}{\mathbf{V}} = \mathbf{\Sigma} \Rightarrow$$

$$\underset{L\times L}{\mathbf{U}^T\mathbf{U}}\ \underset{L\times M}{\mathbf{G}}\ \underset{M\times M}{\mathbf{V}\mathbf{V}^T} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T,$$

$$\mathbf{U}^T\mathbf{U} = \mathbf{I},\quad \mathbf{V}\mathbf{V}^T = \mathbf{I}$$

or in other words, $\mathbf{U}$ and $\mathbf{V}$ are othogonal matrices. Hence $\mathbf{G} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sum_{i=1}^{w}\sigma_i\mathbf{u}_i\mathbf{v}^T$, where w is the rank of $\mathbf{G}$. Note that $\underset{L\times M}{\Sigma} = \begin{bmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$, $\mathbf{S} = \text{diag}(\sigma_1,\ \dots,\ \sigma_2)$.

(b)

$$\begin{aligned}
\mathbf{G}^\dagger &= (\mathbf{G}^T\mathbf{G})^{-1}\mathbf{G}^T \\
&= (\mathbf{V}\mathbf{\Sigma}^T \underbrace{\mathbf{U}^T\mathbf{U}}_{=\mathbf{I}} \mathbf{\Sigma}\mathbf{V}^T)^{-1}\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T \\
&= (\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T)^{-1}\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T \\
&= \mathbf{V}^{-T}\mathbf{\Sigma}^{-2}\underbrace{\mathbf{V}^{-1}\mathbf{V}}_{=\mathbf{I}}\mathbf{\Sigma}^T\mathbf{U}^T \\
&= \mathbf{V}\mathbf{\Sigma}^{-2}\mathbf{\Sigma}^T\mathbf{U}^T = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T = \mathbf{V}\mathbf{\Sigma}^\dagger\mathbf{U}^T
\end{aligned}$$

Note that here $\mathbf{\Sigma}^{-1} = \mathbf{\Sigma}^\dagger = \text{diag}(1/\sigma_1, \ldots, 1/\sigma_w, 0, \ldots, 0)$ and not the usual inverse of matrix. The above works only because $\mathbf{\Sigma}$ is a diagonal matrix.

(c)

$$\begin{aligned}
\mathbf{G}\mathbf{G}^T &= \mathbf{U}\mathbf{\Sigma}\underbrace{\mathbf{V}^T\mathbf{V}}_{=\mathbf{I}}\mathbf{\Sigma}^T\mathbf{U} = \mathbf{U}\underbrace{\mathbf{\Sigma}\mathbf{\Sigma}^T}_{=\text{diag}(\sigma_1^2, \ldots, \sigma_w^2)}\mathbf{U}^T \\
\mathbf{G}\mathbf{G}^T\mathbf{U} &= \mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^T
\end{aligned}$$

$$\begin{aligned}
\mathbf{G}^T\mathbf{G} &= \mathbf{V}\mathbf{\Sigma}^T\underbrace{\mathbf{U}\mathbf{U}^T}_{=\mathbf{I}}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\underbrace{\mathbf{\Sigma}^T\mathbf{\Sigma}}_{=\text{diag}(\sigma_1^2, \ldots, \sigma_w^2, 0, \ldots, 0)}\mathbf{V}^T \\
\mathbf{G}^T\mathbf{G}\mathbf{V} &= \mathbf{V}\mathbf{\Sigma}^T\mathbf{\Sigma}
\end{aligned}$$

T-61.3030 Principles of Neural Computing

Raivio, Koskela, Pöllä

Exercise 11

1. The weights of the neurons of a Self-organizing map (SOM) are updated according to the following learning rule:

   $$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i(\mathbf{x})}(n)(\mathbf{x} - \mathbf{w}_j(n)),$$

   where j is the index of the neuron to be updated, $\eta(n)$ is the learning-rate parameter, $h_{j,i(\mathbf{x})}$ is the neighborhood function, and $i(\mathbf{x})$ is the index of the winning neuron for the given input vector $\mathbf{x}$. Consider an example where scalar values are inputted to a SOM consisting of three neurons. The initial values of the weights are

   $$w_1(0) = 0.5,\ w_2(0) = 1.5,\ w_3(0) = 2.5$$

   and the inputs are randomly selected from the set:

   $$X = \{0.5, 1.5, 2.0, 2.5, 2.75, 3.0, 3.25, 3.5, 3.75, 4.0, 4.25, 4.5\}.$$

   The Kronecker delta function is used as a neighborhood function. The learning-rate parameter has a constant value 0.02. Calculate a few iteration steps with the SOM learning algorithm. Do the weights converge? Assume that some of the initial weight values are so far from the input values that they are never updated. How such a situation could be avoided?

2. Consider a situation in which scalar inputs of a one-dimensional SOM are distibuted according to the probability distribution function $p(x)$. A stationary state of the SOM is reached when the expected changes in the weight values become zero:

   $$E[h_{j,i(x)}(x - w_j)] = 0$$

   What are the stationary weight values in the following cases:

   (a) $h_{j,i(x)}$ is a constant for all j and $i(x)$, and

   (b) $h_{j,i(x)}$ is the Kronecker delta function?

3. Assume that the input and weight vectors of a SOM consisting of $N \times N$ units are d-dimensional and they are compared by using Euclidean metric. How many multiplication and adding operations are required for finding the winning neuron. Calculate also how many operations are required in the updating phase as a function of the width parameter σ of the neighborhood. Assume then that of $N = 15$, $d = 64$, and $\sigma = 3$. Is it computationally more demanding to find the winning neuron or update the weights?

