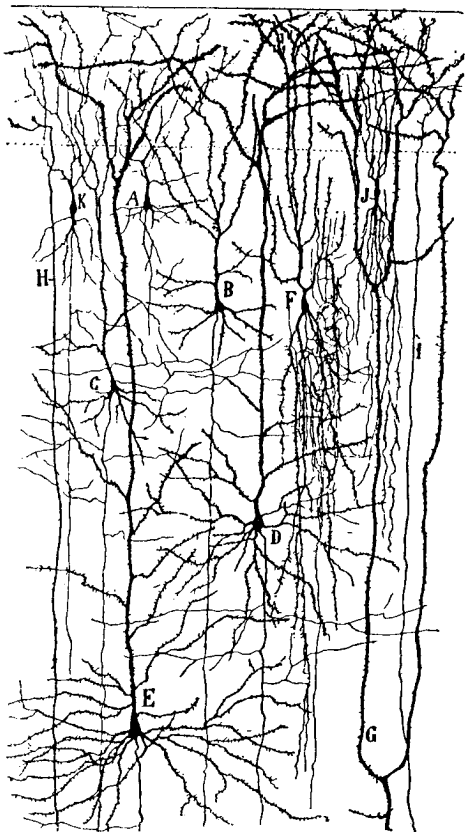
Advanced Neural Networks

Lecture Notes of Course G30/CMNN15

September 2002



ACC Coolen
Department of Mathematics
King's College London

Contents

Chapter 1

Introduction

This lecture course is the sequel to the core course 'Neural Networks', which can be regarded as a prerequisite, in which we expand both in breadth and in depth the material covered so far. Apart from the first of the four chapters in the present notes (and in contrast to the bulk of 'Neural Networks'), this course 'Advanced Neural Networks' deals with relatively novel research results which go back less than ten years.

Expansion in breadth refers to the addition of new neural network classes and new learning rules (Competitive Non-Supervised Learning Processes, this material is not too mathematical because there is little theory on these systems; Support Vector Machines, again a subject with only a modest amount of theory).

Expansion in depth refers to a more solid statistical understanding, interpretation, and increased potential for analysis and prediction of the most popular system types (Bayesian Techniques in Supervised Learning, and its spin-off Gaussian Processes). These latter subjects, although more mathematical in nature, have generated the main progress in industrial and commercial neural network applications over the last ten years, since they have removed in a rigorous way the problems related to data noise and to the quantification of the reliability of neural network decisions.

Chapter 2

Competitive Non-Supervised Learning Processes

In this first chapter we will introduce and study a couple of so-called 'unsupervised' learning processes, where the problem is not to learn to associate some 'correct' response to each possible incoming signal (specified by a 'teacher' or 'supervisor' signal, as in e.g. feed-forward networks of the Perceptron or Multi-Layer Perceptron type), but rather to build some alternative (more efficient, more compact or more structured) representation of the data vectors fed into the system. The first two procedures to do this (Vector Quantization and Soft Vector Quantization) aim to achieve data reduction for initially unknown data distributions (with typical applications in communication, where compact representations are obviously cheaper to transmit than non-compact ones, since it will allow more signals to be communicated via the same channel). The third class of systems (Self-Organizing Maps, or Feature Maps, or Kohonen Maps) aim to create a topologically correct but low-dimensional internal representation of a given data distribution. This has applications in e.g. biology (the brain is known to create such maps for sensory signals), data-base mining, medical diagnostics, etc.

2.1 Vector Quantization

Data-Reduction via Code-Book Vectors. Imagine we have a source of real-valued data vectors $\boldsymbol{x} = (x_1, \ldots, x_n) \in \Re^n$, with statistics described by some probability density $p(\boldsymbol{x})$ (with $\int d\boldsymbol{x}\ p(\boldsymbol{x}) = 1$, and with the definition $\langle f(\boldsymbol{x}) \rangle = \int d\boldsymbol{x}\ p(\boldsymbol{x}) f(\boldsymbol{x})$). Alternatively, if the data can only take values from a discrete set, we would have $p(\boldsymbol{x})$ representing probabilities, with $\sum_{\boldsymbol{x}} p(\boldsymbol{x}) = 1$ and $\langle f(\boldsymbol{x}) \rangle = \sum_{\boldsymbol{x}} p(\boldsymbol{x}) f(\boldsymbol{x})$ (see appendix on elementary probability theory). Especially if n is very large, we would like to represent the real data $\boldsymbol{x}$ by alternative (and simpler) signals from which the $\boldsymbol{x}$ can be re-constructed, with some loss of accuracy, but with a reduction in dimensionality. One way to achieve this is the following:

- We 'cover' the data space by a suitable (small) set of characteristic data points, the so-called 'code-book vectors' $\boldsymbol{m}_i \in \Re^n$. Here i labels the individual code-book vectors, i.e. if we have N code-book vectors then $i = 1, \ldots, N$.
- We then approximate (or 'round off') each data point $\boldsymbol{x}$ by the *nearest* code-book vector, i.e. by that particular $\boldsymbol{m}_i$ for which $|\boldsymbol{x} - \boldsymbol{m}_i|$ is minimal.

