NEURAL NETWORKS

Ivan F Wilde

Mathematics Department

King's College London

London, WC2R 2LS, UK

ivan.wilde@kcl.ac.uk

Contents

1 Matrix Memory 1

2 Adaptive Linear Combiner 21

3 Artificial Neural Networks 35

4 The Perceptron 45

5 Multilayer Feedforward Networks 75

6 Radial Basis Functions 95

7 Recurrent Neural Networks 103

8 Singular Value Decomposition 115

Bibliography 121

Chapter 1

Matrix Memory

We wish to construct a system which possesses so-called associative memory. This is definable generally as a process by which an input, considered as a "key", to a memory system is able to evoke, in a highly selective fashion, a specific response associated with that key, at the system output. The signal-response association should be "robust", that is, a "noisy" or "incomplete" input signal should none the less invoke the correct response—or at least an acceptable response. Such a system is also called a content addressable memory.

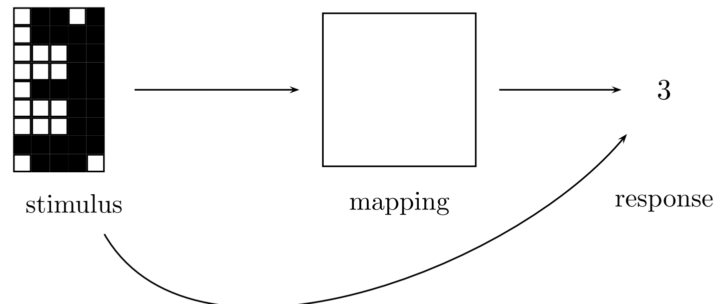


Figure 1.1: A content addressable memory.

The idea is that the association should not be defined so much between the individual stimulus-response pairs, but rather embodied as a whole collection of such input-output patterns—the system is a distributive associative memory (the input-output pairs are "distributed" throughout the system memory rather than the particular input-output pairs being somehow represented individually in various different parts of the system).

To attempt to realize such a system, we shall suppose that the input key (or prototype) patterns are coded as vectors in $\mathbb{R}^n$, say, and that the responses are coded as vectors in $\mathbb{R}^m$. For example, the input might be a digitized photograph comprising a picture with 100×100 pixels, each of which may assume one of eight levels of greyness (from white ($= 0$) to black

(= 7)). In this case, by mapping the screen to a vector, via raster order, say, the input is a vector in $\mathbb{R}^{10000}$ and whose components actually take values in the set $\{0, \dots, 7\}$. The desired output might correspond to the name of the person in the photograph. If we wish to recognize up to 50 people, say, then we could give each a binary code name of 6 digits—which allows up to $2^6 = 64$ different names. Then the output can be considered as an element of $\mathbb{R}^6$.

Now, for any pair of vectors $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, we can effect the map $x \mapsto y$ via the action of the $m \times n$ matrix

$$M^{(x,y)} = y\, x^T$$

where x is considered as an $n \times 1$ (column) matrix and y as an $m \times 1$ matrix. Indeed,

$$\begin{aligned} M^{(x,y)} x &= y\, x^T x \\ &= \alpha\, y, \end{aligned}$$

where $\alpha = x^T x = \|x\|^2$, the squared Euclidean norm of x. The matrix yx^T is called the outer product of x and y. This suggests a model for our "associative system".

Suppose that we wish to consider p input-output pattern pairs, $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, ..., $(x^{(p)}, y^{(p)})$. Form the $m \times n$ matrix

$$M = \sum_{i=1}^{p} y^{(i)}\, x^{(i)T}.$$

M is called the correlation memory matrix (corresponding to the given pattern pairs). Note that if we let $X = (x^{(1)} \cdots x^{(p)})$ and $Y = (y^{(1)} \cdots y^{(p)})$ be the $n \times p$ and $m \times p$ matrices with columns given by the vectors $x^{(1)}, \dots, x^{(p)}$ and $y^{(1)}, \dots, y^{(p)}$, respectively, then the matrix $\sum_{i=1}^{p} y^{(i)}\, x^{(i)T}$ is just YX^T. Indeed, the jk-element of YX^T is

$$\begin{aligned} (YX^T)_{jk} &= \sum_{i=1}^{p} Y_{ji} (X^T)_{ik} = \sum_{i=1}^{p} Y_{ji} X_{ki} \\ &= \sum_{i=1}^{p} y_j^{(i)} x_k^{(i)} \end{aligned}$$

which is precisely the jk-element of M.

When presented with the input signal $x^{(j)}$, the output is

$$\begin{aligned} Mx^{(j)} &= \sum_{i=1}^{p} y^{(i)}\, x^{(i)T} x^{(j)} \\ &= y^{(j)} x^{(j)T} x^{(j)} + \sum_{\substack{i=1 \\ i \neq j}}^{p} (x^{(i)T} x^{(j)}) y^{(i)}. \end{aligned}$$

In particular, if we agree to "normalize" the key input signals so that $x^{(i)T}x^{(i)} = \|x^{(i)}\|^2 = 1$, for all $1 \le i \le p$, then the first term on the right hand side above is just $y^{(j)}$, the desired response signal. The second term on the right hand side is called the "cross-talk" since it involves overlaps (i.e., inner products) of the various input signals.

If the input signals are pairwise orthogonal vectors, as well as being normalized, then $x^{(i)T}x^{(j)} = 0$ for all $i \neq j$. In this case, we get

$$Mx^{(j)} = y^{(j)}$$

that is, perfect recall. Note that $\mathbb{R}^n$ contains at most n mutually orthogonal vectors.

Operationally, one can imagine the system organized as indicated in the figure.

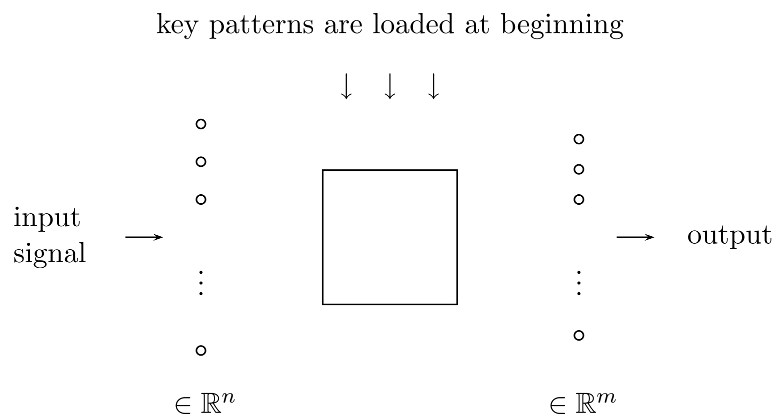


Figure 1.2: An operational view of the correlation memory matrix.

- start with $M = 0$,
- load the key input patterns one by one

  $M \leftarrow M + y^{(i)}x^{(i)T}, \quad i = 1, \dots, p,$
- finally, present any input signal and observe the response.

Note that additional signal-response patterns can simply be "added in" at any time, or even removed—by adding in $-y^{(j)}x^{(j)T}$. After the second stage above, the system has "learned" the signal-response pattern pairs. The collection of pattern pairs $(x^{(1)}, y^{(1)})$, ..., $(x^{(p)}, y^{(p)})$ is called the training set.

Remark 1.1. In general, the system is a heteroassociative memory $x^{(i)} \rightsquigarrow y^{(i)}$, $1 \le i \le p$. If the output is the prototype input itself, then the system is said to be an autoassociative memory.

We wish, now, to consider a quantitative account of the robustness of the autoassociative memory matrix. For this purpose, we shall suppose that the prototype patterns are bipolar vectors in $\mathbb{R}^n$, i.e., the components of the $x^{(i)}$ each belong to $\{-1, 1\}$. Then $\|x^{(i)}\|^2 = \sum_{j=1}^n x_j^{(i)2} = n$, for each $1 \le i \le p$, so that $(1/\sqrt{n})x^{(i)}$ is normalized. Suppose, further, that the prototype vectors are pairwise orthogonal (—this requires that n be even). The correlation memory matrix is

$$M = \frac{1}{n} \sum_{i=1}^{p} x^{(i)}\, x^{(i)T}$$

and we have seen that M has perfect recall, $Mx^{(j)} = x^{(j)}$ for all $1 \le j \le p$. We would like to know what happens if M is presented with x, a corrupted version of one of the $x^{(j)}$. In order to obtain a bipolar vector as output, we process the output vector Mx as follows:

$$Mx \rightsquigarrow \Phi(Mx)$$

where $\Phi : \mathbb{R}^n \to \{-1, 1\}^n$ is defined by

$$\Phi(z)_k = \begin{cases} 1, & \text{if } z_k \ge 0 \\ -1, & \text{if } z_k < 0 \end{cases}$$

for $1 \le k \le n$ and $z \in \mathbb{R}^n$. Thus, the matrix output is passed through a (bipolar) signal quantizer, Φ. To proceed, we introduce the notion of Hamming distance between pairs of bipolar vectors.

Let $a = (a_1, \dots, a_n)$ and $b = (b_1, \dots, b_n)$ be elements of $\{-1, 1\}^n$, i.e., bipolar vectors. The set $\{-1, 1\}^n$ consists of the 2^n vertices of a hypercube in $\mathbb{R}^n$. Then

$$a^T b = \sum_{i=1}^{n} a_i b_i = \alpha - \beta$$

where α is the number of components of a and b which are the same, and β is the number of differing components ($a_i b_i = 1$ if and only if $a_i = b_i$, and $a_i b_i = -1$ if and only if $a_i \ne b_i$). Clearly, $\alpha + \beta = n$ and so $a^T b = n - 2\beta$.

Definition 1.2. The Hamming distance between the bipolar vectors a, b, denoted $\rho(a, b)$, is defined to be

$$\rho(a, b) = \tfrac{1}{2} \sum_{i=1}^{n} |a_i - b_i|.$$

Evidently (thanks to the factor $\frac{1}{2}$), $\rho(a,b)$ is just the total number of mismatches between the components of a and b, i.e., it is equal to β, above. Hence

$$a^T b = n - 2\rho(a,b).$$

Note that the Hamming distance defines a metric on the set of bipolar vectors. Indeed, $\rho(a,b) = \frac{1}{2}\|a-b\|_1$, where $\|\cdot\|_1$ is the ℓ^1-norm defined on $\mathbb{R}^n$ by $\|z\|_1 = \sum_{i=1}^n |z_i|$, for $z = (z_1,\dots,z_n) \in \mathbb{R}^n$. The ℓ^1-norm is also known as the Manhatten norm—the distance between two locations is the sum of lengths of the east-west and north-south contributions to the journey, inasmuch as diagonal travel is not possible in Manhatten.

Hence, using $x^{(i)T}x = n - 2\rho(x^{(i)},x)$, we have

$$\begin{aligned} Mx &= \frac{1}{n}\sum_{i=1}^p x^{(i)} x^{(i)T} x \\ &= \frac{1}{n}\sum_{i=1}^p (n - 2\rho_i(x))\, x^{(i)}, \end{aligned}$$

where $\rho_i(x) = \rho(x^{(i)},x)$, the Hamming distance between the input vector x and the prototype pattern vector $x^{(i)}$.

Given x, we wish to know when $x \rightsquigarrow x^{(m)}$, that is, when $x \rightsquigarrow Mx \rightsquigarrow \Phi(Mx) = x^{(m)}$. According to our bipolar quantization rule, it will certainly be true that $\Phi(Mx) = x^{(m)}$ whenever the corresponding components of Mx and $x^{(m)}$ have the same sign. This will be the case when $(Mx)_j\, x_j^{(m)} > 0$, that is, whenever

$$\frac{1}{n}(n - 2\rho_m(x))\, \underbrace{x_j^{(m)} x_j^{(m)}}_{=1} + \frac{1}{n}\sum_{\substack{i=1\\ i\neq m}}^p (n - 2\rho_i(x))\, x_j^{(i)} x_j^{(m)} > 0$$

for all $1 \le j \le n$. This holds if

$$\left|\sum_{\substack{i=1\\ i\neq m}}^p (n - 2\rho_i(x))\, x_j^{(i)} x_j^{(m)}\right| < n - 2\rho_m(x) \qquad ((*))$$

for all $1 \le j \le n$ (—we have used the fact that if $s > |t|$ then certainly $s + t > 0$).

We wish to find conditions which ensure that the inequality $(*)$ holds. By the triangle inequality, we get

$$\left|\sum_{\substack{i=1\\ i\neq m}}^p (n - 2\rho_i(x))\, x_j^{(i)} x_j^{(m)}\right| \le \sum_{\substack{i=1\\ i\neq m}}^p |n - 2\rho_i(x)| \qquad (**)$$

since $|x_j^{(i)} x_j^{(m)}| = 1$ for all $1 \le j \le n$. Furthermore, using the orthogonality of $x^{(m)}$ and $x^{(i)}$, for $i \ne m$, we have

$$0 = x^{(i)T} x^{(m)} = n - 2\rho(x^{(i)}, x^{(m)})$$

so that

$$\begin{aligned} |n - 2\rho_i(x)| &= |2\rho(x^{(i)}, x^{(m)}) - 2\rho_i(x)| \\ &= 2|\rho(x^{(i)}, x^{(m)}) - \rho(x^{(i)}, x)| \\ &\le 2\rho(x^{(m)}, x), \end{aligned}$$

where the inequality above follows from the pair of inequalities

$$\rho(x^{(i)}, x^{(m)}) \le \rho(x^{(i)}, x) + \rho(x, x^{(m)}) \quad \text{and}$$
$$\rho(x^{(i)}, x) \le \rho(x^{(i)}, x^{(m)}) + \rho(x^{(m)}, x).$$

Hence, we have

$$|n - 2\rho_i(x)| \le 2\rho_m(x) \tag{$*{*}*$}$$

for all $i \ne m$. This, together with $(**)$ gives

$$\begin{aligned} \Bigg| \sum_{\substack{i=1 \\ i \ne m}}^{p} (n - 2\rho_i(x))\, x_j^{(i)} x_j^{(m)} \Bigg| &\le \sum_{\substack{i=1 \\ i \ne m}}^{p} |n - 2\rho_i(x)| \\ &\le 2(p-1)\rho_m(x). \end{aligned}$$

It follows that whenever $2(p-1)\rho_m(x) < n - 2\rho_m(x)$ then $(*)$ holds which means that $Mx = x^{(m)}$. The condition $2(p-1)\rho_m(x) < n - 2\rho_m(x)$ is just that $2p\rho_m(x) < n$, i.e., the condition that $\rho_m(x) < n/2p$.

Now, we observe that if $\rho_m(x) < n/2p$, then, for any $i \ne m$,

$$\begin{aligned} n - 2\rho_i(x) &\le 2\rho_m(x) \quad \text{by } (*{*}*), \text{ above,} \\ &< \frac{n}{p} \end{aligned}$$

and so $n - 2\rho_i(x) < n/p$. Thus

$$\begin{aligned} \rho_i(x) > \frac{1}{2}\left(n - \frac{n}{p}\right) &= \frac{n}{2}\left(\frac{p-1}{p}\right) \\ &\ge \frac{n}{2p}, \end{aligned}$$

assuming that $p \ge 2$, so that $p - 1 \ge 1$. In other words, if x is within Hamming distance of $(n/2p)$ from $x^{(m)}$, then its Hamming distance to every other prototype input vector is greater (or equal to) $(n/2p)$. We have thus proved the following theorem (L. Personnaz, I. Guyon and G. Dreyfus, Phys. Rev. A 34, 4217–4228 (1986)).

Theorem 1.3. *Suppose that* $\{x^{(1)}, x^{(2)}, \dots, x^{(p)}\}$ *is a given set of mutually orthogonal bipolar patterns in* $\{-1, 1\}^n$*. If* $x \in \{-1, 1\}^n$ *lies within Hamming distance* $(n/2p)$ *of a particular prototype vector* $x^{(m)}$*, say, then* $x^{(m)}$ *is the nearest prototype vector to* x*.*

Furthermore, if the autoassociative matrix memory based on the patterns $\{x^{(1)}, x^{(2)}, \dots, x^{(p)}\}$ *is augmented by subsequent bipolar quantization, then the input vector* x *invokes* $x^{(m)}$ *as the corresponding output.*

This means that the combined memory matrix and quantization system can correctly recognize (slightly) corrupted input patterns. The nonlinearity (induced by the bipolar quantizer) has enhanced the system performance—small background "noise" has been removed. Note that it could happen that the output response to x is still $x^{(m)}$ even if x is further than $(n/2p)$ from $x^{(m)}$. In other words, the theorem only gives sufficient conditions for x to recall $x^{(m)}$.

As an example, suppose that we store 4 patterns built from a grid of 8×8 pixels, so that $p = 4$, $n = 8^2 = 64$ and $(n/2p) = 64/8 = 8$. Each of the 4 patterns can then be correctly recalled even when presented with up to 7 incorrect pixels.

Remark 1.4. If x is close to $-x^{(m)}$, then the output from the combined autocorrelation matrix memory and bipolar quantizer is $-x^{(m)}$.

Proof. Let $\widehat{M} = \frac{1}{n} \sum_{i=1}^{p} (-x^{(i)})(-x^{(i)})^T$. Then clearly $\widehat{M} = M$, i.e., the autoassociative correlation memory matrix for the patterns $-x^{(1)}, \dots, -x^{(p)}$ is exactly the same as that for the patterns $x^{(1)}, \dots, x^{(p)}$. Applying the above theorem to the system with the negative patterns, we get that $\Phi(\widehat{M}x) = -x^{(m)}$, whenever x is within Hamming distance $(n/2p)$ of $-x^{(m)}$. But then $\Phi(Mx) = x^{(m)}$, as claimed. ■

A memory matrix, also known as a linear associator, can be pictured as a network as in the figure.

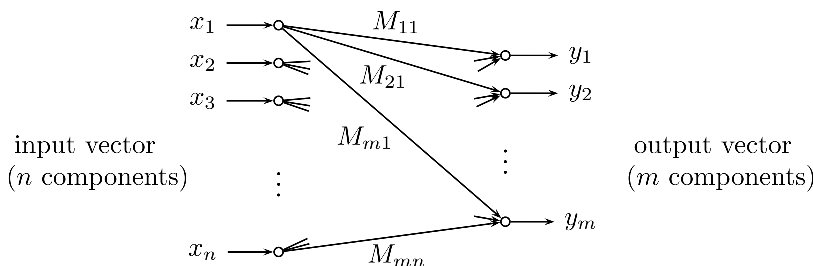


Figure 1.3: The memory matrix (linear associator) as a network.

"Weights" are assigned to the connections. Since $y_i = \sum_j M_{ij} x_j$, this suggests that we assign the weight M_{ij} to the connection joining input node j to output node i; $M_{ij} = \text{weight}(j \to i)$.

The correlation memory matrix trained on the pattern pairs $(x^{(1)}, y^{(1)})$,..., $(x^{(p)}, y^{(p)})$ is given by $M = \sum_{m=1}^{p} y^{(m)} x^{(m)T}$, which has typical term

$$M_{ij} = \sum_{m=1}^{p} (y^{(m)} x^{(m)T})_{ij}$$
$$= \sum_{m=1}^{p} y_i^{(m)} x_j^{(m)}.$$

Now, Hebb's law (1949) for "real" i.e., biological brains says that if the excitation of cell j is involved in the excitation of cell i, then continued excitation of cell j causes an increase in its efficiency to excite cell i. To encapsulate a crude version of this idea mathematically, we might hypothesise that the weight between the two nodes be proportional to the excitation values of the nodes. Thus, for pattern label m, we would postulate that the weight, weight(input $j \to$ output i), be proportional to $x_j^{(m)} y_i^{(m)}$.

We see that M_{ij} is a sum, over all patterns, of such terms. For this reason, the assignment of the correlation memory matrix to a content addressable memory system is sometimes referred to as generalized Hebbian learning, or one says that the memory matrix is given by the generalized Hebbian rule.

Capacity of autoassociative Hebbian learning

We have seen that the correlation memory matrix has perfect recall provided that the input patterns are pairwise orthogonal vectors. Clearly, there can be at most n of these. In practice, this orthogonality requirement may not be satisfied, so it is natural ask for some kind of guide as to the number of patterns that can be stored and effectively recovered. In other words, how many patterns can there be before the cross-talk term becomes so large that it destroys the recovery of the key patterns? Experiment confirms that, indeed, there is a problem here. To give some indication of what might be reasonable, consider the autoassociative correlation memory matrix based on p bipolar pattern vectors $x^{(1)}, \dots, x^{(p)} \in \{-1, 1\}^n$, followed by bipolar quantization, Φ. On presentation of pattern $x^{(m)}$, the system output is

$$\Phi(M\, x^{(m)}) = \Phi\Big(\frac{1}{n} \sum_{i=1}^{p} x^{(i)} x^{(i)T} x^{(m)}\Big).$$

Consider the kth bit. Then $\Phi(Mx^{(m)})_k = x_k^{(m)}$ whenever $x_k^{(m)}(Mx^{(m)})_k > 0$, that is whenever

$$\frac{1}{n}\,\underbrace{x_k^{(m)}x_k^{(m)}}_{1}\;\underbrace{x^{(m)T}x^{(m)}}_{n} + \underbrace{\frac{1}{n}\sum_{\substack{i=1\\ i\neq j}}^{p} x_k^{(m)}x_k^{(i)}x^{(i)T}x^{(m)}}_{C_k} > 0$$

or

$$1 + C_k > 0.$$

In order to consider a "typical" situation, we suppose that the patterns are random. Thus, $x^{(1)},\dots,x^{(p)}$ are selected at random from $\{-1,1\}^n$, with all pn bits being chosen independently and equally likely. The output bit $\Phi(Mx^{(m)})_k$ is therefore incorrect if $1 + C_k < 0$, i.e., if $C_k < -1$. We shall estimate the probability of this happening.

We see that C_k is a sum of many terms

$$C_k = \frac{1}{n}\sum_{\substack{i=1\\ i\neq m}}^{p}\sum_{j=1}^{n} X_{m,k,i,j}$$

where $X_{m,k,i,j} = x_k^{(m)}x_k^{(i)}x_j^{(i)}x_j^{(m)}$. We note firstly that, with $j = k$,

$$X_{m,k,i,k} = x_k^{(m)}x_k^{(i)}x_k^{(i)}x_k^{(m)} = 1.$$

Next, we see that, for $j \neq k$, each $X_{m,k,i,j}$ takes the values ± 1 with equal probability, namely, $\frac{1}{2}$, and that these different Xs form an independent family. Therefore, we may write C_k as

$$C_k = \frac{p-1}{n} + \frac{1}{n}\,S$$

where S is a sum of $(n-1)(p-1)$ independent random variables, each taking the values ± 1 with probability $\frac{1}{2}$. Each of the Xs has mean 0 and variance $\sigma^2 = 1$. By the Central Limit Theorem, it follows that the random variable $(S/(n-1)(p-1))/(\sigma/\sqrt{(n-1)(p-1)}) = S/\sqrt{(n-1)(p-1)}$ has an approximate standard normal distribution (for large n).

Hence, if we denote by Z a standard normal random variable,

$$\begin{aligned}
\text{Prob}(C_k < -1) &= \text{Prob}\Big(\frac{p-1}{n} + \frac{S}{n} < -1\Big) = \text{Prob}\big(S < -n - (p-1)\big) \\
&= \text{Prob}\Big(\frac{S}{\sqrt{(n-1)(p-1)}} < \frac{-n-(p-1)}{\sqrt{(n-1)(p-1)}}\Big) \\
&= \text{Prob}\Big(\frac{S}{\sqrt{(n-1)(p-1)}} < -\sqrt{\frac{n^2}{(n-1)(p-1)}} - \sqrt{\frac{p-1}{n-1}}\Big) \\
&\sim \text{Prob}\Big(Z < -\sqrt{\frac{n^2}{(n-1)(p-1)}} - \sqrt{\frac{p-1}{n-1}}\Big) \\
&\sim \text{Prob}\Big(Z < -\sqrt{\frac{n}{p}}\Big),
\end{aligned}$$

where we have ignored terms in $1/n$ and replaced $p-1$ by p. Using the symmetry of the standard normal distribution, we can rewrite this as

$$\text{Prob}(C_k < -1) = \text{Prob}\Big(\, Z > \sqrt{\frac{n}{p}}\,\Big).$$

Suppose that we require that the probability of an incorrect bit be no greater than 0.01 (or 1%). Then, from statistical tables, we find that $\text{Prob}(Z > \sqrt{n/p}) \leq 0.01$ requires that $\sqrt{n/p} \geq 2.326$. That is, we require $n/p \geq (2.326)^2$ or $p/n \leq 0.185$. Now, to say that any particular bit is incorrectly recalled with probability 0.01 is to say that the average number of incorrect bits (from a large sample) is 1% of the total. We have therefore shown that if we are prepared to accept up to 1% bad bits in our recalled patterns (on average) then we can expect to be able to store no more than $p = 0.185n$ patterns in our autoassociative system. That is, the storage capacity (with a 1% error tolerance) is $0.185n$.

Generalized inverse matrix memory

We have seen that the success of the correlation memory matrix, or Hebbian learning, is limited by the appearance of the cross-talk term. We shall derive an alternative system based on the idea of minimization of the output distortion or error.

Let us start again (and with a change of notation). We wish to construct an associative memory system which matches input patterns $a^{(1)},\dots,\ a^{(p)}$ (from $\mathbb{R}^n$) with output pattern vectors $b^{(1)},\dots,\ b^{(p)}$ (in $\mathbb{R}^m$), respectively. The question is whether or not we can find a matrix $M \in \mathbb{R}^{m\times n}$, the set of $m \times n$ real matrices, such that

$$Ma^{(i)} = b^{(i)}$$

for all $1 \le i \le p$. Let $A \in \mathbb{R}^{n \times p}$ be the matrix whose columns are the vectors $a^{(1)}, \dots, a^{(p)}$, i.e., $A = (a^{(1)} \cdots a^{(p)})$, and let $B \in \mathbb{R}^{m \times p}$ be the matrix with columns given by the $b^{(i)}$ s, $B = (b^{(1)} \cdots b^{(p)})$, thus $A_{ij} = a_i^{(j)}$ and $B_{ij} = b_i^{(j)}$. Then it is easy to see that $Ma^{(i)} = b^{(i)}$, for all i, is equivalent to $MA = B$. The problem, then, is to solve the matrix equation

$$MA = B,$$

for $M \in \mathbb{R}^{m \times n}$, for given matrices $A \in \mathbb{R}^{n \times p}$ and $B \in \mathbb{R}^{m \times p}$.

First, we observe that for a solution to exist, the matrices A and B cannot be arbitrary. Indeed, if $A = 0$, then so is MA no matter what M is—so the equation will not hold unless B also has all zero entries.

Suppose next, slightly more subtly, that there is some non-zero vector $v \in \mathbb{R}^p$ such that $Av = 0$. Then, for any M, $MAv = 0$. In general, it need not be true that $Bv = 0$.

Suppose then that there is *no* such non-zero $v \in \mathbb{R}^p$ such that $Av = 0$, i.e., we are supposing that $Av = 0$ implies that $v = 0$. What does this mean? We have

$$\begin{aligned}(Av)_i &= \sum_{j=1}^{p} A_{ij} v_j \\ &= v_1 A_{i1} + v_2 A_{i2} + \cdots + v_p A_{ip} \\ &= v_1 a_i^{(1)} + v_2 a_i^{(2)} + \cdots + v_p a_i^{(p)} \\ &= i^{\text{th}} \text{ component of } (v_1 a^{(1)} + \cdots + v_p a^{(p)}).\end{aligned}$$

In other words,

$$Av = v_1 a^{(1)} + \cdots + v_p a^{(p)}.$$

The vector Av is a linear combination of the columns of A, considered as elements of $\mathbb{R}^n$.

Now, the statement that $Av = 0$ if and only if $v = 0$ is equivalent to the statement that $v_1 a^{(1)} + \cdots + v_p a^{(p)} = 0$ if and only if $v_1 = v_2 = \cdots = v_p = 0$ which, in turn, is equivalent to the statement that $a^{(1)}, \dots, a^{(p)}$ are linearly independent vectors in $\mathbb{R}^n$.

Thus, the statement, $Av = 0$ if and only if $v = 0$, is true if and only if the columns of A are linearly independent vectors in $\mathbb{R}^n$.

Proposition 1.5. *For any $A \in \mathbb{R}^{n \times p}$, the $p \times p$ matrix $A^T A$ is invertible if and only if the columns of A are linearly independent in $\mathbb{R}^n$.*

Proof. The square matrix $A^T A$ is invertible if and only if the equation $A^T A v = 0$ has the unique solution $v = 0$, $v \in \mathbb{R}^p$. (Certainly the invertibility of $A^T A$ implies the uniqueness of the zero solution to $A^T A v = 0$. For the converse, first note that the uniqueness of this zero solution implies that

A^TA is a one-one linear mapping from $\mathbb{R}^p$ to $\mathbb{R}^p$. Moreover, using linearity, one readily checks that the collection $A^TAu_1, \ldots, A^TAu_p$ is a linearly independent set for any basis $u_1, \ldots, u_p$ of $\mathbb{R}^p$. This means that it is a basis and so A^TA maps $\mathbb{R}^p$ onto itself. Hence A^TA has an inverse. Alternatively, one can argue that since A^TA is symmetric it can be diagonalized via some orthogonal transformation. But a diagonal matrix is invertible if and only if every diagonal entry is non-zero. In this case, these entries are precisely the eigenvalues of A^TA. So A^TA is invertible if and only if none of its eigenvalues are zero.)

Suppose that the columns of A are linearly independent and that $A^TAv = 0$. Then it follows that $v^TA^TAv = 0$ and so $Av = 0$, since $v^TA^TAv = \sum_{i=1}^n (Av)_i^2 = \|Av\|_2^2$, the square of the Euclidean length of the n-dimensional vector Av. By the argument above, Av is a linear combination of the columns of A, and we deduce that $v = 0$. Hence A^TA is invertible.

On the other hand, if A^TA is invertible, then $Av = 0$ implies that $A^TAv = 0$ and so $v = 0$. Hence the columns of A are linearly independent. ∎

We can now derive the result of interest here.

Theorem 1.6. *Let A be any $n \times p$ matrix whose columns are linearly independent. Then for any $m \times p$ matrix B, there is an $m \times n$ matrix M such that $MA = B$.*

Proof. Let

$$M = \underbrace{B}_{m\times p}\underbrace{(A^TA)^{-1}}_{p\times p}\underbrace{A^T}_{p\times n}.$$

Then M is well-defined since $A^TA \in \mathbb{R}^{p\times p}$ is invertible, by the proposition. Direct substitution shows that $MA = B$. ∎

So, with this choice of M we get perfect recall, provided that the input pattern vectors are linearly independent.

Note that, in general, the solution above is not unique. Indeed, for any matrix $C \in \mathbb{R}^{m\times n}$, the $m \times n$ matrix $M' = C\big(\mathbb{1}_n - A(A^TA)^{-1}A^T\big)$ satisfies

$$M'A = C\big(A - A(A^TA)^{-1}A^TA\big) = C(A - A) = 0 \in \mathbb{R}^{m\times p}.$$

Hence $M + M'$ satisfies $(M + M')A = B$.

Can we see what M looks like in terms of the patterns $a^{(i)}$, $b^{(i)}$? The answer is "yes and no". We have $A = (a^{(1)} \cdots a^{(p)})$ and $B = (b^{(1)} \cdots b^{(p)})$. Then

$$\begin{aligned}(A^TA)_{ij} &= \sum_{k=1}^n A_{ik}^T A_{kj} = \sum_{k=1}^n A_{ki}A_{kj} \\ &= \sum_{k=1}^n a_k^{(i)} a_k^{(j)}\end{aligned}$$

which gives A^TA directly in terms of the $a^{(i)}$s. Let $Q = A^TA \in \mathbb{R}^{p\times p}$. Then $M = BQ^{-1}A^T$, so that

$$\begin{aligned} M_{ij} &= \sum_{k,\ell=1}^{p} B_{ik}(Q^{-1})_{k\ell}A_{\ell j} \\ &= \sum_{k,\ell=1}^{p} b_i^{(k)}(Q^{-1})_{k\ell}a_j^{(\ell)} \quad \text{since } A^T_{\ell j} = A_{j\ell}. \end{aligned}$$

This formula for M, valid for linearly independent input patterns, expresses M more or less in terms of the patterns. The appearance of the inverse, Q^{-1}, somewhat lessens its appeal, however.

To discuss the case where the columns of A are not necessarily linearly independent, we need to consider the notion of generalized inverse.

Definition 1.7. For any given matrix $A \in \mathbb{R}^{m\times n}$, the matrix $X \in \mathbb{R}^{n\times m}$ is said to be a generalized inverse of A if

(i) $AXA = A$,

(ii) $XAX = X$,

(iii) $(AX)^T = AX$,

(iv) $(XA)^T = XA$.

The terms pseudoinverse or Moore-Penrose inverse are also commonly used for such an X.

Examples 1.8.

1. If $A \in \mathbb{R}^{n\times n}$ is invertible, then A^{-1} is the generalized inverse of A.

2. If $A = \alpha \in \mathbb{R}^{1\times 1}$, then $X = 1/\alpha$ is the generalized inverse provided $\alpha \neq 0$. If $\alpha = 0$, then $X = 0$ is the generalized inverse.

3. The generalized inverse of $A = 0 \in \mathbb{R}^{m\times n}$ is $X = 0 \in \mathbb{R}^{n\times m}$.

4. If $A = u \in \mathbb{R}^{m\times 1}$, $u \neq 0$, then one checks that $X = u^T/(u^Tu)$ is a generalized inverse of u.

The following result is pertinent to the theory.

Theorem 1.9. *Every matrix possesses a unique generalized inverse.*

Proof. We postpone discussion of existence (which can be established via the Singular Value Decomposition) and just show uniqueness. This follows

by repeated use of the defining properties (i),...,(iv). Let $A \in \mathbb{R}^{m\times n}$ be given and suppose that $X, Y \in \mathbb{R}^{n\times m}$ are generalized inverses of A. Then

$$\begin{aligned}
X &= XAX, \quad \text{by (i)},\\
&= X(AX)^T, \quad \text{by (iii)},\\
&= XX^T\underline{A^T}\\
&= XX^TA^T\underline{Y^TA^T}, \quad \text{by (i)}^T,\\
&= X\underline{X^TA^T}AY, \quad \text{by (iii)},\\
&= \underline{XAX}AY, \quad \text{by (iii)},\\
&= X\underline{A}Y, \quad \text{by (ii)},\\
&= XA\underline{YA}Y, \quad \text{by (i)},\\
&= \underline{XA}A^TY^TY, \quad \text{by (iv)},\\
&= \underline{A^TX^TA^T}Y^TY, \quad \text{by (iv)},\\
&= \underline{A^TY^T}Y, \quad \text{by (i)}^T,\\
&= YAY, \quad \text{by (iv)},\\
&= Y, \quad \text{by (ii)},
\end{aligned}$$

as required. ∎

Notation For given $A \in \mathbb{R}^{m\times n}$, we denote its generalized inverse by $A^{\#}$. It is also often written as A^g, A^+ or $A^\dagger$.

Proposition 1.10. *For any $A \in \mathbb{R}^{m\times n}$, $AA^{\#}$ is the orthogonal projection onto* $\operatorname{ran} A$*, the linear span in $\mathbb{R}^m$ of the columns of A, i.e., if $P = AA^{\#} \in \mathbb{R}^{m\times m}$, then $P = P^T = P^2$ and P maps $\mathbb{R}^m$ onto* $\operatorname{ran} A$*.*

Proof. The defining property (iii) of the generalized inverse $A^{\#}$ is precisely the statement that $P = AA^{\#}$ is symmetric. Furthermore,

$$\begin{aligned}
P^2 &= AA^{\#}AA^{\#} = AA^{\#}, \quad \text{by condition (i)},\\
&= P
\end{aligned}$$

so P is idempotent. Thus P is an orthogonal projection.