4. The function $g(y_j)$ denotes a nonlinear function of the response y_j, which is used in the SOM algorithm as described in Equation (9.9):

   $$\Delta\mathbf{w}_j = \eta y_j \mathbf{x} - g(y_j)\mathbf{w}_j.$$

   Discuss the implications of what could happen if the constant term in the Taylor series of $g(y_j)$ is nonzero. (Haykin, Problem 9.1)

T-61.3030 Principles of Neural Computing

Answers to Exercise 11

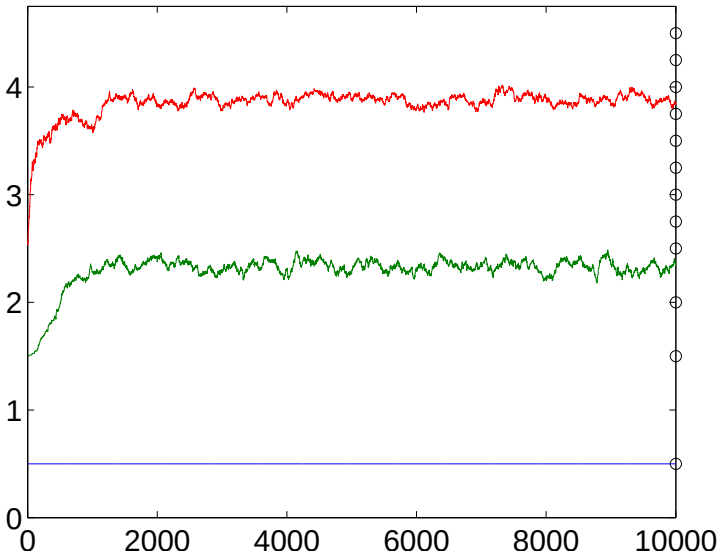
1. The updating rule:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i(\mathbf{x})}(n)(\mathbf{x}(n) - \mathbf{w}_j(n))$$

The inital values of the weights: $w_1(0) = 0.5,\ w_2(0) = 1.5,\ w_3(0) = 2.5$ and$h_{j,i(\mathbf{x})}(n) = \begin{cases} 1, \text{ if } j = i(\mathbf{x}(n)) \\ 0, \text{ otherwise} \end{cases}$, $\eta(n) = 0.02$.

$n+1$	$\mathbf{x}(n)$	$i(\mathbf{x}(n))$	$(\mathbf{x}(n) - \mathbf{w}_{i(\mathbf{x}(n))}\eta(n)$	$\mathbf{w}_{i(\mathbf{x}(n))}(n+1)$
1	2.5	3	(2.5-2.5)0.002=0	2.5
2	4.5	3	(4.5-2.5)0.002=0.04	2.54
3	2.0	2	(2.0-1.5)0.002=0.01	1.61
4	0.5	1	(0.5-0.5)0.002=0	0.5
5	3.25	3	(3.25-2.54)0.002=0.0142	2.5542
6	3.75	3	(3.75-2.5542)0.002=0.02392	2.578116
7	2.75	3	(2.75-2.578116)0.002=0.00344	2.58155368
8	4.0	3	(4.0-2.58155368)0.002=0.002837	2.609923
9	4.25	3	(4.25-2.609923)0.002=0.0328	2.642742
10	1.5	2	(1.5-1.61)0.002=-0.0022	1.6078
11	3.0	3	(3.0-2.642742)0.002=0.007	2.64987
12	3.5	3	(3.5-2.64987)0.002=0.017	2.666872

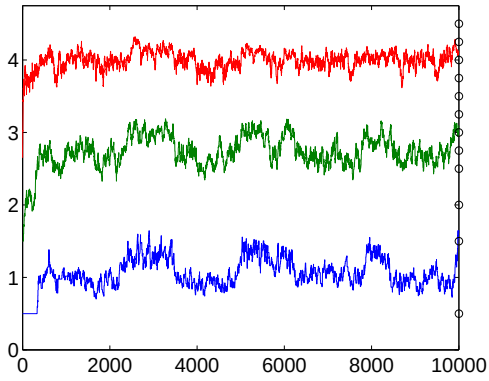
Matlab gives the following result:



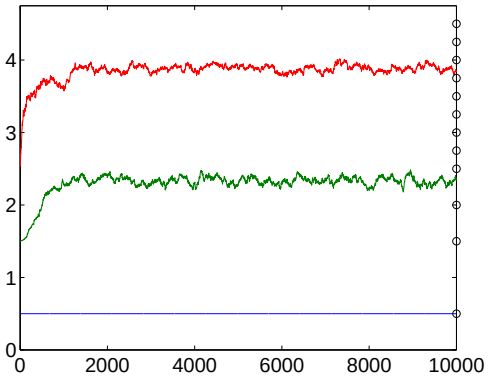
Situations in which some of the neurons are not updated can be avoided if the range of the inputs is examined and the inital values are set according to it. Also, a different neighborhood definition might help: if a single input have an effect on more than one neuron, it is not so probable that some of the neurons stay still.

```
function cbt=train_csom(cb,data,alpha,max_rounds)
%data=[.5,1.5,2.0,2.5,2.75,3.0,
3.25,3.5,3.75,4.0,4.25,4.5];
%cb=[.5,1.5,2.5];
%alpha=.02;
N=length(data);
M=length(cb);
cbt=zeros(max_rounds,M);
for t=1:max_rounds,
  i=ceil(rand(1)*N);
  d=abs(ones(1,M)*data(i)-cb);
  [f,wi]=min(d);
  cb(wi)=cb(wi)+alpha*(data(i)-cb(wi));
  cbt(t,:)=cb;
end
```