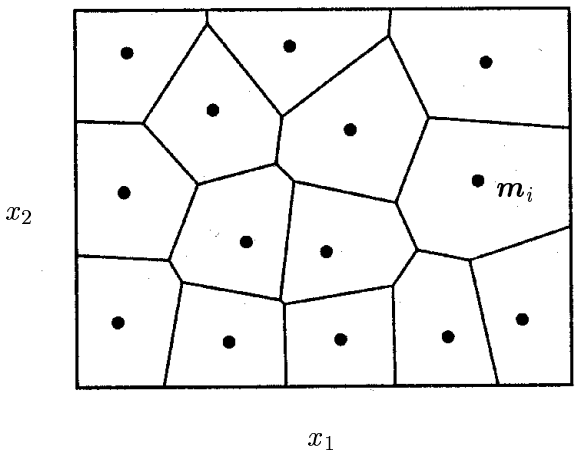


Figure 2.1: Example of a Voronoi tessalation of data space for $n = 2$, i.e. $\boldsymbol{x} = (x_1, x_2)$. The points • represent the code-book vectors. The 'Voronoi cell' associated with code-book vector $\boldsymbol{m}_i$, $V_i[\{\boldsymbol{m}\}]$, is the compartment surrounding point $\boldsymbol{m}_i$.

It will be clear why in communication this might be desirable: instead of the n real numbers $(x_1, \ldots, x_n)$ we would now need to send only a single integer number: the index i. The price we pay for this is a reduction in accuracy; after all we have approximated $\boldsymbol{x}$ by $\boldsymbol{m}_i$, and have thereby lost information. We can no longer be sure of the exact value of $\boldsymbol{x}$, since many more data points $\boldsymbol{x}$ would have been replaced by $\boldsymbol{m}_i$ (all those which are close to $\boldsymbol{m}_i$). The amount of information lost will obviously be less if we have a larger number or a more efficiently distributed set of code-book vectors.

The above procedure gives us what is know as a Voronoi tessalation of data space. This is a partitioning into convex subsets $V_i[\{\boldsymbol{m}\}]$, controlled by the choice $\{\boldsymbol{m}\}$ made for the N code-book vectors, defined as

$$V_i[\{\boldsymbol{m}\}] = \{\boldsymbol{x} \in \Re^n |\ \forall j \neq i :\ |\boldsymbol{x} - \boldsymbol{m}_i| < |\boldsymbol{x} - \boldsymbol{m}_j|\} \tag{2.1}$$

(see figure 2.1).

A good set of code-book vectors is one with the property that the density of code-book vectors in a given region of $\Re^n$ is proportional to the density of data-points in that region. In other words, given the statistics of the data and given the number N of code-book vectors we are willing to invest, we aim for a set $\{\boldsymbol{m}\}$ such that

$$\text{Prob}\,[\boldsymbol{x} \in V_i[\{\boldsymbol{m}\}]] = \text{Prob}\,[\boldsymbol{x} \in V_j[\{\boldsymbol{m}\}]] \tag{2.2}$$

In this way all code-book vectors are used equally often, and no resources are wasted. The problem addressed by the VQ and SVQ algorithms is how to find a proper set of code-book vectors, via adaptive processes; i.e. both are based on gradually 'learning' a proper positioning of the code-book vectors by observing the data.

The VQ Algorithm and its Properties. The Vector Quantization (VQ) algorithm is defined as follows. First we initialize the N code-book vectors $\boldsymbol{m}_i \in \Re^n$ (randomly, or according to a recipe to be given below). Then we iterate the following stochastic process:

step 1: pick a data point $\boldsymbol{x}$ at random, according to the probability density $p(\boldsymbol{x})$

step 2: find the Voronoi cell containing $\boldsymbol{x}$, i.e. find i such that $|\boldsymbol{x} - \boldsymbol{m}_i| < |\boldsymbol{x} - \boldsymbol{m}_j| \ \forall j \neq i$

step 3: move $\boldsymbol{m}_i$ towards $\boldsymbol{x}$: $\boldsymbol{m}_i \to \boldsymbol{m}_i + \eta(\boldsymbol{x} - \boldsymbol{m}_i)$

step 4: return to 1

The parameter $\eta > 0$ (the learning rate) controls the magnitude of the changes, one takes $\eta \ll 1$ to suppress fluctuations. The uncertainty (stochasticity) of the process is only in the realization of the subsequent data points we draw. Note that there would be a problem with data points $\boldsymbol{x}$ which are exactly on the border of two Voronoi cells. In practice this is not an issue: firstly, the probability for this to happen is generally very small (unless we have a pathological distribution $p(\boldsymbol{x})$), and secondly, one could simply decide in those instances not to make a change.

The VQ algorithm can be seen to have the following properties:

(i) A code-book vector will only become mobile when we pick a data point in its Voronoi cell, i.e. sufficiently close to it. Hence the process of moving the $\boldsymbol{m}_i$ will proceed slowly in areas where the data density $p(\boldsymbol{x})$ is low.

(ii) Unless we reduce η during the process, the code-book vectors will continue to move stochastically, although the *density* of code-book vectors in any given region should become stationary.

(iii) VQ is very simple and (as we will see below) effective, and has just three parameters to be chosen: N, η, and the duration of the process.

As a consequence of (i) we also conclude that it is not necessarily optimal to initialize the N code-book vectors randomly: those which are initialized in regions where $p(\boldsymbol{x})$ is zero will *never* be used. Alternatively one could initialize the $\boldsymbol{m}_i$ by putting them at the locations of the first N observed data points (by construction they can now never be initialized in regions where there are no data).