For any $x \in \mathbb{R}^m$, we have that $Px = AA^{\#}x \in \operatorname{ran} A$, so that $P : \mathbb{R}^m \to \operatorname{ran} A$. On the other hand, if $x \in \operatorname{ran} A$, there is some $z \in \mathbb{R}^n$ such that $x = Az$. Hence $Px = PAz = AA^{\#}Az = Az = x$, where we have used condition (i) in the penultimate step. Hence P maps $\mathbb{R}$ onto $\operatorname{ran} A$. ∎

Proposition 1.11. *Let $A \in \mathbb{R}^{m\times n}$.*

(i) *If* $\operatorname{rank} A = n$*, then $A^{\#} = (A^TA)^{-1}A^T$.*

(ii) *If* $\operatorname{rank} A = m$*, then $A^{\#} = A^T(AA^T)^{-1}$.*

Proof. If $\operatorname{rank} A = n$, then A has linearly independent columns and we know that this implies that $A^T A$ is invertible (in $\mathbb{R}^{n\times n}$). It is now a straightforward matter to verify that $(A^T A)^{-1} A^T$ satisfies the four defining properties of the generalized inverse, which completes the proof of (i).

If $\operatorname{rank} A = m$, we simply consider the transpose instead. Let $B = A^T$. Then $\operatorname{rank} B = m$, since A and A^T have the same rank, and so, by the argument above, $B^{\#} = (B^T B)^{-1} B^T$. However, $A^{T\#} = A^{\#T}$, as is easily checked (again from the defining conditions). Hence

$$\begin{aligned} A^{\#} &= A^{\#TT} = (A^T)^{\#T} \\ &= B^{\#T} = B(B^T B)^{-1} \\ &= A^T (AA^T)^{-1} \end{aligned}$$

which establishes (ii). ∎

Definition 1.12. The $\|\cdot\|_F$-norm on $\mathbb{R}^{m\times n}$ is defined by

$$\|A\|_F^2 = \operatorname{Tr}(A^T A) \quad \text{for } A \in \mathbb{R}^{m\times n},$$

where $\operatorname{Tr}(B)$ is the trace of the square matrix B; $\operatorname{Tr}(B) = \sum_i B_{ii}$. This norm is called the Frobenius norm and sometimes denoted $\|\cdot\|_2$.

We see that

$$\begin{aligned} \|A\|_F^2 = \operatorname{Tr}(A^T A) &= \sum_{i=1}^{n} (A^T A)_{ii} \\ &= \sum_{i=1}^{n} \sum_{j=1}^{m} A^T_{ij} A_{ji} \\ &= \sum_{i=1}^{n} \sum_{j=1}^{m} A_{ij}^2 \end{aligned}$$

since $A^T_{ij} = A_{ji}$. Hence

$$\|A\|_F = \sqrt{(\text{sum of squares of all entries of } A)}\,.$$

We also note, here, that clearly $\|A\|_F = \|A^T\|_F$.

Suppose that $A = u \in \mathbb{R}^{m\times 1}$, an m-component vector. Then $\|A\|_F^2 = \sum_{i=1}^m u_i^2$, that is, $\|A\|_F$ is the usual Euclidean norm in this case. Thus the notation $\|\cdot\|_2$ for this norm is consistent. Note that, generally, $\|A\|_F$ is just the Euclidean norm of $A \in \mathbb{R}^{m\times n}$ when A is "taken apart" row by row and considered as a vector in $\mathbb{R}^{mn}$ via the correspondence $A \leftrightarrow (A_{11}, A_{12}, \ldots, A_{1n}, A_{21}, \ldots, A_{mn})$.

The notation $\|A\|_2$ is sometimes used in numerical analysis texts (and in the computer algebra software package Maple) to mean the norm of

A as a linear map from $\mathbb{R}^n$ into $\mathbb{R}^m$, that is, the value $\sup\{\|Ax\| : x \in \mathbb{R}^n \text{ with } \|x\| = 1\}$. One can show that this value is equal to the square root of the largest eigenvalue of A^TA whereas $\|A\|_F$ is equal to the square root of the sum of the eigenvalues of A^TA.

Remark 1.13. Let $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{m\times n}$, and $X \in \mathbb{R}^{p\times p}$. Then it is easy to see that

(i) $\operatorname{Tr}(AB) = \operatorname{Tr}(BA)$,

(ii) $\operatorname{Tr}(X) = \operatorname{Tr}(X^T)$,

(iii) $\operatorname{Tr}(AC^T) = \operatorname{Tr}(C^TA) = \operatorname{Tr}(A^TC) = \operatorname{Tr}(CA^T)$.

The equalities in (iii) can each be verified directly, or alternatively, one notices that (iii) is a consequence of (i) and (ii) (replacing B by C^T).

Lemma 1.14. *For $A \in \mathbb{R}^{m\times n}$, $A^\# AA^T = A^T$.*

Proof. We have

$$\begin{aligned}(A^\# A)A^T &= (A^\# A)^T A^T \quad \text{by condition (iv)} \\ &= (A(A^\# A))^T \\ &= A^T \quad \text{by condition (i)}\end{aligned}$$

as required. ∎

Theorem 1.15. *Let $A \in \mathbb{R}^{n\times p}$ and $B \in \mathbb{R}^{m\times p}$ be given. Then $X = BA^\#$ is an element of $\mathbb{R}^{m\times n}$ which minimizes the quantity $\|XA - B\|_F$.*

Proof. We have

$$\begin{aligned}\|XA - B\|_F^2 &= \|(X - BA^\#)A + B(A^\# A - \mathbb{1}_p)\|_F^2 \\ &= \|(X - BA^\#)A\|_F^2 + \|B(A^\# A - \mathbb{1}_p)\|_F^2 \\ &\qquad + 2\operatorname{Tr}\big(A^T(X - BA^\#)^T B(A^\# A - \mathbb{1}_p)\big) \\ &= \|(X - BA^\#)A\|_F^2 + \|B(A^\# A - \mathbb{1}_p)\|_F^2 \\ &\qquad + 2\operatorname{Tr}\big((X - BA^\#)^T B\,\underbrace{(A^\# A - \mathbb{1}_p)A^T}_{=0 \text{ by the lemma}}\big).\end{aligned}$$

Hence

$$\|XA - B\|_F^2 = \|(X - BA^\#)A\|_F^2 + \|B(A^\# A - \mathbb{1}_p)\|_F^2$$

which achieves its minimum, $\|B(A^\# A - \mathbb{1}_p)\|_F^2$, when $X = BA^\#$. ∎

Note that any X satisfying $XA = BA^{\#}A$ gives a minimum solution. If A^T has full column rank (or, equivalently, A^T has no kernel) then AA^T is invertible. Multiplying on the right by $A^T(AA^T)^{-1}$ gives $X = BA^{\#}$. So under this condition on A^T, we see that there is a unique solution $X = BA^{\#}$ minimizing $\|XA - B\|_F$.

In general, one can show that $BA^{\#}$ is that element with minimal $\|\cdot\|_F$-norm which minimizes $\|XA - B\|_F$, i.e., if $Y \neq BA^{\#}$ and $\|YA - B\|_F = \|BA^{\#}A - B\|_F$, then $\|BA^{\#}\|_F < \|Y\|_F$.

Now let us return to our problem of finding a memory matrix which stores the input-output pattern pairs $(a^{(i)}, b^{(i)})$, $1 \le i \le p$, with each $a^{(i)} \in \mathbb{R}^n$ and each $b^{(i)} \in \mathbb{R}^m$. In general, it may not be possible to find a matrix $M \in \mathbb{R}^{m \times n}$ such that $Ma^{(i)} = b^{(i)}$, for each i. Whatever our choice of M, the system output corresponding to the input $a^{(i)}$ is just $Ma^{(i)}$. So, failing equality $Ma^{(i)} = b^{(i)}$, we would at least like to minimize the error $b^{(i)} - Ma^{(i)}$. A measure of such an error is $\|b^{(i)} - Ma^{(i)}\|_2^2$ the squared Euclidean norm of the difference. Taking all p patterns into account, the total system recall error is taken to be

$$\sum_{i=1}^{p} \|b^{(i)} - Ma^{(i)}\|_2^2.$$

Let $A = (a^{(1)} \cdots a^{(p)}) \in \mathbb{R}^{n \times p}$ and $B = (b^{(1)} \cdots b^{(p)}) \in \mathbb{R}^{m \times p}$ be the matrices whose columns are given by the pattern vectors $a^{(i)}$ and $b^{(i)}$, respectively. Then the total system recall error, above, is just

$$\|B - MA\|_F^2.$$

We have seen that this is minimized by the choice $M = BA^{\#}$, where $A^{\#}$ is the generalized inverse of A. The memory matrix $M = BA^{\#}$ is called the optimal linear associative memory (OLAM) matrix.

Remark 1.16. If the patterns $\{a^{(1)}, \dots, a^{(p)}\}$ constitute an orthonormal family, then A has independent columns and so $A^{\#} = (A^TA)^{-1}A^T = \mathbb{1}_p A^T$, so that the OLAM matrix is $BA^{\#} = BA^T$ which is exactly the correlation memory matrix.

In the autoassociative case, $b^{(i)} = a^{(i)}$, so that $B = A$ and the OLAM matrix is given as

$$M = AA^{\#}.$$

We have seen that $AA^{\#}$ is precisely the projection onto the range of A, i.e., onto the subspace of $\mathbb{R}^n$ spanned by the prototype patterns. In this case, we say that M is given by the projection rule.

Any input $x \in \mathbb{R}^n$ can be written as

$$x = \underbrace{AA^{\#}x}_{\text{OLAM system output}} + \underbrace{(\mathbb{1} - AA^{\#})x}_{\text{``novelty''}}.$$

Kohonen calls $\mathbb{1} - AA^{\#}$ the novelty filter and has applied these ideas to image-subtraction problems such as tumor detection in brain scans. Non-null novelty vectors may indicate disorders or anomalies.

Pattern classification

We have discussed the distributed associative memory (DAM) matrix as an autoassociative or as a heteroassociative memory model. The first is mathematically just a special case of the second. Another special case is that of so-called classification. The idea is that one simply wants an input signal to elicit a response "tag", typically coded as one of a collection of orthogonal unit vectors, such as given by the standard basis vectors of $\mathbb{R}^m$.

- In operation, the input x induces output Mx, which is then associated with that tag vector corresponding to its maximum component. In other words, if $(Mx)_j$ is the maximum component of Mx, then the output Mx is associated with the jth tag.

Examples of various pattern classification tasks have been given by T. Kohonen, P. Lehtiö, E. Oja, A. Kortekangas and K. Mäkisara, Demonstration of pattern processing properties of the optimal associative mappings, *Proceedings of the International Conference on Cybernetics and Society*, Washington, D. C., 581–585 (1977). (See also the article "Storage and Processing of Information in Distributed Associative Memory Systems" by T. Kohonen, P. Lehtiö and E. Oja in "Parallel Models of Associative Memory" edited by G. Hinton and J. Anderson, published by Lawrence Erlbaum Associates, (updated edition) 1989.)

In one such experiment, ten people were each photographed from five different angles, ranging from 45° to −45°, with 0° corresponding to a fully frontal face. These were then digitized to produce pattern vectors with eight possible intensity levels for each pixel. A distinct unit vector, a tag, was associated with each person, giving a total of ten tags, and fifty patterns. The OLAM matrix was constructed from this data.

The memory matrix was then presented with a digitized photograph of one of the ten people, but *taken from a different angle* to any of the original five prototypes. The output was then classified according to the tag associated with its largest component. This was found to give correct identification.

The OLAM matrix was also found to perform well with autoassociation. Pattern vectors corresponding to one hundred digitized photographs were

used to construct the autoassociative memory via the projection rule. When presented with incomplete or fuzzy versions of the original patterns, the OLAM matrix satisfactorily reconstructed the correct image.

In another autoassociative recall experiment, twenty one different prototype images were used to construct the OLAM matrix. These were each composed of three similarly placed copies of a subimage. New pattern images, consisting of just one part of the usual triple features, were presented to the OLAM matrix. The output images consisted of slightly fuzzy versions of the single part but triplicated so as to mimic the subimage positioning learned from the original twenty one prototypes.

An analysis comparing the performance of the correlation memory matrix with that of the generalized inverse matrix memory has been offered by Cherkassky, Fassett and Vassilas (IEEE Trans. on Computers, 40, 1429 (1991)). Their conclusion is that the generalized inverse memory matrix performs better than the correlation memory matrix for autoassociation, but that the correlation memory matrix is better for classification. This is contrary to the widespread belief that the generalized inverse memory matrix is the superior model.

Chapter 2

Adaptive Linear Combiner

We wish to consider a memory matrix for the special case of one-dimensional output vectors. Thus, we consider input pattern vectors $x^{(1)}, \dots, x^{(p)} \in \mathbb{R}^\ell$, say, with corresponding desired outputs $y^{(1)}, \dots, y^{(p)} \in \mathbb{R}$ and we seek a memory matrix $M \in \mathbb{R}^{1\times\ell}$ such that

$$Mx^{(i)} = y^{(i)},$$

for $1 \le i \le p$. Since $M \in \mathbb{R}^{1\times\ell}$, we can think of M as a row vector $M = (m_1, \dots, m_\ell)$. The output corresponding to the input $x = (x_1, \dots, x_\ell) \in \mathbb{R}^\ell$ is just

$$y = Mx = \sum_{i=1}^{\ell} m_i x_i \,.$$

Such a system is known as the adaptive linear combiner (ALC).

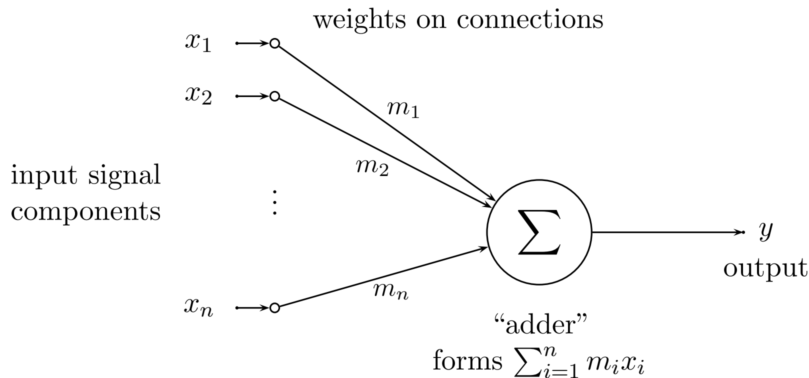


Figure 2.1: The Adaptive Linear Combiner.

We have seen that we may not be able to find M which satisfies the exact input-output relationship $Mx^{(i)} = y^{(i)}$, for each i. The idea is to look for an M which is in a certain sense optimal. To do this, we seek $m_1, \dots, m_\ell$

such that (one half) the average mean-squared error

$$\mathcal{E} \equiv \tfrac{1}{2}\tfrac{1}{p}\sum_{i=1}^{p}|y^{(i)} - z^{(i)}|^2$$

is minimized—where $y^{(i)}$ is the desired output corresponding to the input vector $x^{(i)}$ and $z^{(i)} = Mx^{(i)}$ is the actual system output. We already know, from the results in the last chapter, that this is achieved by the OLAM matrix based on the input-output pattern pairs, but we wish here to develop an algorithmic approach to the construction of the appropriate memory matrix. We can write out $\mathcal{E}$ in terms of the m_i as follows

$$\begin{aligned}
\mathcal{E} &= \tfrac{1}{2p}\sum_{i=1}^{p}|y^{(i)} - \sum_{j=1}^{\ell} m_j x_j^{(i)}|^2 \\
&= \tfrac{1}{2p}\sum_{i=1}^{p}\Big(y^{(i)2} + \sum_{j,k=1}^{\ell} m_j m_k x_j^{(i)} x_k^{(i)} - 2\sum_{j=1}^{\ell} y^{(i)} m_j x_j^{(i)}\Big) \\
&= \tfrac{1}{2}\sum_{j,k=1}^{\ell} m_j A_{jk} m_k - \sum_{j=1}^{\ell} b_j m_j + \tfrac{1}{2}c
\end{aligned}$$

where $A_{jk} = \frac{1}{p}\sum_{i=1}^{p} x_j^{(i)} x_k^{(i)}$, $b_j = \frac{1}{p}\sum_{i=1}^{p} y^{(i)} x_j^{(i)}$ and $c = \frac{1}{p}\sum_{i=1}^{p} y^{(i)2}$. Note that $A = (A_{jk}) \in \mathbb{R}^{\ell\times\ell}$ is symmetric. The error $\mathcal{E}$ is a non-negative quadratic function of the m_i. For a minimum, we investigate the equalities $\partial\mathcal{E}/\partial m_i = 0$, that is,

$$0 = \frac{\partial\mathcal{E}}{\partial m_i} = \sum_{k=1}^{\ell} A_{ik} m_k - b_i\,,$$

for $1 \le i \le \ell$, where we have used the symmetry of (A_{ik}) here. We thus obtain the so-called Gauss-normal or Wiener-Hopf equations

$$\sum_{k=1}^{\ell} A_{ik} m_k = b_i\,, \qquad \text{for } 1 \le i \le \ell,$$

or, in matrix form,

$$Am = b\,,$$

with $A = (A_{ik}) \in \mathbb{R}^{\ell\times\ell}$, $m = (m_1, \dots, m_\ell) \in \mathbb{R}^\ell$ and $b = (b_1, \dots, b_\ell) \in \mathbb{R}^\ell$. If A is invertible, then $m = A^{-1}b$ is the unique solution. In general, there may be many solutions. For example, if A is diagonal with $A_{11} = 0$, then necessarily $b_1 = 0$ (otherwise $\mathcal{E}$ could not be non-negative as a function of the m_i) and so we see that m_1 is arbitrary. To relate this to the OLAM matrix, write $\mathcal{E}$ as

$$\mathcal{E} = \tfrac{1}{2p}\|MX - Y\|_F^2,$$

where $X = (x^{(1)} \cdots x^{(p)}) \in \mathbb{R}^{\ell \times p}$ and $Y = (y^{(1)} \cdots y^{(p)}) \in \mathbb{R}^{1\times p}$. This, we know, is minimized by $M = YX^{\#} \in \mathbb{R}^{1\times \ell}$. Therefore $m = M^T$ must be a solution to the Wiener-Hopf equations above. We can write A, b and c in terms of the matrices X and Y. One finds that $A = \frac{1}{p}XX^T$, $b^T = \frac{1}{p}YX^T$ and $c = \frac{1}{p}Y^TY$. The equation $Am = b$ then becomes $XX^Tm = XY^T$ giving $m = (XX^T)^{-1}XY^T$, provided that A is invertible. This gives $M = m^T = YX^T(XX^T)^{-1} = YX^{\#}$, as above.

One method of attack for finding a vector m_* minimizing $\mathcal{E}$ is that of gradient-descent. The idea is to think of $\mathcal{E}(m_1, \dots, m_\ell)$ as a bowl-shaped surface above the ℓ-dimensional $m_1, \dots, m_\ell$-space. Pick any value for m. The vector $\operatorname{grad}\mathcal{E}$, when evaluated at m, points in the direction of maximum increase of $\mathcal{E}$ in the neighbourhood of m. That is to say, for small α (and a vector v of given length), $\mathcal{E}(m+\alpha v) - \mathcal{E}(m)$ is maximized when v points in the same direction as $\operatorname{grad}\mathcal{E}$ (as is seen by Taylor's theorem). Now, rather than increasing $\mathcal{E}$, we wish to minimize it. So the idea is to move a small distance from m to $m - \alpha \operatorname{grad}\mathcal{E}$, thus inducing maximal "downhill" movement on the error surface. By repeating this process, we hope to eventually reach a value of m which minimizes $\mathcal{E}$.

The strategy, then, is to consider a sequence of vectors $m(n)$ given algorithmically by

$$m(n+1) = m(n) - \alpha \operatorname{grad}\mathcal{E}\,, \qquad \text{for } n = 1, 2, \dots,$$

with $m(1)$ arbitrary and where the parameter α is called the learning rate. If we substitute for $\operatorname{grad}\mathcal{E}$, we find

$$m(n+1) = m(n) + \alpha\big(b - Am(n)\big).$$

Now, A is symmetric and so can be diagonalized. There is an orthogonal matrix $U \in \mathbb{R}^{\ell\times\ell}$ such that

$$UAU^T = D = \operatorname{diag}(\lambda_1, \dots, \lambda_\ell)$$

and we may assume that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_\ell$. We have

$$\begin{aligned}\mathcal{E} &= \tfrac{1}{2}m^TAm - b^Tm + \tfrac{1}{2}c \\ &= \tfrac{1}{2}m^TU^TUAU^TUm - b^TU^TUm + \tfrac{1}{2}c \\ &= \tfrac{1}{2}z^TDz - v^Tz + \tfrac{1}{2}c\end{aligned}$$

where $z = Um$ and $v = Ub$

$$= \tfrac{1}{2}\sum_{i=1}^{\ell}\lambda_i z_i^2 - \sum_{i=1}^{\ell} v_i z_i + \tfrac{1}{2}c.$$

Since $\mathcal{E} \geq 0$, it follows that all $\lambda_i \geq 0$—otherwise $\mathcal{E}$ would have a negative leading term. The recursion formula for $m(n)$, namely,

$$m(n+1) = m(n) + \alpha\big(b - Am(n)\big),$$

gives

$$Um(n+1) = Um(n) + \alpha\big(Ub - UAU^TUm(n)\big).$$

In terms of z, this becomes

$$z(n+1) = z(n) + \alpha\big(v - Dz(n)\big).$$

Hence, for any $1 \leq j \leq \ell$,

$$\begin{aligned} z_j(n+1) &= z_j(n) + \alpha\big(v_j - \lambda_j z_j(n)\big) \\ &= (1 - \alpha\lambda_j) z_j(n) + \alpha v_j. \end{aligned}$$

Setting $\mu_j = (1 - \alpha\lambda_j)$, we have

$$\begin{aligned} z_j(n+1) &= \mu_j z_j(n) + \alpha v_j \\ &= \mu_j(\mu_j z_j(n-1) + \alpha v_j) + \alpha v_j \\ &= \mu_j^2 z_j(n-1) + (\mu_j + 1)\alpha v_j \\ &= \mu_j^2(\mu_j z_j(n-2) + \alpha v_j) + (\mu_j + 1)\alpha v_j \\ &= \mu_j^3 z_j(n-2) + (\mu_j^2 + \mu_j + 1)\alpha v_j \\ &= \quad \cdots \\ &= \mu_j^n z_j(1) + (\mu_j^{n-1} + \mu_j^{n-2} + \cdots + \mu_j + 1)\alpha v_j. \end{aligned}$$

This converges, as $n \to \infty$, provided $|\mu_j| < 1$, that is, provided $-1 < 1 - \alpha\lambda_j < 1$. Thus, convergence demands the inequalities $0 < \alpha\lambda_j < 2$ for all $1 \leq j \leq \ell$. We therefore have shown that the algorithm

$$m(n+1) = m(n) + \alpha\big(b - Am(n)\big), \quad n = 1, 2, \ldots,$$

with $m(1)$ arbitrary, converges provided $0 < \alpha < \dfrac{2}{\lambda_{\max}}$, where $\lambda_{\max}$ is the maximum eigenvalue of A.

Suppose that $m(1)$ is given and that α does indeed satisfy the inequalities $0 < \alpha < 2/\lambda_{\max}$. Let m_* denote the limit $\lim_{n\to\infty} m(n)$. Then, letting $n \to \infty$ in the recursion formula for $m(n)$, we see that

$$m_* = m_* + \alpha\big(b - Am_*\big),$$

that is, m_* satisfies $Am_* = b$ and so $m(n)$ does, indeed, converge to a value minimizing $\mathcal{E}$. Indeed, if m_* satisfies $Am_* = b$, then we can complete the square and write $2\mathcal{E}$ as

$$\begin{aligned} 2\mathcal{E} &= m^T Am - 2b^T m + c \\ &= m^T Am - m_*^T Am - m^T Am_* + c, \text{ using } b^T m = m^T b \\ &= (m - m_*)^T A(m - m_*) - m_*^T m_* + c, \end{aligned}$$

which is certainly minimized when $m = m_*$.

The above analysis requires a detailed knowledge of the matrix A. In particular, its eigenvalues must be determined in order for us to be able to choose a valid value for the learning rate α. We would like to avoid having to worry too much about this detailed structure of A.

We recall that $A = (A_{jk})$ is given by

$$A_{jk} = \frac{1}{p} \sum_{i=1}^{p} x_j^{(i)} x_k^{(i)}.$$

This is an average of $x_j^{(i)} x_k^{(i)}$ taken over the patterns. Given a particular pattern $x^{(i)}$, we can think of $x_j^{(i)} x_k^{(i)}$ as an *estimate* for the average A_{jk}. Similarly, we can think of $b_j = \frac{1}{p} \sum_{i=1}^{p} y^{(i)} x_j^{(i)}$ as an average, and $y^{(i)} x_j^{(i)}$ as an estimate for b_j. Accordingly, we change our algorithm for updating the memory matrix to the following.

Select an input-output pattern pair, $(x^{(i)}, y^{(i)})$, say, and use the previous algorithm but with A_{jk} and b_j "estimated" as above. Thus,

$$m_j(n+1) = m_j(n) + \alpha\big(y^{(i)} x_j^{(i)} - \sum_{k=1}^{\ell} x_j^{(i)} x_k^{(i)} m_k(n)\big)$$

for $1 \le j \le n$. That is,

$$m_j(n+1) = m_j(n) + \alpha \delta^{(i)} x_j^{(i)}$$

where

$$\begin{aligned} \delta^{(i)} &= \big(y^{(i)} - \sum_{k=1}^{\ell} x_k^{(i)} m_k(n)\big) \\ &= (\text{desired output} - \text{actual output}) \end{aligned}$$

is the output error for pattern pair i. This is known as the delta-rule, or the Widrow-Hoff learning rule, or the least mean square (LMS) algorithm.

The learning rule is then as follows.

Widrow-Hoff (LMS) algorithm

- First choose a value for α, the learning rate (in practice, this might be 0.1 or 0.05, say).

- Start with $m_j(1) = 0$ for all j, or perhaps with small random values.

- Keep selecting input-output pattern pairs $x^{(i)}, y^{(i)}$ and update $m(n)$ by the rule
$$m_j(n+1) = m_j(n) + \alpha\delta^{(i)} x_j^{(i)}, \qquad 1 \le j \le \ell,$$
where $\delta^{(i)} = y^{(i)} - \sum_{k=1}^{\ell} m_k(n) x_k^{(i)}$ is the output error for the pattern pair (i) as determined by the memory matrix in operation at iteration step n. Ensure that every pattern pair is regularly presented and continue until the output error has reached and appears to remain at an acceptably small value.

- *The actual question of convergence still remains to be discussed!*

Remark 2.1. If α is too small, we might expect the convergence to be slow—the adjustments to m are small if α is small. Of course, this is assuming that there *is* convergence. Similarly, if the output error δ is small then changes to m will also be small, thus slowing convergence. This could happen if m enters an error surface "valley" with an almost flat bottom.

On the other hand, if α is too large, then the ms may overshoot and oscillate about an optimal solution. In practice, one might start with a largish value for α but then gradually decrease it as the learning progresses. These comments apply to any kind of gradient-descent algorithm.

Remark 2.2. Suppose that, instead of basing our discussion on the error function $\mathcal{E}$, we present the ith input vector $x^{(i)}$ and look at the immediate output "error"
$$\mathcal{E}^{(i)} = \tfrac{1}{2} \big| y^{(i)} - \sum_{k=1}^{\ell} m_k x_k^{(i)} \big|^2 .$$
Then we calculate
$$\frac{\partial \mathcal{E}^{(i)}}{\partial m_j} = -\underbrace{\big(y^{(i)} - \sum_{k=1}^{\ell} m_k x_k^{(i)} \big)}_{\delta^{(i)}} \, x_j^{(i)},$$
for $1 \le j \le \ell$. So we might try the "one pattern at a time" gradient-descent algorithm
$$m_j(n+1) = m_j(n) + \alpha\delta^{(i)} x_j^{(i)}, \qquad 1 \le j \le \ell,$$

with $m(1)$ arbitrary. This is exactly what we have already arrived at above. It should be clear from this point of view that there is no reason a priori to suppose that the algorithm converges. Indeed, one might be more inclined to suspect that the m-values given by this rule simply "thrash about all over the place" rather than settling down towards a limiting value.

Remark 2.3. We might wish to consider the input-output patterns x and y as random variables taking values in $\mathbb{R}^\ell$ and $\mathbb{R}$, respectively. In this context, it would be natural to consider the minimization of $\mathbb{E}((y - m^T x)^2)$. The analysis proceeds exactly as above, but now with $A_{jk} = \mathbb{E}(x_j x_k)$ and $b_j = \mathbb{E}(y x_j)$. The idea of using the current, i.e., the observed, values of x and y to construct estimates for A and b is a common part of standard statistical theory. The algorithm is then

$$m_j(n+1) = m_j(n) + \alpha\big(y^{(n)} x_j^{(n)} - \sum_{k=1}^{\ell} x_j^{(n)} x_k^{(n)} m_k(n)\big)$$

where $x^{(n)}$ and $y^{(n)}$ are the input-output pattern pair presented at step n. If we assume that these patterns presented at the various steps are independent, then, from the algorithm, we see that $m_k(n)$ only depends on the patterns presented before step n and so is independent of $x^{(n)}$. Taking expectations we obtain the vector equation

$$\mathbb{E}\, m(n+1) = \mathbb{E}\, m(n) + \alpha(\, b - A\, \mathbb{E}\, m(n)\,).$$

It follows, as above, that if $0 < \alpha < 2/\lambda_{\max}$, then $\mathbb{E}\, m(n)$ converges to m_* which minimizes the mean square error $\mathbb{E}((y - m^T x)^2)$.

We now turn to a discussion of the convergence of the LMS algorithm (see Z-Q. Luo, Neural Computation 3, 226–245 (1991)). Rather than just looking at the ALC system, we shall consider the general heteroassociative problem with p input-output pattern pairs $(a^{(1)}, b^{(1)}), \dots, (a^{(p)}, b^{(p)})$ with $a^{(i)} \in \mathbb{R}^\ell$ and $b^{(i)} \in \mathbb{R}^m$, $1 \le i \le p$. Taking $m = 1$, we recover the ALC, as above. We seek an algorithmic approach to minimizing the total system "error"

$$\mathcal{E}(M) = \sum_{i=1}^{p} \mathcal{E}^{(i)}(M) = \sum_{i=1}^{p} \tfrac{1}{2} \|b^{(i)} - M a^{(i)}\|^2$$

where $\mathcal{E}^{(i)}(M) = \frac{1}{2}\|b^{(i)} - M a^{(i)}\|^2$ is the error function corresponding to pattern i.

We have seen that $\mathcal{E}(M) = \frac{1}{2}\|B - MA\|_F^2$, where $\|\cdot\|_F$ is the Frobenius norm, $A = (a^{(1)} \cdots a^{(p)}) \in \mathbb{R}^{\ell \times p}$ and $B = (b^{(1)} \cdots b^{(p)}) \in \mathbb{R}^{m \times p}$ and that a solution to the problem is given by $M = BA^{\#}$.

Each $\mathcal{E}^{(i)}$ is a function of the elements M_{jk} of the memory matrix M. Calculating the partial derivatives gives

$$\begin{aligned}\frac{\partial \mathcal{E}^{(i)}}{\partial M_{jk}} &= \frac{\partial}{\partial M_{jk}} \ \tfrac{1}{2} \sum_{r=1}^{m} \big(b_r^{(i)} - \sum_{s=1}^{\ell} M_{rs} a_s^{(i)}\big)^2 \\ &= -\big(b_j^{(i)} - \sum_{s=1}^{\ell} M_{js} a_s^{(i)}\big) a_k^{(i)} \\ &= \big(\big(M a^{(i)} - b^{(i)}\big) a^{(i)T}\big)_{jk}.\end{aligned}$$

This leads to the following LMS learning algorithm.

- The patterns $(a^{(i)}, b^{(i)})$ are presented one after the other, $(a^{(1)}, b^{(1)})$, $(a^{(2)}, b^{(2)})$,..., $(a^{(p)}, b^{(p)})$—thus constituting a (pattern) cycle. This cycle is to be repeated.

- Let $M^{(i)}(n)$ denote the memory matrix just before the pattern pair $(a^{(i)}, b^{(i)})$ is presented in the nth cycle. On presentation of $(a^{(i)}, b^{(i)})$ to the network, the memory matrix is updated according to the rule

$$M^{(i+1)}(n) = M^{(i)}(n) - \alpha_n \big(M^{(i)}(n) a^{(i)} - b^{(i)}\big) a^{(i)T}$$

  where α_n is the (n-dependent) learning rate, and we agree that the matrix $M^{(p+1)}(n) = M^{(1)}(n+1)$, that is, presentation of pattern $p+1$ in cycle n is actually the presentation of pattern 1 in cycle $n+1$.

Remark 2.4. The gradient of the total error function $\mathcal{E}$ is given by the terms

$$\frac{\partial \mathcal{E}}{\partial M_{jk}} = \sum_{i=1}^{p} \frac{\partial \mathcal{E}^{(i)}}{\partial M_{jk}}.$$

When the ith example is being learned, only the terms $\partial \mathcal{E}^{(i)}/\partial M_{jk}$, for fixed i, are used to update the memory matrix. So at this step the value of $\mathcal{E}^{(i)}$ will decrease but it could happen that $\mathcal{E}$ actually *increases*. The point is that the algorithm is *not* a standard gradient-descent algorithm and so standard convergence arguments are not applicable. A separate proof of convergence must be given.

Remark 2.5. When $m = 1$, the output vectors are just real numbers and we recover the adaptive linear combiner and the Widrow-Hoff rule as a special case.

Remark 2.6. The algorithm is "local" in the sense that it only involves information available at the time of each presentation, i.e., it does not need to remember any of the previously seen examples.

The following result is due to Kohonen.

Theorem 2.7. *Suppose that $\alpha_n = \alpha > 0$ is fixed. Then, for each $i = 1, \dots, p$, the sequence $M^{(i)}(n)$ converges to some matrix $M_\alpha^{(i)}$ depending on α and i. Moreover,*

$$\lim_{\alpha \downarrow 0} M_\alpha^{(i)} = BA^{\#},$$

for each $i = 1, \dots, p$.

Remark 2.8. In general, the limit matrices $M_\alpha^{(i)}$ are different for different i.

We shall investigate a simple example to illustrate the theory (following Luo).

Example 2.9. Consider the case when there is a single input node, so that the memory matrix $M \in \mathbb{R}^{1\times 1}$ is just a real number, m, say.

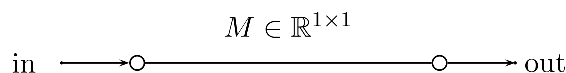


Figure 2.2: The ALC with one input node.