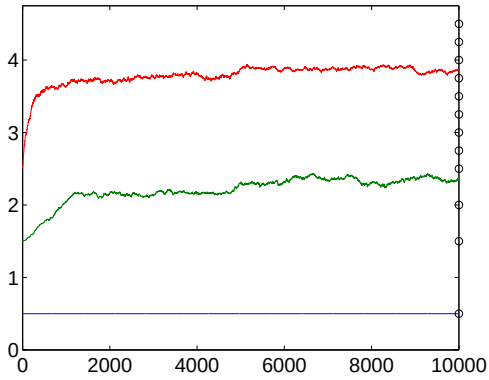
alpha=.10



alpha=.02

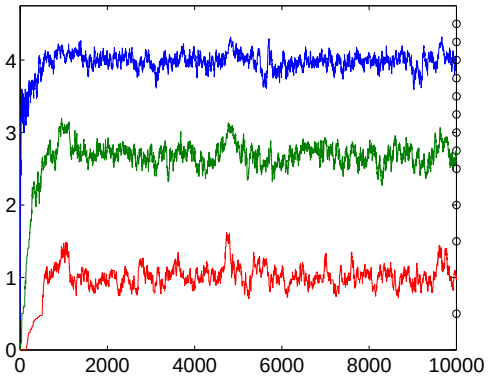


alpha=.01

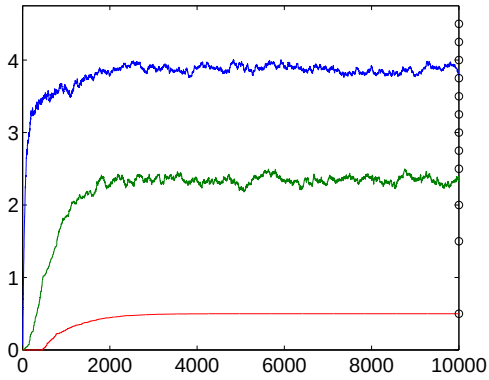


---

cb=[0,0,0]
alpha=.1



alpha=.02



2. (a)

$$\begin{aligned} E[\underbrace{h_{j,i(x)}}_{=constant}(x - wj)] &= 0 \\ \Rightarrow E[(x - w_j)] &= \int_{\mathcal{X}} (x - w_j)p(x)dx = 0 \\ \Rightarrow w_j &= \int_{\mathcal{X}} xp(x)dx = E[x] \end{aligned}$$

This means that all the weights converge to the same point.

(b)

$$\begin{aligned} E[\underbrace{h_{j,i(x)}}_{=\delta(j,i(x))}(x - wj)] &= 0 \\ \Rightarrow E[(x - w_j)] &= \int_{\mathcal{X}} \delta(j, i(x))(x - w_j)p(x)dx = 0 \end{aligned}$$

Let $\mathcal{X} = \bigcup_j \mathcal{X}_j$, where $\mathcal{X}_j$ is the part of the input space where w_j is the nearest weight. (Voronoi region of w_j).

$$\begin{aligned} &\Rightarrow \sum_j \int_{\mathcal{X}_j} (x - w_j)p(x)dx = 0 \\ &\Rightarrow \int_{\mathcal{X}_j} (x - w_j)p(x)dx = 0 \\ &\Rightarrow w_j = \frac{\int_{\mathcal{X}_j} xp(x)dx}{\int_{\mathcal{X}_j} p(x)dx} \end{aligned}$$

3. For finding the winning neuron $i(\mathbf{x})$, the squared Euclidean distance to the input vector $\mathbf{x}$ has to be calculated for each of the neurons:

$$\|\mathbf{x} - \mathbf{w}_j\|^2 = \sum_{i=1}^{d} (x_i - w_{ji})^2.$$

This involves d multiplication operations and $2d-1$ adding operations per a neuron. Thus, in total, for finding the winner neuron, dN^2 multiplication operations and $(2d-1)N^2$ adding operations are required.

Next, the definition of the topological neighborhood $h_{j,i(\mathbf{x})}$:

If all the neurons lie inside a hypercube ("radius"=σ) around the winning neuron are updated for input $\mathbf{x}$, $(2\sigma+1)^2$ neurons are updated at the same time. This is the so called bubble neighborhood.

An other choise for the topological neighborhood definition is to use a Gaussian function:

$$h_{j,i(\mathbf{x})} = e^{-d_{j,i(\mathbf{x})}/2\sigma^2},$$

where $d_{j,i(\mathbf{x})}$ is the lateral distance between neurons j and $i(\mathbf{x})$. In this case, all the neurons are updated for every input $\mathbf{x}$.

For findin out which of the neurons belong to the buble neighborhood of the winning neuron ($d_{jh,i(\mathbf{x})} = \|\mathbf{n}_j - \mathbf{n}_{i(\mathbf{x})}\|^2 \leq \sigma^2$, $2N^2$ multiplication and $3N^2$ adding operations are required as the dimension of the map is 2.

Calculating the gaussian function fro all the neurons involves $3N^2$ multiplication and adding operations.