Examples of VQ in Action. The figures below illustrate the functioning of the VQ algorithm for a number of simple examples with $n = 2$ (i.e. where one has data points $\boldsymbol{x} = (x_1, x_2)$ in a plane) and $N = 100$ (i.e. a population of 100 code-book vectors), but for different choices of the data distribution $p(\boldsymbol{x})$ and with different initialization strategies. Firstly, figures 2.2 and 2.3 show examples of simple data distributions and random initialization, where the process is still seen to work fine, simply because in this case no code-book vector happens to have been initialized in data-free regions. Figures 2.4, 2.5 and 2.6 refer to strongly inhomogeneous data distributions, but now with non-random initialization (so that the inhomogeneities cannot disrupt the functioning of VQ). The process is again seen to work fine. In contrast, in figures 2.7, 2.8 and 2.9 the same three inhomogeneous distributions were considered, but now with random (i.e. inappropriate) initialization of the code-book vectors. The resulting locations of the code-book vectors illustrates quite clearly how for random initialization and inhomogeneous data distributions the VQ process fails to use its resources effectively.

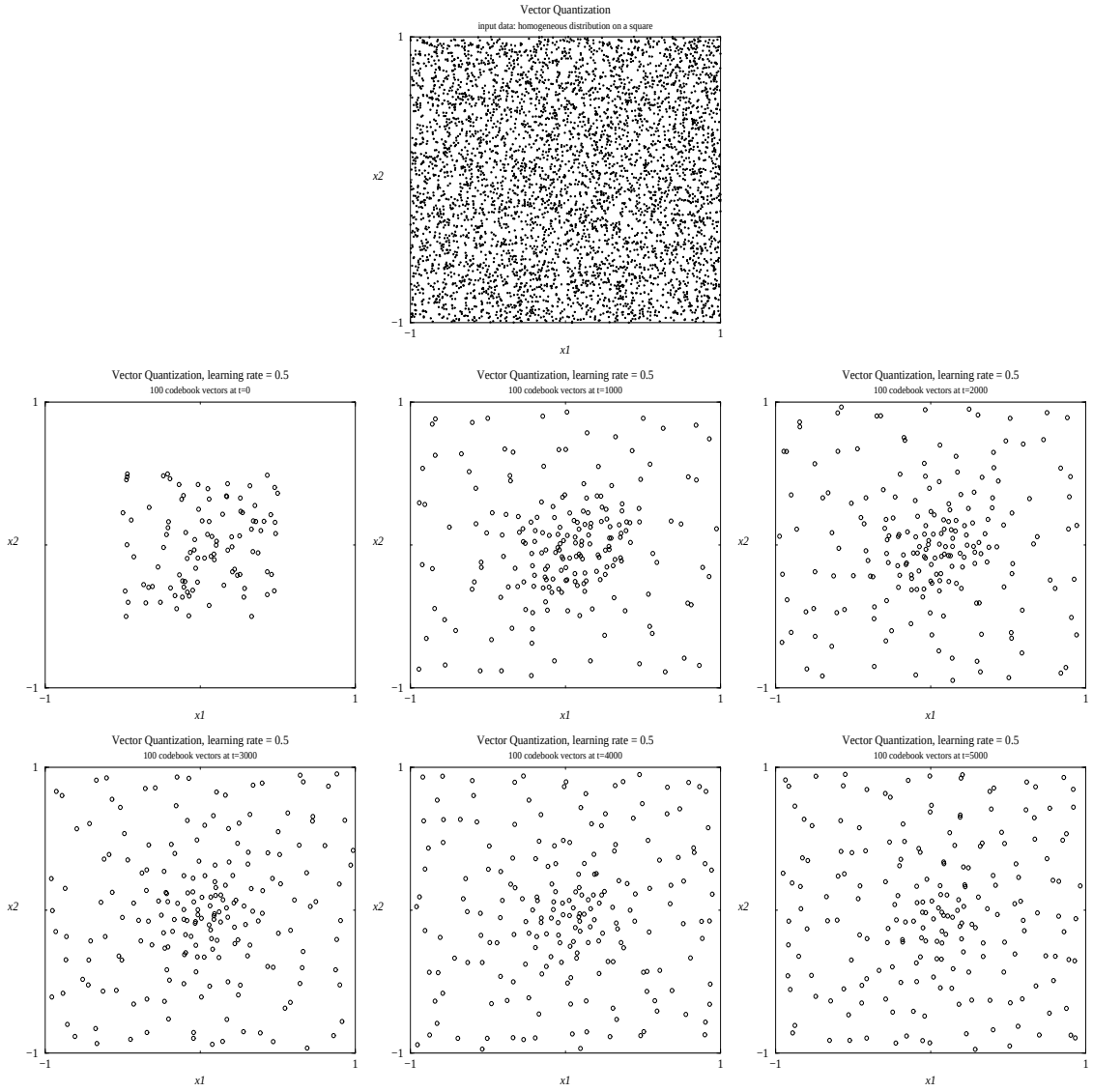


Figure 2.2: Example: numerical simulation of Vector Quantization, with $n = 2$, $\eta = 0.5$ and $N = 100$. Top graph: data distribution $p(\boldsymbol{x})$, homogeneously distributed over the square $[-1, 1] \times [-1, 1]$. Bottom six graphs: locations of the code-book vectors $\boldsymbol{m}_i$ during the course of the process ($i = 1, \ldots, 100$), at times $t = 0$ (middle row, left), $t = 1000$ (middle row, center), $t = 2000$ (middle row, right), $t = 3000$ (bottom row, left), $t = 4000$ (bottom row, middle) and $t = 5000$ (bottom row, right). Times are measures in the number of iterations. Initialisation of code-book vector allocations: randomly in the square $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$.