We shall suppose that the system is to learn the two pattern pairs $(1, c_1)$ and $(-1, c_2)$. Then the total system error function is

$$\mathcal{E} = \tfrac{1}{2}(c_1 - m)^2 + \tfrac{1}{2}(c_2 + m)^2$$

where $M_{11} = m$, as above. The LMS algorithm, in this case, becomes

$$\begin{aligned} m^{(2)}(n) &= m^{(1)}(n) - \alpha(m^{(1)}(n) - c_1) \\ m^{(3)}(n) = m^{(1)}(n+1) &= m^{(2)}(n) - \alpha(m^{(2)}(n) + c_2) \end{aligned}$$

with the initialization $m^{(1)}(1) = 0$. Hence

$$\begin{aligned} m^{(1)}(n+1) &= (1-\alpha)m^{(2)}(n) - \alpha c_2 \\ &= (1-\alpha)\big((1-\alpha)m^{(1)}(n) + \alpha c_1\big) - \alpha c_2 \\ &= \underbrace{(1-\alpha)^2}_{=\lambda,\text{ say}} m^{(1)}(n) + \underbrace{(1-\alpha)\alpha c_1 - \alpha c_2}_{=\beta,\text{ say}} \end{aligned}$$

giving

$$\begin{aligned} m^{(1)}(n+1) &= \lambda m^{(1)}(n) + \beta \\ &= \lambda\big(\lambda m^{(1)}(n-1) + \beta\big) + \beta \\ &= \lambda^2 m^{(1)}(n-1) + \lambda\beta + \beta \\ &= \quad \cdots \\ &= \underbrace{\lambda^n m^{(1)}(1)}_{=0} + (\lambda^{n-1} + \cdots + \lambda + 1)\beta \\ &= \frac{(1-\lambda^n)}{(1-\lambda)}\beta. \end{aligned}$$

We see that $\lim_{n\to\infty} m^{(1)}(n+1) = \beta/(1-\lambda)$, provided that $|\lambda| < 1$. This condition is equivalent to $(1-\alpha)^2 < 1$, or $|1-\alpha| < 1$, which is the same as $0 < \alpha < 2$. The limit is

$$m_\alpha^{(1)} \equiv \frac{\beta}{1-\lambda} = \frac{(1-\alpha)\alpha c_1 - \alpha c_2}{1-(1-\alpha)^2}$$

which simplifies to

$$m_\alpha^{(1)} = \frac{(1-\alpha)c_1 - c_2}{2-\alpha}.$$

Now, for $m^{(2)}(n)$, we have

$$\begin{aligned} m^{(2)}(n) &= m^{(1)}(n)(1-\alpha) + \alpha c_1 \\ &\to m_\alpha^{(2)} \equiv m_\alpha^{(1)}(1-\alpha) + \alpha c_1 \\ &= \frac{c_1 - (1-\alpha)c_2}{2-\alpha} \end{aligned}$$

as $n \to \infty$, provided $0 < \alpha < 2$. This shows that if we keep the learning rate fixed, then we do *not* get convergence, $m_\alpha^{(1)} \neq m_\alpha^{(2)}$, unless $c_1 = -c_2$. The actual optimal solution m_* to minimizing the error $\mathcal{E}$ is got from solving $d\mathcal{E}/dm = 0$, i.e., $-(c_1-m)+(c_2+m) = 0$, which gives $m_* = \frac{1}{2}(c_1-c_2)$. This is the value for the OLAM "matrix". If $c_1 \neq c_2$ and $\alpha \neq 0$, then $m_\alpha^{(1)} \neq m_*$ and $m_\alpha^{(2)} \neq m_*$. Notice that both $m_\alpha^{(1)}$ and $m_\alpha^{(2)}$ converge to the OLAM solution m_* as $\alpha \to 0$, and also the average $\frac{1}{2}(m^{(1)}(n) + m^{(2)}(n))$ converges to m_*.

Next, we shall consider a dynamically variable learning rate α_n. In this case the algorithm becomes

$$\begin{aligned} m^{(2)}(n) &= (1-\alpha_n)m^{(1)}(n) + \alpha_n c_1 \\ m^{(1)}(n+1) &= (1-\alpha_n)m^{(2)}(n) - \alpha_n c_2. \end{aligned}$$

Hence

$$m^{(1)}(n+1) = (1-\alpha_n)\big[(1-\alpha_n)m^{(1)}(n) + \alpha_n c_1\big] - \alpha_n c_2,$$

giving

$$m^{(1)}(n+1) = (1-\alpha_n)^2 m^{(1)}(n) + (1-\alpha_n)\alpha_n c_1 - \alpha_n c_2.$$

We wish to examine the convergence of $m^{(1)}(n)$ (and $m^{(2)}(n)$) to $m_* = \Big(\frac{c_1 - c_2}{2}\Big)$. So if we set $y_n = m^{(1)}(n) - \Big(\frac{c_1 - c_2}{2}\Big)$, then we would like to show that $y_n \to 0$, as $n \to \infty$. The recursion formula for y_n is

$$y_{n+1} + \Big(\frac{c_1 - c_2}{2}\Big) = (1-\alpha_n)^2\Big(y_n + \Big(\frac{c_1 - c_2}{2}\Big)\Big) + (1-\alpha_n)\alpha_n c_1 - \alpha_n c_2.$$

which simplifies to

$$y_{n+1} = (1-\alpha_n)^2 y_n - \alpha_n^2\Big(\frac{c_1 + c_2}{2}\Big).$$

Next, we impose suitable conditions on the learning rates, α_n, which will ensure convergence.

- Suppose that $0 \le \alpha_n < 1$, for all n, and that

$$\text{(i)} \quad \sum_{n=1}^{\infty} \alpha_n = \infty\,, \qquad \text{(ii)} \quad \sum_{n=1}^{\infty} \alpha_n^2 < \infty\,,$$

  that is, the series $\sum_{n=1}^{\infty} \alpha_n$ is divergent, whilst the series $\sum_{n=1}^{\infty} \alpha_n^2$ is convergent.

An example is provided by the assignment $\alpha_n = 1/n$. The intuition is that condition (i) ensures that the learning rate is always sufficiently large to push the iteration towards the desired limiting value, whereas condition (ii) ensures that its influence is not too strong that it might force the scheme into some kind of endless oscillatory behaviour.

Claim The sequence (y_n) converges to 0, as $n \to \infty$.

Proof. Let $r_n = -\Big(\frac{c_1 + c_2}{2}\Big)\,\alpha_n^2$. Then, by condition (ii), $\sum_{n=1}^{\infty} |r_n| < \infty$. The algorithm for y_n is

$$\begin{aligned}
y_{n+1} &= (1-\alpha_n)^2 y_n + r_n \\
&= (1-\alpha_n)^2((1-\alpha_{n-1})^2 y_{n-1} + r_{n-1}) + r_n \\
&= (1-\alpha_n)^2(1-\alpha_{n-1})^2 y_{n-1} + (1-\alpha_n)^2 r_{n-1} + r_n \\
&= \quad \cdots \\
&= (1-\alpha_n)^2(1-\alpha_{n-1})^2 \cdots (1-\alpha_1)^2 y_1 + (1-\alpha_n)^2 \cdots (1-\alpha_2)^2 r_1 \\
&\qquad + (1-\alpha_n)^2 \cdots (1-\alpha_3)^2 r_2 + \cdots + (1-\alpha_n)^2 r_{n-1} + r_n\,.
\end{aligned}$$

For convenience, set $y_1 = r_0$ and $\beta_j = (1-\alpha_j)^2$. Then we can write y_{n+1} as

$$y_{n+1} = r_0\beta_1\beta_2\cdots\beta_n + r_1\beta_2\cdots\beta_n + r_2\beta_3\cdots\beta_n + \cdots + r_{n-1}\beta_n + r_n.$$

Let $\varepsilon > 0$ be given. We must show that there is some integer N such that $|y_{n+1}| < \varepsilon$ whenever $n > N$. The idea of the proof is to split the sum in the expression for y_{n+1} into two parts, and show that each can be made small for sufficiently large n. Thus, we write

$$\begin{aligned} y_{n+1} = (r_0\beta_1\beta_2\cdots\beta_n + \cdots + r_m\beta_{m+1}\cdots\beta_n) \\ + (r_{m+1}\beta_{m+2}\cdots\beta_n + \cdots + r_{n-1}\beta_n + r_n) \end{aligned}$$

and seek m so that each of the two bracketed terms on the right hand side is smaller than $\varepsilon/2$.

We know that $\sum_{j=1}^{\infty} |r_j| < \infty$ and so there is some integer m such that $\sum_{j=m+1}^{n} |r_j| < \varepsilon/2$, for $n > m$. Furthermore, the inequality $|1-\alpha_j| \le 1$ implies that $0 < \beta_j \le 1$ and so, with m as above,

$$\begin{aligned} |(r_{m+1}\beta_{m+2}\cdots\beta_n + \cdots + r_{n-1}\beta_n + r_n)| &\le |r_{m+1}| + \cdots + |r_n| \\ &= \sum_{j=m+1}^{n} |r_j| < \frac{\varepsilon}{2} \end{aligned}$$

which deals with the second bracketed term in the expression for y_{n+1}.

To estimate the first term, we rewrite it as

$$(r_0' + r_1' + \cdots + r_m')\beta_1\beta_2\cdots\beta_n\,,$$

where we have set $r_0' = r_0$ and $r_j' = \dfrac{r_j}{\beta_1\cdots\beta_j}$, for $j > 0$.

We claim that $\beta_1\cdots\beta_n \to 0$ as $n \to \infty$. To see this, we use the inequality

$$\log(1-t) \le -t, \qquad \text{for } 0 \le t < 1,$$

which can be derived as follows. We have

$$\begin{aligned} -\log(1-t) &= \int_{1-t}^{1} \frac{dx}{x} \\ &\ge \int_{1-t}^{1} dx, \quad \text{since } \frac{1}{x} \ge 1 \text{ in the range of integration,} \\ &= t, \end{aligned}$$

which gives $\log(1-t) \le -t$, as required. Using this, we may say that $\log(1-\alpha_j) \le -\alpha_j$, and so $\sum_{j=1}^{n} \log(1-\alpha_j) \le -\sum_{j=1}^{n} \alpha_j$. Thus

$$\log(\beta_1\cdots\beta_n) = \log\prod_{j=1}^{n}(1-\alpha_j)^2 = 2\sum_{j=1}^{n}\log(1-\alpha_j) \le -2\sum_{j=1}^{n}\alpha_j.$$

But $\sum_{j=1}^{n} \alpha_j \to \infty$ as $n \to \infty$, which means that $\log(\beta_1\cdots\beta_n) \to -\infty$ as $n \to \infty$, which, in turn, implies that $\beta_1\cdots\beta_n \to 0$ as $n \to \infty$, as claimed.

Finally, we observe that for m as above, the numbers $r'_0, r'_1, \dots, r'_m$ do not depend on n. Hence there is N, with $N > m$, such that

$$|(r'_0 + r'_1 + \cdots + r'_m)\beta_1\beta_2\cdots\beta_n| < \frac{\varepsilon}{2}$$

whenever $n > N$. This completes the proof that $y_n \to 0$ as $n \to \infty$. ∎

It follows, therefore, that $m^{(1)}(n) \to m_* = \left(\frac{c_1 - c_2}{2}\right)$, as $n \to \infty$. To investigate $m^{(2)}(n)$, we use the relation

$$m^{(2)}(n) = \underbrace{(1-\alpha_n)}_{\to 1}\underbrace{m^{(1)}(n)}_{\to m_*} + \underbrace{\alpha_n}_{\to 0} c_1$$

to see that $m^{(2)}(n) \to m_*$ also.

Thus, for this special simple example, we have demonstrated the convergence of the LMS algorithm. The statement of the general case is as follows.

Theorem 2.10 (LMS Convergence Theorem). *Suppose that the learning rate α_n in the LMS algorithm satisfies the conditions*

$$\sum_{n=1}^{\infty} \alpha_n = \infty \qquad \text{and} \qquad \sum_{n=1}^{\infty} \alpha_n^2 < \infty.$$

Then the sequence of matrices generated by the algorithm, initialized by the condition $M^{(1)}(1) = 0$, converges to $BA^{\#}$, the OLAM matrix.

We will not present the proof here, which involves the explicit form of the generalized inverse, as given via the Singular Value Decomposition. For the details, we refer to the original paper of Luo.

The ALC can be used to “clean-up” a noisy signal by arranging it as a transverse filter. Using delay mechanisms, the “noisy” input signal is sampled n times, that is, its values at time steps $\tau, 2\tau, \dots, n\tau$ are collected. These n values form the fan-in values for the ALC. The output error is $\varepsilon = |d - y|$, where d is the desired output, i.e., the pure signal. The network is trained to minimize ε, so that the system output y is as close to d as possible (via the LMS error). Once trained, the network produces a “clean” version of the signal.

The ALC has applications in echo cancellation in long distance telephone calls. The ’phone handset contains a special circuit designed to distinguish between incoming and outgoing signals. However, there is a certain amount of signal leakage from the earpiece to the mouthpiece. When the caller speaks, the message is transmitted to the recipient’s earpiece via satellite.

Some of this signal then leaks across to the recipient's mouthpiece and is sent back to the caller. The time taken for this is about half a second, so that the caller hears an echo of his own voice. By appropriate use of the ALC in the circuit, this echo effect can be reduced.

Chapter 3

Artificial Neural Networks

By way of background information, we consider some basic neurophysiology. It has been estimated that the human brain contains some 10^{11} nerve cells, or neurons, each having perhaps as many as 10^4 interconnections, thus forming a densely packed web of fibres.

The neuron has three major components:

- the dendrites (constituting a vastly multibranching tree-like structure which collects inputs from other cells),
- the cell body (the processing part, called the soma),
- the axon (which carries electrical pulses to other cells).

Each neuron has only one axon, but it may branch out and so may be able to reach perhaps thousands of other cells. There are many dendrites (the word dendron is Greek for tree). The diameter of the soma is of the order of 10 microns.

The outgoing signal is in the form of a pulse down the axon. On arrival at a synapse (the junction where the axon meets a dendrite, or indeed, any other part of another nerve cell) molecules, called neurotransmitters are released. These cross the synaptic gap (the axon and receiving neuron do not quite touch) and attach themselves, very selectively, to receptor sites on the receiving neuron. The membrane of the target neuron is chemically affected and its own inclination to fire may be either enhanced or decreased. Thus, the incoming signal can be correspondingly either excitatory or inhibitory. Various drugs work by exploiting this behaviour. For example, curare deposits certain chemicals at particular receptor sites which artificially inhibit motor (muscular) stimulation by the brain cells. This results in the inability to move.

The containing wall of the cell is the cell membrane—a phospholipid bilayer. (A lipid is, by definition, a compound insoluble in water, such as oils and fatty acids.) These bilayers have a phosphoric acid head which is attracted to water and a glyceride tail which is repelled by water. This means that in a water solution they tend to line up in a double layer with the heads pointing outwards.

The membrane keeps most molecules from passing either in or out of the cell, but there are special channels allowing the passage of certain ions such as Na^+, K^+, Cl^- and Ca^{++}. By allowing such ions to pass in and out, a potential difference between the inside and the outside of the cell is maintained. The cell membrane is selectively more favourable to the passage of potassium than to sodium so that the K^+ ions could more easily diffuse out, but negative organic ions inside tend to pull K^+ ions into the cell. The net result is that the K^+ concentration is higher inside than outside whereas the reverse is true of Na^+ and Cl^- ions. This results in a resting potential inside relative to outside of about -70mV across the cell wall.

When an action potential reaches a synapse it causes a change in the permeability of the membrane of the cell carrying the pulse (the presynaptic membrane) which results in an influx of Ca^{++} ions. This leads to a release of neurotransmitters into the synaptic cleft which diffuse across the gap and attach themselves at receptor sites on the membrane of the receiving cell (the postsynaptic membrane). As a consequence, the permeability of the postsynaptic membrane is altered. An influx of positive ions will tend to depolarize the receiving neuron (causing excitation) whereas an influx of negative ions will increase polarization and so inhibit activation.

Each input pulse is of the order of 1 millivolt and these diffuse towards the body of the cell where they are summed at the axon hillock. If there is sufficient depolarization the membrane permeability changes and allows a large influx of Na^+ ions. An action potential is generated and travels down the axon away from the main cell body and off to other neurons. The amplitude of this signal is of the order of tens of millivolts and its presence prevents the axon from transmitting further pulses. The shape and amplitude of this travelling pulse is very stable and is replicated at the branching points of the axon. This would indicate that the pulse seems not to carry any information other than to indicate its presence, i.e., the axon can be thought of as being in an all-or-none state.

Once triggered, the neuron is incapable of re-excitation for about one millisecond, during which time it is restored to its resting potential. This is called the refractory period. The existence of the refractory period limits the frequency of nerve-pulse transmissions to no more than about 1000 per second. In fact, this frequency can vary greatly, being mere tens of pulses per second in some cases. The impulse trains can be in the form of regular spikes, irregular spikes or in bursts.

The big question is how this massively interconnected network constituting the brain can not only control general functional behaviour but also give rise to phenomena such as personality, sleep and consciousness. It is also amazing how the brain can recognize something it has not "seen" before. For example, a piece of badly played music, or writing roughly done, say, can nevertheless be perfectly recognized in the sense that there is no doubt in one's mind (*sic*) what the tune or the letters actually "are". (Indeed, surely no real-life experience can ever be an *exact* replica of a previous experience.) This type of ability seems to be very hard indeed to reproduce by computer. One should take care in discussions of this kind, since we are apparently talking about the functioning of the brain in self-referential terms. After all, perhaps if we knew (whatever "know" means) what a tune "is", i.e., how it relates to the brain via our hearing it (or, indeed, seeing the musical score written down) then we might be able to understand how we can recognize it even in some new distorted form.

In this connection, certain cells do seem to perform as so-called "feature detectors". One example is provided by auditory cells located at either side of the back of the brain near the base and serve to locate the direction of sounds. These cells have two groups of dendrites receiving inputs originating, respectively, from the left ear and the right ear. For those cells in the left side of the brain, the inputs from the left ear inhibit activation, whereas those from the right are excitatory. The arrangement is reversed for those in the right side of the brain. This means that a sound coming from the right, say, will reach the right ear first and hence initially excite those auditory cells located in the left side of the brain but inhibit those in the right side. When the sound reaches the left ear, the reverse happens, the cells on the left become inhibited and those on the right side of the brain become excited. The change from strong excitation to strong inhibition can take place within a few hundred microseconds.

Another example of feature detection is provided by certain visual cells. Imagine looking at a circular region which is divided into two by a smaller concentric central disc and its surround. Then there are cells in the visual system which become excited when a light appears in the centre but for which activation is inhibited when a light appears in the surround. These are called "on centre–off surround" cells. There are also corresponding "off centre–on surround" cells.

We would like to devise mathematical models of networks inspired by (our understanding of) the workings of the brain. The study of such artificial neural networks may then help us to gain a greater understanding of the workings of the brain. In this connection, one might then strive to make the models more biologically realistic in a continuing endeavour to model the brain. Presumably one might imagine that sooner or later the detailed biochemistry of the neuron will have to be taken into account. Perhaps one

might even have to go right down to a quantum mechanical description. This seems to be a debatable issue in that there is a school of thought which suggests that the details are not strictly relevant and that it is the overall cooperative behaviour which is important for our understanding of the brain. This situation is analogous to that of the study of thermodynamics and statistical mechanics. The former deals essentially with gross behaviour of physical systems whilst the latter is concerned with a detailed atomic or molecular description in the hope of explaining the former. It turned out that the detailed (and quantum) description was needed to explain certain phenomena such as superconductivity. Perhaps this will turn out to be the case in neuroscience too.

On the other hand, one could simply develop the networks in any direction whatsoever and just consider them for their own sake (as part of a mathematical structure), or as tools in artificial intelligence and expert systems. Indeed, artificial neural networks have been applied in many areas including medical diagnosis, credit validation, stock market prediction, wine tasting and microwave cookers.

To develop the basic model, we shall think of the nervous system as mediated by the passage of electrical impulses between a vast web of interconnected cells—neurons. This network receives input from receptors, such as the rods and cones of the eye, or the hot and cold touch receptors of the skin. These inputs are then processed in some way by the neural net within the brain, and the result is the emission of impulses that control so-called effectors, such as muscles, glands etc., which result in the response. Thus, we have a three-stage system: input (via receptors), processing (via neural net) and output (via effectors).

To model the excitatory/inhibitory behaviour of the synapse, we shall assign suitable positive weights to excitatory synapses and negative weights to the inhibitory ones. The neuron will then "fire" if its total weighted input exceeds some threshold value. Having fired, there is a small delay, the refractory period, before it is capable of firing again. To take this into account, we consider a discrete time evolution by dividing the time scale into units equal to the refractory period. Our concern is whether any given neuron has "spiked" or not within one such period. This has the effect of "clocking" the evolution. We are thus led to the caricature illustrated in the figure.

The symbols $x_1, \dots, x_n$ denote the input values, $w_1, \dots, w_n$ denote the weights associated with the connections (terminating at the synapses). $u = \sum_{i=1}^{n} w_i x_i$ is the net (weighted) input, θ is the threshold, $v = u - \theta$ is called the activation potential, and $\varphi(\cdot)$ the activation function. The output y is given by

$$y = \varphi(\underbrace{u - \theta}_{v}).$$

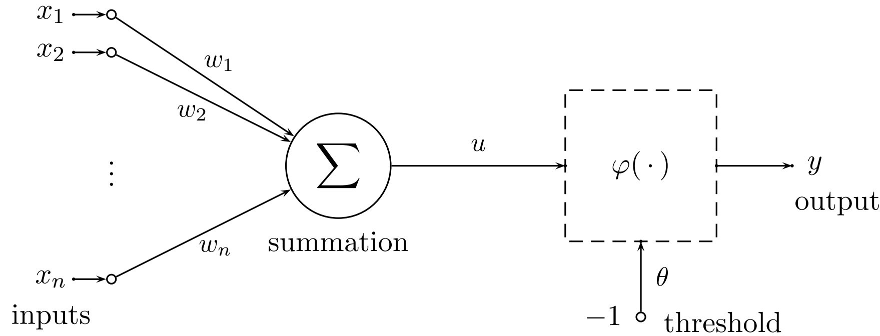


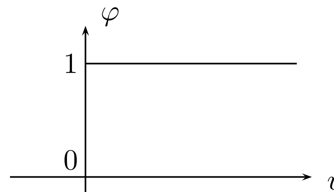
Figure 3.1: A model of a neuron as a processing device.

Typically, $\varphi(\cdot)$ is a *non-linear* function. Commonly used forms for $\varphi(\cdot)$ are the binary and bipolar threshold functions, the piece-wise linear function ("hard-limited" linear function), and the so-called sigmoid function. Examples of these are as follows.

Examples 3.1.

1. Binary threshold function:

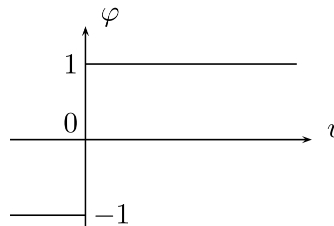
$$\varphi(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0. \end{cases}$$



   Thus, the output y is 1 if $u - \theta \geq 0$, i.e., if $u \geq \theta$, or, equivalently, if $\sum w_i x_i \geq 0$. A neuron with such an activation function is known as a McCulloch-Pitts neuron. It is an "all or nothing" neuron. We will sometimes write step(v) for such a binary threshold function $\varphi(v)$.

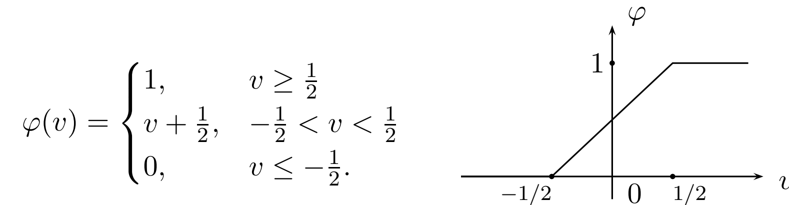
2. Bipolar threshold function:

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



   This is just like the binary version, but the "off" output is represented as -1 rather than as 0. We might call this a bipolar McCulloch-Pitts neuron, and denote the function $\varphi(\cdot)$ by sign$(\cdot)$.

3. Hard-limited linear function:



4. Sigmoid function

   A sigmoid function is any differentiable function $\varphi(\cdot)$, say, such that $\varphi(v) \to 0$ as $v \to -\infty$, $\varphi(v) \to 1$ as $v \to \infty$ and $\varphi'(v) > 0$.

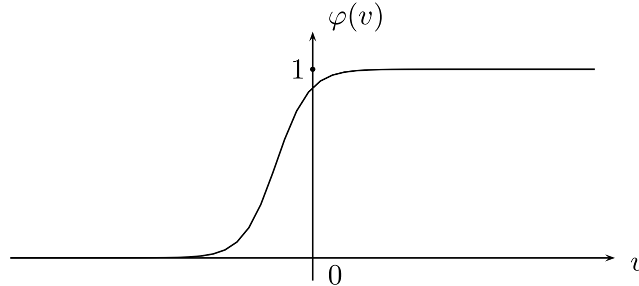


Figure 3.2: A sigmoid function.

A specific example of a sigmoid function is given by

$$\varphi : v \mapsto \varphi(v) = \frac{1}{1 + \exp(-\alpha v)}.$$

The larger the value of the constant parameter α, the greater is the slope. (The slope is sometimes called the "gain".) In the limit when $\alpha \to \infty$, this sigmoid function becomes the binary threshold function (except for the single value $v = 0$, for which $\varphi(0)$ is equal to $1/2$, for all α). One could call this the threshold limit of φ.

If we want the output to vary between -1 and $+1$, rather than 0 and 1, we could simply change the definition to demand that $\varphi(v) \to -1$, as $v \to \infty$, thus defining a bipolar sigmoid function. One can easily transform between binary and bipolar sigmoid functions. For example, if φ is a binary sigmoid function, then $2\varphi - 1$ is a bipolar sigmoid function.

For the explicit example above, we see that

$$2\varphi(v) - 1 = \frac{1 - \exp(-\alpha v)}{1 + \exp(\alpha v)} = \tanh\Big(\frac{\alpha v}{2}\Big).$$

We turn now to a slightly more formal discussion of neural network. We will not attempt a definition in the axiomatic sense (there seems little point at this stage), but rather enumerate various characteristics that typify a neural network.

- Essentially, a neural network is a decorated directed graph.
- A directed graph is a set of objects, called nodes, together with a collection of directed links, that is, ordered pairs of nodes. Thus, the ordered pair of nodes $\{u, v\}$ is thought of as a line joining the node u to the node v in the direction *from u to v*. A link is usually called a connection (or wire) and the nodes are called processing units (or elements), neurons or artificial neurons.
- Each connection can carry a signal, i.e., to each connection we may assign a real number, called a signal. The signal is thought of as travelling in the direction of the link.
- Each node has "local memory" in the form of a collection of real numbers, called weights, each of which is assigned to a corresponding terminating i.e., incoming connection and represents the synaptic efficacy.
- Moreover, to each node there is associated some activation function which determines the signals along its outgoing connections based only on local memory—such as the weights associated with the incoming connections and the incoming signals. All outgoing signals from a particular node are equal in value.
- Some nodes may be specified as input nodes and others as output nodes. In this way, the neural network can communicate with the external world. One could consider a node with only outgoing links as an input node (source) and one with only incoming links as an output node (sink).
- In practical applications, a neural network is expected to be an interconnected network of very many (possibly thousands) of relatively simple processing units. That is to say, the effectiveness of the network is expected to come about because of the complexity of the interconnections rather than through any particularly clever behaviour of the individual neurons. The performance of a neural network is also expected to be robust in the sense of relative insensitivity to the removal of a small number of connections or variations in the values of a few of the weights.

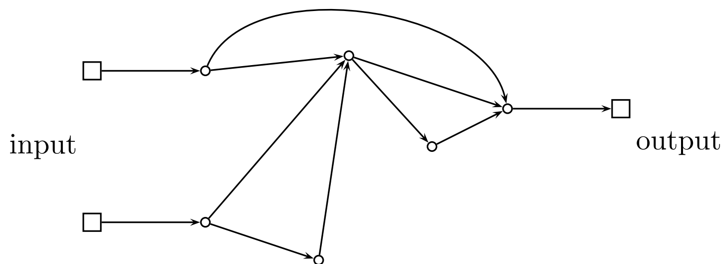


Figure 3.3: An example neural network.

Often neural networks are arranged in layers such that the connections are only between consecutive layers, all in the same direction, and there being no connections within any given layer. Such neural networks are called feedforward neural networks.

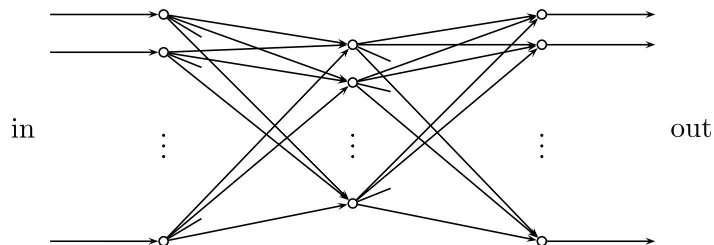


Figure 3.4: A three layer feedforward neural network.

Note that sometimes the input layer of a multilayer feedforward neural network is not counted as a layer—this is because it usually serves just to provide "place-holders" for the inputs. Thus, the example in the figure might sometimes be referred to as a two-layer feedforward neural network. We will always count the first layer.

A neural network is said to be recurrent if it possesses at least one feedback connection.

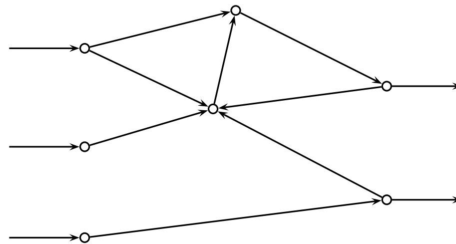


Figure 3.5: An example of a recurrent neural network.

Chapter 4

The Perceptron

The perceptron is a simple neural network invented by Frank Rosenblatt in 1957 and subsequently extensively studied by Marvin Minsky and Seymour Papert (1969). It is driven by McCulloch-Pitts threshold processing units equipped with a particular learning algorithm.

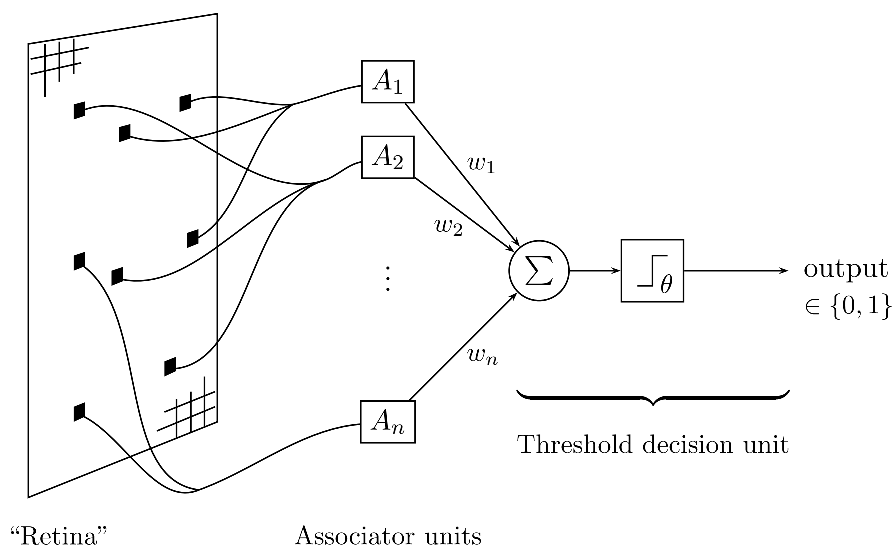


Figure 4.1: The perceptron architecture.

The story goes that after much hype about the promise of neural networks in the fifties and early sixties (for example, that artificial brains would soon be a reality), Minsky and Papert's work, elucidating the theory and vividly illustrating the limitations of the perceptron, was the catalyst for the decline of the subject and certainly (which is almost the same thing) the withdrawal of U. S. government funding. This led to a lull in research into neural networks, as such, during the period from the end of the sixties to the beginning of the eighties, but work did continue under the headings of adap-

tive signal processing, pattern recognition, and biological modelling. By the early to mid-eighties the subject was revived mainly thanks to the discovery of possible methods of surmounting the shortcomings of the perceptron and the contributions from the physics community.

The figure illustrates the architecture of a so-called simple perceptron, i.e., a perceptron with only a single output line.

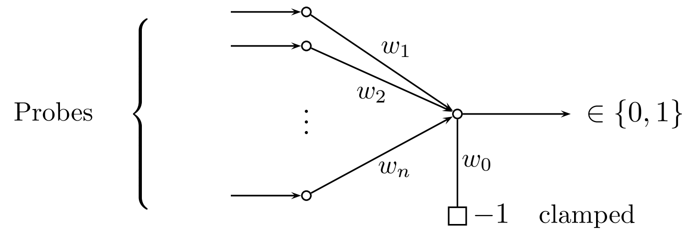
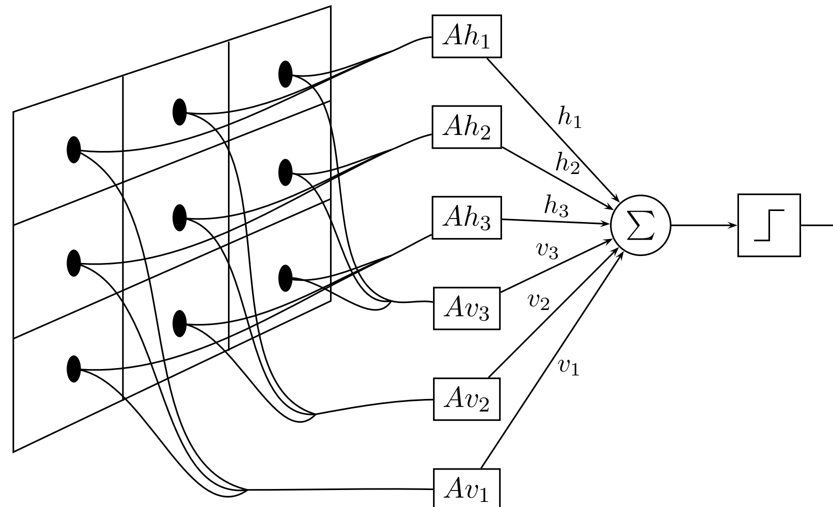


Figure 4.2: The simple perceptron.

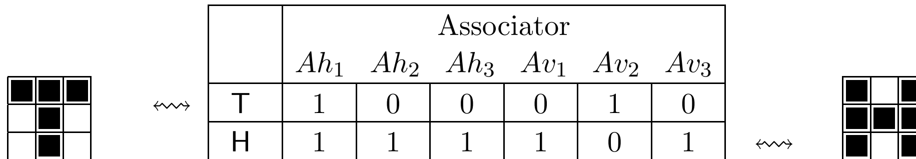
The associator units are thought of as "probing" the outside world, which we shall call the "retina", for pictorial simplicity, and then transmitting a binary signal depending on the result. For example, a given associator unit might probe a group of $m \times n$ pixels on the retina and output 1 if they are all "on", but otherwise output 0. Having set up a scheme of associators, we might then ask whether the system can distinguish between vowels and consonants drawn on the screen (retina).

Example 4.1. Consider a system comprising a retina formed by a 3×3 grid of pixels and six associator units $Ah_1, \dots, Av_3$ (see Aleksander, I. and H. Morton, 1991).



The three associator units Ah_i, $i = 1, 2, 3$, each have a 3-point horizontal receptive field (the three rows, respectively) and each Av_j, $j = 1, 2, 3$ has a

3-point vertical receptive field (the three columns). Each associator fires if and only if a majority of its probes are "on", i.e., each produces an output of 1 if and only if at least 2 of its 3 pixels are black. We wish to assign weights $h_1, \ldots, v_3$, to the six connections to the binary threshold unit so that the system can successfully distinguish between the letters T and H so that the network has an output of 1, say, corresponding to T, and an output of 0 corresponding to H.



	Associator					
	Ah_1	Ah_2	Ah_3	Av_1	Av_2	Av_3
T	1	0	0	0	1	0
H	1	1	1	1	0	1

To induce the required output, we seek values for $h_1, \ldots, v_3$ and θ such that

$$1h_1 + 0h_2 + 0h_3 + 0v_1 + 1v_2 + 0v_3 > \theta,$$

but

$$1h_1 + 1h_2 + 1h_3 + 1v_1 + 0v_2 + 1v_3 < \theta.$$

That is, $h_1 + v_2 > \theta$ and $h_1 + h_2 + h_3 + v_1 + v_3 < \theta$.

This will ensure that the binary decision output unit responds correctly. There are many possibilities here. One such is to try $h_1 = v_2 = 1$. Then the first of these inequalities demands that $2 > \theta$, whereas the second inequality requires that $1 + h_2 + h_3 + v_1 + v_3 < \theta$. Setting $h_2 = h_3 = v_1 = v_3 = -1$ and $\theta = 0$ does the trick.

The retina image corresponds to the associator output vector given by $(Ah_1, \ldots, Av_3) = (1, 0, 0, 0, 1, 0)$, which we see induces the weighted net input $h_1 + v_2 = 1 + 1 = 2$ to the binary decision unit. Thus the output is 1, which is associated with the letter T.

On the other hand, the retina image leads to the associator output vector $(Ah_1, \ldots, Av_3) = (1, 1, 1, 1, 0, 1)$ which induces the weighted net input $h_1 + h_2 + h_3 + v_1 + v_3 = 1 - 1 - 1 - 1 - 1 = -3$ to the binary decision unit. The output is 0 which is associated with the letter H.