The weights are updated according to the following rule:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i(\mathbf{x})}(\mathbf{x} - \mathbf{w}_j(n)),$$

which involves $d+1$ multiplications in the case of Gaussian neighborhood and d multiplication operations in the case of bubble neighborhood, and $2d$ adding operations in both cases.

($N = 15$, $d = 64$, $\sigma = 3$)

	multiplications	additions
winner search	$dN^2 = 14400$	$(2d-1)N^2 = 28575$
updating (bubble)	$2N^2 + d(2\sigma+1)^2 = 3586$	$3N^2 + 2d(2\sigma+1)^2 = 6722$
updating (Gaussian)	$(d+4)N^2 = 15300$	$(2d+3)N^2 = 29475$

4. Expanding the function $g(y_j)$ in a Tayler series around $y_j = 0$, we get

$$g(y_j) = g(0) + g^{(1)}(0) y_j + \frac{1}{2} g^{(2)}(0) y_j^2 + \ldots, \tag{44}$$

where $g^{(k)}(0) = \left. \frac{\partial^k g(y_j)}{\partial y_j^k} \right|_{y_j=0}$, $k = 1, 2, \ldots$.

Let $y_j = \begin{cases} 1, \text{ neuron } j \text{ is on} \\ 0, \text{ neuron } j \text{ is off} \end{cases}$. Then , we may rewrite equation 44 as

$$g(y_j) = \begin{cases} g(0) + g^{(1)}(0) + \frac{1}{2} g^{(2)}(0) + \ldots, \text{ neuron } j \text{ is on} \\ g(0), \text{ neuron } j \text{ is off} \end{cases} .$$

Correspondingly, we may write equation (9.9)

$$\Delta \mathbf{w}_j = \eta y_j \mathbf{x} - g(y_j)\mathbf{x}_j = \begin{cases} \eta \mathbf{x} - \mathbf{w}_j [g(0) + g^{(1)}(0) + \frac{1}{2} g^{(2)}(0) + \ldots], \text{ neuron } j \text{ is on} \\ -g(0)\mathbf{w}_j, \text{neuron } j \text{ is off} \end{cases} .$$

Consequently, a nonzero $g(0)$ has the effect of making $\Delta \mathbf{w}_j$ assume a nonzero value when neuron j is off, thereby violating the desired form described in equation (9.13):

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i(\mathbf{x})}(n)(\mathbf{x} - \mathbf{w}_j(n)).$$

To achieve this desired from, we have to make $g(0) = 0$.

T-61.3030 Principles of Neural Computing

Raivio, Koskela, Pöllä

Exercise 12

1. It is sometimes said that the SOM algorithm preserves the topological relationships that exist in the input space. Strictly speaking, this property can be guaranteed only for an input space of equal or lower dimensionality than that of the neural lattice. Discuss the validity of this statement. (Haykin, Problem 9.3)

2. It is said that the SOM algorithm based on competitive learning lacks any tolerance against hardware failure, yet the algorithm is error tolerant in that a small perturbation applied to the input vector causes the output to jump from the winning neuron to a neighboring one. Discuss the implications of these two statements. (Haykin, Problem 9.4)

3. In this problem we consider the optimized form of the learning vector quantization algorithm (see Section 9.7, Haykin) developed by Kohonen. We wish to arrange for the effects of the corrections to the Voronoi vectors, made at different times, to have equal influence when referring to the end of the learning period.

   (a) First, show that Equation (9.30)

   $$\mathbf{w}_c(n+1) = \mathbf{w}_c(n) + \alpha_n[\mathbf{x}_i - \mathbf{w}_c(n)]$$

   and Equation (9.31)

   $$\mathbf{w}_c(n+1) = \mathbf{w}_c(n) - \alpha_n[\mathbf{x}_i - \mathbf{w}_c(n)]$$

   may be integrated into a single equation, as follows:

   $$\mathbf{w}_c(n+1) = (1 - s_n\alpha_n)\mathbf{w}_c(n) + s_n\alpha_n\mathbf{x}(n).$$

   In the above equations, $\mathbf{w}_c$ is the Voronoi vector closest to the input vector $\mathbf{x}_i$, $0 < \alpha_n < 1$ is a learning constant, and s_n is a sign function depending on the classification result of the nth input vector $\mathbf{x}(n)$: $s_n = +1$ if classification is correct, otherwise $s_n = -1$.

   (b) Hence, show that the optimization criterion described at the beginning of the problem is satisfied if

   $$\alpha_n = (1 - s_n\alpha_n)\alpha_{n-1}$$

   which yields the optimized value of the learning constant α_n as

   $$\alpha_n^{\text{opt}} = \frac{\alpha_{n-1}^{\text{opt}}}{1 + s_n\alpha_{n-1}^{\text{opt}}}.$$

   (Haykin, Problem 9.6)

4. The following algorithm introduced by J. Friedman can be used for speeding up winner search in the SOM algorithm: 1) Evaluate the Euclidean distance between the input vector and weight vector whose projections on some coordinate axis are least apart, 2) Examine the weight vectors in the increasing order of their projected distances to the input vector. Continue this until a weight vector whose projected distance to the input vector is greater than the smallest Euclidean distance calculated so far is found. At this point of the algorithm, the winning neuron has been found.

   Apply the algorithm described above for the problem illustrated in Figure 13.