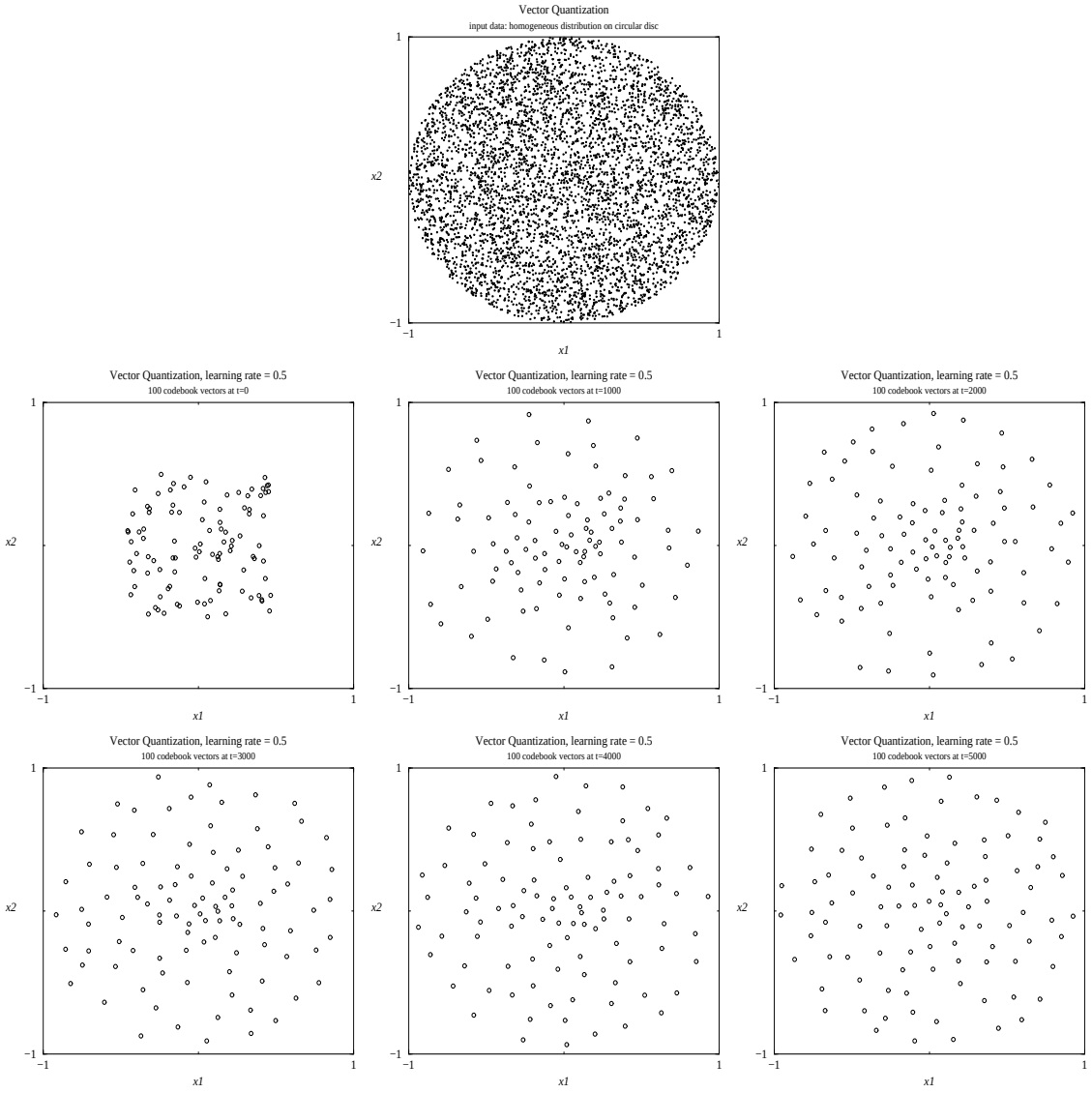


Figure 2.3: Example: numerical simulation of Vector Quantization, with $n = 2$, $\eta = 0.5$ and $N = 100$. Top graph: data distribution $p(\boldsymbol{x})$, homogeneously distributed over the disk $|\boldsymbol{x}| < 1$. Bottom six graphs: locations of the code-book vectors $\boldsymbol{m}_i$ during the course of the process ($i = 1, \ldots, 100$), at times $t = 0$ (middle row, left), $t = 1000$ (middle row, center), $t = 2000$ (middle row, right), $t = 3000$ (bottom row, left), $t = 4000$ (bottom row, middle) and $t = 5000$ (bottom row, right). Times are measures in the number of iterations. Initialisation of code-book vector allocations: randomly in the square $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$.