It is of interest to look at an example of the limitation of the perceptron (after Minsky and Papert). We shall illustrate an example of a global property (connectedness) that cannot be measured by local probes. Let us say that an associator unit has diameter d if the smallest square containing its support pixels (i.e., those pixels in the retina on whose values it depends) has side of d pixels. We shall now see that if a perceptron has diameter d, then, no matter how many associator units it has, there are tasks it simply cannot perform, such as check for connectedness.

Theorem 4.2. *It is impossible to implement "connectedness" on a retina of width $r > d$ by a perceptron whose associator units have diameter no greater than d.*

Proof. The proof is by contradiction. Suppose that we have a perceptron whose probes have diameters not greater than d and that this perceptron can recognise connectedness, that is, its output is 1 when it sees a connected figure on the retina and 0 otherwise. Consider the four figures indicated on a retina of width $r > d$.

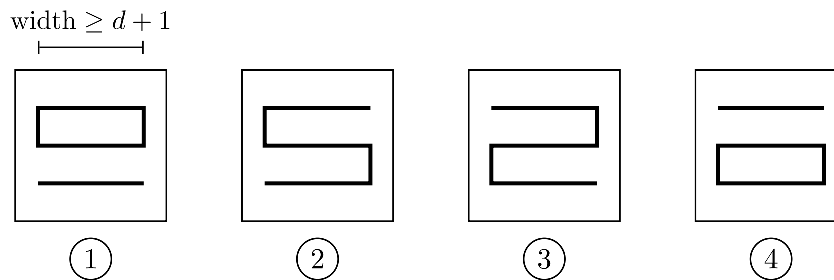


Figure 4.3: Figures to fox the perceptron.

The perceptron associator units can be split into three distinct groups:

A — those which probe the left-hand end of the figure,

B — those which probe neither end,

C — those which probe the right-hand end of the figure.

The contribution to the output unit's activation can therefore be represented as a sum of the three corresponding contributions, symbolically,

$$\Sigma = \Sigma_A + \Sigma_B + \Sigma_C \,.$$

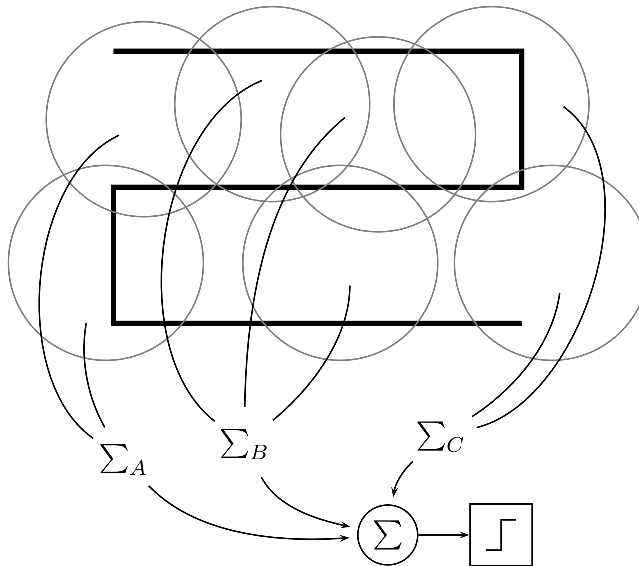


Figure 4.4: Contributions to the output activation.

On presentation of pattern (1), the output should be 0, so that $\Sigma^{(1)} < \theta$, that is,

$$\Sigma_A^{(1)} + \Sigma_B^{(1)} + \Sigma_C^{(1)} < \theta.$$

On presentation of patterns (2) or (3), the output should be 1, so

$$\Sigma_A^{(2)} + \Sigma_B^{(2)} + \Sigma_C^{(2)} > \theta$$
$$\Sigma_A^{(3)} + \Sigma_B^{(3)} + \Sigma_C^{(3)} > \theta.$$

Finally, presentation of pattern (4) should give output 0, so

$$\Sigma_A^{(4)} + \Sigma_B^{(4)} + \Sigma_C^{(4)} < \theta.$$

However, the contributions $\Sigma_B^{(1)}$, $\Sigma_B^{(2)}$, $\Sigma_B^{(3)}$ and $\Sigma_B^{(4)}$ are all equal because the class B probes cannot tell the difference between the four figures.

Similarly, we see that $\Sigma_A^{(1)} = \Sigma_A^{(2)}$ and $\Sigma_C^{(1)} = \Sigma_C^{(3)}$, and $\Sigma_C^{(2)} = \Sigma_C^{(4)}$ and $\Sigma_A^{(3)} = \Sigma_A^{(4)}$. To simplify the notation, let X, X' and Y, Y' denote the two different A and C contributions, respectively, and set $\alpha = \theta - \Sigma_B^{(1)}$. Then

we can write our four inequalities as

$$\begin{aligned} X + Y &< \alpha && \text{from figure (1)} \\ X + Y' &> \alpha && \text{from figure (2)} \\ X' + Y &> \alpha && \text{from figure (3)} \\ X' + Y' &< \alpha && \text{from figure (4).} \end{aligned}$$

Clearly, the first two inequalities imply that $Y' > Y$ which together with the third gives $X' + Y' > X' + Y > \alpha$. This contradicts the fourth inequality. We conclude that there can be no such perceptron. ■

Two-class classification

We wish to set up a network which will be able to differentiate inputs from one of two categories. To this end, we shall consider a simple perceptron comprising many input units and a single binary decision threshold unit. We shall ignore the associator units as such and just imagine them as providing inputs to the binary decision unit. (One could imagine the system comprising as many associator units as pixels and where each associator unit probes just one pixel to see if it is "on" or not.) However, we shall allow the input values to be any real numbers, rather than just binary digits.

Suppose then that we have a finite collection of vectors in $\mathbb{R}^n$, each of which is classified as belonging to one of two categories, $\mathcal{S}_1$, $\mathcal{S}_2$, say. If the vector $(x_1, \dots, x_n)$ is presented to a linear binary threshold unit with weights $w_1, \dots, w_n$ and threshold θ, then the output, z, say, is

$$z = \text{step}\Big(\textstyle\sum_{i=1}^n w_i x_i - \theta\Big) = \begin{cases} 1, & \text{if } \sum_{i=1}^n w_i x_i \ge \theta \\ 0, & \text{otherwise.} \end{cases}$$

We wish to find values of $w_1, \dots, w_n$ and θ such that inputs in one of the categories give output $z = 1$, whilst those in the other category yield output $z = 0$. The means of attempting to achieve this will be to modify the weights w_i and the threshold θ via an error-correction learning procedure—the so-called "perceptron learning rule".

For notational convenience, we shall reformulate the problem. Let $w_0 = \theta$ and set $x_0 = -1$. Then

$$\sum_{i=1}^n w_i x_i - \theta = \sum_{i=0}^n w_i x_i.$$

For given $x \in \mathbb{R}^n$, let $y = (x_0, x) = (-1, x) \in \mathbb{R}^{n+1}$ and let $w \in \mathbb{R}^{n+1}$ be the vector given by $w = (w_0, w_1, \dots, w_n)$. The two categories of vectors $\mathcal{S}_1$ and $\mathcal{S}_2$ in $\mathbb{R}^n$ determine two categories, $\mathcal{C}_1$ and $\mathcal{C}_2$, say, in $\mathbb{R}^{n+1}$ via the above

augmentation, $x \mapsto (-1, x)$. The problem is to find $w \in \mathbb{R}^{n+1}$ such that $w \cdot y > 0$ for $y \in \mathcal{C}_1$ and $w \cdot y < 0$ for $y \in \mathcal{C}_2$, where

$$w \cdot y = w^T y = w_0 y_0 + \cdots + w_n y_n$$

is the inner product of the vectors w and y in $\mathbb{R}^{n+1}$.

To construct an error-correction scheme, suppose that y is one of the input pattern vectors we wish to classify. If $y \in \mathcal{C}_1$, then we want $w \cdot y > 0$ for correct classification. If this is, indeed, the case, then we do not change the value of w. However, if the classification is incorrect, then we must have $w \cdot y < 0$. The idea is to change w to $w' = w + y$. This leads to

$$w' \cdot y = (w + y) \cdot y = w \cdot y + y \cdot y > w \cdot y$$

so that even if $w' \cdot y$ still fails to be positive, it will be less negative than $w \cdot y$ (note that $y = (x_0, x) \neq 0$, since $x_0 = -1$). In other words, w' is "nearer" to giving the correct classification for the particular pattern y than w was.

Similarly, for $y \in \mathcal{C}_2$, misclassification means that $w \cdot y \geq 0$ rather than $w \cdot y < 0$. Setting $w' = w - y$, gives

$$w' \cdot y = (w - y) \cdot y = w \cdot y - y \cdot y,$$

so that w' is nearer to the correct classification than w.

The perceptron error-correction rule, then, is to present the pattern vector y and leave w unchanged if y is correctly classified, but otherwise change w to w', where

$$w' = \begin{cases} w + y, & \text{if } y \in \mathcal{C}_1 \\ w - y, & \text{if } y \in \mathcal{C}_2. \end{cases}$$

The discussion can be simplified somewhat by introducing

$$\mathcal{C} = \mathcal{C}_1 \cup (-\mathcal{C}_2) = \{y : y \in \mathcal{C}_1, \text{ or } -y \in \mathcal{C}_2\}.$$

Then $w \cdot y > 0$ for all $y \in \mathcal{C}$ would give correct classification. The error-correction rule now becomes the following.

The perceptron error-correction rule:

- present the pattern $y \in \mathcal{C}$ to the network;
- leave the weight vector w unchanged if $w \cdot y > 0$,
- otherwise change w to $w' = w + y$.

One would like to present the input patterns one after the other, each time following this error-correction rule. Unfortunately, whilst changes in the vector w may enhance classification for one particular input pattern, these changes may spoil the classification of other patterns and this correction procedure may simply go on forever. The content of the Perceptron Convergence Theorem is that, in fact, under suitable circumstances, this endless loop does not occur. After a finite number of steps, further corrections become unnecessary.

Definition 4.3. We say that two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ in $\mathbb{R}^n$ are linearly separable if and only if there is some $(w_1, \dots, w_n) \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ such that $\sum_{i=1} w_i x_i > \theta$ for each $x \in \mathcal{S}_1$ and $\sum_{i=1} w_i x_i < \theta$ for each $x \in \mathcal{S}_2$.

Thus, $\mathcal{S}_1$ and $\mathcal{S}_2$ are linearly separable if and only if they lie on opposite sides of some hyperplane $\sum_{i=1} w_i x_i = \theta$ in $\mathbb{R}^n$. For example, in two dimensions, $\mathcal{S}_1$ and $\mathcal{S}_2$ are linearly separable if and only if they lie on opposite sides of some line $w_1 x_1 + w_2 x_2 = \theta$.

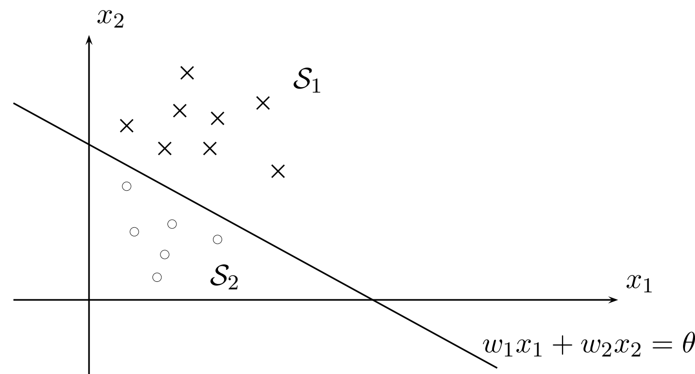


Figure 4.5: The line separates the two sets of points.

Notice that the introduction of the threshold as an extra parameter allows the construction of such a separating line. Without the threshold term, the line would have to pass through the origin, severely limiting its separation ability.

Theorem 4.4 (Perceptron Convergence Theorem). *Suppose that $\mathcal{S}_1$ and $\mathcal{S}_2$ are linearly separable finite subsets of pattern vectors in $\mathbb{R}^n$. Let $\mathcal{C} = \mathcal{C}_1 \cup (-\mathcal{C}_2)$, where $\mathcal{C}_1$ and $\mathcal{C}_2$ are the augmentations of $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively, via the map $x \mapsto y = (-1, x) \in \mathbb{R}^{n+1}$. Let $(y^{(k)})$ be any given sequence of pattern vectors such that $y^{(k)} \in \mathcal{C}$ for each $k \in \mathbb{N}$, and such that each $y \in \mathcal{C}$ occurs infinitely often in $(y^{(k)})$. Let $w^{(1)} \in \mathbb{R}^{n+1}$ be arbitrary, and define*

$w^{(k+1)}$ *by the perceptron error-correction rule*

$$w^{(k+1)} = \begin{cases} w^{(k)}, & \text{if } w^{(k)} \cdot y^{(k)} > 0, \\ w^{(k)} + y^{(k)}, & \text{otherwise.} \end{cases}$$

Then there is N such that $w^{(N)} \cdot y > 0$ for all $y \in \mathcal{C}$, that is, after at most N update steps, the weight vector correctly classifies all the input patterns and no further weight changes take place.

Proof. The perceptron error-correction rule can be written as

$$w^{(k+1)} = w^{(k)} + \alpha^{(k)} y^{(k)},$$

where $\alpha^{(k)} = \begin{cases} 0, & \text{if } w^{(k)} \cdot y^{(k)} > 0 \\ 1, & \text{otherwise,} \end{cases}$ for $k = 1, 2, \dots$.

Notice that $\alpha^{(k)} y^{(k)} \cdot w^{(k)} \le 0$, for all k. We have

$$\begin{aligned} w^{(k+1)} &= w^{(k)} + \alpha^{(k)} y^{(k)} \\ &= w^{(k-1)} + \alpha^{(k-1)} y^{(k-1)} + \alpha^{(k)} y^{(k)} \\ &= \qquad \cdots \\ &= w^{(1)} + \alpha^{(1)} y^{(1)} + \alpha^{(2)} y^{(2)} + \cdots + \alpha^{(k)} y^{(k)}. \end{aligned}$$

By hypothesis, $\mathcal{S}_1$ and $\mathcal{S}_2$ are linearly separable. This means that there is some $\widehat{w} \in \mathbb{R}^{n+1}$ such that $\widehat{w} \cdot y > 0$ for each $y \in \mathcal{C}$. Let $\delta = \min\{\widehat{w} \cdot y : y \in \mathcal{C}\}$. Then $\delta > 0$ and $\widehat{w} \cdot y^{(k)} \ge \delta$ for all $k \in \mathbb{N}$. Hence

$$\begin{aligned} w^{(k+1)} \cdot \widehat{w} &= w^{(1)} \cdot \widehat{w} + \alpha^{(1)} y^{(1)} \cdot \widehat{w} + \alpha^{(2)} y^{(2)} \cdot \widehat{w} + \cdots + \alpha^{(k)} y^{(k)} \cdot \widehat{w} \\ &\ge w^{(1)} \cdot \widehat{w} + (\alpha^{(1)} + \alpha^{(2)} + \cdots + \alpha^{(k)})\delta. \end{aligned}$$

However, by Schwarz' inequality, $|w^{(k+1)} \cdot \widehat{w}| \le \|w^{(k+1)}\| \|\widehat{w}\|$ and so

$$\big(w^{(1)} \cdot \widehat{w} + (\alpha^{(1)} + \alpha^{(2)} + \cdots + \alpha^{(k)})\delta\big) \le \|w^{(k+1)}\| \, \|\widehat{w}\|. \qquad (*)$$

On the other hand,

$$\begin{aligned} \|w^{(k+1)}\|^2 &= w^{(k+1)} \cdot w^{(k+1)} \\ &= (w^{(k)} + \alpha^{(k)} y^{(k)}) \cdot (w^{(k)} + \alpha^{(k)} y^{(k)}) \\ &= \|w^{(k)}\|^2 + \underbrace{2\alpha^{(k)} y^{(k)} \cdot w^{(k)}}_{\le 0} + \alpha^{(k)2} \|y^{(k)}\|^2 \\ &\le \|w^{(k)}\|^2 + \alpha^{(k)} \|y^{(k)}\|^2, \text{ since } \alpha^{(k)2} = \alpha^{(k)}, \\ &\le \qquad \cdots \\ &\le \|w^{(1)}\|^2 + \alpha^{(1)} \|y^{(1)}\|^2 + \alpha^{(2)} \|y^{(2)}\|^2 + \cdots + \alpha^{(k)} \|y^{(k)}\|^2 \\ &\le \|w^{(1)}\|^2 + \big(\alpha^{(1)} + \alpha^{(2)} + \cdots + \alpha^{(k)}\big) K \qquad (**) \end{aligned}$$

where $K = \max\{\|y\|^2 : y \in \mathcal{C}\}$. Let $N^{(k)} = \alpha^{(1)} + \alpha^{(2)} + \cdots + \alpha^{(k)}$. Then $N^{(k)}$ is the total number of non-zero weight adjustments that have been made during the first k applications of the perceptron correction rule. Combining the inequalities $(*)$ and $(**)$, we obtain an inequality for $N^{(k)}$ of the form

$$N^{(k)} \le b + c\sqrt{N^{(k)}}\,,$$

where b and c are constants, i.e., independent of k. In fact, b and c depend only on $w^{(1)}$, $\widehat{w}$, δ and K. It follows from this inequality that there is an upper bound to the possible values of $N^{(k)}$, that is, there is γ, independent of k, such that $N^{(k)} < \gamma$ for all k. Since $N^{(k+1)} \ge N^{(k)}$, it follows that $(N^{(k)})$ is eventually constant, which means that $\alpha^{(k)}$ is eventually zero. Thus, there is $N \in \mathbb{N} \cup \{0\}$ such that $\alpha^{(N)} = \alpha^{(N+1)} = \cdots = 0$, i.e., $w^{(N)} \cdot y > 0$ for all $y \in \mathcal{C}$ and no weight changes are needed after the Nth step of the error-correction algorithm. ■

Remark 4.5. The assumption of linear separability ensures the existence of a suitable $\widehat{w}$. We do *not* need to know what $\widehat{w}$ actually is, its mere existence suffices to guarantee convergence of the algorithm to a fixed point after a finite number of iterations.

By keeping track of the constants in the inequalities, one can estimate N, the upper bound for the number of required steps to ensure convergence, in terms of $\widehat{w}$, but if we knew $\widehat{w}$ then there would be no need to perform the algorithm in the first place. We could simply assign $w = \widehat{w}$ to obtain a valid network.

It is possible to extend the theorem to allow for an infinite number of patterns, but the hypotheses must be sharpened slightly. The relevant concept needed is that of strict linear separability.

Definition 4.6. The non-empty subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ of $\mathbb{R}^\ell$ are said to be strictly linearly separable if and only if there is $v \in \mathbb{R}^\ell$, $\theta \in \mathbb{R}$ and some $\delta > 0$ such that

$$v \cdot x > \theta + \delta \text{ for all } x \in \mathcal{S}_1$$

and

$$v \cdot x < \theta - \delta \text{ for all } x \in \mathcal{S}_2.$$

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be the subsets of $\mathbb{R}^{\ell+1}$ obtained by augmenting $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively, via the map $x \mapsto (-1, x)$, as usual. Then $\mathcal{S}_1$ and $\mathcal{S}_2$ are strictly linearly separable if and only if there is $\widehat{w} \in \mathbb{R}^{\ell+1}$ and $\delta > 0$ such that $\widehat{w} \cdot y > \delta$ for all $y \in \mathcal{C}_1 \cup (-\mathcal{C}_2)$. The perceptron convergence theorem for infinitely-many input pattern vectors can be stated as follows.

Theorem 4.7. *Let $\mathcal{C}$ be any bounded subset of $\mathbb{R}^{\ell+1}$ such that there is some $\widehat{w} \in \mathbb{R}^{\ell+1}$ and $\delta > 0$ such that $\widehat{w} \cdot y > \delta$ for all $y \in \mathcal{C}$. For any given sequence $(y^{(k)})$ in $\mathcal{C}$, define the sequence $(w^{(k)})$ by*

$$w^{(k+1)} = w^{(k)} + \alpha^{(k)} y^{(k)}, \text{ where } \alpha^{(k)} = \begin{cases} 0, & \text{if } w^{(k)} \cdot y^{(k)} > 0 \\ 1, & \text{otherwise,} \end{cases}$$

and $w^{(1)} \in \mathbb{R}^{\ell+1}$ is arbitrary. Then there is M (not depending on the particular sequence $(y^{(k)})$) such that $N^{(k)} = \alpha^{(1)} + \alpha^{(2)} + \cdots + \alpha^{(k)} \le M$ for all k. In other words, $\alpha^{(k)} = 0$ for all sufficiently large k and so there can be at most a finite number of non-zero weight changes.

Proof. By hypothesis, $\mathcal{C}$ is bounded and so there is some constant K such that $\|y\|^2 < K$ for all $y \in \mathcal{C}$. Exactly as before, we obtain an inequality

$$N^{(k)} \le b + c\sqrt{N^{(k)}}$$

where b and c only depend on $w^{(1)}$, $\widehat{w}$, K and δ. ■

Remark 4.8. This means that if $\mathcal{S}_1$ and $\mathcal{S}_2$ are strictly linearly separable, bounded subsets of $\mathbb{R}^\ell$, then there is an upper bound to the number of corrections that the perceptron learning rule is able to make. This does *not* mean that the perceptron will necessarily learn to separate $\mathcal{S}_1$ and $\mathcal{S}_2$ in a finite number of steps—after all, the update sequence may only "sample" some of the data a few times, or perhaps not even at all! Indeed, having chosen $w^{(1)} \in \mathbb{R}^{\ell+1}$ and the sequence $y^{(1)}, y^{(2)}, \dots$ of sample data, it may be that $w^{(1)} \cdot y^{(j)} > 0$ for all j, but nonetheless, if $y^{(1)}, y^{(2)}, \dots$ does not exhaust $\mathcal{C}$, there could be (possibly infinitely-many) points $y \in \mathcal{C}$ with $w^{(1)} \cdot y < 0$. Thus, the algorithm, starting with this particular $w^{(1)}$ and based on the particular sample patterns $y^{(1)}, y^{(2)}, \dots$ will not lead to a successful classification of $\mathcal{S}_1$ and $\mathcal{S}_2$.

Extended two-class classification

So far, we have considered classification into one of two possible classes. By allowing many output units, we can use a perceptron to classify patterns as belonging to one of a number of possible classes. Consider a fully connected neural network with n input and m output threshold units.

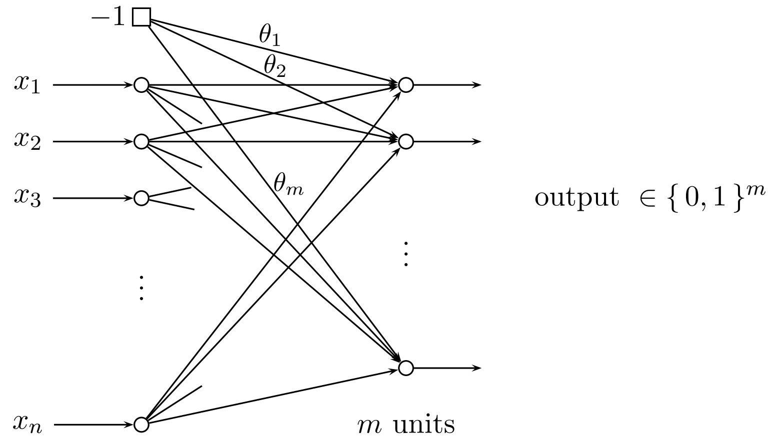


Figure 4.6: A perceptron with m output units.

The idea is to find weights and thresholds such that presentation of pattern j induces output for which only the jth unit is "on", i.e., all output units are zero except for the jth which has value 1. Notice that the system is equivalent to m completely independent simple perceptrons but with each being presented with the *same* input pattern. We are led to the following multiclass perceptron error-correction rule. As usual, we work with the augmented pattern vectors.

The extended two-class perceptron error-correction algorithm

- start with arbitrary weights (incorporating the threshold as zeroth component);
- present the first (augmented) pattern vector and observe the output values;
- using the usual (simple) perceptron error-correction rule, update all those weights which connect to an output with incorrect value;
- now present the next input pattern and repeat, cycling through all the patterns again and again, as necessary.

Of course, the question is whether or not this procedure ever leads to the correct classification of *all* m patterns. The relevant theorem is entirely analogous to the binary decision case—and is based on pairwise mutual linear separability. (We will consider a less restrictive formulation later.)

Theorem 4.9. *Let $\mathcal{S} = \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_m$ be a collection of m classes of pattern vectors in $\mathbb{R}^n$ and suppose that the subsets $\mathcal{S}_i$ and $\mathcal{S}_i' = \mathcal{S} \setminus \mathcal{S}_i$ are linearly separable, for each $1 \le i \le m$. Then the extended two-class perceptron error-correction algorithm applied to an n-input m-output perceptron reaches a fixed point after a finite number of iterations and thus correctly classifies each of the m patterns.*

Proof. As we have already noted, the neural network is equivalent to m independent simple perceptrons, each being presented with the same input pattern. By the perceptron convergence theorem, we know that the jth of these will correctly distinguish the jth pattern from the rest after a finite number N_j, say, of iterations. Since there are only a finite number of patterns to be classified, the whole set will be correctly classified after at most $\max\{N_1, \dots, N_m\}$ updates. ∎

It is therefore of interest to find conditions which guarantee the linear separability of m patterns as required by the theorem. It turns out that linear independence suffices for this, as we now show.

Definition 4.10. A linear map ℓ from $\mathbb{R}^n$ to $\mathbb{R}$ is called a linear functional.

Remark 4.11. Let ℓ be a linear functional on $\mathbb{R}^n$, and let $e_1, \dots, e_n$ be a basis for $\mathbb{R}^n$. Thus, for any $x \in \mathbb{R}^n$, there is $\alpha_1, \dots, \alpha_n \in \mathbb{R}$ such that

$$x = \alpha_1 e_1 + \cdots + \alpha_n e_n.$$

Then we have

$$\begin{aligned} \ell(x) &= \ell(\alpha_1 e_1 + \cdots + \alpha_n e_n) \\ &= \ell(\alpha_1 e_1) + \cdots + \ell(\alpha_n e_n) \\ &= \alpha_1 \ell(e_1) + \cdots + \alpha_n \ell(e_n). \end{aligned}$$

It follows that ℓ is completely determined by its values on the elements of any basis in $\mathbb{R}^n$. In particular, if $\{e_1, \dots, e_n\}$ is the standard orthonormal basis of $\mathbb{R}^n$, so that $x = (x_1, \dots, x_n) = x_1 e_1 + \cdots + x_n e_n$, then $\ell(x) = v \cdot x$ ($= v^T x$, in matrix notation), where $v \in \mathbb{R}^n$ is the vector $v = \big(\ell(e_1), \dots, \ell(e_n)\big)$.

On the other hand, for any $u \in \mathbb{R}^n$, it is clear that the map $x \mapsto u \cdot x$ ($= u^T x$) defines a linear functional on $\mathbb{R}^n$.

Proposition 4.12. *For any linearly independent set of vectors* $\{x^{(1)},\dots,x^{(m)}\}$ *in* $\mathbb{R}^n$*, the augmented vectors* $\{\overline{x}^{(1)},\dots,\overline{x}^{(m)}\}$*, with* $\overline{x}^{(i)}=(-1,x^{(i)})\in\mathbb{R}^{n+1}$*, form a linearly independent collection.*

Proof. Suppose that $\sum_{j=1}^m \alpha_j \overline{x}^{(j)}=0$. This is a vector equation and so each of the $n+1$ components of the left hand side must vanish, $\sum_{j=1}^m \alpha_j \overline{x}_i^{(j)}=0$, for $0\le i\le n$. Note that the first component of $\overline{x}^{(j)}$ has been given the index 0.

In particular, $\sum_{j=1}^m \alpha_j \overline{x}_i^{(j)}=0$, for $1\le i\le n$, that is, $\sum_{j=1}^m \alpha_j x^{(j)}=0$. By hypothesis, $x^{(1)},\dots,x^{(m)}$ are linearly independent and so $\alpha_1=\dots=\alpha_m=0$ which means that $\overline{x}^{(1)},\dots,\overline{x}^{(m)}$ are linearly independent, as required. ∎

Remark 4.13. The converse is false. For example, if $x^{(1)}=(1,1,\dots,1)\in\mathbb{R}^n$, and $x^{(2)}=(2,2,\dots,2)\in\mathbb{R}^n$, then clearly $x^{(1)}$ and $x^{(2)}$ are *not* linearly independent. However, $\overline{x}^{(1)}$ and $\overline{x}^{(2)}$ *are* linearly independent.

Proposition 4.14. *Suppose that* $x^{(1)},\dots,x^{(m)}\in\mathbb{R}^n$ *are linearly independent pattern vectors. Then, for each* $k=1,2,\dots,m$*, there is* $\widehat{w}^{(k)}\in\mathbb{R}^n$ *and* $\theta_k\in\mathbb{R}$ *such that*

$$\widehat{w}^{(k)}\cdot x^{(k)}>\theta_k$$

and

$$\widehat{w}^{(k)}\cdot x^{(j)}<\theta_k$$

for $j\ne k$.

Proof. By the previous result, the augmented vectors $\overline{x}^{(1)},\dots,\overline{x}^{(m)}\in\mathbb{R}^{n+1}$ are linearly independent. Let $y^{(m+1)},\dots,y^{(n+1)}$ be $n-m$ linearly independent vectors such that the collection $\{\overline{x}^{(1)},\dots,\overline{x}^{(m)},y^{(m+1)},\dots,y^{(n+1)}\}$ forms a basis for $\mathbb{R}^{n+1}$.

Define the linear functional $\ell_1:\mathbb{R}^{n+1}\to\mathbb{R}$ by assigning its values on the basis given above according to $\ell_1(\overline{x}^{(1)})=1$ and $\ell_1(\overline{x}^{(2)})=\dots=\ell(y^{n+1})=-1$. Now, we know that there is $v=(v_0,v_1,\dots,v_n)\in\mathbb{R}^{n+1}$ such that ℓ_1 can be written as

$$\ell_1(x)=v\cdot x$$

for all $x\in\mathbb{R}^{n+1}$. Set $\theta_1=v_0$ and $\widehat{w}^{(1)}=(v_1,\dots,v_n)\in\mathbb{R}^n$. Then

$$\begin{aligned}1=\ell_1(\overline{x}^{(1)})&=v\cdot\overline{x}^{(1)}\\&=v_0\overline{x}_0^{(1)}+v_1\overline{x}_1^{(1)}+\dots+v_n\overline{x}_n^{(1)}\\&=-\theta_1+\widehat{w}^{(1)}\cdot x^{(1)},\ \text{since } \overline{x}_0^{(1)}=-1,\end{aligned}$$

so that $\widehat{w}^{(1)}\cdot x^{(1)}>\theta_1$.

Similarly, for any $j\ne1$, $-1=\ell_1(\overline{x}^{(j)})=-\theta_1+\widehat{w}^{(1)}\cdot x^{(j)}$ giving $\widehat{w}^{(1)}\cdot x^{(j)}<\theta_1$. An identical argument holds for any $x^{(k)}$ rather than $x^{(1)}$. ∎

This tells us that an m-output perceptron is capable of correctly classifying m linearly independent pattern vectors. For example, we could correctly classify the capital letters of the alphabet using a perceptron with 26 output units by representing the letters as 26 linearly independent vectors. This might be done by drawing each character on a retina (or grid) of sufficiently high resolution (perhaps 6×6 pixels) and transforming the resulting image into a binary vector whose components correspond to the on-off state of the corresponding pixels. Of course, one should check that the 26 binary vectors one gets are, indeed, linearly independent.

The multi-class error correction algorithm

Suppose that we have a collection of patterns $\mathcal{S}$ in $\mathbb{R}^n$ comprising m classes $\mathcal{S}_1, \dots, \mathcal{S}_m$. The aim is to construct a network which can correctly classify these patterns. To do this, we set up a "winner takes all" network built from m linear units.

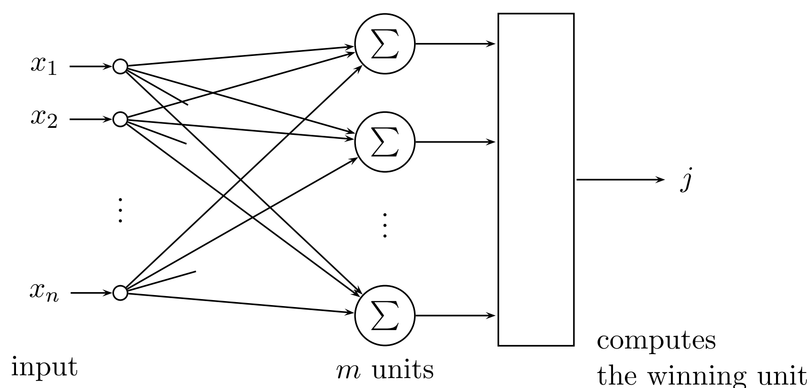


Figure 4.7: Winner takes all network.

There are n inputs which feed into m linear units, i.e., each unit simply records its net weighted input. The network output is the index of the unit which has maximum such activation. Thus, if $w_1, \dots, w_m$ are the weights to the m linear units and the input pattern is x, then the system output is the "winning unit", i.e., that value of j with

$$w_j \cdot x > w_i \cdot x \quad \text{for all } i \neq j.$$

In the case of a tie, one assigns a rule of convenience. For example, one might choose one of the winners at random, or alternatively, always choose the smallest index possible, say.

Multi-class error correction algorithm

- Choose the initial set of weight vectors $w_1^{(1)}, \dots, w_m^{(m)}$, arbitrarily.
- Step k: Present a pattern $x^{(k)}$ from $\mathcal{S}$ and observe the winning unit. Suppose that $x^{(k)} \in \mathcal{S}_i$. If the winning unit is unit i, then do nothing. If the winning unit is $j \neq i$, then update the weights according to

$$\begin{aligned} w_i^{(k+1)} &= w_i^{(k)} + x^{(k)} \quad \text{reinforce unit } i \\ w_j^{(k+1)} &= w_j^{(k)} - x^{(k)} \quad \text{anti-reinforce unit } j \end{aligned}$$

  ... and repeat.

The question is whether or not the algorithm leads to a set of weights which correctly classify the patterns. Under certain separability conditions, the answer is yes.

Theorem 4.15 (Multi-class error-correction convergence theorem). *Suppose that the m classes $\mathcal{S}_i$ are linearly separable in the sense that there exist weight vectors $w_1^*, \dots, w_m^* \in \mathbb{R}^n$ such that*

$$w_i^* \cdot x > w_j^* \cdot x \quad \text{for all } j \neq i$$

whenever $x \in \mathcal{S}_i$. Suppose that the pattern presentation sequence $(x^{(k)})$, from the finite set $\mathcal{S} = \mathcal{S}_1 \cup \dots \cup \mathcal{S}_m$, is such that every pattern in $\mathcal{S}$ appears infinitely-often. Then, after a finite number of iterations, the multi-class error-correction algorithm provides a set of weight vectors which correctly classify all patterns in $\mathcal{S}$.

Proof. The idea of the proof (Kesler) is to relate this system to a two-class problem (in a high dimensional space) solved by the usual perceptron error-correction algorithm.