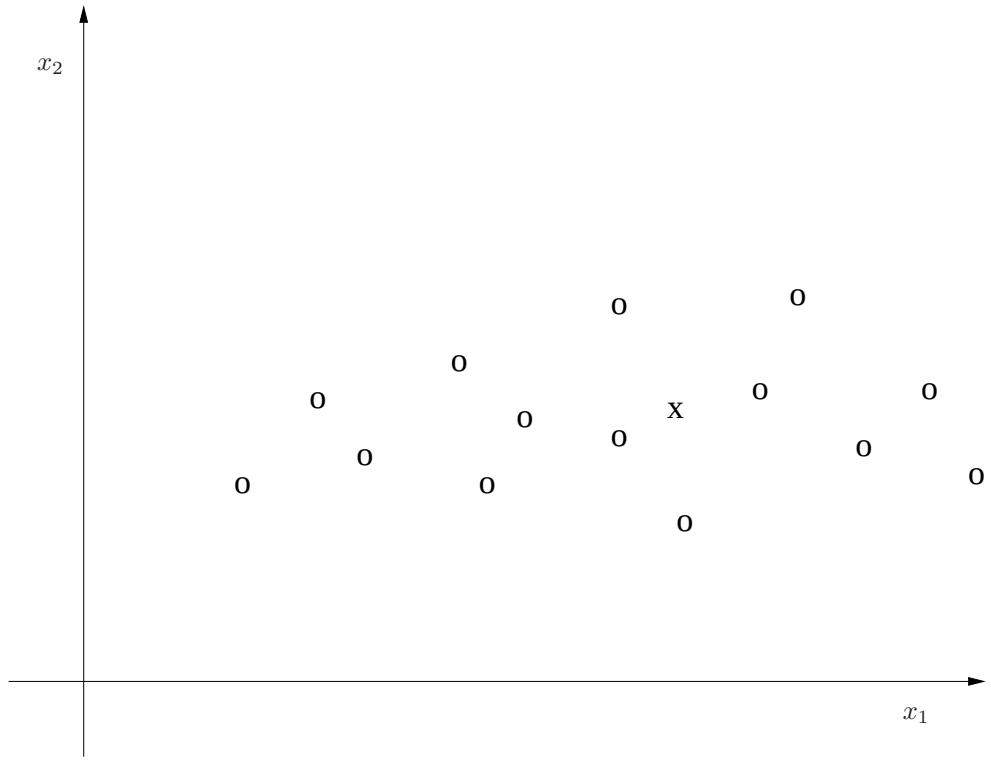


Figure 13: The weight vectors (o) among which the nearest one to the input vector (x) has to be found.

(a) Find the winning neuron when the weight vectors and the input vector are projected onto x_1-axis. Show that the weight vector found by the algorithm is indeed the winning one. How many Euclidean distances one must evaluate for finding it?

(b) Repeat the search but project the vectors onto x_2-axis this time.

(c) Which one of the two searches was the fastest? Are there some general rules on how the projection axis should be chosen?

T-61.3030 Principles of Neural Computing

Answers to Exercise 12

1. Consider the "peano curve" shown in figure 14**c**. The self-organizing map has a one-dimensional lattice that is taught with two-dimensional data. We can see that the data points inside the circle map to either unit A or unit B on the map. Some of the data points are mapped very far from each other on the lattice of the som even though they are close by in the input space. Thus the topological relationships are not preserved by the SOM. This is true in general for any mehtod that does dimensinoality reduction. It is not possible to preserve all relationships in the data.