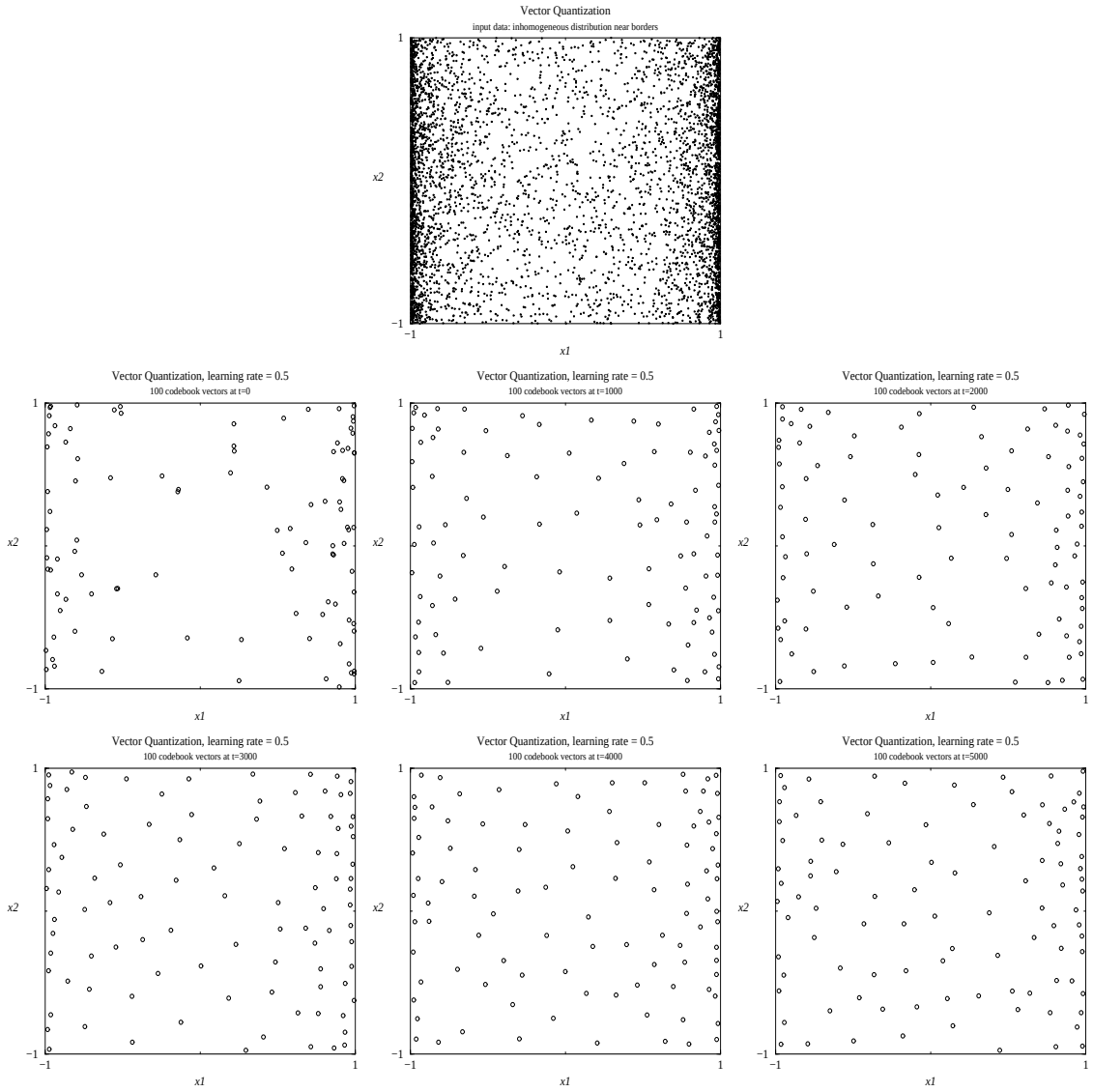


Figure 2.4: Example: numerical simulation of Vector Quantization, with $n = 2$, $\eta = 0.5$ and $N = 100$. Top graph: data distribution $p(\boldsymbol{x})$, inhomogeneously distributed over the square $[-1, 1] \times [-1, 1]$ (with highest data density near the borders $|x_1| = 1$. Bottom six graphs: locations of the code-book vectors $\boldsymbol{m}_i$ during the course of the process ($i = 1, \ldots, 100$), at times $t = 0$ (middle row, left), $t = 1000$ (middle row, center), $t = 2000$ (middle row, right), $t = 3000$ (bottom row, left), $t = 4000$ (bottom row, middle) and $t = 5000$ (bottom row, right). Times are measures in the number of iterations. Initialisation of code-book vector allocations: positioning at the locations of the first 100 data points picked by the process.