For any given $i \in \{1, \dots, m\}$ and $x \in \mathcal{S}_i$, we construct $m-1$ vectors, $\widehat{x}^{(j)} \in \mathbb{R}^{mn}$, $1 \le j \le m$ but $j \neq i$, as follows. Set

$$\widehat{x}^{(j)} = u_1 \oplus u_2 \oplus \dots \oplus u_m \in \mathbb{R}^{mn}$$

where $u_i = x$, $u_j = -x$ and all other u_ℓs are zero in $\mathbb{R}^n$. Let

$$\widehat{w}^* = w_1^* \oplus \dots \oplus w_m^* \in \mathbb{R}^{mn}.$$

Then

$$\widehat{w}^* \cdot \widehat{x}^{(j)} = w_i^* \cdot x - w_j^* \cdot x > 0$$

by (the separation) hypothesis. Let $\mathcal{C}$ denote the set of all possible $\widehat{x}^{(j)}$s as x runs over $\mathcal{S}$. Then $\mathcal{C}$ is "linearly separable" in the sense of the perceptron convergence theorem;

$$\widehat{w}^* \cdot y > 0 \quad \text{for all } y \in \mathcal{C}.$$

Suppose we start with $\widehat{w} = w_1^{(1)} \oplus \cdots \oplus w_m^{(1)}$ and apply the perceptron error correction algorithm with patterns drawn from $\mathcal{C}$:

$$\widehat{w}^{(k+1)} = \widehat{w}^{(k)} + \alpha^{(k)} \widehat{x}^{(j)}$$

where $\alpha^{(k)} = 0$ if $\widehat{w}^{(k)} \cdot \widehat{x}^{(j)} > 0$, and otherwise $\alpha^{(k)} = 1$. The correction $\alpha^{(k)} \widehat{x}^{(j)}$ gives

$$\widehat{w}^{(k+1)} = (w_1^{(k)} + v_1) \oplus \dots (w_m^{(k)} + v_m)$$

where

$$\begin{aligned} v_i &= \alpha^{(k)} x, \quad \text{assuming } x \in \mathcal{S}_i \\ v_j &= -\alpha^{(k)} x, \\ v_\ell &= 0, \quad \text{all } \ell \neq i, j. \end{aligned}$$

Thus, the components of $\widehat{w}^{(k+1)}$ are given by

$$\begin{aligned} w_i^{(k+1)} &= w_i^{(k)} + \alpha^{(k)} x \\ w_j^{(k+1)} &= w_j^{(k)} - \alpha^{(k)} x \\ w_\ell^{(k+1)} &= w_\ell^{(k)}, \quad \ell \neq i, \, \ell \neq j \end{aligned}$$

which is precisely the multi-class error correction rule when unit j is the winning unit. In other words, the weight changes given by the two system algorithms are the same. If one is non-zero, then neither is the other. It follows that every misclassified pattern (from $\mathcal{S}$) presented to the multi-class system induces a misclassified "super-pattern" (in $\mathcal{C} \subset \mathbb{R}^{mn}$) for the perceptron system thus triggering a non-zero weight update.

However, the perceptron convergence theorem assures us that the "super-pattern" system can accommodate at most a finite number of non-zero weight changes. It follows that the multi-class algorithm can undergo at most a finite number of weights changes. We deduce that after a finite number of steps, the winner-takes-all system must correctly classify all patterns from $\mathcal{S}$. ∎

Mapping implementation

Let us turn now to the perceptron viewed as a mechanism for implementing binary-valued mappings, $f : x \mapsto f(x) \in \{0,1\}$.

Example 4.16. Consider the function $f : \{0,1\}^2 \to \{0,1\}$ given by

$$f : \begin{cases} (0,0) & \mapsto 0 \\ (0,1) & \mapsto 0 \\ (1,0) & \mapsto 0 \\ (1,1) & \mapsto 1. \end{cases}$$

If we think of 0 and 1 as "off" and "on", respectively, then f is "on" if and only if both its arguments are "on". For this reason, f is called the "AND" function.

We wish to implement f using a simple perceptron with two input neurons and a binary threshold output neuron. Thus, we seek weights w_1, w_2 and a threshold θ such that $\text{step}(w_1 x + w_2 y - \theta) = 1$ for $(x,y) = (1,1)$, but otherwise is zero, that is, we require that $w_1 x + w_2 y - \theta$ is positive for $(x,y) = (1,1)$, but otherwise is negative. Geometrically, this means that the point $(1,1)$ lies above the line $w_1 x + w_2 y - \theta = 0$ in the two-dimensional (x,y)-plane, whereas the other three points lie below this line. The line $w_1 x + w_2 y - \theta = 0$ is a "hyperplane" linearly separating the two classes $\{(1,1)\}$ and $\{(0,0),(0,1),(1,0)\}$. There are many possibilities—one such is shown in the figure.

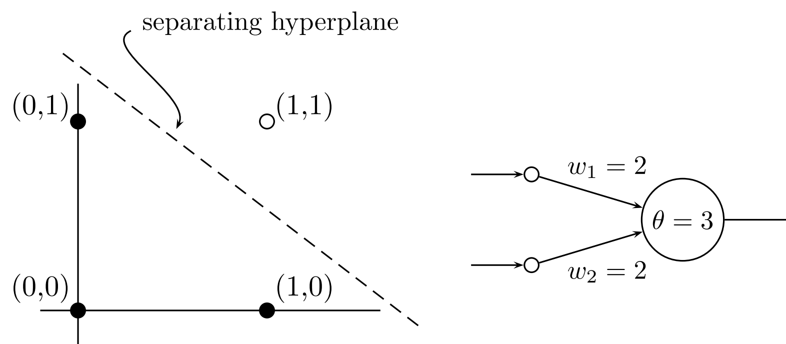


Figure 4.8: A line separating the classes, and the corresponding simple perceptron which implements the "AND" function.

Example 4.17. Next we consider the so-called "OR" function, f on $\{(0,1)\}^2$. This is given by

$$f : \begin{cases} (0,0) & \mapsto 0 \\ (0,1) & \mapsto 1 \\ (1,0) & \mapsto 1 \\ (1,1) & \mapsto 1. \end{cases}$$

Thus, f is "on" if and only at least one of its inputs is also "on", i.e., one or the other, or both. As above, we seek a line separating the point $(0,0)$ from the rest. A solution is indicated in the figure.

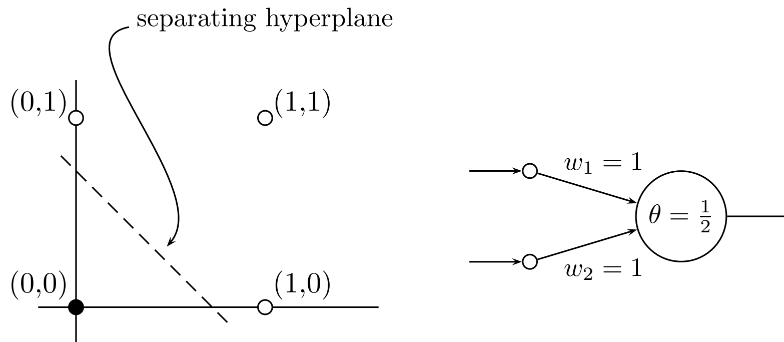


Figure 4.9: A separating line for the "OR" function and the corresponding simple perceptron.

Example 4.18. Consider, now, the 2-parity or "XOR" (exclusive-OR) function. This is defined by

$$f : \begin{cases} (0,0) & \mapsto 0 \\ (0,1) & \mapsto 1 \\ (1,0) & \mapsto 1 \\ (1,1) & \mapsto 0. \end{cases}$$

Here, the function is "on" only if *exactly one* of the inputs is "on". (We discuss the n-parity function later.) It is clear from a diagram that it is impossible to find a line separating the two classes $\{(0,0),(1,1)\}$ and $\{(0,1),(1,0)\}$. We can also see this algebraically as follows. If w_1, w_2 and θ were weights and threshold values implementing the "XOR" function, then we would have

$$\begin{aligned} 0\,w_1 + 0\,w_2 &< \theta \\ 0\,w_1 + w_2 &\ge \theta \\ w_1 + 0\,w_2 &\ge \theta \\ w_1 + w_2 &< \theta. \end{aligned}$$

Adding the second and third of these inequalities gives $w_1 + w_2 \geq 2\theta$, which is incompatible with the fourth inequality. Thus we come to the conclusion that the simple perceptron (two input nodes, one output node) is *not* able to implement the "XOR" function.

However, it *is* possible to implement the "XOR" function using slightly more complicated networks. Two such examples are shown in the figure.

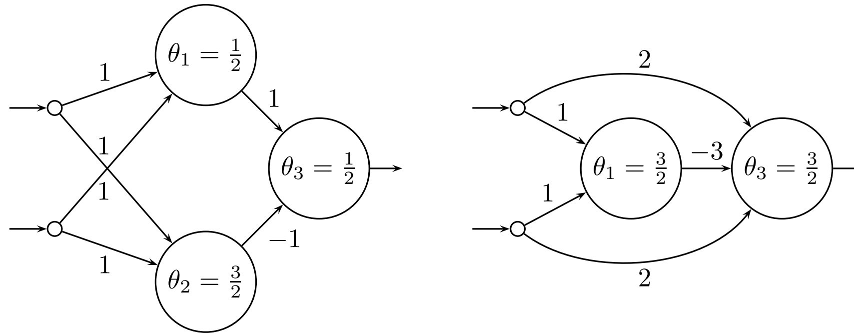


Figure 4.10: The "XOR" function can be implemented by a perceptron with a middle layer of two binary threshold units. Alternatively, we can use just one middle unit but with a non-conventional feed-forward structure.

Definition 4.19. The n-parity function is the binary map f on the hypercube $\{0,1\}^n$, $f : \{0,1\}^n \to \{0,1\}$, given by $f(x_1, \ldots, x_n) = 1$ if the number of x_k s which are equal to 1 is odd, and $f(x_1, \ldots, x_n) = 0$ otherwise. Thus, f can be written as

$$f(x_1, \ldots, x_n) = \tfrac{1}{2}\big(1 - (-1)^{x_1 + x_2 + \cdots + x_n}\big),$$

bearing in mind that each x_k is a binary digit, so that $x_1 + x_2 + \cdots + x_n$ is precisely the number of 1 s in $(x_1, \ldots, x_n)$. Note that 2-parity is just the "XOR" function on $\{0,1\}^2$.

Theorem 4.20. *The n-parity function cannot be implemented by an n-input simple perceptron.*

Proof. The statement of the theorem is equivalent to the statement that n-parity is not linearly separable, that is, the two sets $\{x \in \{0,1\}^n : f(x) = 0\}$ and $\{x \in \{0,1\}^n : f(x) = 1\}$ are not linearly separable in $\mathbb{R}^n$. To prove the theorem, suppose that f *can* be implemented by a simple perceptron (with n-input units and a single binary threshold output unit). Then there are weights $w_1, \ldots, w_n$ and a threshold θ such that $w_1 x_1 + \cdots + w_n x_n > \theta$ if the

number of x_ks equal to 1 is odd, i.e., $x_1 + \cdots + x_n$ is odd, but otherwise $w_1x_1 + \cdots + w_nx_n < \theta$.

Setting $x_3 = \cdots = x_n = 0$, we see that the network with the inputs x_1 and x_2, weights w_1, w_2 and threshold θ implements the 2-parity function, i.e., the "XOR" function. However, this we have seen is impossible. We conclude that such a perceptron as hypothesized above cannot exist, and the proof is complete. ∎

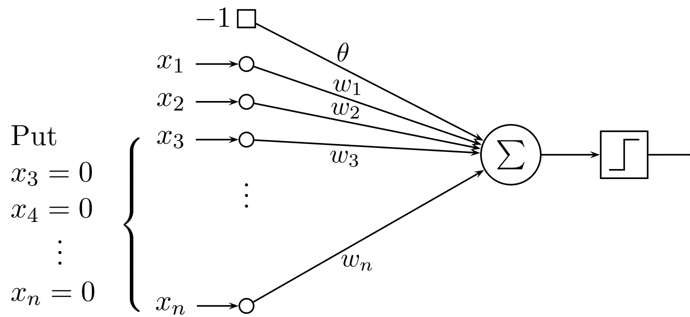


Figure 4.11: If the n-parity function can be implemented by a simple perceptron, then so can the "XOR" function (2-parity).

Theorem 4.21. *The n-parity function can be implemented by a three-layer network of binary threshold units.*

Proof. The network is a feed-forward network with n input units, n binary threshold units in the middle layer, and a single binary threshold output unit.

All weights from the inputs to the the middle layer are set equal to 1, whilst those from the middle layer to the output are given the values $+1, -1, +1, -1, \ldots$, respectively, that is, the weight from middle unit j to the output unit is set equal to $(-1)^{j+1}$.

The threshold values of the middle units are $1 - \frac{1}{2}, 2 - \frac{1}{2}, \ldots, n - \frac{1}{2}$ and that for the output unit is set to $\frac{1}{2}$. When k input units are "on", i.e., have value 1, then the total input (net activation) to the jth middle unit is equal to k. Thus the units $1, 2, \ldots, k$ in the middle layer all fire (since $k - (j - \frac{1}{2}) > 0$ for all $1 \le j \le k$). The middle units $k+1, \ldots, n$ do *not* fire. The net activation potential of the output unit is therefore equal to

$$\sum_{j=1}^{k} (-1)^{j+1} = \underbrace{1 - 1 + 1 - \ldots}_{k \text{ terms}} = \begin{cases} 1, & \text{if } k \text{ is odd} \\ 0, & \text{if } k \text{ is even.} \end{cases}$$

Evidently, because of the threshold of $\frac{1}{2}$ at the output unit, this will fire if and only if k is odd, as required. ∎

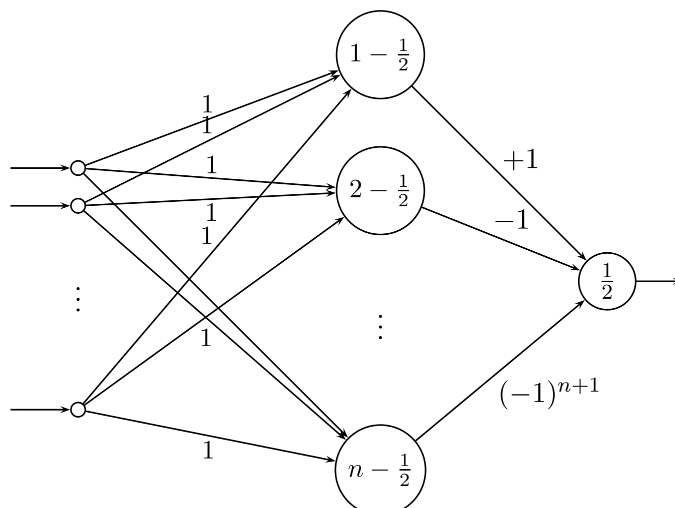


Figure 4.12: A 3-layer network implementing the n-parity function. All weights between the first and second layers are equal to 1, whereas those between the middle and output unit alternate between the values ± 1. The threshold values of the units are as indicated.

Next we shall show that any binary mapping on the binary hypercube $\{0,1\}^N$ can be implemented by a 3-layer feedforward neural network of binary threshold units. The space $\{0,1\}^N$ contains 2^N points (each a string of N binary digits). Each of these is mapped into either 0 or 1 under a binary function, and each such assignment defines a binary function. It follows that there are 2^{2^N} binary-valued functions on the binary hypercube $\{0,1\}^N$. We wish to show that any such function can be implemented by a suitable network. First we shall show that it is possible to construct perceptrons which are very selective.

Theorem 4.22 (Grandmother cell). *Let $z \in \{0,1\}^n$ be given. Then there is an n-input perceptron which fires if and only if it is presented with z.*

Proof. Suppose that $z = (b_1, \dots, b_n) \in \{0,1\}^n$ is given. We must find weights $w_1, \dots, w_n$ and a threshold θ such that

$$w_1 b_1 + \cdots + w_n b_n - \theta > 0$$

but

$$w_1 x_1 + \cdots + w_n x_n - \theta < 0$$

for every $x \neq z$. Define w_i, $i = 1, \dots, n$ by

$$w_i = \begin{cases} 1 & b_i = 1 \\ -1 & \text{otherwise.} \end{cases}$$

This assignment can be rewritten as $w_i = (2b_i - 1)$. Then we see that

$$\sum_{i=1}^{n} w_i x_i = \sum_{i=1}^{n} (2b_i - 1)x_i$$
$$= \sum_{i=1}^{n} (2b_i x_i - x_i).$$

Now, if $x = z$, then $x_i = b_i$, $1 \le i \le n$, so that $b_i x_i = b_i^2 = b_i$, since $b_i \in \{0,1\}$. Hence, in this case,

$$\sum_{i=1}^{n} w_i x_i = \sum_{i=1}^{n} (2b_i - b_i) = \sum_{i=1}^{n} b_i.$$

On the other hand, if $x \neq z$, then there is some j such that $x_j \neq b_j$ so that $b_j x_j = 0$ (since one of b_j, x_j must be zero). But, for such a term, $(2b_j - 1)x_j = -x_j < b_j$ (because $b_j + x_j = 1$). Hence, if $x \neq z$, we have

$$\sum_{i=1}^{n} w_i x_i = \sum_{i=1}^{n} (2b_i - 1)x_i < \sum_{i=1}^{n} b_i.$$

Now, both sides of the above inequality are integers, so if we set $\theta = \sum_{i=1}^{n} b_i - \frac{1}{2}$, it follows that

$$\sum_{i=1}^{n} w_i x_i < \theta < \sum_{i=1}^{n} b_i = \sum_{i=1}^{n} w_i b_i$$

for any $x \neq z$ and the construction is complete. ∎

In the jargon, a neuron such as this is sometimes called a grandmother cell (from the biological idea that some neurons might serve just a single specialized purpose, such as responding to precisely one's grandmother). It is not thought that such neurons really exist in real brains.

Theorem 4.23. *Any mapping $f : \{0,1\}^N \to \{0,1\}$ can be implemented by a 3-layer feedforward neural network of binary threshold units.*

Proof. Let $f : \{0,1\}^N \to \{0,1\}$ be given. We construct the appropriate neural network as follows.

The neural network has N inputs, a middle (hidden) layer of 2^N binary threshold units and a single binary threshold output unit. The output unit is assigned threshold $\frac{1}{2}$. Label the hidden units by the 2^N elements z in $\{0,1\}^N$ and construct each to be the corresponding grandmother unit for the binary vector z. The weight from each such unit to the output unit is set to the value $f(z)$.

Then if the input is z, only the hidden unit labelled by z will fire. The total input to the output unit will then be precisely $f(z)$. The output unit will fire if and only if the value of $f(z)$ is 1 (and not 0). ∎

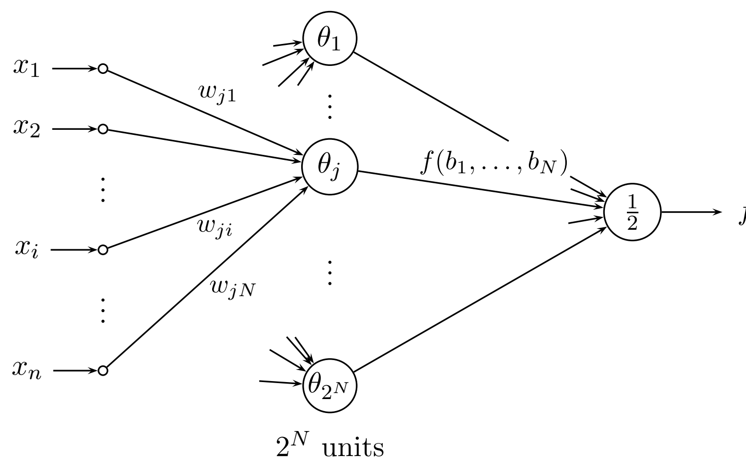


Figure 4.13: A 3-layer neural network implementing the function f.

Remark 4.24. Notice that those hidden units labelled by those z with $f(z) = 0$ are effectively not connected to the output unit (their weights are zero). We may just as well leave out these units altogether. This leads to the observation that, in fact, at most 2^{N-1} units are required in the hidden layer.

Indeed, suppose that $\{0,1\}^N = A \cup B$ where $f(x) = 1$ for $x \in A$ and $f(x) = 0$ for $x \in B$. If the number of elements in A is not greater than that in B, we simply throw away all hidden units labelled by members of B, as suggested above. In this case, we have no more than 2^{N-1} units left in the hidden layer.

On the other hand, if B has fewer members than A, we wish to throw away those units labelled by A. However, we first have to slightly modify the network—otherwise, we will throw the grandmother out with the bath water.

We make the following modifications. Change θ to $-\theta$, and then change all weights from the A-labelled hidden units to output to zero, and change all weights from B-labelled hidden units to the output unit (from the previous value of zero) to the value -1. Then we see that the system fires only for inputs $x \in A$. We now throw away all the A-labelled hidden units (they have zero weights out from them, anyway) to end up with fewer than 2^{N-1} hidden units, as required.

Testing for linear separability

The perceptron convergence theorem tells us that *if* two classes of patterns are linearly separable, then the perceptron error-correction rule will eventually lead to a set of weights giving correct classification. Unfortunately, it sheds no light on whether or not the classes actually are linearly separable. Of course, one could ignore this in practice, and continue with the hope that they are. We have seen that linear independence suffices and so, provided we embed our patterns into a space of sufficiently high dimension, there would seem to be a good chance that the classes will be linearly separable and so we can run the perceptron algorithm with abandon. It would be nice, however, if it were possible to somehow discover, as we go along, whether or not our efforts are doomed to failure. We will discuss now the so-called Ho-Kashyap algorithm, which does precisely that.

The problem is to classify two sets of patterns $\mathcal{S}_1$ and $\mathcal{S}_2$ in $\mathbb{R}^\ell$. As usual, these are augmented to give $\mathcal{C}_1$ and $\mathcal{C}_2$, and then we set $\mathcal{C} = \mathcal{C}_1 \cup (-\mathcal{C}_2)$. Thus, we seek $w \in \mathbb{R}^{\ell+1}$ such that $w \cdot y > 0$ for all $y \in \mathcal{C}$. Suppose $\mathcal{C} = \{y^{(1)}, \dots, y^{(N)}\}$. Then we want $w \in \mathbb{R}^{\ell+1}$ such that $w \cdot y^{(j)} > 0$ for each $1 \le j \le N$. Evidently, this problem is equivalent to that of finding a set of real numbers $b_1 > 0, \dots, b_N > 0$ such that $w \cdot y^{(j)} = b_j$, $1 \le j \le N$. Expressing this in matrix form, we seek $w \in \mathbb{R}^{\ell+1}$ and $b \in \mathbb{R}^N$ such that b has strictly positive entries, and

$$Yw = b, \text{ where } Y = \begin{pmatrix} y^{(1)T} \\ y^{(2)T} \\ \vdots \\ y^{(N)T} \end{pmatrix} \in \mathbb{R}^{N \times (\ell+1)},$$

where we have used the equality $w \cdot y^{(j)} = y^{(j)T} w$.

To develop an algorithm for finding such vectors w and b, we observe that, trivially, any such pair minimizes the quantity

$$J = \tfrac{1}{2}\|Yw - b\|^2 = \tfrac{1}{2}\|Yw - b\|_F^2 .$$

So to attempt to find a suitable w and b, we employ a kind of gradient-descent based technique. Writing

$$\begin{aligned} J &= \tfrac{1}{2}\|Yw - b\|^2 = \tfrac{1}{2}(Yw - b)^T (Yw - b) \\ &= \tfrac{1}{2}\sum_{j=1}^{N} (y^{(j)T} w - b_j)^2 \end{aligned}$$

we see that $\partial J / \partial b_j = -(y^{(j)T} w - b_j)$, $1 \le j \le N$, and so the gradient of J, as a function of the vector b, is

$$\frac{\partial J}{\partial b} = -(Yw - b) \in \mathbb{R}^N.$$

Suppose that $w^{(1)}, w^{(2)}, \ldots,$ and $b^{(1)}, b^{(2)}, \ldots$ are iterations for w and b, respectively. Then a gradient-descent technique for b might be to construct the $b^{(k)}$s by the rule

$$\begin{aligned} b^{(k+1)} &= b^{(k)} - \alpha \frac{\partial J}{\partial b} \\ &= b^{(k)} + \alpha (Y w^{(k)} - b^{(k)}), \quad k = 1, 2, \ldots, \end{aligned}$$

where α is a positive constant. Given $b^{(k)}$, we choose $w^{(k)}$ so that J is minimized (for given Y and $b^{(k)}$). This we know, from general theory, to be achieved by $w^{(k)} = Y^{\#} b^{(k)}$. Indeed, for given matrices A and B, $\|XA - B\|_F$ is minimized by $X = BA^{\#}$ and so, for given Y and b, $\|Yw - b\|_F = \|(Yw - b)^T\|_F = \|w^T Y^T - b^T\|_F$ is minimized by $w^T = b^T Y^{T\#} = b^T Y^{\# T}$, that is, for $w = Y^{\#} b$.

However, we want to ensure that $b^{(k)}$ has strictly positive components. To this end, we modify the formula for $b^{(k+1)}$ above, to the following. We set

$$b^{(k+1)} = b^{(k)} + \alpha \Delta b^{(k)}$$

where $\Delta b^{(k)}{}_j = \begin{cases} (Y w^{(k)} - b^{(k)})_j, & \text{if } (Y w^{(k)} - b^{(k)})_j > 0 \\ 0, & \text{otherwise} \end{cases}$ and where $b^{(1)}$ is such that $b^{(1)}{}_j > 0$ for all $1 \le j \le N$. Note that $b^{(k+1)}{}_j \ge b^{(k)}{}_j$ for each $1 \le j \le N$, that is, the components of $b^{(k)}$ never decrease at any step of the iteration. The algorithm then is as follows.

The Ho-Kashyap algorithm

Let $b^{(1)}$ be arbitrary subject to $b^{(1)}{}_j > 0$, $1 \le j \le N$. For each $k = 1, 2, \ldots$ set $w^{(k)} = Y^{\#} b^{(k)}$ and

$$b^{(k+1)} = b^{(k)} + \alpha \Delta b^{(k)}$$

where the constant $\alpha > 0$ will be chosen to ensure convergence, and where $\Delta b^{(k)}$ is given as above.

For notational convenience, let $e^{(k)} = Y w^{(k)} - b^{(k)}$. Then $\Delta b^{(k)}$ is the vector with components given by $\Delta b^{(k)}{}_j = e^{(k)}{}_j$ if $e^{(k)}{}_j > 0$, but otherwise $\Delta b^{(k)}{}_j = 0$. It follows that $e^{(k)T} \Delta b^{(k)} = \Delta b^{(k)T} \Delta b^{(k)}$.

Theorem 4.25. *Suppose that $\mathcal{S}_1$ and $\mathcal{S}_2$ are linearly separable classes. Then, for any k, the inequalities $e^{(k)}{}_j \le 0$ for all $j = 1, \ldots, N$ imply that $e^{(k)} = 0$.*

Proof. By hypothesis, there is some $\widehat{w} \in \mathbb{R}^{\ell+1}$ such that $\widehat{w} \cdot y > 0$ for all $y \in \mathcal{C}$, i.e., $\widehat{w} \cdot y^{(j)} > 0$ for all $1 \le j \le N$. In terms of the matrix Y, this

implies that the vector $Y\widehat{w}$ has strictly positive components, $(Y\widehat{w})_j > 0$, $1 \le j \le N$. Now, from the definition of $e^{(k)}$, we have

$$\begin{aligned} Y^T e^{(k)} &= Y^T (Y w^{(k)} - b^{(k)}) \\ &= Y^T (Y Y^{\#} b^{(k)} - b^{(k)}) \\ &= (Y^T Y Y^{\#} - Y^T) b^{(k)}. \end{aligned}$$

But $YY^{\#} = (YY^{\#})^T$ and so

$$\begin{aligned} Y^T Y Y^{\#} &= Y^T (Y Y^{\#})^T \\ &= \big((Y Y^{\#} Y\big)^T \\ &= Y^T. \end{aligned}$$

Hence $Y^T e^{(k)} = 0$, and so $e^{(k)T} Y = 0$. (We will also use this last equality later.) It follows that $e^{(k)T} \widehat{w} = 0$, that is,

$$\sum_{j=1}^{N} e^{(k)}{}_j (Y\widehat{w})_j = 0.$$

The inequalities $(Y\widehat{w})_j > 0$ and $e^{(k)}{}_j \le 0$, for all $1 \le j \le N$, force $e^{(k)}{}_j = 0$ for all $1 \le j \le N$. ∎

Remark 4.26. This result means that if, during execution of the algorithm, we find that $e^{(k)}$ has strictly negative components, then $\mathcal{S}_1$ and $\mathcal{S}_2$ are *not* linearly separable.

Next we turn to a proof of convergence of the algorithm under the assumption that $\mathcal{S}_1$ and $\mathcal{S}_2$ are linearly separable.

Theorem 4.27. *Suppose that $\mathcal{S}_1$ and $\mathcal{S}_2$ are linearly separable, and let $0 < \alpha < 2$. The algorithm converges to a solution after a finite number of steps.*

Proof. By definition,

$$\begin{aligned} e^{(k)} &= Y w^{(k)} - b^{(k)} \\ &= (Y Y^{\#} - \mathbb{1}_N) b^{(k)} \\ &= (P - \mathbb{1}_N) b^{(k)} \end{aligned}$$

where $P = YY^{\#}$ satisfies $P = P^T = P^2$, i.e., P is an orthogonal projection. Therefore

$$\begin{aligned} e^{(k+1)} &= (P - \mathbb{1}_N) b^{(k+1)} \\ &= (P - \mathbb{1}_N)(b^{(k)} + \alpha \Delta b^{(k)}) \\ &= e^{(k)} + \alpha (P - \mathbb{1}_N) \Delta b^{(k)}. \end{aligned}$$

Hence

$$\begin{aligned}\|e^{(k+1)}\|^2 = \|e^{(k)}\|^2 + \alpha^2 \|(P - \mathbb{1}_N)\Delta b^{(k)}\|^2 \\ + 2\alpha e^{(k)T}(P - \mathbb{1}_N)\Delta b^{(k)}.\end{aligned}$$

To evaluate the right hand side, we calculate

$$\begin{aligned}\|(P - \mathbb{1}_N)\Delta b^{(k)}\|^2 &= \Delta b^{(k)T} P^T P \Delta b^{(k)} \\ &\qquad - 2\Delta b^{(k)} P \Delta b^{(k)} + \Delta b^{(k)T}\Delta b^{(k)} \\ &= -\|P\Delta b^{(k)}\|^2 + \|\Delta b^{(k)}\|^2, \text{ using } P = P^T P.\end{aligned}$$

Furthermore,

$$\begin{aligned}e^{(k)T}(P - \mathbb{1}_N)\Delta b^{(k)} &= e^{(k)T}(YY^{\#} - \mathbb{1}_N)\Delta b^{(k)} \\ &= -e^{(k)T}\Delta b^{(k)}, \text{ since } e^{(k)T} Y = 0, \\ &= -\Delta b^{(k)T}\Delta b^{(k)} \\ &= -\|\Delta b^{(k)}\|^2.\end{aligned}$$

Therefore, we can write $\|e^{(k+1)}\|^2$ as

$$\|e^{(k+1)}\|^2 = \|e^{(k)}\|^2 - \alpha^2\|P\Delta b^{(k)}\|^2 - \alpha(2-\alpha)\|\Delta b^{(k)}\|^2 \qquad (*)$$

Choosing $0 < \alpha < 2$, we see that

$$\|e^{(k+1)}\| \le \|e^{(k)}\|, \text{ for } k = 1, 2, \dots .$$

The sequence $(\|e^{(k)}\|)_{k\in\mathbb{N}}$ is a non-increasing sequence of non-negative terms, and so converges as $k \to \infty$. In particular, it follows from $(*)$ that $\|\Delta b^{(k)}\| \to 0$, i.e., $\Delta b^{(k)} \to 0$ in $\mathbb{R}^N$ as $k \to \infty$. (The difference $\|e^{(k)}\|^2 - \|e^{(k+1)}\|^2$ converges to 0. By $(*)$, this is equal to a sum of two non-negative terms, both of which must therefore also converge to 0.)

Moreover, the sequence $(e^{(k)})_{k\in\mathbb{N}}$ is a bounded sequence in $\mathbb{R}^N$ and so has a convergent subsequence. That is, there is some $f \in \mathbb{R}^N$ and a subsequence $(e^{(k_n)})_{n\in\mathbb{N}}$ such that $e^{(k_n)} \to f$ in $\mathbb{R}^N$ as $n \to \infty$. In particular, the components ${e^{(k_n)}}_j$ converge to the corresponding component f_j of f, as $n \to \infty$. From its definition, we can write down the components of $\Delta b^{(k)}$ as

$$\Delta {b^{(k)}}_j = \tfrac{1}{2}({e^{(k)}}_j + |{e^{(k)}}_j|), \quad j = 1, 2, \dots, N.$$

Hence

$$\begin{aligned}\Delta {b^{(k_n)}}_j &= \tfrac{1}{2}({e^{(k_n)}}_j + |{e^{(k_n)}}_j|) \\ &\to \tfrac{1}{2}(f_j + |f_j|)\end{aligned}$$

as $n \to \infty$, for each $1 \le j \le N$. However, we know that the sequence $(\Delta b^{(k)})_{k \in \mathbb{N}}$ converges to 0 in $\mathbb{R}^N$, and so, therefore, does the subsequence $(\Delta b^{(k_n)})_{n \in \mathbb{N}}$. It follows that $f_j + |f_j| = 0$ for each $1 \le j \le N$. In other words, *all* components of f are non-positive.

By hypothesis, $\mathcal{S}_1$ and $\mathcal{S}_2$ are linearly separable and so there is $\widehat{w} \in \mathbb{R}^{\ell+1}$ such that $Y\widehat{w}$ has strictly positive components. However, $e^{(k)T}Y = 0$ for all k, and so, in particular,

$$f^T Y \widehat{w} = \lim_{n \to \infty} e^{(k_n)T} Y \widehat{w} = 0,$$

i.e., $\sum_{j=1}^N f_j (Y\widehat{w})_j = 0$, where $(Y\widehat{w})_j > 0$ for all $1 \le j \le N$. Since $f_j \le 0$, we must have $f_j = 0$, $j = 1, \dots, N$, and we conclude that $e^{(k_n)} \to 0$ as $n \to \infty$. But then the inequality $\|e^{(k+1)}\| \le \|e^{(k)}\|$, for all $k = 1, 2, \dots$, implies that the whole sequence $(e^{(k)})_{k \in \mathbb{N}}$ converges to 0. (For any given $\varepsilon > 0$, there is N_0 such that $\|e^{(k_n)}\| < \varepsilon$ whenever $n > N_0$. Put $N_1 = k_{N_0+1}$. Then for any $k > N_1$, we have $\|e^{(k)}\| \le \|e^{(k_{N_0+1})}\| < \varepsilon$.)

Let $\mu = \max\{b^{(1)}{}_j : 1 \le j \le N\}$ be the maximum value of the components of $b^{(1)}$. Then $\mu > 0$ by our choice of $b^{(1)}$. Since $e^{(k)} \to 0$, as $k \to \infty$, there is k_0 such that $\|e^{(k)}\| < \frac{1}{2}\mu$ whenever $k > k_0$. In particular, $-\frac{1}{2}\mu < e^{(k)}{}_j < \frac{1}{2}\mu$ for each $1 \le j \le N$ whenever $k > k_0$. But for any $j = 1, \dots, N$, $b^{(k)}{}_j \ge b^{(1)}{}_j$, and so

$$\begin{aligned}
(Yw^{(k)})_j &= b^{(k)}{}_j + e^{(k)}{}_j \\
&> \mu - \tfrac{1}{2}\mu = \tfrac{1}{2}\mu \\
&> 0
\end{aligned}$$

whenever $k > k_0$. Therefore the vector $w^{(k_0+1)}$ satisfies $y^{(j)T} w^{(k_0+1)} > 0$ for all $1 \le j \le N$ and determines a separating vector for $\mathcal{S}_1$ and $\mathcal{S}_2$ which completes the proof. ■

Chapter 5

Multilayer Feedforward Networks

We have seen that there are limitations on the computational abilities of the perceptron which seem not to be shared by neural networks with hidden layers. For example, we know that any boolean function can be implemented by some multilayer neural network (and, in fact, three layers suffice). The question one must ask for multilayer networks is how are the weights to be assigned or learned. We shall discuss, in this section, a very popular method of learning in multilayer feedforward neural networks, the so-called error back-propagation algorithm. This is a supervised learning algorithm based on a suitable error or cost function, with values determined by the actual and desired outputs of the network, which is to be minimized via a gradient-descent method. For this, we require that the activation functions of the neurons in the network be differentiable and it is customary to use some kind of sigmoid function. The idea is very simple, but the analysis is complicated by the abundance of indices. To illustrate the method, we shall consider the following three layer feedforward neural network.