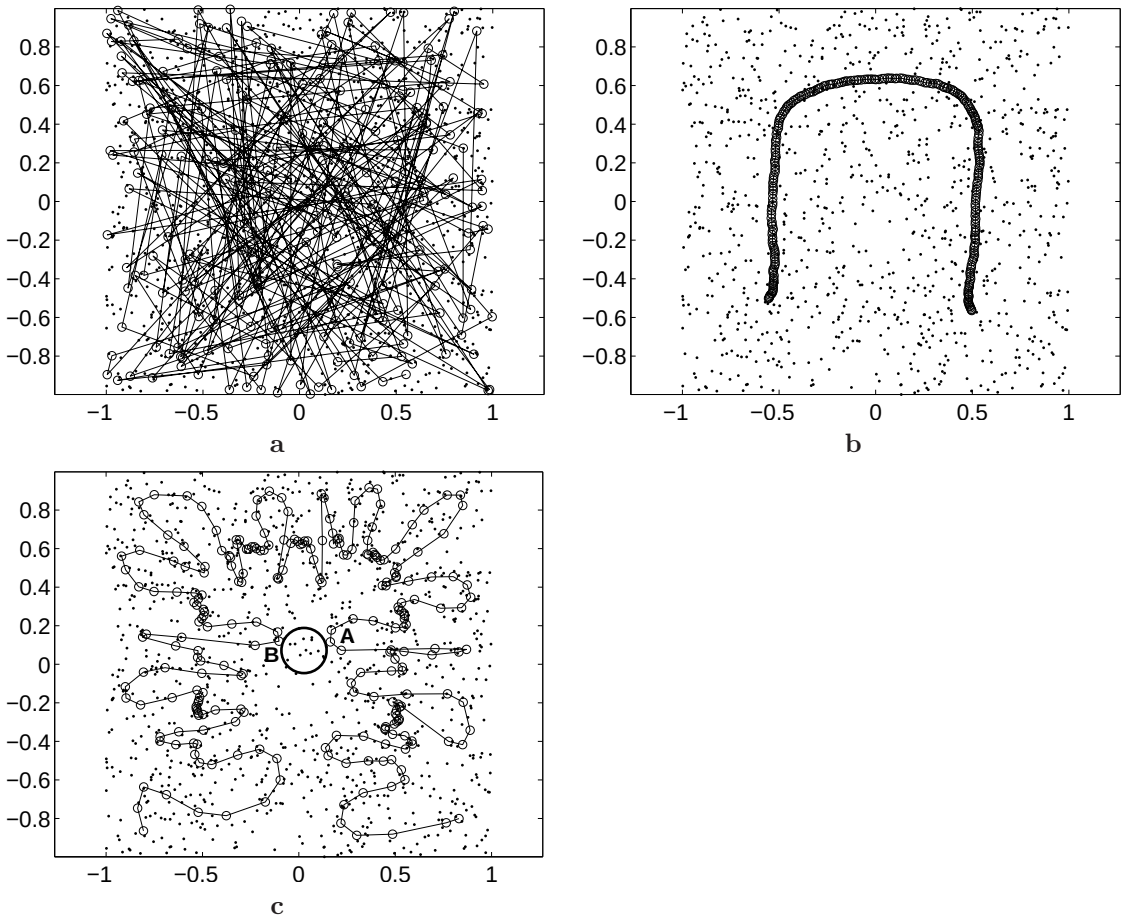


Figure 14: An example of a one-dimensinal SOM with two-dimensional input data. **a**) Random initializatin. **b**) After the ordering phase. **c**) End of the convergence space. The circle marks one area where the SOM mapping violates the topology of the input space.

2. Consider for ecample a two-dimensional lattice using the SOM algorithm to learn a two-dimensional input distribution as in Figure 15**a** and **b**. Suppose that two neurons break down by getting stuck at the initial position. The effect is illustrated in Figure 15**c**. The organization of the SOM is distrubed. This is evident by the bending of the lattice around the broken units. If the unit breaks completely, i.e. it can not even be selected as a best match the effect is not as dramatic (Figure 15**d**). This type of break down effectively creates holes in the lattice of the SOM, but the effect on the overall organization is not as large. On the other hand small perturbations to the data samples leaves the map lattice essentially unchanged.

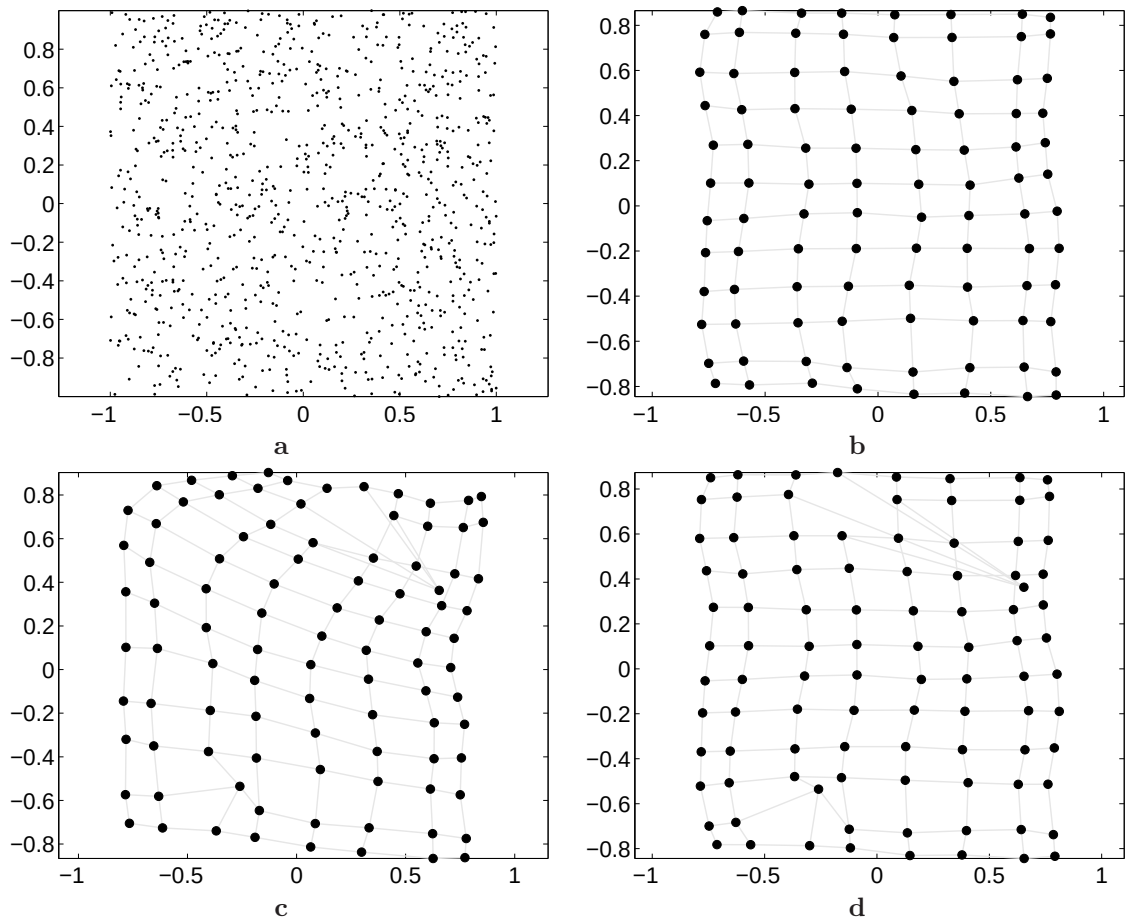


Figure 15: An example the effects caused by hardware erros on a two-dimensinal SOM with two-dimensional input data. **a**) Input data. **b**) SOM with no hardware problems **c**) Two units are stuck to their initial positions. **d**) The same two units are completely dead (they are not updated and they can't be best matches). The same initialization was used in all cases.