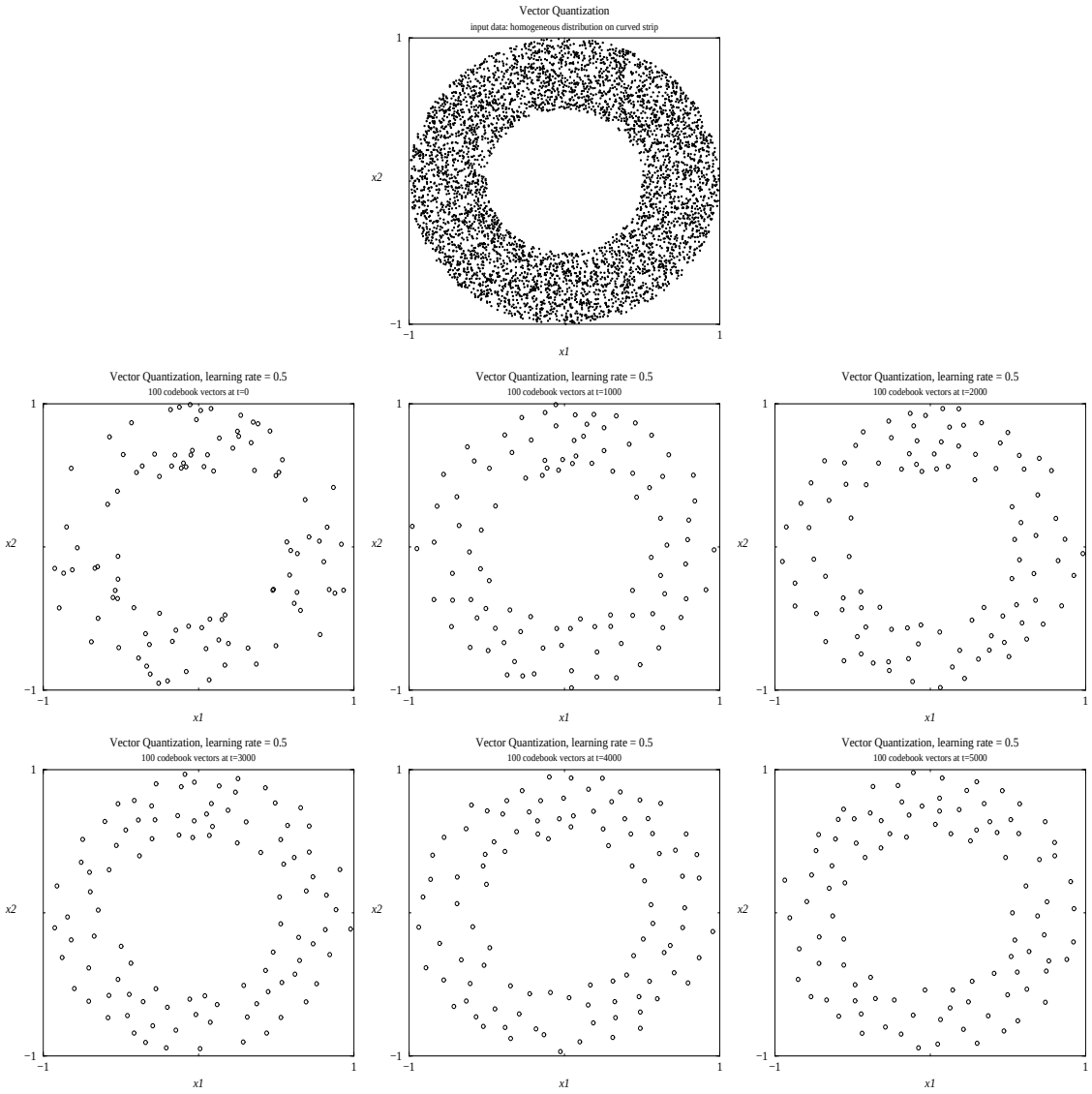


Figure 2.5: Example: numerical simulation of Vector Quantization, with $n = 2$, $\eta = 0.5$ and $N = 100$. Top graph: data distribution $p(\boldsymbol{x})$, homogeneously distributed over the region $\frac{1}{2} < |\boldsymbol{x}| < 1$. Bottom six graphs: locations of the code-book vectors $\boldsymbol{m}_i$ during the course of the process ($i = 1, \ldots, 100$), at times $t = 0$ (middle row, left), $t = 1000$ (middle row, center), $t = 2000$ (middle row, right), $t = 3000$ (bottom row, left), $t = 4000$ (bottom row, middle) and $t = 5000$ (bottom row, right). Times are measures in the number of iterations. Initialisation of code-book vector allocations: positioning at the locations of the first 100 data points picked by the process.