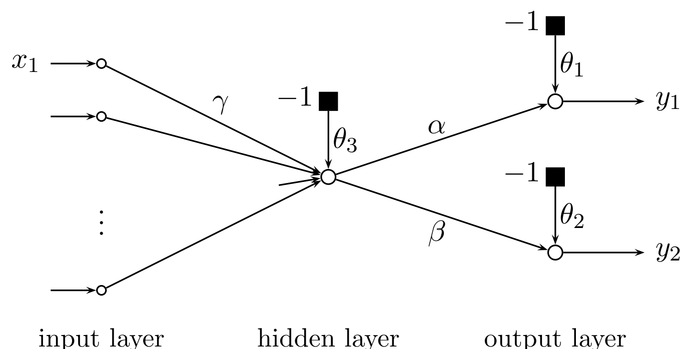


Figure 5.1: A small 3-layer feedforward neural network.

The network has n input units, one unit in the middle layer and two in the output layer. The various weights and thresholds are as indicated.

We suppose that the activation functions of the neurons in the second and third layers are given by the sigmoid function φ, say. The nodes in the input layer serve, as usual, merely as placeholders for the distribution of the input values to the units in the second (hidden) layer. Suppose that the input pattern $x = (x_1, \dots, x_n)$ is to induce the desired output (d_1, d_2). Let $y = (y_1, y_2)$ denote the actual output. Then the sum of squared differences

$$\mathcal{E} = \tfrac{1}{2}\big((d_1 - y_1)^2 + (d_2 - y_2)^2\big)$$

is a measure of the error between the actual and desired outputs. We seek to find weights and thresholds which minimize this. The approach is to use a gradient-descent method, that is, we use the update algorithm

$$w \mapsto w + \Delta w,$$

with $\Delta w = -\lambda \operatorname{grad} \mathcal{E}$, where w denotes the "weight" vector formed by all the individual weights and thresholds of the network. (So the dimension of w is given by the total number of connections and thresholds.) As usual, λ is called the learning parameter. To find $\operatorname{grad} \mathcal{E}$, we must calculate the partial derivatives of $\mathcal{E}$ with respect to all the weights and thresholds.

Consider $\partial \mathcal{E} / \partial \alpha$ first. We have $y_1 = \varphi(v_1)$, with $v_1 = u_1 - \theta_1$ and $u_1 = (\sum w_{1i} z_i) = \alpha z$ where z is the output from the (single) middle unit (so that there is, in fact, no summation but just the one term). Thus $y_1 = \varphi(\alpha z - \theta_1)$. Hence

$$\begin{aligned}
\frac{\partial \mathcal{E}}{\partial \alpha} &= -(d_1 - y_1)\, \frac{\partial y_1}{\partial \alpha} \\
&= -(d_1 - y_1)\, \varphi'(v_1)\, \frac{\partial v_1}{\partial \alpha} \\
&= -(d_1 - y_1)\, \varphi'(v_1)\, z \\
&= -\Delta_1\, z
\end{aligned}$$

where we have set $\Delta_1 = (d_1 - y_1)\varphi'(v_1)$. In an entirely similar way, we get

$$\begin{aligned}
\frac{\partial \mathcal{E}}{\partial \beta} &= -(d_2 - y_2)\, \varphi'(v_2)\, z \\
&\equiv -\Delta_2\, z
\end{aligned}$$

where we have used $v_2 = \beta z - \theta_2$. Further,

$$\frac{\partial \mathcal{E}}{\partial \theta_1} = -(d_1 - y_1)\, \varphi'(v_2)\, (-1)$$

and

$$\frac{\partial \mathcal{E}}{\partial \theta_2} = -(d_2 - y_2)\, \varphi'(v_2)\, (-1).$$

Next, we must calculate

$$\frac{\partial \mathcal{E}}{\partial \gamma} = -(d_1 - y_1)\frac{\partial y_1}{\partial \gamma} - (d_2 - y_2)\frac{\partial y_2}{\partial \gamma}.$$

Now,

$$\begin{aligned}
\frac{\partial y_1}{\partial \gamma} &= \frac{\partial \varphi(v_1)}{\partial \gamma} = \varphi'(v_1)\frac{\partial v_1}{\partial \gamma} \\
&= \varphi'(v_1)\alpha\frac{\partial z}{\partial \gamma} = \varphi'(v_1)\alpha\frac{\partial \varphi(v_3)}{\partial \gamma} \\
&= \varphi'(v_1)\alpha\varphi'(v_3)\frac{\partial v_3}{\partial \gamma} \\
&= \varphi'(v_1)\alpha\varphi'(v_3)x_1, \text{ using } v_3 = u_3 - \theta_3 = (x_1\gamma + \dots) - \theta_3.
\end{aligned}$$

Similarly, we find

$$\frac{\partial y_2}{\partial \gamma} = \varphi'(v_2)\beta\varphi'(v_3)x_1.$$

Hence

$$\begin{aligned}
\frac{\partial \mathcal{E}}{\partial \gamma} &= -\big(d_1 - \varphi(v_1)\big)\varphi'(v_1)\alpha\varphi'(v_3)x_1 \\
&\qquad - \big(d_2 - \varphi(v_2)\big)\varphi'(v_2)\alpha\varphi'(v_3)x_1 \\
&= -(\Delta_1\alpha + \Delta_2\beta)\varphi'(v_3)x_1 \\
&= -\overline{\Delta}_1\, x_1
\end{aligned}$$

where $\overline{\Delta}_1 \equiv (\Delta_1\alpha + \Delta_2\beta)\varphi'(v_3)$.

In the same way, we calculate

$$\frac{\partial \mathcal{E}}{\partial \theta_3} = -\overline{\Delta}_1\,(-1).$$

Thus, the gradient-descent algorithm is to update the weights according to

$$\begin{aligned}
\alpha &\longrightarrow \alpha + \lambda\Delta_1 z, \\
\beta &\longrightarrow \beta + \lambda\Delta_2 z, \\
\theta_i &\longrightarrow \theta_i - \lambda\Delta_i, \quad i = 1, 2, \\
\gamma &\longrightarrow \gamma + \lambda\overline{\Delta}_1 x_1, \quad \text{and} \\
\theta_3 &\longrightarrow \theta_3 - \lambda\overline{\Delta}_1.
\end{aligned}$$

We have only considered the weight γ associated with the input to second layer, but the general case is clear. (We could denote these n weights by γ_i, $i = 1, \dots, n$, in which case the adjustment to γ_i is $+\lambda\overline{\Delta}_1 x_i$.) Notice that the weight changes are proportional to the signal strength along the

corresponding connection (where the thresholds are considered as weights on connections with inputs clamped to the value -1). The idea is to picture the Δs as errors associated with the respective network nodes. The values of these are computed from the ouput layer "back" towards the input layer by propagation, with the appropriate weights, along the "backwards" network one layer at a time. Hence the name "error back-propagation algorithm", although it is simply an implementation of gradient-descent.

Now we shall turn to the more general case of a three layer feedforward neural network with n input nodes, M neurons in the middle (hidden) layer and m in the third (output) layer. A similar analysis can be carried out on a general multilayer feedforward neural network, but we will just consider one hidden layer.

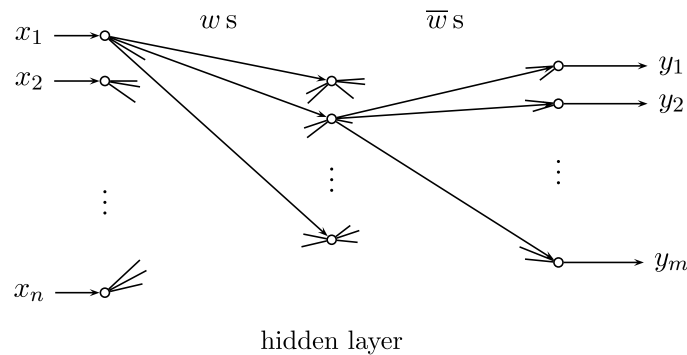


Figure 5.2: The general 3-layer feedforward neural network.

First, we need to set up the notation. Let $x_1, \ldots, x_n$ denote the input values and $d_1, \ldots, d_m$ the desired outputs. Let the weight from input unit i to hidden layer neuron j be denoted by w_{ji} and that from hidden layer neuron k to output neuron ℓ by $\overline{w}_{\ell k}$. All neurons in layers two and three are assumed to have differentiable (usually sigmoid) activation function $\varphi(\cdot)$.

The activation potential (net input to the activation function) of neuron j in the hidden layer is $v_j^{\mathrm{h}} = \sum_{i=0}^{n} w_{ji} x_i$, where $x_0 = -1$ and $w_{j0} = \theta_j$ is the threshold value.

The output from the neuron j in the hidden layer is $\varphi(v_j^{\mathrm{h}}) \equiv z_j$, say.

The activation potential of the output neuron ℓ is $v_\ell^{\mathrm{out}} = \sum_{k=0}^{M} \overline{w}_{\ell k} z_k$, where $z_0 = -1$ and $\overline{w}_{\ell 0} = \overline{\theta_\ell}$, the threshold for the unit.

The output from the neuron ℓ in the output layer is $y_\ell = \varphi(v_\ell^{\mathrm{out}})$.

We consider the error function $\mathcal{E} = \frac{1}{2}\sum_{r=1}^{m}(d_r - y_r)^2$.

The strategy is to try to minimize $\mathcal{E}$ using a gradient-descent algorithm based on the weight variables: $w \mapsto w - \lambda \operatorname{grad}\mathcal{E}$, where the gradient is with respect to all the weights (a total of $nM + M + Mm + m$ variables). We wish to calculate the partial derivatives $\partial\mathcal{E}/\partial w_{ji}$ and $\partial\mathcal{E}/\partial \overline{w}_{\ell k}$. We find

$$\begin{aligned}
\frac{\partial \mathcal{E}}{\partial \overline{w}_{\ell k}} &= \sum_{r=1}^{m} -(d_r - y_r)\frac{\partial y_r}{\partial \overline{w}_{\ell k}} \\
&= -(d_\ell - y_\ell)\frac{\partial \varphi(v_\ell^{\text{out}})}{\partial \overline{w}_{\ell k}} \\
&= -(d_\ell - y_\ell)\varphi'(v_\ell^{\text{out}})\frac{\partial v_\ell^{\text{out}}}{\partial \overline{w}_{\ell k}} \\
&= -(d_\ell - y_\ell)\varphi'(v_\ell^{\text{out}}) z_k \\
&= -\overline{\Delta}_\ell \, z_k,
\end{aligned}$$

where $\overline{\Delta}_\ell = (d_\ell - y_\ell)\varphi'(v_\ell^{\text{out}})$. Next, we have

$$\frac{\partial \mathcal{E}}{\partial w_{ji}} = \sum_{r=1}^{m} -(d_r - y_r)\frac{\partial y_r}{\partial w_{ji}}.$$

But

$$\begin{aligned}
\frac{\partial y_r}{\partial w_{ji}} &= \frac{\partial \varphi(v_r^{\text{out}})}{\partial w_{ji}} = \varphi'(v_r^{\text{out}})\frac{\partial v_r^{\text{out}}}{\partial w_{ji}} \\
&= \varphi'(v_r^{\text{out}})\frac{\partial}{\partial w_{ji}}\Big(\sum_{k=0}^{M}\overline{w}_{rk} z_k\Big) \\
&= \varphi'(v_r^{\text{out}})\overline{w}_{rj}\frac{\partial z_j}{\partial w_{ji}}, \text{ since only } z_j \text{ depends on } w_{ji}, \\
&= \varphi'(v_r^{\text{out}})\overline{w}_{rj}\varphi'(v_j^{\text{h}})\frac{\partial v_j^{\text{h}}}{\partial w_{ji}}, \text{ since } z_j = \varphi(v_j^{\text{h}}), \\
&= \varphi'(v_r^{\text{out}})\overline{w}_{rj}\varphi'(v_j^{\text{h}}) x_i.
\end{aligned}$$

It follows that

$$\begin{aligned}
\frac{\partial \mathcal{E}}{\partial w_{ji}} &= -\sum_{r=1}^{m} -(d_r - y_r)\varphi'(v_r^{\text{out}})\overline{w}_{rj}\varphi'(v_j^{\text{h}}) x_i \\
&= -\Big(\sum_{r=1}^{m}\overline{\Delta}_r \overline{w}_{rj}\Big)\varphi'(v_j^{\text{h}}) x_i \\
&= -\Delta_j x_i,
\end{aligned}$$

where $\Delta_j = \Big(\sum_{r=1}^{m}\overline{\Delta}_r \overline{w}_{rj}\Big)\varphi'(v_j^{\text{h}})$. As we have remarked earlier, the Δ s are calculated *backwards* through the network using the current weights. Thus we may formulate the algorithm as follows.

The Error Back-Propagation algorithm:

- set all weights to small random values;
- present an input-output pattern pair from the training set and then compute all activation potentials and internal unit outputs passing forward through the network, and the network outputs;
- compute all the Δ s by propagation backwards through the network and hence determine the partial derivatives of the error function with respect to all the weights using the *current* values of the weights;
- update all weights by the gradient-descent rule $w \mapsto w - \lambda \operatorname{grad} \mathcal{E}$, where λ is the learning rate;
- select another pattern repeating steps 2, 3 and 4 above until the error function is acceptably small for all patterns in the given training set.

It must be noted at the outset that there are *no known general convergence theorems for the back-propagation algorithm* (unlike the LMS algorithm discussed earlier). This is a game of trial and error. Nonetheless, the method has been widely applied and it could be argued that its study in the mid-eighties led to the recent resurgence of interest (and funding) of artificial neural networks. Some of the following remarks are various cook-book comments based on experiment with the algorithm.

Remarks 5.1.

1. Instead of changing the weights after each presentation, we could take a "batch" of pattern pairs, calculate the weight changes for each (but do not actually *make* the change), sum them and then make an accumulated weight change "in one lump". This would be closer in spirit to a "pure" gradient-descent method. However, such batch updating requires storing the various changes in memory which is considered undesirable so usually the algorithm is performed with an *update after each step.* This is called pattern mode as distinct from batch mode.

2. The learning parameter λ is usually selected in the range 0.05—0.25. One might wish to vary the value of λ as learning proceeds. A further possibility would be to have different values of the learning parameter for the different weights.

3. It is found in practice that the algorithm is slow to converge, i.e., it needs many iterations.

4. There may well be problems with *local* minima. The sequence of changing weights could converge towards a local minimum rather than a global one. However, one might hope that such a local minimum is acceptably close to the global minimum so that it would not really matter, in practice, should the system get stuck in one.

5. If the weights are large in magnitude then the sigmoid $\varphi(v)$ is near saturation (close to its maximum or minimum) so that its derivative will be small. Therefore changes to the weights according to the algorithm will also be correspondingly small and so learning will be slow. It is therefore a good idea to start with weights randomized in a small band around the midpoint of the sigmoid (where its slope is near a maximum). For the usual sigmoid $\varphi(v) = 1/(1+\exp(-v))$, this is at the value $v = 0$.

6. **Validation.** It is perhaps wise to hold back a fraction of the training set of input-output patterns (say, 10%) during training. When training is completed, the weights are fixed and the network can be tested on this remaining 10% of known data. This is the validation set. If the performance is unsatisfactory, then action can be taken, such as further training or retraining on a different 90% of the training set.

7. **Generalization.** It has been found that a neural network trained using this algorithm generalizes well, that is, it appears to produce acceptable results when presented with *input patterns it has not previously seen.* Of course, this will only be the case if the new pattern is "like" the training set. The network cannot be expected to give good results when presented with exceptional patterns.

   The network may suffer from overtraining. It seems that if the network is trained for too long, then it learns the actual training set rather than the underlying features of the patterns. This limits its ability to generalize.

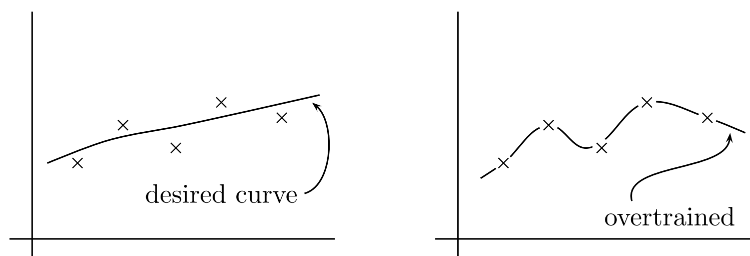


Figure 5.3: Overtraining is like overfitting a curve.

8. If we take the activation function to be the standard sigmoid function given as $\varphi(v) = 1/(1+\exp(-v))$, then

$$\varphi'(v) = \frac{e^{-v}}{(1+e^{-v})^2} = \varphi(v)\frac{e^{-v}}{(1+e^{-v})} = \varphi(v)\Big(1 - \frac{1}{(1+e^{-v})}\Big)$$
$$= \varphi(v)\big(1-\varphi(v)\big).$$

   This means that it is not necessary to compute $\varphi'(v)$ separately once we have found $\varphi(v)$. That is to say, the update algorithm can be rewritten so that the Δs can be calculated from the knowledge of $\varphi(v)$ directly. This saves computation time.

9. To picture how the network operates, it is usual to imagine that somehow certain characteristic *features* of the training set have been encapsulated in the network weights. The network has extracted the essence of the training set.

   Data compression. Suppose the network has 100 input units, say, 10 hidden units and 100 output units, and is trained on a set of input-output patterns where the output is the same as the input, i.e., the network is trained to learn the identity function. Such a neural network will have learned to perform 10:1 data compression. Instead of transmitting the 100 signal values, one would transmit the 10 values emitted by the hidden units and only on arrival at the receiving end would these values be passed into the final third layer (so there would be two neural networks, one at each end of the transmission line).

10. We could use a different error function if we wish. All that is required by the logic of the procedure is that it have a (global) minimum when the actual output and desired output are equal. The only effect on the calculations would be to replace the terms $-2(d_r - y_r)$ by the more general expression $\partial\mathcal{E}/\partial y_r$. This will change the formula for the "output unit Δ s", $\overline{\Delta}_\ell$, but once this has been done the formulae for the remaining weight updates are unchanged.

Instead of just the weight update $\Delta w_{ji}(t+1) = -\lambda \partial\mathcal{E}(t)/\partial w_{ji}$ at the iteration step $(t+1)$, it has often been found better to incorporate a so-called momentum term, $\alpha \Delta w_{ji}(t)$, giving

$$\Delta w_{ji}(t+1) = -\lambda \frac{\partial\mathcal{E}(t)}{\partial w_{ji}} + \alpha \Delta w_{ji}(t),$$

where $\Delta w_{ji}(t)$ is the *previous* weight increment and α is called the momentum parameter (chosen so that $0 \le \alpha \le 1$, but often taken to be 0.9). To see how such a term may prove to be beneficial, suppose that the weights

correspond to a "flat" spot on the error surface, so that $\partial\mathcal{E}(t)/\partial w_{ji}$ is nearly constant (and small). In this situation, we have

$$\begin{aligned}\Delta w_{ji}(t) &= -\lambda\frac{\partial\mathcal{E}}{\partial w_{ji}} + \alpha\Delta w_{ji}(t-1)\\ &= -\lambda\frac{\partial\mathcal{E}}{\partial w_{ji}} + \alpha\Big(-\lambda\frac{\partial\mathcal{E}}{\partial w_{ji}} + \alpha\Delta w_{ji}(t-2)\Big)\\ &= -\lambda\frac{\partial\mathcal{E}}{\partial w_{ji}}\big(1+\alpha+\alpha^2+\cdots+\alpha^{t-2}\big) + \alpha^{t-1}\Delta w_{ji}(1).\end{aligned}$$

We see that

$$\Delta w_{ji}(t) \to -\frac{\lambda}{(1-\alpha)}\frac{\partial\mathcal{E}}{\partial w_{ji}},$$

as $t\to\infty$, giving an effective learning rate of $\lambda/(1-\alpha)$ which is larger than λ if α is close to 1. Indeed, if we take $\alpha = 0.9$, as suggested above, then the effective learning rate is 10λ. This helps to speed up "convergence".

Now let us suppose that $\partial\mathcal{E}(t)/\partial w_{ji}$ fluctuates between successive positive and negative values. Then

$$\begin{aligned}\Delta w_{ji}(t+2) &= -\lambda\frac{\partial\mathcal{E}(t+1)}{\partial w_{ji}} + \alpha\Delta w_{ji}(t+1)\\ &= -\lambda\frac{\partial\mathcal{E}(t+1)}{\partial w_{ji}} - \alpha\lambda\frac{\partial\mathcal{E}(t)}{\partial w_{ji}} + \alpha^2\Delta w_{ji}(t)\\ &= -\lambda\Big(\frac{\partial\mathcal{E}(t+1)}{\partial w_{ji}} + \alpha\frac{\partial\mathcal{E}(t)}{\partial w_{ji}}\Big) + \alpha^2\Delta w_{ji}(t).\end{aligned}$$

If α is close to 1 and if $\partial\mathcal{E}(t+1)/\partial w_{ji} \sim -\partial\mathcal{E}(t)/\partial w_{ji}$ then the first (bracketed) term on the right hand side, above, is small and so the inclusion of the momentum term helps to damp down oscillatory behaviour of the weights.

On the other hand, if $\partial\mathcal{E}/\partial w_{ji}$ has the same sign for two consecutive steps, then the above calculation shows that including the momentum term "accelerates" the weight change.

A vestige of mathematical respectability can be conferred on such use of a momentum term according to the following argument. The idea is not to use just the error function for the current pattern, but rather to incorporate all the error functions for all the patterns presented throughout the history of the iteration process. Denote by $\mathcal{E}^{(1)}, \mathcal{E}^{(2)}, \dots, \mathcal{E}^{(t)}$ the error functions associated with those patterns presented at steps $1, 2, \dots, t$. Let $\mathcal{E}_t = \mathcal{E}^{(1)}(w(t)) + \cdots + \mathcal{E}^{(t)}(w(t))$ be the sum of these error functions *evaluated at the current time step* t. Now base the gradient-descent method on the function $\mathcal{E}_t$.

We get

$$\Delta w_{ji}(t+1) = -\lambda \frac{\partial \mathcal{E}_t}{\partial w_{ji}}$$
$$= -\lambda \frac{\partial \mathcal{E}^{(t)}}{\partial w_{ji}} - \lambda \frac{\partial}{\partial w_{ji}} \Big(\mathcal{E}^{(1)}(w(t)) + \cdots + \mathcal{E}^{(t-1)}(w(t)) \Big).$$

The second term on the right hand side above is like $\partial \mathcal{E}_{t-1}/\partial w_{ji} = \Delta w_{ji}(t)$, a momentum contribution, except that it is evaluated at step t not $t-1$.

Function approximation

We have thought of a multilayer feedforward neural network as a network which learns input-output pattern pairs. Suppose such a network has n units in the input layer and m in the output layer. Then any given function f, say, of n variables and with values in $\mathbb{R}^m$ determines input-output pattern pairs by the obvious pairing $(x, f(x))$. One can therefore consider trying to train a network to learn a given function and so it is of interest to know if and in what sense this can be achieved.

It turns out that there is a theorem of Kolmogorov, extended by Sprecher, on the representation of continuous functions of many variables in terms of linear combinations of a continuous function of linear combinations of functions of one variable.

Theorem 5.2 (Kolmogorov). *For any continuous function $f : [0,1]^n \to \mathbb{R}$ (on the n-dimensional unit cube), there are continuous functions $h_1, \dots, h_{2n+1}$ on $\mathbb{R}$ and continuous monotonic increasing functions g_{ij}, for $1 \le i \le n$, and $1 \le j \le 2n+1$, such that*

$$f(x_1, \dots, x_n) = \sum_{j=1}^{2n+1} h_j \Big(\sum_{i=1}^{n} g_{ij}(x_i) \Big).$$

The functions g_{ij} do not depend on f.

Theorem 5.3 (Sprecher). *For any continuous function on the n-dimensional unit cube, $f : [0,1]^n \to \mathbb{R}$, there is a continuous function h and continuous monotonic increasing functions $g_1, \dots, g_{2n+1}$ and constants $\lambda_1, \dots, \lambda_n$ such that*

$$f(x_1, \dots, x_n) = \sum_{j=1}^{2n+1} h \Big(\sum_{i=1}^{n} \lambda_i g_j(x_i) \Big).$$

The functions g_j and the constants λ_i do not depend on f.

This theorem implies that any continuous function f from a compact subset of $\mathbb{R}^n$ into $\mathbb{R}^m$ can be realized by some four layer feedforward neural network.

Such a feedforward network realization with two hidden layers is illustrated in the diagram (for the case $n = 2$ and $m = 1$).

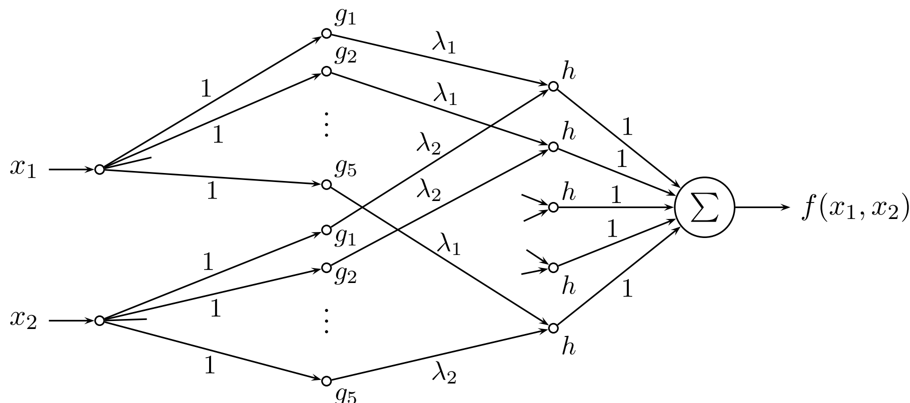


Figure 5.4: An illustration of a Kolmogorov/Sprecher network, with $n = 2$ and $m = 1$.

However, the precise form of the various activation functions and the values of the weights are unknown. That is to say, the theorem provides an (important) existence statement, rather than an explicit construction.

If we are prepared to relax the requirement that the representation be exact, then one can be more explicit as to the nature of the activation functions. That is, there are results to the effect that any f as above can be approximated, in various senses, to within any preassigned degree of accuracy, by a suitable feedforward neural network with certain specified activation functions. However, one usually does not know how many neurons are required. Usually, the more accurate the approximation is required to be, so will the number of units needed increase. We shall consider such function approximation results (E. K. Blum and L. L. Li, Neural Networks, 4 (1991), 511–515, see also K. Hornik, M. Stinchcombe and H. White, Neural Networks, 2 (1989), 359–366). The following example will illustrate the ideas.

Example 5.4. Any right-continuous simple function $f : [a, b] \to \mathbb{R}$ can be implemented by a 3-layer neural network comprising McCulloch-Pitts hidden units and a linear output unit.

By definition, a right-continuous simple function on the interval $[a, b]$ is a function of the form

$$f = f_0 \chi_{[b_0, b_1)} + f_1 \chi_{[b_1, b_2)} + \cdots + f_{n-1} \chi_{[b_{n-1}, b_n)} + f_n \chi_{\{b_n\}}$$

for some n and constants $f_0, \dots, f_n$, and suitable $a = b_0 < b_1 < \dots < b_n = b$, where χ_I denotes the characteristic function of the set I (it is 1 on I and 0 otherwise). Now we observe that

$$\chi_{[\alpha,\beta)}(x) = \mathrm{step}(x-\alpha) - \mathrm{step}(x-\beta)$$

and so for any $x \in [a,b]$ (and using $\mathrm{step}(0) = 1$),

$$\begin{aligned} f(x) = f_0(\mathrm{step}(x-b_0) - \mathrm{step}(x-b_1)) + f_1(\mathrm{step}(x-b_1) - \mathrm{step}(x-b_2)) + \\ \cdots + f_{n-1}(\mathrm{step}(x-b_{n-1}) - \mathrm{step}(x-b_n)) + f_n\,\mathrm{step}(x-b_n) \\ = f_0\,\mathrm{step}(x-b_0) + (f_1 - f_0)\,\mathrm{step}(x-b_1) + \\ \cdots + (f_n - f_{n-1})\,\mathrm{step}(x-b_n). \end{aligned}$$

The neural network implementing this mapping is as shown in the figure.

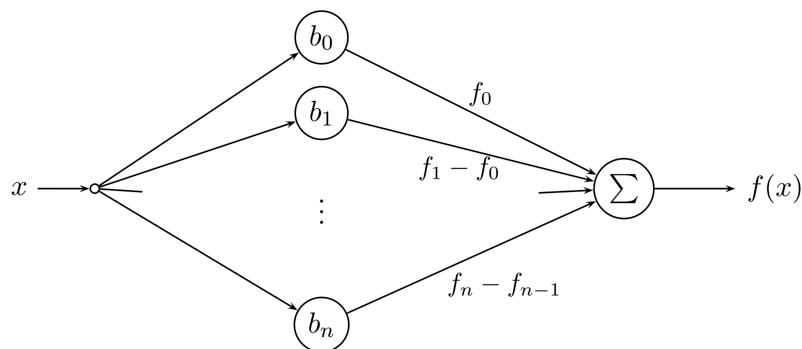


Figure 5.5: A neural network implementing a simple function.

We shall show that any continuous function on a square in $\mathbb{R}^2$ can be approximated by a 3-layer feedforward neural network. The result depends on the Stone-Weierstrass theorem which implies that any continuous function on such a square can be approximated by a sum of cosine functions.

Theorem 5.5 (Stone-Weierstrass). *Suppose that $\mathcal{A}$ is an algebra of continuous real-valued functions on the compact set K and that $\mathcal{A}$ separates points of K and contains constants. Then $\mathcal{A}$ is uniformly dense in $\mathcal{C}(K)$, the normed space of all real-valued continuous functions on K.*

Remark 5.6. To say that $\mathcal{A}$ is an algebra simply means that if f and g belong to $\mathcal{A}$, then so do $\alpha f + g$ and fg, for any $\alpha \in \mathbb{R}$. $\mathcal{A}$ separates points of K means that given any points $x, x' \in K$, with $x \neq x'$, then there is some $f \in \mathcal{A}$ such that $f(x) \neq f(x')$. In other words, $\mathcal{A}$ is sufficiently rich to be able to distinguish different points in K.

The requirement that $\mathcal{A}$ contain constants is to rule out the possibility that all functions may vanish at some common point. Uniform density is the statement that for any given $f \in \mathcal{C}(K)$ and any given $\varepsilon > 0$, there is some $g \in \mathcal{A}$ such that

$$|f(x) - g(x)| < \varepsilon \quad \text{for every } x \in K.$$

We will be interested in the case where K is a square or rectangle in $\mathbb{R}^2$ (or $\mathbb{R}^n$). The trigonometric identity $\cos\alpha\cos\beta = \frac{1}{2}\big(\cos(\alpha+\beta)+\cos(\alpha-\beta)\big)$ implies that the linear span of the family $\cos mx\ \cos ny$, for $m, n \in \mathbb{N}\cup\{0\}$, is, in fact, an algebra. This family clearly contains constants (taking $m = n = 0$) and separates the points of $[0,\pi]^2$. Thus, we get the following application of the Stone-Weierstrass Theorem.

Theorem 5.7 (Application of the Stone-Weierstrass theorem). *Suppose that $f : [0,\pi]^2 \to \mathbb{R}$ is continuous. For any given $\varepsilon > 0$, there is $N \in \mathbb{N}$ and constants a_{mn}, with $0 \le m, n \le N$, such that*

$$\Big|f(x,y) - \sum_{m,n=0}^{N} a_{mn}\cos mx\cos ny\Big| < \varepsilon$$

for all $(x,y) \in [0,\pi]^2$.

We emphasize that this is not Fourier analysis—the a_{mn}s are not any kind of Fourier coefficients.

Theorem 5.8. *Let $f : [0,\pi]^2 \to \mathbb{R}$ be continuous. For any given $\varepsilon > 0$, there is a 3-layer feedforward neural network with McCulloch-Pitts neurons in the hidden layer and a linear output unit which implements the function f on the square $[0,\pi]^2$ to within ε.*

Proof. According to the version of the Stone-Weierstrass theorem above, there is N and constants a_{mn}, with $0 \le, m, n \le N$ such that

$$\Big|f(x,y) - \sum_{m,n=0}^{N} a_{mn}\cos mx\cos ny\Big| < \tfrac{1}{2}\varepsilon$$

for all $(x,y) \in [0,\pi]^2$. Let $K = \max\{|a_{mn}| : 0 \le m, n \le N\}$. We write

$$\cos mx\cos ny = \tfrac{1}{2}(\cos(mx+ny) + \cos(mx-ny))$$

and also note that $|mx \pm ny| \le 2N\pi$ for any $(x,y) \in [0,\pi]^2$.

We can approximate $\frac{1}{2}\cos t$ on $[-2N\pi, 2N\pi]$ by a simple function, $\gamma(t)$, say,

$$|\tfrac{1}{2}\cos t - \gamma(t)| < \frac{\varepsilon}{4(N+1)^2K}$$

for all $t \in [-2N\pi, 2N\pi]$. Hence

$$\Big|\sum_{m,n} a_{mn} \underbrace{\cos mx \cos ny}_{\frac{1}{2}(\cos(mx+ny)+\cos(mx-ny))} - \sum_{m,n} a_{mn}\big(\gamma(mx+ny)+\gamma(mx-ny)\big)\Big|$$

$$\leq \sum_{m,n} |a_{mn}| \Big(\frac{\varepsilon}{4(N+1)^2 K} + \frac{\varepsilon}{4(N+1)^2 K} \Big)$$

$$< \sum_{m,n} \frac{\varepsilon}{2(N+1)^2} = \frac{\varepsilon}{2}.$$

But γ can be written as a sum as in the example above:

$$\gamma(t) = \sum_{j=1}^{M} w_j \operatorname{step}(t - \theta_j)$$

for suitable M, w_j and θ_j. Hence

$$\sum_{m,n} a_{m,n}\big(\gamma(mx+ny)+\gamma(mx-ny)\big)$$

$$= \sum_{m,n,j} \big\{ a_{mn} w_j \operatorname{step}(mx+ny-\theta_j) + a_{mn} w_j \operatorname{step}(mx-ny-\theta_j) \big\}$$

which provides the required network (see the figure). ∎

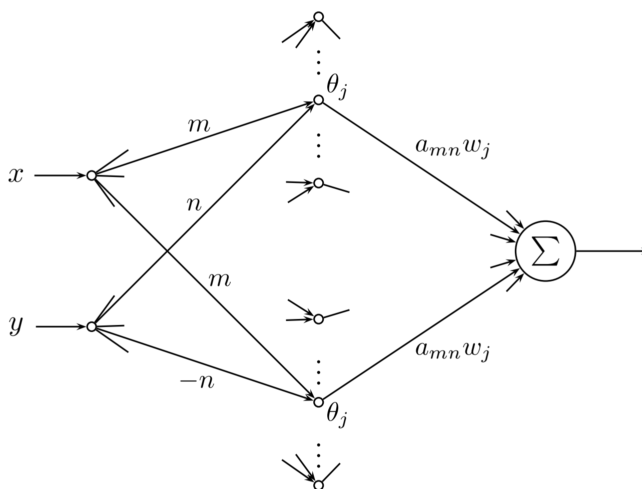


Figure 5.6: The 3-layer neural network implementing f to within ε. The network has $(N+1)^2 \times M + (N+1)^2 \times M$ hidden neurons (doubly labelled by m, n, j). The two weights to the hidden neuron labelled m, n, j in the top half are m and n, whereas those to the hidden neuron m, n, j in the bottom half are m and $-n$.

Remark 5.9. The particular square $[0, \pi]^2$ is not crucial—one can obtain a similar result, in fact, on any bounded region of $\mathbb{R}^2$ by rescaling the variables. Also, there is an analogous result in any number of dimensions (products of cosines in the several variables can always be rewritten as cosines of sums). The theorem gives an approximation in the uniform sense. There are various other results which, for example, approximate f in the mean square sense. It is also worth noting that with a little more work, one can see that sigmoid units could be used instead of the threshold units. One would take γ above to be a sum of "smoothed" step functions, rather a sum of step functions.