3. (a) If the classification is correct, $s_n = +1$, the update equation

$$\mathbf{w}_c(n+1) = (1 - s_n\alpha_n)\mathbf{w}_c(n) + s_n\alpha_n\mathbf{x}(n) \tag{45}$$

reduces to

$$\mathbf{w}_c(n+1) = (1 - \alpha_n)\mathbf{w}_c(n) + \alpha_n\mathbf{x}(n) = \mathbf{w}_c(n) + \alpha_n(\mathbf{x}(n) - \mathbf{w}_c(n)),$$

whitch is the same as the update rule for correct classification.
Next, we note that if $s_n = -1$, then equation 45 reduces to

$$\mathbf{w}_c(n+1) = (1 + \alpha_n)\mathbf{w}_c(n) - \alpha_n\mathbf{x}(n) = \mathbf{w}_c(n) - \alpha_n(\mathbf{x}(n) - \mathbf{w}_c(n)),$$

which is identical to the update rule for incorrect classification.

(b) For equation 45 we note that the updated weight vector $\mathbf{w}_c(n+1)$ contains a "trace" from $\mathbf{x}(\mathbf{n})$ by virtue of the last term $s_n\alpha_n\mathbf{x}(n)$. More over, it contains traces of previous samples, namely, $\mathbf{x}(n-1)$, $\mathbf{x}(n-2)$, ..., $\mathbf{x}(1)$ by virtue of the present value of the weight vector $\mathbf{w}_c(n)$. Consider, for exmaple, $\mathbf{w}_c(n)$ which (according to equation 45) is defined by

$$\mathbf{w}_c(n) = (1 - s_{n-1}\alpha_{n-1})\mathbf{w}_c(n-1) + s_{n-1}\alpha_{n-1}\mathbf{x}(n-1) \tag{46}$$

Hence, substituting equation 46 into 45 and combining terms:

$$\mathbf{w}_c(n+1) = (1-s_n\alpha_n)(1-s_{n-1}\alpha_{n-1})\mathbf{w}_c(n-1)+(1-s_n\alpha_n)s_{n-1}\alpha_{n-1}\mathbf{x}(n-1)+s_n\alpha_n\mathbf{x}(n). \quad (47)$$

It follows therefore that the effect of $\mathbf{x}(n-1)$ on the updated weight vector $\mathbf{w}_c(n+1)$ is scaled by the factor $(1-s_n\alpha_n)\alpha_{n-1}$. On the other hand, the effect of $\mathbf{x}(n)$ is scaled by $s_n\alpha_n$. Accordingly, if we require that $\mathbf{x}(n)$ and $\mathbf{x}(n-1)$ are to have the same effect on $\mathbf{w}_c(n+1)$, then we must have