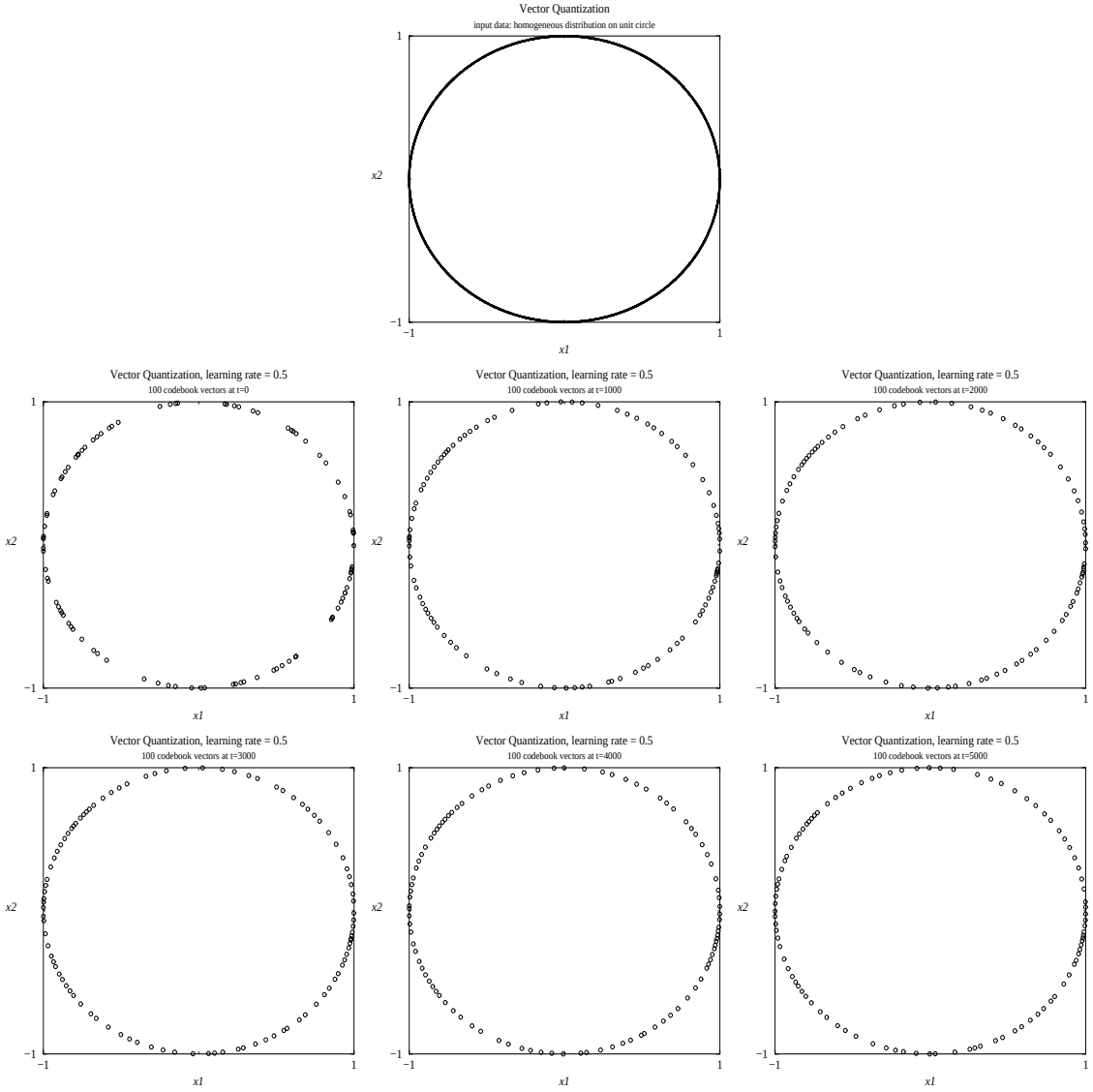


Figure 2.6: Example: numerical simulation of Vector Quantization, with $n = 2$, $\eta = 0.5$ and $N = 100$. Top graph: data distribution $p(\boldsymbol{x})$, homogeneously distributed over the circle $|\boldsymbol{x}| = 1$. Bottom six graphs: locations of the code-book vectors $\boldsymbol{m}_i$ during the course of the process ($i = 1, \ldots, 100$), at times $t = 0$ (middle row, left), $t = 1000$ (middle row, center), $t = 2000$ (middle row, right), $t = 3000$ (bottom row, left), $t = 4000$ (bottom row, middle) and $t = 5000$ (bottom row, right). Times are measures in the number of iterations. Initialisation of code-book vector allocations: positioning at the locations of the first 100 data points picked by the process.

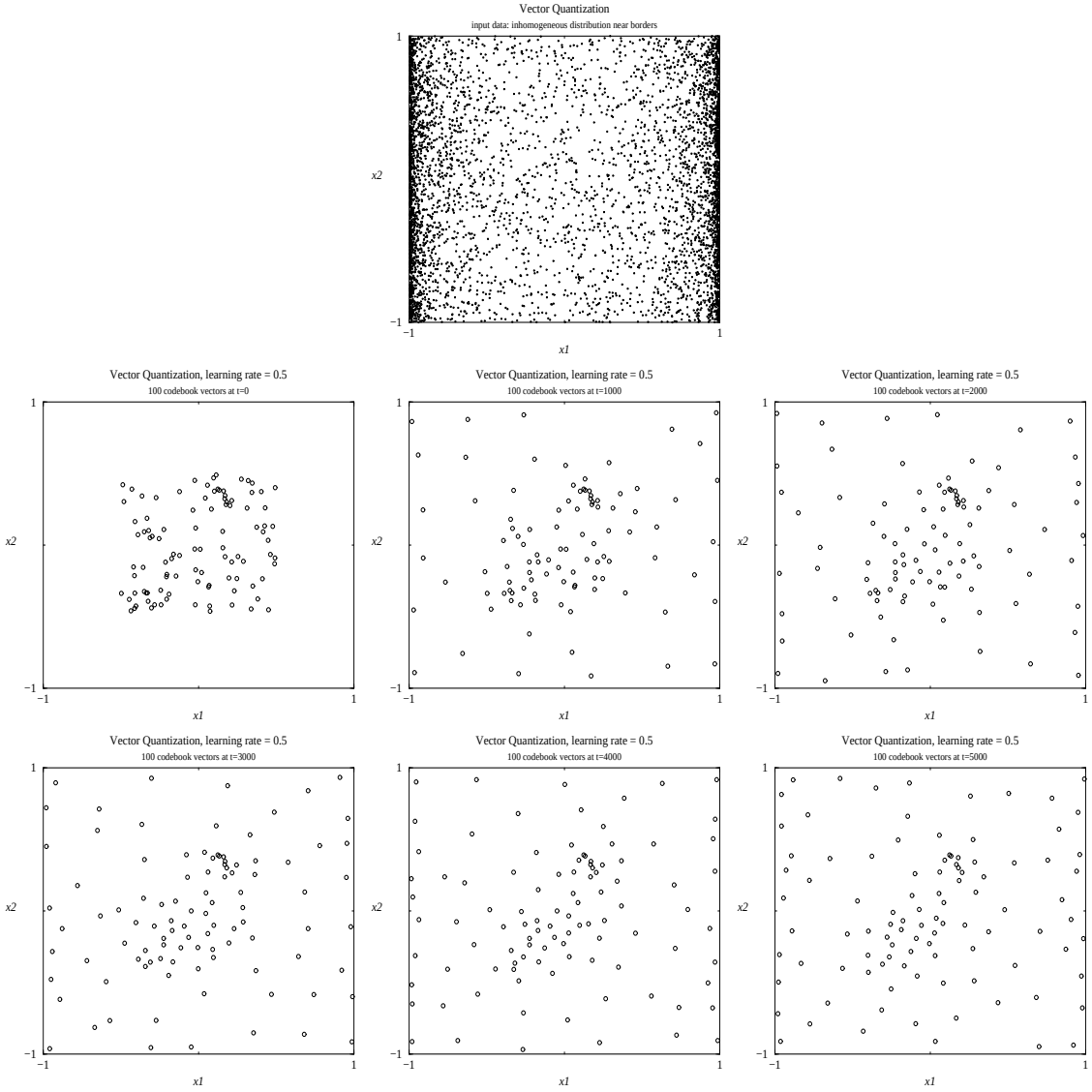


Figure 2.7: Example: numerical simulation of Vector Quantization, with $n = 2$, $\eta = 0.5$ and $N = 100$. Top graph: data distribution $p(\boldsymbol{x})$, inhomogeneously distributed over the square $[-1, 1] \times [-1, 1]$ (with highest data density near the borders $|x_1| = 1$. Bottom six graphs: locations of the code-book vectors $\boldsymbol{m}_i$ during the course of the process ($i = 1, \ldots, 100$), at times $t = 0$ (middle row, left), $t = 1000$ (middle row, center), $t = 2000$ (middle row, right), $t = 3000$ (bottom row, left), $t = 4000$ (bottom row, middle) and $t = 5000$ (bottom row, right). Times are measures in the number of iterations. Initialisation of code-book vector allocations: randomly in the square $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$. Comparison with figure 2.4 shows that code-book vectors initialized in data regions where $p(\boldsymbol{x})$ is small tend to get 'stuck'.