A problem with the above approach is with the difficulty in actually finding the values of N and the a_{mn}s in any particular concrete situation. By admitting an extra layer of neurons, a somewhat more direct approximation can be made. We shall illustrate the ideas in two dimensions.

Proposition 5.10. *Let $f : \mathbb{R}^2 \to \mathbb{R}$ be continuous. Then f is uniformly continuous on any square $S \subset \mathbb{R}^2$.*

Proof. Suppose that S is the square $S = [-R, R] \times [-R, R]$ and let $\varepsilon > 0$ be given. We must show that there is some $\delta > 0$ such that $|x - x'| < \delta$ and $x, x' \in S$ imply that

$$|f(x) - f(x')| < \varepsilon.$$

Suppose that this is not true. Then, no matter what $\delta > 0$ is, there will be points $x, x' \in S$ such that $|x - x'| < \delta$ but $|f(x) - f(x')| \geq \varepsilon$. In particular, if, for any $n \in \mathbb{N}$, we take $\delta = 1/n$, then there will be points x_n and x'_n in S with $|x_n - x'_n| < 1/n$ but $|f(x_n) - f(x'_n)| \geq \varepsilon$.

Now, (x_n) is a sequence in the compact set S and so has a convergent subsequence, $x_{n_k} \to x$ as $k \to \infty$, with $x \in S$. But then

$$|x'_{n_k} - x| \leq |x'_{n_k} - x_{n_k}| + |x_{n_k} - x| < \frac{1}{n_k} + |x_{n_k} - x| \to 0$$

as $k \to \infty$, i.e., $x'_{n_k} \to x$, as $k \to \infty$.

By the continuity of f, it follows that

$$|f(x_{n_k}) - f(x'_{n_k})| \to |f(x) - f(x)| = 0$$

which contradicts the inequality $|f(x_{n_k}) - f(x'_{n_k})| \geq \varepsilon$ and the proof is complete. ■

The idea of the construction is to approximate a given continuous function by step-functions built on small rectangles. Let B be a rectangle of the form

$$B = (a, b] \times (\alpha, \beta].$$

Notice that B contains its upper and right-hand edges, but not the lower or left-hand ones. It is a bit of a nuisance that we have to pay attention to this minor detail.

Let $\mathcal{S}$ denote the collection of finite linear combinations of functions of the form χ_B, for various Bs, where

$$\chi_B(x) = \begin{cases} 1, & x \in B \\ 0, & x \notin B \end{cases}$$

is the characteristic function of the rectangle B. Thus, $\mathcal{S}$ consists of functions of the form $a_1\chi_{B_1} + \cdots + a_m\chi_{B_m}$, with $a_1, \ldots, a_m \in \mathbb{R}$.

Theorem 5.11. *Let $f : \mathbb{R}^2 \to \mathbb{R}$ be continuous. Then for any $R > 0$ and any $\varepsilon > 0$ there is $g \in \mathcal{S}$ such that*

$$|f(x) - g(x)| < \varepsilon, \quad \text{for all } x \in (-R, R) \times (-R, R),$$

i.e., f can be uniformly approximated on the square $(-R, R) \times (-R, R)$ by elements of $\mathcal{S}$.

Proof. By our previous discussion, we know that f is uniformly continuous on the square $A = [-R, R] \times [-R, R]$. Hence there is $\delta > 0$ such that $|f(x) - f(x')| < \varepsilon$ whenever $x, x' \in A$ and $|x - x'| < \delta$. Subdivide A into n smaller squares as shown.

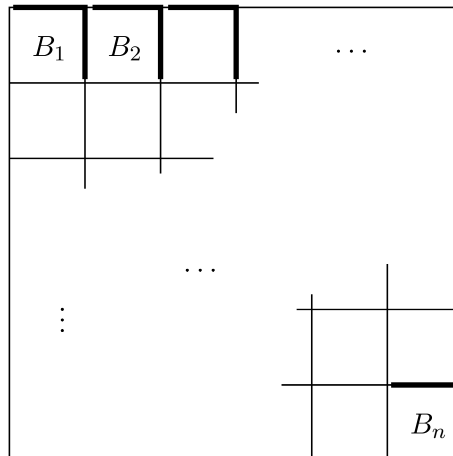


Figure 5.7: Divide A into many small rectangles, $B_1, \ldots, B_n$.

Choose n so large that each B_m has diagonal smaller than δ and let x_m be the centre of B_m. Then $|x - x_m| < \delta$ for any $x \in B_m$.

Put $g_m = f(x_m)$, and set

$$g(x) = \sum_{m=1}^{n} g_m \chi_{B_m}(x).$$

Then, for any $x \in (-R, R) \times (-R, R)$,

$$\begin{aligned} |g(x) - f(x)| &= |g_j - f(x)|, \quad \text{where } j \text{ is such that } x \in B_j, \\ &= |f(x_j) - f(x)| \\ &< \varepsilon \end{aligned}$$

since $|x_j - x| < \delta$. ■

We see that f is approximated on each B_m by its value at the centre. Notice that the approximation is rather more explicit than that of the previous scheme using the Stone-Weierstrass theorem. As a matter of fact, we could use that theorem here also, but as we have just remarked, there would then be no indication as to the value of n nor of the constants a_m.

To construct the appropriate network, we shall need two types of Mc Culloch-Pitts threshold units—differing on the value they take at the jump point. Let

$$H(s) = \begin{cases} 1, & s \geq 0 \\ 0, & s < 0 \end{cases} \qquad \text{and} \qquad H_0(s) = \begin{cases} 1, & s > 0 \\ 0, & s \leq 0. \end{cases}$$

Thus, $H(\cdot)$ is just the function step$(\cdot)$ used already. H_0 differs from H only in its value at 0—it is 0 there. The purpose of introducing these different versions of the step-function will soon become apparent. Indeed, $H_0(s-a) = \chi_{(a,\infty)}(s)$ and $H(-s+b) = \chi_{(-\infty,b]}(s)$ so that

$$\chi_{(a,b]}(s) = \underbrace{H_0(s-a)}_{\chi_{(a,\infty)}(s)} \; \underbrace{H(-s+b)}_{\chi_{(-\infty,b]}(s)}.$$

This means that we can implement the function $\chi_{(a,b]}$ by the network shown.

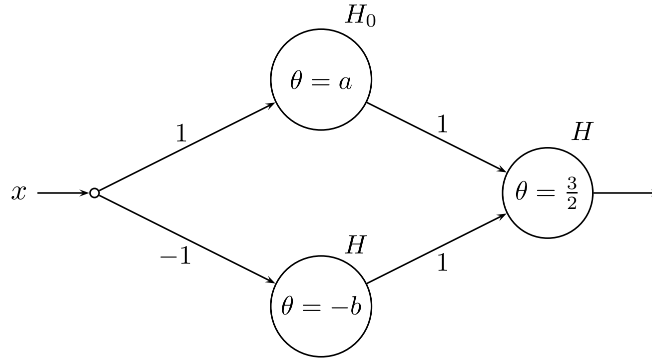


Figure 5.8: Mc Culloch-Pitts units to implement $\chi(a,b](s)$.

In our two dimensional case, we can write each B_j as

$$B_j = \{ (x_1, x_2) : a_1^j < x_1 \leq b_1^j,\ a_2^j < x_2 \leq b_2^j \}.$$

Thus

$$\chi_{B_j}(x_1, x_2) = \chi_{(a_1^j, b_1^j]}(x_1)\, \chi_{(a_2^j, b_2^j]}(x_2)$$

and we can implement such a function by combining the networks above, as shown.

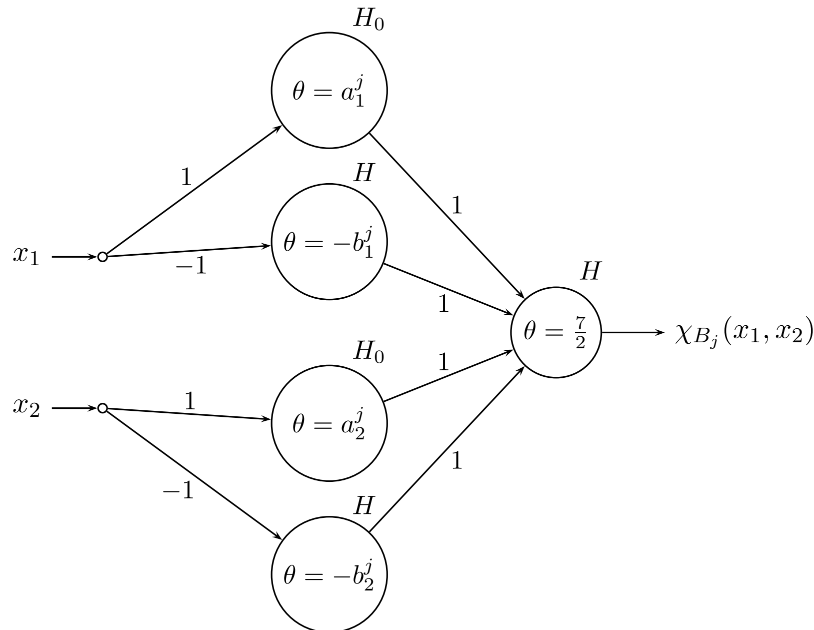


Figure 5.9: Mc Culloch-Pitts units to implement $\chi_{B_j}(x_1, x_2)$.

Theorem 5.12. *Suppose that $f : \mathbb{R}^2 \to \mathbb{R}$ is continuous. Then for any given $R > 0$ and $\varepsilon > 0$ there is a 4-layer feedforward neural network with Mc Culloch-Pitts units in layers 2 and 3 and with a linear output unit which implements f to within error ε uniformly on the square $(-R, R) \times (-R, R)$.*

Proof. We have done all the preparation. Now we just piece together the various parts. Let $g, B_1, \ldots, B_n$ etc. be an approximation to f (to within ε) as given above. We shall implement g using the stated network. Each χ_{B_j} is implemented by a 3-layer network, as above. The outputs of these are weighted by the corresponding g_j and then fed into a linear (summation) unit.

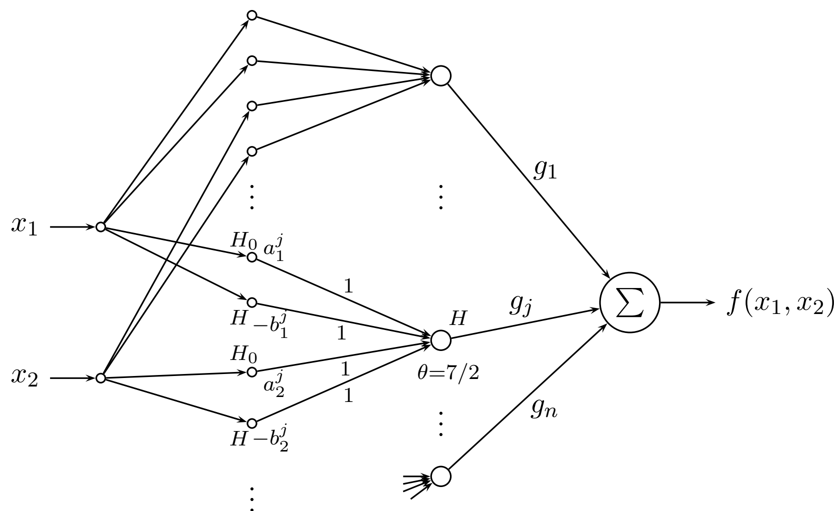


Figure 5.10: The network to implement f to within ε uniformly on the square $(-R, R) \times (-R, R)$.

The first layer consists of place holders for the input, as usual. The second layer consists of $4n$ threshold units (4 for each rectangle B_j, $j = 1, \ldots, n$). The third layer consists of n threshold units, required to complete the implementation of the n various χ_{B_j}s. The final output layer consists of a single linear unit.

For any given input (x_1, x_2) from the rectangle $(-R, R) \times (-R, R)$ precisely one of the units in the third layer will fire—this will be the jth corresponding to the unique j with $(x_1, x_2) \in B_j$. The system output is therefore equal to $\sum_{i=1}^{n} g_i \chi_{B_i}(x_1, x_2) = g_j = g(x_1, x_2)$, as required. ∎

See the work of K. Hornik, M. Stinchecombe and H. White, for example, for further results on function approximation.

Chapter 6

Radial Basis Functions

We shall reconsider the problem of implementing the mapping of a given set of input vectors $x^{(1)}, \dots, x^{(p)}$ in $\mathbb{R}^n$ into target values $y^{(1)}, \dots, y^{(p)}$ in $\mathbb{R}$,

$$x^{(i)} \rightsquigarrow y^{(i)}, \quad i = 1, \dots, p.$$

We seek a function h which not only realizes this association,

$$h(x^{(i)}) = y^{(i)}, \quad i = 1, \dots, p,$$

but which should also "predict" values when applied to new but "similar" input data. This means that we are not interested in finding a mapping h which works *precisely* for just these $x^{(1)}, \dots, x^{(p)}$. We would like a mapping h which will "generalize".

We will look at functions of the form $\varphi(\|x - x^{(i)}\|)$, i.e., functions of the distance between x and the prototype input $x^{(i)}$—so-called basis functions. Thus, we try

$$h(x) = \sum_{i=1}^{p} w_i \varphi(\|x - x^{(i)}\|)$$

and require $y^{(j)} = h(x^{(j)}) = \sum_{i=1}^{p} w_i \varphi(\|x^{(j)} - x^{(i)}\|)$. If we set $(A_{ji}) = (A_{ij}) = (\varphi(\|x^{(j)} - x^{(i)}\|)) \in \mathbb{R}^{p \times p}$, then our requirement is that

$$y^{(j)} = \sum_{i=1}^{p} A_{ji} w_i = (Aw)_j,$$

that is, $y = Aw$. If A is invertible, we get $w = A^{-1} y$ and we have found h.

A common choice for φ is a Gaussian function:

$$\varphi(s) = e^{-s^2/2\sigma^2}$$

—a so-called localized basis function.

With this choice, we see that

$$\begin{aligned} h(x^{(j)}) &= \sum_{i=1}^{p} w_i \varphi(\|x^{(j)} - x^{(i)}\|) \\ &= w_j \underbrace{\varphi(\|x^{(j)} - x^{(j)}\|)}_{=1} + \sum_{\substack{i=1 \\ i \neq j}}^{p} w_i \, \varphi(\|x^{(j)} - x^{(i)}\|) \\ &= w_j + \varepsilon \end{aligned}$$

where ε is small provided that the prototypes $x^{(j)}$ are reasonably well-separated. The various basis functions therefore "pick out" input patterns in the vicinity of specified spatial locations.

To construct a mapping h from $\mathbb{R}^n$ to $\mathbb{R}^m$, we simply construct suitable combinations of basis functions for each of the m components of the vector-valued map $h = (h_1, \dots, h_m)$. Thus,

$$h_k(x) = \sum_{i=1}^{p} w_{ki} \, \varphi(\|x - x^{(i)}\|).$$

This leads to

$$\begin{aligned} y_k^{(j)} = h_k(x^{(j)}) &= \sum_{i=1}^{p} w_{ki} \underbrace{\varphi(\|x^{(j)} - x^{(i)}\|)}_{A_{ji}} \\ &= \sum_{i=1}^{p} w_{ki} A_{ij} \\ &= (WA)_{kj} \end{aligned}$$

giving $Y = WA$, where $Y = (y^{(1)}, \dots, y^{(p)}) \in \mathbb{R}^{m \times p}$.

The combination of basis functions has been constructed so that it passes through all input-output data points. This means that h has "learned" these particular pairs—but we really want a system to learn the "essence" of the input-output association rather than the intricate details of a number of specific examples. We want the system to somehow capture the "features" of the data source, rather than any actual numbers—which may well be "noisy" (or slightly inaccurate) anyway.

Radial basis network

A so-called radial basis network is designed with the hope of incorporating this idea of realistic generalization ability. It has the following features.

- The number M of basis functions is chosen to be typically much smaller than p.
- The centres of the basis functions are not necessarily constrained to be precisely at the input data vectors. They may be determined during the training process itself.
- The basis functions are allowed to have different widths (σ)—these may also be determined by the training data.
- Biases or thresholds are introduced into the linear sum.

Thus, we are led to consider mappings of the form

$$x \mapsto y_k(x) = \sum_{j=1}^{M} w_{kj}\varphi_j(x) + w_{k0} \quad k = 1, 2, \ldots, m,$$

where $\varphi_j(x) = \exp(-\|x-\mu_j\|^2/2\sigma_j^2)$ and $\mu_j \in \mathbb{R}^n$. The corresponding 3-layer network is shown in the diagram. The first layer is the usual place-holder layer. The second layer has units with activation functions given by the basis functions φ_j, and the final layer consists of linear (summation) units.

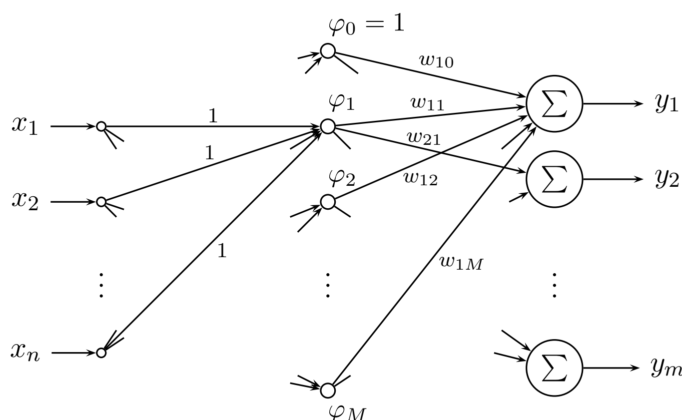


Figure 6.1: A radial basis network.

Training

To train such a network, the layers are treated quite differently. The second layer is subjected to unsupervised training—the input data being used to assign values to μ_j and σ_j. These are then held fixed and the weights to the output layer are found in the second phase of training.

Once a suitable set of values for the μ_js and σ_js has been found, we seek suitable w_{kj}s. These are given by

$$y_k(x) = \sum_{j=0}^{M} w_{kj}\varphi_j(x) \quad (\text{with } \varphi_0 \equiv 1)$$

i.e., $y = W\varphi$. This is just a linear network problem—for given x, we calculate $\varphi_j(x)$, so we know the hidden layer to output pattern pairs. We can therefore find W by minimizing the mean square error $\rightsquigarrow$ generalized inverse memory (OLAM) (or we can use the LMS algorithm to find this).

Returning to the problem of assigning values to the μ_js and σ_js—this can be done using the k-means clustering algorithm (MacQueen), which assigns vectors to one of a number of clusters.

k-means clustering algorithm

Suppose that the training set consists of N data vectors, each in $\mathbb{R}^n$, which are assumed to constitute k clusters. The centroid of a cluster is, by definition, the mean of its members.

- Take the first k data points as k cluster centres (giving clusters each with one member).
- Assign each of the remaining $N - k$ data points one by one to the cluster with the nearest centroid. After each assignment, recompute the centroid of the gaining cluster.
- Select each data point in turn and compute the distances to all cluster centroids. If the nearest centroid is not that particular data point's parent cluster, then reassign the data point (to the cluster with the nearest centroid) and recompute the centroids of the losing and gaining clusters.
- Repeat the above step until convergence—that is, until a full cycle through all data points in the training set fails to trigger any further cluster membership reallocations.
- Note: in the case of ties, any choice can be made.

This algorithm actually does converge, as we show next. The idea is to introduce an "energy" or "cost" function which decreases with every cluster reallocation.

Theorem 6.1 (k-means convergence theorem). *The k-means algorithm is convergent, i.e., the algorithm induces at most a finite number of cluster membership reallocations and then "stops".*

Proof. Suppose that after a number of steps of the algorithm the data set is partitioned into the k clusters $A_1, \dots, A_k$ with corresponding centroids $a_1, \dots, a_k$, so that

$$a_j = \frac{1}{n_j} \sum_{x \in A_j} x,$$

where n_j is the number of data points in A_j, $j = 1, \dots, k$. Let

$$E = E(A_1) + \cdots + E(A_k)$$

where $E(A_j) = \sum_{x \in A_j} \|x - a_j\|^2$, $j = 1, \dots, k$. Suppose that the data point x' is selected next and that $x' \in A_i$. If $\|x' - a_i\| \leq \|x' - a_j\|$, all j, then the algorithm makes no change.

On the other hand, if there is $\ell \neq i$ such that $\|x' - a_\ell\| < \|x' - a_i\|$, then the algorithm removes x' from A_i and reallocates it to A_m, say, where m is such that

$$\|x' - a_m\| = \min\{\, \|x' - a_j\| : j = 1, 2, \dots, k \,\}.$$

In this case, $\|x' - a_m\| < \|x' - a_i\|$.

We estimate the change in E. To do this, let $A'_i = A_i \setminus \{x'\}$ and let $A'_m = A_m \cup \{x'\}$, the reallocated clusters, and let a'_i and a'_m denote their centroids, respectively. We find

$$\begin{aligned} E(A_i) + E(A_m) &= \sum_{x \in A_i} \|x - a_i\|^2 + \sum_{x \in A_m} \|x - a_m\|^2 \\ &= \sum_{x \in A'_i} \|x - a_i\|^2 + \|x' - a_i\|^2 + \sum_{x \in A_m} \|x - a_m\|^2 \\ &> \sum_{x \in A'_i} \|x - a_i\|^2 + \|x' - a_m\|^2 + \sum_{x \in A_m} \|x - a_m\|^2 \\ &\geq \sum_{x \in A'_i} \|x - a'_i\|^2 + \sum_{x \in A'_m} \|x - a'_m\|^2 = E(A'_i) + E(A'_m), \end{aligned}$$

since $\sum \|x - \alpha\|^2$ is minimized when α is the centroid of the xs. Thus, every "active" step of the algorithm demands a strict decrease in the "energy" E of the overall cluster arrangement.

Clearly, a given finite number of data points can be assigned to k-clusters in a finite number of ways, and so E can assume only a finite number of possible values. It follows that the algorithm can only trigger a finite number of reallocations, and the result follows. ∎

Having found the centroids $c_1, \dots, c_k$, via the k-means algorithm, we set $\mu_1 = c_1, \dots, \mu_k = c_k$. The values of the various σ_js can then be calculated as the standard deviation of the data vectors in the appropriate cluster j, i.e.,

$$\sigma_j^2 = \frac{1}{n_j} \sum_{i=1}^{n_j} \|x^{(i)} - c_j\|^2, \quad j = 1, 2, \dots, k,$$

where n_j is the number of data vectors in cluster j.

We see that radial basis function networks are somewhat different from the standard feedforward sigmoidal-type neural network and are simpler to train. As some reassurance regarding their usefulness, we have the following "universal approximation theorem".

Theorem 6.2. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is a continuous function. Then for any $R > 0$ and $\varepsilon > 0$ there is a radial basis function network which implements f to within ε everywhere in the ball $\{ x : \|x\| \le R \}$.*

Proof. Let $\varphi(x, \mu, \sigma) = \exp(-(x-\mu)^2/2\sigma^2)$. Then we see that

$$\varphi(x, \mu, \sigma)\, \varphi(x, \mu', \sigma') = c\, \varphi(x, \lambda, \nu),$$

for suitable c, λ and ν. It follows that the linear span S of 1 and Gaussian functions of the form $\varphi(x, \mu, \sigma)$ is, in fact, an algebra. That is, if S denotes the collection of functions of the form

$$w_0 + w_1 \varphi(x, \mu_1, \sigma_1) + \cdots + w_j \varphi(x, \mu_j, \sigma_j)$$

for $j \in \mathbb{N}$ and $w_i \in \mathbb{R}$, $0 \le i \le j$, then S is an algebra.

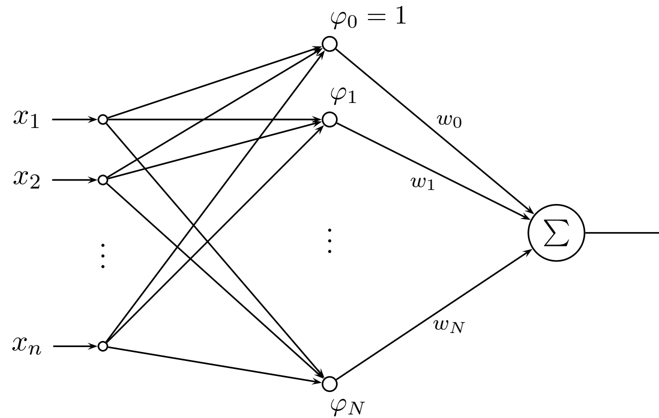


Figure 6.2: The radial basis network approximating the function f uniformly on the ball $\{ x : \|x\| \le R \}$.

Clearly, S contains constants and also S separates points of $\mathbb{R}^n$. Indeed, if $z \neq z'$ in $\mathbb{R}^n$, then

$$\varphi(z, z, 1) = 1 \neq \varphi(z', z, 1).$$

We can now apply the Stone-Weierstrass theorem to conclude that f can be uniformly approximated on any ball $\{ x : \|x\| \le R \}$ by an element from S. Thus, for given $\varepsilon > 0$, there is $N \in \mathbb{N}$ and $w_0, w_1, \dots, w_N$, μ_i and σ_i for $i = 1, \dots, N$ such that

$$\Big| f(x) - \sum_{i=0}^{N} w_i \varphi_i(x) \Big| < \varepsilon$$

for all x with $\|x\| \le R$, where $\varphi_0 = 1$ and $\varphi_i(x) = \varphi(x, \mu_i, \sigma_i)$ for $i = 1, \dots, N$. Such an approximating sum is implemented by a radial basis network, as shown. ∎

Chapter 7

Recurrent Neural Networks

In this chapter, we reconsider the problem of content addressable memory (CAM). The idea is to construct a neural network which has stored (or learned) a number of prototype patterns (also called exemplars) so that when presented with a possible "noisy" or "corrupt" version of one of these, it nonetheless correctly recalls the appropriate pattern—autoassociation.

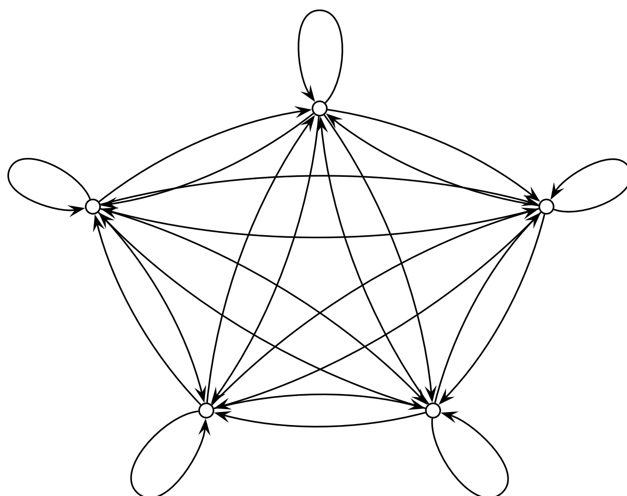


Figure 7.1: A recurrent neural network with 5 neurons.

We consider a recurrent neural network of n bipolar threshold units. Once the exemplars are loaded, the system is fixed, that is, the weights and thresholds are kept fixed—there is no further learning. The purpose of the network is to simply recall a suitable pattern appropriate to the input. The state of the network is just the knowledge of which neurons are firing and which are not. We shall model such states of the system by elements of the bipolar hypercube $\{-1, 1\}^n$, where the value 1 corresponds to "firing" and -1 to "not firing".

If the state of the system is $x = (x_1, \dots, x_n)$, then the net internal activation potential of neuron i is equal to $\sum_{k=1}^n w_{ik} x_k - \theta_i$, where w_{ik} is the connection weight from neuron k to neuron i, and θ_i is the threshold of neuron i. Passing this through the bipolar threshold function gives the signal value $\operatorname{sign}(\sum_{k=1}^n w_{ik} x_k - \theta_i)$, where

$$\operatorname{sign}(v) = \begin{cases} 1, & v \geq 0 \\ -1, & v < 0. \end{cases}$$

The neural network can be realized as a dynamical system as follows. Consider the state of the system at discrete time steps $t = 0, 1, 2, \dots$. If $x(t) = (x_1(t), \dots, x_n(t))$ in $\{-1, 1\}^n$ denotes the state of the system at time t, we define the state at time $t+1$ by the assignment

$$x_i(t+1) = \operatorname{sign}\Big(\sum_{k=1}^n w_{ik} x_k(t) - \theta_i \Big),$$

$1 \leq i \leq n$, with the agreement that $x_i(t+1) = x_i(t)$ if $\sum_{k=1}^n w_{ik} x_k(t) - \theta_i = 0$. Thus, as time goes by, the state of the system hops about on the vertices of the hypercube $[-1, 1]^n$. The time evolution set out above is called the synchronous (or parallel) dynamics, because all components x_i are updated *simultaneously*.

An alternative possibility is to update only *one* unit at a time, that is, to consider the neurons individually rather than en masse. This is called sequential (or asynchronous) dynamics and is defined by

$$x_i(t+1) = \operatorname{sign}\Big(\sum_{k=1}^{i-1} w_{ik} x_k(t+1) + \sum_{k=i}^{n} w_{ik} x_k(t) - \theta_i \Big),$$

for $1 \leq i \leq n$. Thus, first x_1 and only x_1, is updated, then just x_2 is updated using the latest value of x_1, then x_3, using the latest values of x_1 and x_2, and so on. Then the transition $(x_1(t), \dots, x_n(t))$ to $(x_1(t+1), \dots, x_n(t+1))$ proceeds in n intermediate steps. At each such step, the state change of only *one* neuron is considered. As a consequence, the state vector x hops about on the vertices of the hypercube $[-1, 1]^n$ but can either remain where it is or only move to a nearest-neighbour vertex at each such step. If we think of n such intermediate steps as constituting a cycle, then we see that each neuron is "activated" once and only once per cycle. (A variation on this theme is to update the neurons in a random order so that each neuron is only updated once per cycle *on average*.)

Such systems were considered by J. Hopfield in the early eighties and this provided impetus to the study of neural networks. A recurrent neural network as above and with the asynchronous (or random) mode of operation is known as the Hopfield model.

There is yet a further possibility—known as block-sequential dynamics. Here the idea is to divide the neurons into a number of groups and then update each group synchronously, but the groups sequentially, for example, synchronously update group one, then likewise group two, but using the latest values of the states of the neurons in group one, and so on.

So far we have not said anything about the values to be assigned to the weights. Before doing this, we first consider the network abstractly and make the following observation. The state vector moves around in the space $\{-1,1\}^n$ which contains only finitely-many points (2^n, in fact) and its position at $t+1$ depends only on where it is at time t (and, in the sequential case, which neuron is to be updated next). Thus, either it will reach some point and *stop*, or it will reach a configuration for a second time and then *cycle repeatedly forever*.

To facilitate further investigation, we define

$$\begin{aligned} E(x) &= \tfrac{1}{2} x^T W x + x^T \theta \\ &= -\tfrac{1}{2} \sum_{i,j=1}^{n} x_i w_{ij} x_j + \sum_{i=1}^{n} x_i \theta_i \end{aligned}$$

where $x = (x_1, \dots, x_n)$ is the state vector of the network, $W = (w_{ij})$ and $\theta = (\theta_i)$.

The function E is called the energy associated with x (by analogy with lattice magnetic (or spin) systems). We can now state the basic result for sequential dynamics.

Theorem 7.1. *Suppose that the synaptic weight matrix (w_{ij}) is symmetric with non-negative diagonal terms. Then the sequential mode of operation of the recurrent neural network has no cycles.*

Proof. Consider a single update $x_i \mapsto x_i'$, and all other x_j s remain unchanged. Then we calculate the energy difference

$$\begin{aligned} E(x') - E(x) = -\tfrac{1}{2} \sum_{\substack{j=1 \\ j \neq i}}^{n} (x_i' w_{ij} x_j + x_j w_{ji} x_i') - \tfrac{1}{2} x_i' w_{ii} x_i' + x_i' \theta_i \\ + \tfrac{1}{2} \sum_{\substack{j=1 \\ j \neq i}}^{n} (x_i w_{ij} x_j + x_j w_{ji} x_i) + \tfrac{1}{2} x_i w_{ii} x_i - x_i \theta_i \end{aligned}$$

(all terms not involving x_i or x'_i cancel out)

$$= -\sum_{\substack{j=1\\ j\neq i}}^{n} x'_i w_{ij} x_j - \tfrac{1}{2} x'_i w_{ii} x'_i + x'_i \theta_i$$

$$+ \sum_{\substack{j=1\\ j\neq i}}^{n} x_i w_{ij} x_j + \tfrac{1}{2} x_i w_{ii} x_i - x_i \theta_i$$

using the symmetry of (w_{ij})

$$= -\sum_{j=1}^{n} x'_i w_{ij} x_j + x'_i w_{ii} x_i - \tfrac{1}{2} x'_i w_{ii} x'_i + x'_i \theta_i$$

$$+ \sum_{j=1}^{n} x_i w_{ij} x_j - x_i w_{ii} x_i + \tfrac{1}{2} x_i w_{ii} x_i - x_i \theta_i$$

$$= (x_i - x'_i) \sum_{j=1}^{n} w_{ij} x_j + \tfrac{1}{2} w_{ii} (2x'_i x_i - x'_i x'_i - x_i x_i) + (x'_i - x_i)\theta_i$$

$$= -(x'_i - x_i) \sum_{j=1}^{n} w_{ij} x_j - \tfrac{1}{2} w_{ii} (x'_i - x_i)^2 + (x'_i - x_i)\theta_i$$

$$= -\tfrac{1}{2} w_{ii} (x'_i - x_i)^2 - (x'_i - x_i)\Big(\sum_{j=1}^{n} w_{ij} x_j - \theta_i \Big).$$

Suppose that $x'_i \neq x_i$. Then $x'_i - x_i = 2$ if $x'_i = 1$ but $x'_i - x_i = -2$ if $x'_i = -1$. It follows that $\operatorname{sign}(x'_i) = \operatorname{sign}(x'_i - x_i)$. Furthermore, by definition of the evolution, $x'_i = \operatorname{sign}(\sum_{j=1}^{n} w_{ij} x_j - \theta_i)$, and so we see that $(x'_i - x_i)$ and $(\sum_{j=1}^{n} w_{ij} x_j - \theta_i)$ have the *same* sign.

By hypothesis, $w_{ii} \geq 0$, and so if $x'_i \neq x_i$ then $E(x') < E(x)$. This means that any asynchronous change in x leads to a strict decrease in the energy function E. It follows that the system can never leave and then subsequently return to any state configuration. In particular, there can be no cycles. ■

This leads immediately to the following corollary.

Corollary 7.2. *A recurrent neural network with symmetric synaptic weight matrix with non-negative diagonal elements reaches a fixed point after a finite number of steps when operated in sequential mode.*

Proof. The state space is finite (and there are only a finite number of neurons to be updated), so if there are no cycles the iteration process must simply stop, i.e., the system reaches a fixed point. ■

We turn now to the question of choosing the synaptic weights. We set all thresholds to zero, $\theta_i = 0$, $1 \le i \le n$. Consider the problem of storing a single pattern $z = (z_1, \dots, z_n)$. We would like input patterns (starting configurations) close to z to converge to z via the asynchronous dynamics. In particular, we would like z to be a fixed point. Now, according to the evolution, if $x(0) = z$ and the ith component of $x(0)$ is updated first, then

$$\begin{aligned} x_i(1) &= \operatorname{sign}\Big(\sum_{k=1}^{n} w_{ik} x_k(0)\Big) \\ &= \operatorname{sign}\Big(\sum_{k=1}^{n} w_{ik} z_k\Big). \end{aligned}$$

For z to be a fixed point, we must have $z_i = \operatorname{sign}\Big(\sum_{k=1}^{n} w_{ik} z_k\Big)$ (or possibly $\sum_{k=1}^{n} w_{ik} z_k = 0$). This requires that

$$\sum_{k=1}^{n} w_{ik} z_k = \alpha z_i$$

for some $\alpha \ge 0$. Guided by our experience with the adaptive linear combiner (ALC) network, we try $w_{ik} = \alpha z_i z_k$, that is, we choose the synaptic matrix to be $(w_{ik}) = \alpha z z^T$, the outer product (or Hebb rule). We calculate

$$\sum_{k=1}^{n} w_{ik} z_k = \sum_{k=1}^{n} \alpha z_i z_k z_k = \alpha n z_i$$

since $z_k^2 = 1$. It follows that the choice $(w_{ik}) = \alpha z z^T$, $\alpha \ge 0$, does indeed give z as a fixed point.