$$\alpha_n = (1 - s_n\alpha_n)\alpha_{n-1}.$$

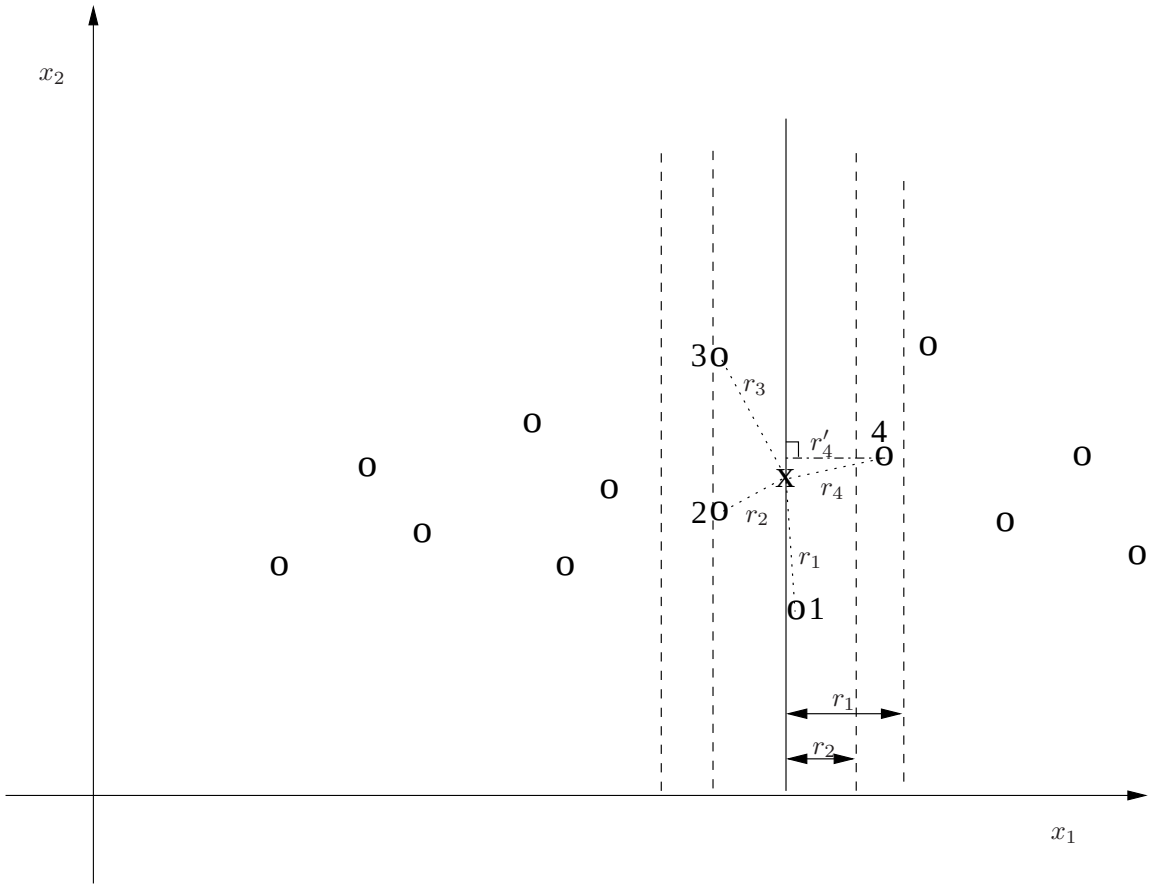
Solving for α_n, we get the optimum

$$\alpha_n^{opt} = \frac{\alpha_{n-1}^{opt}}{1 + s_n\alpha_{n-1}^{opt}}.$$

4. (a) The Friedman's algorithm is based on the following fact:

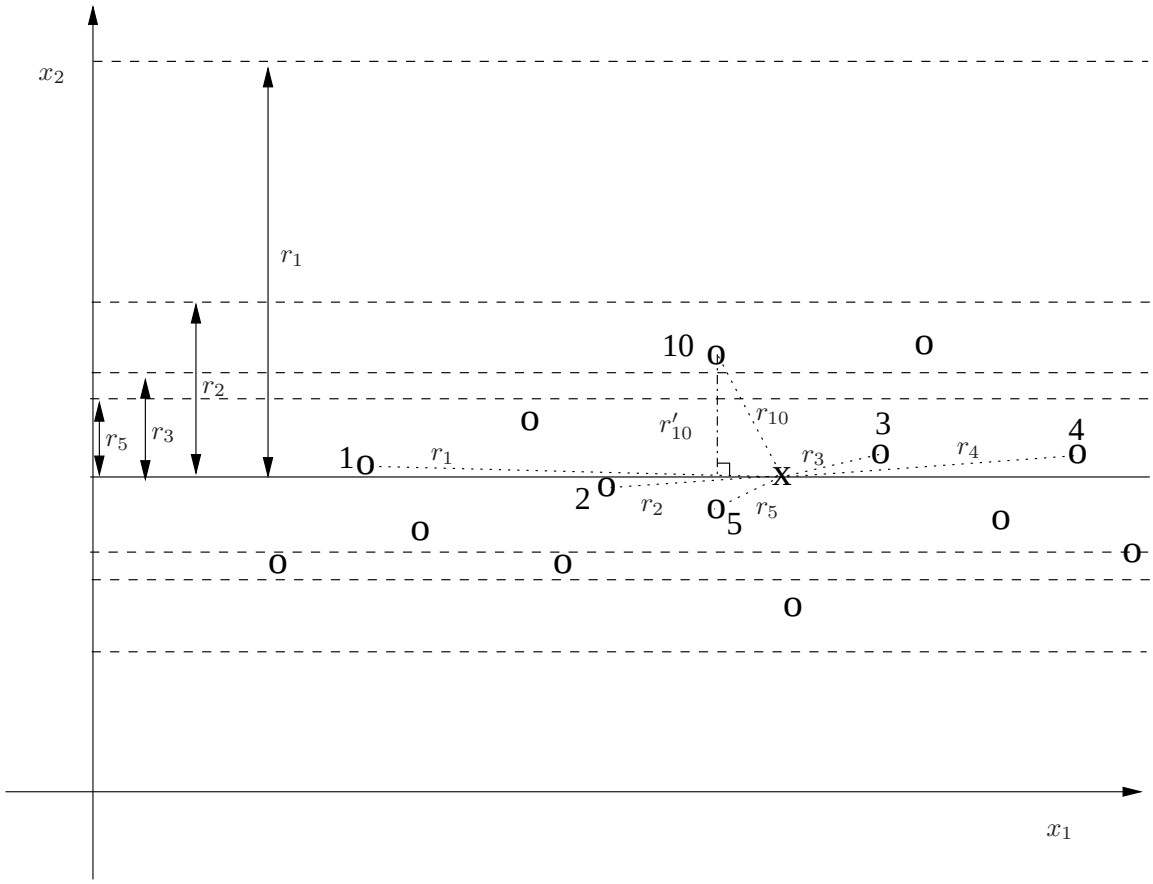
$$d(\mathbf{w}, \mathbf{x}) = \left[\sum_{i=1}^{d}(w_i - x_i)^2\right]^{-\frac{1}{2}} \geq [(w_k - x_k)^2]^{-\frac{1}{2}} = d(w_k, x_k),\ 1 \leq k \leq d$$

Friedman's algorithm on using the x_1 axis:



$r_4' > r_2 \Rightarrow 3$ Euclidean distances must be evaluated before the winning neuron is found.

(b) Now apply Friedman's algorithm on using the x_2 axis:



$r'_1 0 > r_5 \Rightarrow 9$ Euclidean distances must be evaluated before the winning neuron is found.

(c) The first search was faster as only 3 Euclidean distances had to be evaluated, which is considerably less than the 9 evaluations in the second case. As a general rule, the projection axis should be chosen so that the variance of the projected distances is maximized. The variance can be approximated by the maximum distance between the projected points. This way, the projected points are located more sparcely on the axis.