Next, we consider what happens if the system starts out not exactly in the state z, but in some perturbation of z, say $\widehat{z}$. We can think of $\widehat{z}$ as z but with some of its bits flipped. Taking $x(0) = \widehat{z}$, we get

$$x_i(1) = \operatorname{sign}\Big(\sum_{k=1}^{n} w_{ik} \widehat{z}_k\Big).$$

The choice $\alpha = 0$ would give $w_{ik} = 0$ and so $x_i(1) = \widehat{z}_i$ and $\widehat{z}$ (and indeed *any* state) would be a fixed point. The dynamics would be trivial—nothing moves. This is no good for us—we want states $\widehat{z}$ sufficiently close to z to "evolve" into z. Incidentally, the same remark applies to the choice of synaptic matrix $(w_{ik}) = \mathbb{1}_n$, the unit $n \times n$ matrix. For such a choice, it is clear that *all* vectors are fixed points. We want the stored patterns to act as "attractors" each with non-trivial "basin of attraction". Substituting

$(w_{ik}) = \alpha z z^T$, with $\alpha > 0$, we find

$$\begin{aligned} x_i(1) &= \operatorname{sign}\Big(\sum_{k=1}^{n} \alpha z_i z_k \widehat{z}_k\Big) \\ &= \operatorname{sign}\Big(z_i \sum_{k=1}^{n} z_k \widehat{z}_k\Big), \text{ since } \alpha > 0, \\ &= \operatorname{sign}\big(z_i(n - 2\rho(z,\widehat{z}))\big), \end{aligned}$$

where $\rho(z,\widehat{z})$ is the Hamming distance between z and $\widehat{z}$, i.e., the number of differing bits. It follows that if $\rho(z,\widehat{z}) < n/2$, then $\operatorname{sign}\big(z_i(n - 2\rho(z,\widehat{z}))\big) = \operatorname{sign}(z_i) = z_i$, so that $x(1) = z$. In other words, any vector $\widehat{z}$ with $\rho(z,\widehat{z}) < n/2$ is mapped onto the fixed point z directly, in *one* iteration cycle. (Whenever a component of $\widehat{z}$ is updated, either it agrees with the corresponding component of z and so remains unchanged, or it differs and is then mapped immediately onto the corresponding component of z. Each flip moves $\widehat{z}$ towards z.) We have therefore shown that the basin of direct attraction of z contains the disc $\{\widehat{z} : \rho(z,\widehat{z}) < n/2\}$.

Can we say anything about the situation when $\rho(z,\widehat{z}) > n/2$? Evidently, $\operatorname{sign}\big(z_i(n - 2\rho(z,\widehat{z}))\big) = \operatorname{sign}(-z_i) = -z_i$, and so $x(1) = -z$. Is $-z$ a fixed point? According to our earlier discussion, the state $-z$ is a fixed point if we take the synaptic matrix to be given by $(-z)(-z)^T$. But $(-z)(-z)^T = zz^T$ and we conclude that $-z$ is a fixed point for the network above. (Alternatively, we could simply check this using the definition of the dynamics.)

We can also use this to see that $\widehat{z}$ is directly attracted to $-z$ whenever $\rho(z,\widehat{z}) > n/2$. Indeed, $\rho(z,\widehat{z}) > n/2$ implies that $\rho(-z,\widehat{z}) < n/2$ and so, arguing as before, we deduce that $\widehat{z}$ is directly mapped onto the fixed point $-z$. If $\rho(z,\widehat{z}) = n/2$, then there is no change, by definition of the dynamics.

How can we store many patterns? We shall try the Hebbian rule,

$$W \equiv (w_{ik}) = \sum_{\mu=1}^{p} z^{(\mu)} z^{(\mu)T}$$

so that $w_{ik} = \sum_{\mu=1}^{p} z_i^{(\mu)} z_k^{(\mu)}$, where $z^{(1)}, \dots, z^{(p)}$ are the patterns to be stored. Starting with $x(0) = z$, we find

$$\begin{aligned} z_i(1) &= \operatorname{sign}((Wz)_i) = \operatorname{sign}\Big(\sum_{k=1}^{n} w_{ik} z_k\Big) \\ &= \operatorname{sign}\Big(\sum_{k=1}^{n}\sum_{\mu=1}^{p} z_i^{(\mu)} z_k^{(\mu)} z_k\Big). \end{aligned}$$

Notice that if $z^{(1)}, \dots, z^{(p)}$ are pairwise orthogonal, then

$$\sum_{\mu=1}^{p} z^{(\mu)} z^{(\mu)T} z^{(j)} = z^{(j)} \|z^{(j)}\|^2$$

so that if $z = z^{(j)}$, then

$$\begin{aligned} z_i(1) &= \operatorname{sign}\big(\|z^{(j)}\|^2 z_i^{(j)}\big) \\ &= \operatorname{sign}(z_i^{(j)}) \\ &= z_i^{(j)}. \end{aligned}$$

It follows that each exemplar $z^{(1)}, \dots, z^{(p)}$ is a fixed point if they are pairwise orthogonal. In general (not necessarily pairwise orthogonal), we have

$$\begin{aligned} \sum_{k=1}^{n} w_{ik} z_k^{(j)} &= \sum_{\mu=1}^{p} \sum_{k=1}^{n} z_i^{(\mu)} z_k^{(\mu)} z_k^{(j)} \\ &= z_k^{(j)} \sum_{k=1}^{n} z_i^{(j)} z_k^{(j)} + \sum_{\mu \neq j}^{p} z_i^{(\mu)} \sum_{k=1}^{n} z_k^{(\mu)} z_k^{(j)} \\ &= n\Big(z_i^{(j)} + \frac{1}{n} \sum_{\mu \neq j}^{p} z_i^{(\mu)} \sum_{k=1}^{n} z_k^{(\mu)} z_k^{(j)} \Big) \qquad (*) \end{aligned}$$

The right hand side has the same sign as $z_i^{(j)}$ whenever the first term is dominant. To get a rough estimate of the situation, let us suppose that the protoype vectors are chosen at random—this gives an "average" indication of what we might expect to happen (a "typical" situation). Then the second term in the brackets on the right hand side of $(*)$ involves a sum of $n(p-1)$ terms which can take on the values ± 1 with equal probability. By the Central Limit Theorem, the term

$$\frac{1}{n} \sum_{\mu \neq j}^{p} z_i^{(\mu)} \sum_{k=1}^{n} z_k^{(\mu)} z_k^{(j)}$$

has approximately a normal distribution with mean 0 and variance $(p-1)/n$. This means that if p/n is small, then this term will be small with high probability, so that the pattern $z^{(j)}$ will be stable.

Consider now a recurrent neural network operating under the synchronous (parallel) mode,

$$x_i(t+1) = \operatorname{sign}\Big(\sum_{k=1}^{n} w_{ik} x_k(t) - \theta_i\Big).$$

The dynamics is described by the following theorem.

Theorem 7.3. *A recurrent neural network with symmetric synaptic matrix operating in synchronous mode converges to a fixed point or to a cycle of period two.*

Proof. The method uses an energy function, but now depending on the state of the network at two consecutive time steps. We define

$$\begin{aligned} G(t) &= -\sum_{i,j=1}^{n} x_i(t) w_{ij} x_j(t-1) + \sum_{i=1}^{n} \big(x_i(t) + x_i(t-1)\big)\theta_i \\ &= -x^T(t) W x(t-1) + \big(x^T(t) + x^T(t-1)\big)\theta. \end{aligned}$$

Hence

$$\begin{aligned} G(t+1) - G(t) &= -x^T(t+1) W x(t) + \big(x^T(t+1) + x^T(t)\big)\theta \\ &\qquad + x^T(t) W x(t-1) - \big(x^T(t) + x^T(t-1)\big)\theta \\ &= \big(x^T(t-1) - x^T(t+1)\big) W x(t) \\ &\qquad + \big(x^T(t+1) - x^T(t-1)\big)\theta, \text{ using } W = W^T, \\ &= -\big(x^T(t+1) - x^T(t-1)\big)(W x(t) - \theta) \\ &= -\sum_{i=1}^{n} \big(x_i(t+1) - x_i(t-1)\big)\Big(\sum_{k=1}^{n} w_{ik} x_k(t) - \theta_i\Big). \end{aligned}$$

But, by definition of the dynamics,

$$x_i(t+1) = \operatorname{sign}\Big(\sum_{k=1}^{n} w_{ik} x_k(t) - \theta_i\Big),$$

which means that if $x(t+1) \neq x(t-1)$ then $x_i(t+1) \neq x_i(t-1)$ for some i and so

$$\big(x_i(t+1) - x_i(t-1)\big)\Big(\sum_{k=1}^{n} w_{ik} x_k(t) - \theta_i\Big) > 0$$

and we conclude that $G(t+1) < G(t)$. (We assume here that the threshold function is strict, that is, the weights and threshold are such that $x = (x_1, \dots, x_n) \mapsto \sum_k w_{ik} x_k - \theta_i$ never vanishes on $\{-1, 1\}^n$.) Since the state

space $\{-1,1\}^n$ is finite, G cannot decrease indefinitely and so eventually $x(t+1) = x(t-1)$. It follows that either $x(t+1)$ is a fixed point or

$$\begin{aligned} x(t+2) &= \text{ the image of } x(t+1) \text{ under one iteration} \\ &= \text{ the image of } x(t-1), \text{ since } x(t+1) = x(t-1) \\ &= x(t) \end{aligned}$$

and we have a cycle $x(t) = x(t+2) = x(t+4) = \dots$ and $x(t-1) = x(t+1) = x(t+3) = \dots$, that is, a cycle of period 2. ■

We can understand G and this result (and rederive it) as follows. We shall design a network which will function like two copies of our original. We consider $2n$ nodes, with corresponding state vector $(z_1, \dots, z_{2n}) \in \{0,1\}^{2n}$. The synaptic weight matrix $\widehat{W} \in \mathbb{R}^{2n \times 2n}$ is given by

$$\widehat{W} = \begin{pmatrix} 0 & W \\ W & 0 \end{pmatrix}$$

so that $\widehat{W}_{ij} = 0$ and $\widehat{W}_{(n+i)(n+j)} = 0$ for all $1 \le i, j \le n$ and $\widehat{W}_{i(n+j)} = w_{ij}$ and $\widehat{W}_{(n+i)j} = w_{ij}$. Notice that $\widehat{W}$ is symmetric and has zero diagonal entries. The thresholds are set as $\widehat{\theta}_i = \widehat{\theta}_{n+i} = \theta_i$, $1 \le i \le n$.

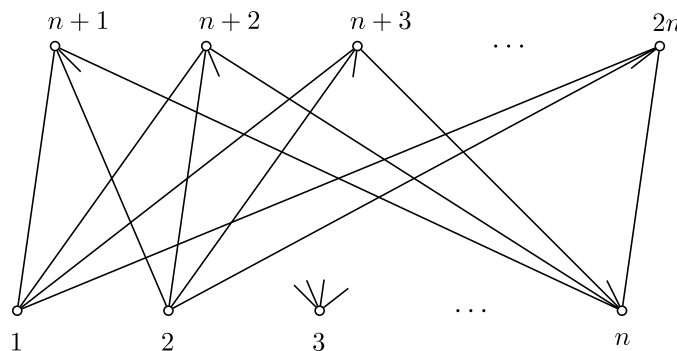


Figure 7.2: A "doubled" network. There are no connections between nodes $1, \dots, n$ nor between nodes $n+1, \dots, 2n$. If we were to "collapse" nodes $n+i$ onto i, then we would recover the original network.

Let $x(0)$ be the initial configuration of our original network (of n nodes). Set $z(0) = x(0) \oplus x(0)$, that is, $z_i(0) = x_i(0) = z_{n+i}(0)$ for $1 \le i \le n$.

We update the larger (doubled) network sequentially in the order node $n+1, \dots, 2n, 1, \dots, n$. Since there are no connections within the set of nodes $1, \dots, n$ and within the set $n+1, \dots, 2n$, we see that the outcome is

$$\begin{aligned} z(0) = x(0) \oplus x(0) &\longmapsto x(2) \oplus x(1) = z(1) \\ &\longmapsto x(4) \oplus x(3) = z(2) \\ &\longmapsto x(6) \oplus x(5) = z(3) \\ &\qquad \vdots \\ &\longmapsto x(2t) \oplus x(2t-1) = z(t) \end{aligned}$$

where $x(s)$ is the state of our original system run in *parallel* mode. By the theorem, the larger system reaches a fixed point so that $z(t) = z(t+1)$ for all sufficiently large t. Hence

$$x(2t) = x(2t+2) \text{ and } x(2t-1) = x(2t+1)$$

for all sufficiently large t—which means that the original system has a cycle of length 2 (or a fixed point).

The energy function for the larger system is

$$\begin{aligned} \widehat{E}(z) &= -\tfrac{1}{2} z^T \widehat{W} z + z^T \widehat{\theta} \\ &= -\tfrac{1}{2} \big(x(2t) \oplus x(2t-1)\big)^T \widehat{W} \big(x(2t) \oplus x(2t-1)\big) \\ &\qquad\qquad + \big(x(2t) \oplus x(2t-1)\big)^T \widehat{\theta} \\ &= -\tfrac{1}{2} \big(x(2t) \oplus x(2t-1)\big)^T \big(Wx(2t-1) \oplus Wx(2t)\big) \\ &\qquad\qquad + \big(x(2t) \oplus x(2t-1)\big)^T \widehat{\theta} \\ &= -\tfrac{1}{2} \big(x(2t)^T W x(2t-1) x(2t-1)^T W x(2t)\big) \\ &\qquad\qquad + x(2t)^T \theta + x(2t-1)^T \theta \\ &= -x(2t)^T W x(2t-1) + \big(x(2t) \\ &= x(2t-1)\big)^T \theta \\ &= G(2t) \end{aligned}$$

since $W = W^T$. We know that $\widehat{E}$ is decreasing. Thus we have shown that parallel dynamics leads to a fixed point or a cycle of length (at most) 2 *without* the extra condition that the threshold function be “strict”.

The BAM network

A special case of a recurrent neural network is the bidirectional associative memory (BAM). The BAM architecture is as shown.

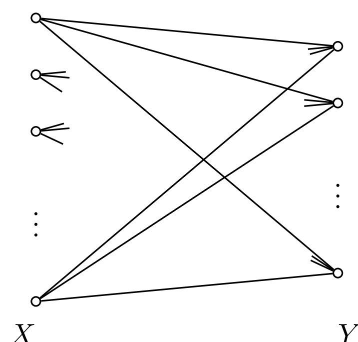


Figure 7.3: The BAM network.

It consists of two subsets of bipolar threshold units, each unit of one subset being fully connected to all units in the other subset, but with *no* connections between the neurons *within* each of the two subsets. Let us denote these subsets of neurons by $X = \{x_1, \dots, x_n\}$ and $Y = \{y_1, \dots, y_m\}$ and let w_{ji}^{XY} denote the weight corresponding to the connection from x_i to y_j and w_{ij}^{YX} that corresponding to the connection from y_j to x_i. These two values are taken to be equal.

The operation of the BAM is via block sequential dynamics—first the y s are updated, and then the x s (based on the latest values of the y s). Thus,

$$y_j(t+1) = \operatorname{sign}\Big(\sum_{k=1}^{n} w_{jk}^{XY} x_k(t)\Big)$$

and then

$$x_i(t+1) = \operatorname{sign}\Big(\sum_{\ell=1}^{m} w_{i\ell}^{YX} y_\ell(t+1)\Big)$$

with the usual convention that there is no change if the argument to the function $\operatorname{sign}(\cdot)$ is zero.

This system can be regarded as a special case of a recurrent neural network operating under asynchronous dynamics, as we now show. First, let $z = (x_1, \dots, x_n, y_1, \dots, y_m) \in \{-1, 1\}^{n+m}$. For i and j in $\{1, 2, \dots, n+m\}$,

define w_{ji} as follows

$$
\begin{aligned}
w_{(n+r)i} &= w_{ri}^{XY}, \text{ for } 1 \le i \le n \text{ and } 1 \le r \le m, \\
w_{i(n+r)} &= w_{ir}^{YX}, \text{ for } 1 \le i \le n \text{ and } 1 \le r \le m
\end{aligned}
$$

and all other $w_{ji} = 0$. Thus

$$
(w_{ji}) = \begin{pmatrix} 0 & W^T \\ W & 0 \end{pmatrix}
$$

where W is the $m \times n$ matrix $W_{ri} = w_{ri}^{XY}$. Next we consider the mode of operation. Since there are no connections between the ys, the y updates can be considered as the result of m sequential updates. Similarly, the x updates can be thought of as the result of n sequential updates. So we can consider the whole y and x update as $m+n$ sequential updates taken in the order of ys first and then the xs. It follows that the dynamics is governed by the asynchronous dynamics of a recurrent neural network whose synaptic matrix is symmetric and has nonnegative diagonal terms (zero in this case). Hence, in particular, the system converges to a fixed point.

One usually considers the patterns stored by the BAM to consist of pairs of vectors, corresponding to the x and y decomposition of the network, rather than as vectors in $\{-1, 1\}^{n+m}$, although this is a mathematically trivial distinction.

Chapter 8

Singular Value Decomposition

We have discussed the uniqueness of the generalized inverse of a matrix, but have still to demonstrate its existence. We shall turn to a discussion of this problem here. The solution rests on the existence of the so-called singular value decomposition of any matrix, which is of interest in its own right. It is a standard result of linear algebra that any symmetric matrix can be diagonalized via an orthogonal transformation. The singular value decomposition can be thought of as a generalization of this.

Theorem 8.1 (Singular Value Decomposition). *For any given non-zero matrix $A \in \mathbb{R}^{m\times n}$, there exist orthogonal matrices $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ and positive real numbers $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_r > 0$, where $r = \operatorname{rank} A$, such that*

$$A = U\,D\,V^T$$

where $D \in \mathbb{R}^{m\times n}$ has entries $D_{ii} = \lambda_i$, $1 \le i \le r$ and all other entries are zero.

Proof. Suppose that $m \ge n$. Then $A^TA \in \mathbb{R}^{n\times n}$ and $A^TA \ge 0$. Hence there is an orthogonal $n \times n$ matrix V such that

$$A^TA = V\,\Sigma\,V^T$$

where $\Sigma \in \mathbb{R}^{n\times n}$ is given by $\Sigma = \begin{pmatrix} \mu_1 & & 0 \\ & \ddots & \\ 0 & & \mu_n \end{pmatrix}$ where $\mu_1 \ge \mu_2 \ge \cdots \ge \mu_n$ are the eigenvalues of A^TA, counted according to multiplicity. If $A \ne 0$, then $A^TA \ne 0$ and so has at least one non-zero eigenvalue. Thus, there is $0 < r \le n$ such that $\mu_1 \ge \mu_2 \ge \cdots \ge \mu_r > \mu_{r+1} = \cdots = \mu_n = 0$. Write $\Sigma = \begin{pmatrix} \Lambda & 0 \\ 0 & 0 \end{pmatrix}$, where $\Lambda = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix}$, with $\lambda_1^2 = \mu_1, \ldots, \lambda_r^2 = \mu_r$. Partition V as $V = \begin{pmatrix} V_1 & V_2 \end{pmatrix}$ where $V_1 \in \mathbb{R}^{n\times r}$ and $V_2 \in \mathbb{R}^{n\times(n-r)}$. Since V is

orthogonal, its columns form pairwise orthogonal vectors, and so $V_1^T V_2 = 0$. We have

$$\begin{aligned} A^T A &= V \, \Sigma \, V^T \\ &= \begin{pmatrix} V_1 & V_2 \end{pmatrix} \begin{pmatrix} \Lambda^2 & 0 \\ 0 & 0 \end{pmatrix} V^T \\ &= \begin{pmatrix} V_1 \Lambda^2 & 0 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix} \\ &= V_1 \Lambda^2 V_1^T . \end{aligned}$$

Hence $V_2^T A^T A V_2 = \underbrace{V_2^T V_1}_{(V_1^T V_2)^T = 0} \Lambda^2 \underbrace{V_1^T V_2}_{=0}$, so that $V_2^T A^T A V_2 = 0$ and so it follows that $AV_2 = 0$.

Now, the equality $A^T A = V_1 \Lambda^2 V_1^T$ suggests at first sight that we might hope that $A = \Lambda V_1^T$. However, this cannot be correct, in general, since $A \in \mathbb{R}^{m \times n}$, whereas $\Lambda V_1^T \in \mathbb{R}^{r \times n}$, and so the dimensions are incorrect. However, if $U \in \mathbb{R}^{k \times r}$ satisfies $U^T U = \mathbb{1}_r$, then $V_1 \Lambda^2 V_1^T = V_1 \Lambda U^T U \Lambda V_1^T$ and we might hope that $A = U \Lambda V_1^T$. We use this idea to *define* a suitable matrix U. Accordingly, we define

$$U_1 = A V_1 \Lambda^{-1} \in \mathbb{R}^{m \times r},$$

so that $A = U_1 \Lambda V_1^T$, as discussed above. We compute

$$U_1^T U_1 = \Lambda^{-1} \underbrace{V_1^T A^T A V_1}_{\Lambda^2} \Lambda^{-1} = \mathbb{1}_r .$$

This means that the r columns of U_1 form an orthonormal set of vectors in $\mathbb{R}^m$. Let $U_2 \in \mathbb{R}^{m \times (m-r)}$ be such that $U = (U_1 \ U_2)$ is orthogonal (in $\mathbb{R}^{m \times m}$)—thus the columns of U_2 are made up of $(m-r)$ orthonormal vectors such that these, together with those of U_1, form an orthonormal set of m vectors. Thus, $U_2^T U_1 = 0 \in \mathbb{R}^{(m-r) \times r}$ and $U_1^T U_2 = 0 \in \mathbb{R}^{r \times (m-r)}$. Hence we have

$$\begin{aligned} U^T A V &= \begin{pmatrix} U_1^T \\ U_2^T \end{pmatrix} A \, (V_1 \ V_2) \\ &= \begin{pmatrix} U_1^T A \\ U_2^T A \end{pmatrix} (V_1 \ V_2) \\ &= \begin{pmatrix} U_1^T A V_1 & U_1^T A V_2 \\ U_2^T A V_1 & U_2^T A V_2 \end{pmatrix} \\ &= \begin{pmatrix} U_1^T A V_1 & 0 \\ U_2^T A V_1 & 0 \end{pmatrix}, \quad \text{since } AV_2 = 0, \\ &= \begin{pmatrix} \Lambda & 0 \\ U_2^T U_1 \Lambda & 0 \end{pmatrix}, \end{aligned}$$

using $U_1 = AV_1\Lambda^{-1}$ and $U_1^T U_1 = \mathbb{1}_r$ so that $\Lambda = U_1^T A V_1$,

$$= \begin{pmatrix} \Lambda & 0 \\ 0 & 0 \end{pmatrix}, \quad \text{using } U_2^T U_1 = 0.$$

Hence,

$$A = U \begin{pmatrix} \Lambda & 0 \\ 0 & 0 \end{pmatrix} V^T,$$

as claimed. Note that the condition $m \geq n$ means that $m \geq n \geq r$, and so the dimensions of the various matrices are all valid.

If $m < n$, consider $B = A^T$ instead. Then, by the above argument, we get that

$$A^T = B = U' \begin{pmatrix} \Lambda' & 0 \\ 0 & 0 \end{pmatrix} V'^T,$$

for orthogonal matrices $U' \in \mathbb{R}^{n \times n}$, $V' \in \mathbb{R}^{m \times m}$ and where Λ'^2 holds the positive eigenvalues of AA^T. Taking the transpose, we have

$$A = V' \begin{pmatrix} \Lambda' & 0 \\ 0 & 0 \end{pmatrix} U'^T.$$

Finally, we observe that from the given form of the matrix A, it is clear that the dimension of $\operatorname{ran} A$ is exactly r, that is, $\operatorname{rank} A = r$. ■

Remark 8.2. From this result, we see that the matrices $A^T A = U \begin{pmatrix} \Lambda^2 & 0 \\ 0 & 0 \end{pmatrix} V^T$ and $AA^T = V \begin{pmatrix} \Lambda^2 & 0 \\ 0 & 0 \end{pmatrix} U^T$ have the same non-zero eigenvalues, counted according to multiplicity. We can also see that

$$\operatorname{rank} A = \operatorname{rank} A^T = \operatorname{rank} AA^T = \operatorname{rank} A^T A\,.$$

Theorem 8.3. *Let $A \in \mathbb{R}^{m \times n}$ and let $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$, $\Lambda \in \mathbb{R}^{r \times r}$ be as given above via the singular value decomposition of A, so that $A = UDV^T$ where $D = \begin{pmatrix} \Lambda & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{m \times n}$. Then the generalized inverse of A is given by*

$$A^\# = V \underbrace{\begin{pmatrix} \Lambda^{-1} & 0 \\ 0 & 0 \end{pmatrix}}_{n \times m} U^T.$$

Proof. We just have to check that $A^\#$, as given above, really does satisfy the defining conditions of the generalized inverse. We will verify two of the four conditions by way of illustration.

Put $X = VHU^T$, where $H = \begin{pmatrix} \Lambda^{-1} & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{n \times m}$. Then

$$\begin{aligned} AXA &= UDV^T VHU^T UDV^T \\ &= U \underbrace{\begin{pmatrix} \mathbb{1}_r & 0 \\ 0 & 0 \end{pmatrix}}_{m \times m} \underbrace{\begin{pmatrix} \Lambda & 0 \\ 0 & 0 \end{pmatrix}}_{m \times n} V^T = A. \end{aligned}$$

Similarly, one finds that $XAX = X$. Next, we consider

$$\begin{aligned} XA &= VHU^T UDV^T \\ &= V \underbrace{\begin{pmatrix} \Lambda^{-1} & 0 \\ 0 & 0 \end{pmatrix}}_{n \times m} \underbrace{U^T U}_{\mathbb{1}_m} \underbrace{\begin{pmatrix} \Lambda & 0 \\ 0 & 0 \end{pmatrix}}_{m \times n} V^T \\ &= V \underbrace{\begin{pmatrix} \mathbb{1}_r & 0 \\ 0 & 0 \end{pmatrix}}_{n \times n} V^T \end{aligned}$$

which is clearly symmetric. Similarly, one verifies that $AX = (AX)^T$, and the proof is complete. ∎

We have seen how the generalized inverse appears in the construction of the OLAM matrix memory, via minimization of the output error. Now we can show that this choice is privileged in a certain precise sense.

Theorem 8.4. *For given $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{\ell \times n}$, let $\psi : \mathbb{R}^{\ell \times m} \to \mathbb{R}$ be the map $\psi(X) = \|XA - B\|_F$. Among those matrices X minimizing ψ, the matrix $X = BA^{\#}$ has the least $\|\cdot\|_F$-norm.*

Proof. We have already seen that the choice $X = BA^{\#}$ does indeed minimize $\psi(X)$. Let $A = UDV^T$ be the singular value decomposition of A with the notation as above—so that, for example, Λ denotes the top left $r \times r$ block of D. We have

$$\begin{aligned} \psi(X)^2 &= \|XA - B\|_F^2 \\ &= \|XUDV^T - B\|_F^2 \\ &= \|XUD - BV\|_F^2, \text{ since } V \text{ is orthogonal,} \\ &= \|YD - C\|_F^2, \text{ where } Y = XU \in \mathbb{R}^{\ell \times m},\ C = BV \in \mathbb{R}^{\ell \times n}, \\ &= \left\|(Y_1\ Y_2) \begin{pmatrix} \Lambda & 0 \\ 0 & 0 \end{pmatrix} - (C_1\ C_2)\right\|_F^2, \end{aligned}$$

where we have partitioned the matrices Y and C into $(Y_1\ Y_2)$ and $(C_1\ C_2)$ with $Y_1, C_1 \in \mathbb{R}^{\ell\times r}$,

$$
\begin{aligned}
&= \|(Y_1\Lambda\ 0) - (C_1\ C_2)\|_F^2 \\
&= \|(Y_1\Lambda - C_1 \vdots - C_2)\|_F^2 \\
&= \|Y_1\Lambda - C_1\|_F^2 + \|C_2\|_F^2.
\end{aligned}
$$

This is evidently minimized by any choice of Y (equivalently X) for which $Y_1\Lambda = C_1$. Let $\widehat{Y} = (C_1\Lambda^{-1} \vdots 0)$ and let $Y = (C_1\Lambda^{-1} \vdots Y_2)$, where $Y_2 \in \mathbb{R}^{\ell\times m-r}$ is subject to $Y_2 \neq 0$ but otherwise is arbitrary. Both $\widehat{Y}$ and Y correspond to a minimum of $\psi(X)$, where the X and Y matrices are related by $Y = XU$, as above. Now,

$$
\|\widehat{Y}\|_F^2 = \|C_1\Lambda^{-1}\|_F^2 < \|C_1\Lambda^{-1}\|_F^2 + \|Y_2\|_F^2 = \|Y'\|_F^2.
$$

It follows that among those Y matrices minimizing $\psi(X)$, $\widehat{Y}$ is the one with the least $\|\cdot\|_F$-norm. But if $Y = XU$, then $\|Y\|_F = \|X\|_F$ and so $\widehat{X}$ given by $\widehat{X} = \widehat{Y}U^T$ is the X matrix which has the least $\|\cdot\|_F$-norm amongst those which minimize $\psi(X)$. We find

$$
\begin{aligned}
\widehat{X} &= \widehat{Y}U^T \\
&= (C_1\Lambda^{-1} \vdots 0)U^T \\
&= (C_1\ C_2)\begin{pmatrix}\Lambda^{-1} & 0 \\ 0 & 0\end{pmatrix}U^T \\
&= BV\begin{pmatrix}\Lambda^{-1} & 0 \\ 0 & 0\end{pmatrix}U^T \\
&= BA^{\#}
\end{aligned}
$$

which completes the proof. ■

Bibliography

We list, here, a selection of books available. Many are journalistic-style overviews of various aspects of the subject and convey little quantitative feel for what is supposed to be going on. Many declare a deliberate attempt to avoid anything mathematical.

For further references a good place to start might be the bibliography of the book of R. Rojas (or that of S. Haykin).

Aarts, E. and J. Korst, *Simulated Annealing and Boltzmann Machines*, J. Wiley 1989. *Mathematics of Markov chains and Boltzmann machines.*

Amit, D. J., *Modeling Brain Function, The World of Attractor Neural Networks*, Cambridge University Press 1989. *Statistical physics.*

Aleksander, I. and H. Morton, *An Introduction to Neural Computing, 2nd ed.*, Thompson International, London 1995.

Anderson, J. A., *An Introduction to Neural Networks*, M.I.T. Press 1995. *Minimal mathematics with emphasis on the uses of neural network algorithms. Indeed, the author questions the appropriateness of using mathematics in this field.*

Arbib, M. A., *Brains, Machines and Mathematics, 2nd ed.*, Springer-Verlag 1987. *Nice overview of the perceptron.*

Aubin, J.-P., *Neural Networks and Qualitative Physics*, Cambridge University Press 1996. *Quite abstract.*

Bertsekas, D. P. and J. N. Tsitsiklis, *Parallel and Distributed Computation–Numerical Methods*, Prentice-Hall International Inc. 1989.

Bishop, C. M., *Neural networks for Pattern Recognition*, Oxford University Press 1995. *Primarily concerned with Bayesian statistics.*

Blum, A., *Neural Networks in C^{++}—An Object Oriented Framework for Building Connectionist Systems*, J. Wiley 1992. *Computing hints.*

Bose, N. K. and P. Liang, *Neural Network Fundamentals with Graphs, Algorithms and Applications*, McGraw-Hill 1996. *Emphasises the use of graph theory.*

Dayhoff, J., *Neural Computing Architectures*, Van Nostrand Reinhold 1990. *A nice discussion of neurophysiology.*

De Wilde, P., *Neural Network Models, 2nd ed.*, Springer 1997. *Gentle introduction.*

Devroye, L., Györfi, G. and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*, Springer 1996. *A mathematics book, full of inequalities to do with nonparametric estimation.*

Dotsenko, V., *An Introduction to the Theory of Spin Glasses and Neural Networks*, World Scientific 1994. *Statistical physics.*

Duda, R. O. and P. E. Hart, *Pattern Classification and Scene Analysis*, J. Wiley 1973. *Well worth looking at. It puts later books into perspective.*

Fausett, L., *Fundamentals of Neural Networks*, Prentice Hall 1994. *Nice detailed descriptions of the standard algorithms with many examples.*

Freeman, J. A., *Simulating Neural Networks with Mathematica*, Addison-Wesley 1994. *Set Mathematica to work on some algorithms.*

Freeman, J. A. and D. M. Skapura, *Neural Networks: Algorithms, Applications, and Programming Techniques*, Addison-Wesley 1991. *Well worth a look.*

Golub, G. H. and C. F. van Loan, *Matrix Computations*, The John Hopkins University Press 1989. *Superb book on matrices; you will discover entries in your matrix you never knew you had.*

Hassoun, M. H., *Fundamentals of Artificial Neural Networks*, The MIT Press 1995. *Quite useful to dip into now and again.*

Haykin, S., *Neural Networks, A Comprehensive Foundation*, Macmillan 1994. *Looks very promising, but you soon realise that you need to spend a lot of time tracking down the original papers.*

Hecht-Nielsen, R., *Neurocomputing*, Addison-Wesley 1990.

Hertz, J., A. Krogh and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley 1991. *Quite good account but arguments are sometimes replaced by references.*

Kamp, Y. and M. Hasler, *Recursive Neural Networks for Associative Memory*, J. Wiley 1990. *Well-presented readable account of recurrent networks.*

Khanna, T., *Foundations of Neural Networks*, Addison-Wesley 1990.

Kohonen, T., *Self-Organization and Associative Memory*, Springer 1988. *Discusses the OLAM.*

Kohonen, T., *Self-Organizing Maps*, Springer 1995. *Describes a number of algorithms and various applications. There are hundreds of references.*

Kosko, B., *Neural Networks and Fuzzy Systems*, Prentice-Hall 1992. *Disc included.*

Kung, S. Y., *Digital Neural Networks*, Prentice-Hall 1993. *Very concise account for electrical engineers.*

Looney, C. G., *Pattern Recognition Using Neural Networks*, Oxford University Press 1997. *Quite nice account—concerned with practical application of algorithms.*

Masters, T., *Practical Neural Network Recipes in* C^{++}, Academic Press 1993. *Hints and ideas about implementing algorithms; with disc.*

McClelland J. L., D. E. Rumelhart and the PDP Research Group, *Parallel Distributed Processing, Vols 1 and 2*, MIT Press 1986. *Has apparently sold many copies.*

Minsky, M. L. and S. A. Papert, *Perceptrons*, Expanded ed. MIT Press 1988. *Everyone should at least browse through this book.*

Müller, B. and J. Reinhardt, *Neural Networks: An Introduction*, Springer 1990. *Quite a nice account; primarily statistical physics.*

Pao, Y-H., *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley 1989.

Peretto, P., *An Introduction to the Modeling of Neural Networks*, Cambridge University Press 1992. *Heavy-going statistical physics.*

Ritter, H., T. Martinez and K. Schulten, *Neural Computation and Self-Organizing Maps*, Addison-Wesley 1992.

Rojas, R., *Neural Networks – A Systematic Introduction*, Springer 1996. *A very nice introduction—but do not expect too many details.*

Simpson, P. K., *Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations*, Pergamon Press 1990. *A compilation of algorithms.*

Vapnik, V., *The Nature of Statistical Learning Theory*, Springer 1995. *An overview (no proofs) of the statistical theory of learning and generalization.*

Wasserman, P. D., *Neural Computing, Theory and Practice*, Van Nostrand Reinhold 1989.

Welstead, S., *Neural Network and Fuzzy Logic Applications in C/C++*, J. Wiley 1994. *Programmes built around Borland's Turbo Vision; disc included.*