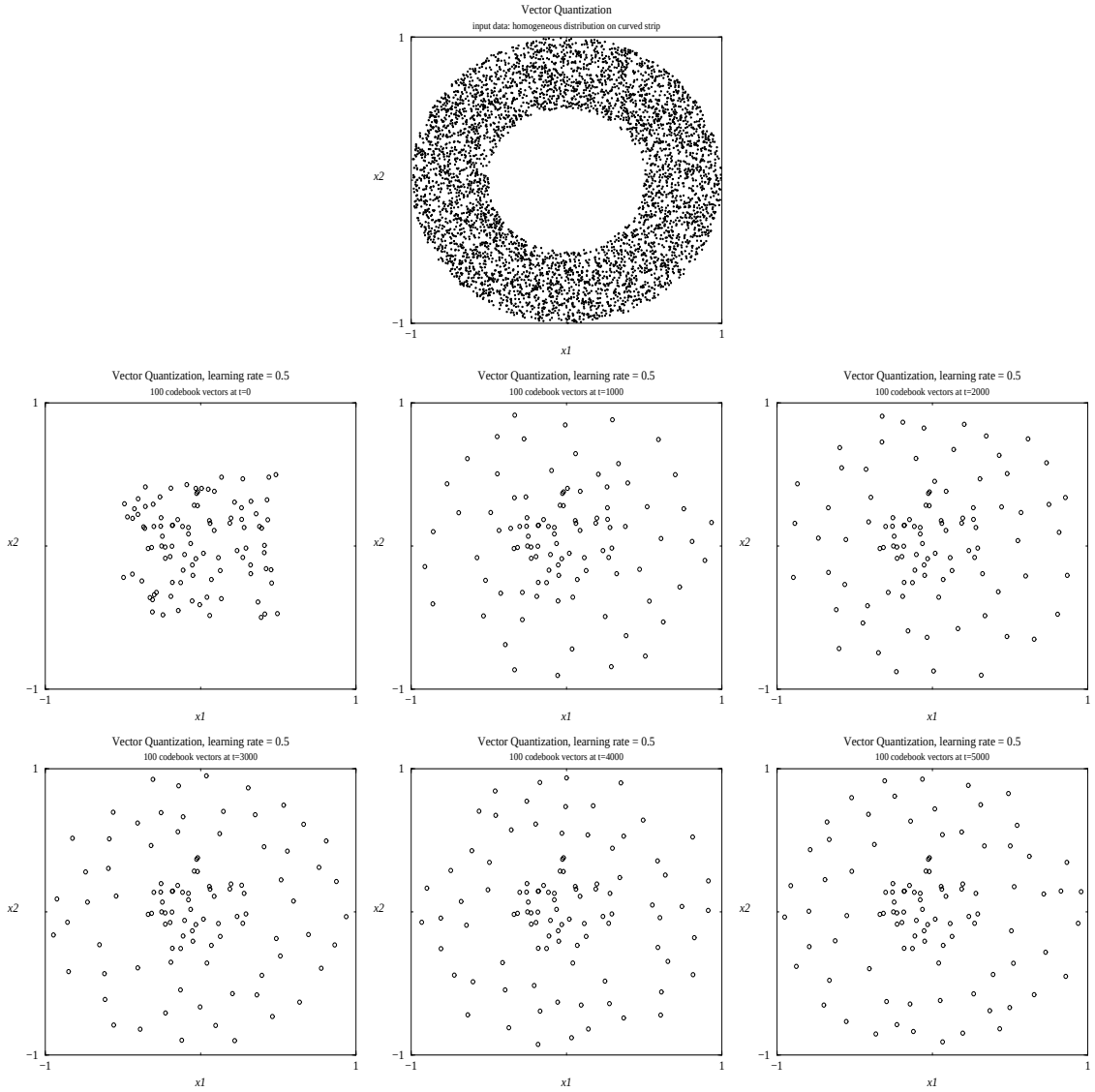


Figure 2.8: Example: numerical simulation of Vector Quantization, with $n = 2$, $\eta = 0.5$ and $N = 100$. Top graph: data distribution $p(\boldsymbol{x})$, homogeneously distributed over the region $\frac{1}{2} < |\boldsymbol{x}| < 1$. Bottom six graphs: locations of the code-book vectors $\boldsymbol{m}_i$ during the course of the process ($i = 1, \ldots, 100$), at times $t = 0$ (middle row, left), $t = 1000$ (middle row, center), $t = 2000$ (middle row, right), $t = 3000$ (bottom row, left), $t = 4000$ (bottom row, middle) and $t = 5000$ (bottom row, right). Times are measures in the number of iterations. Initialisation of code-book vector allocations: randomly in the square $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$. Comparison with figure 2.5 shows that code-book vectors initialized in data regions where $p(\boldsymbol{x})$ is zero tend to get 'stuck'.

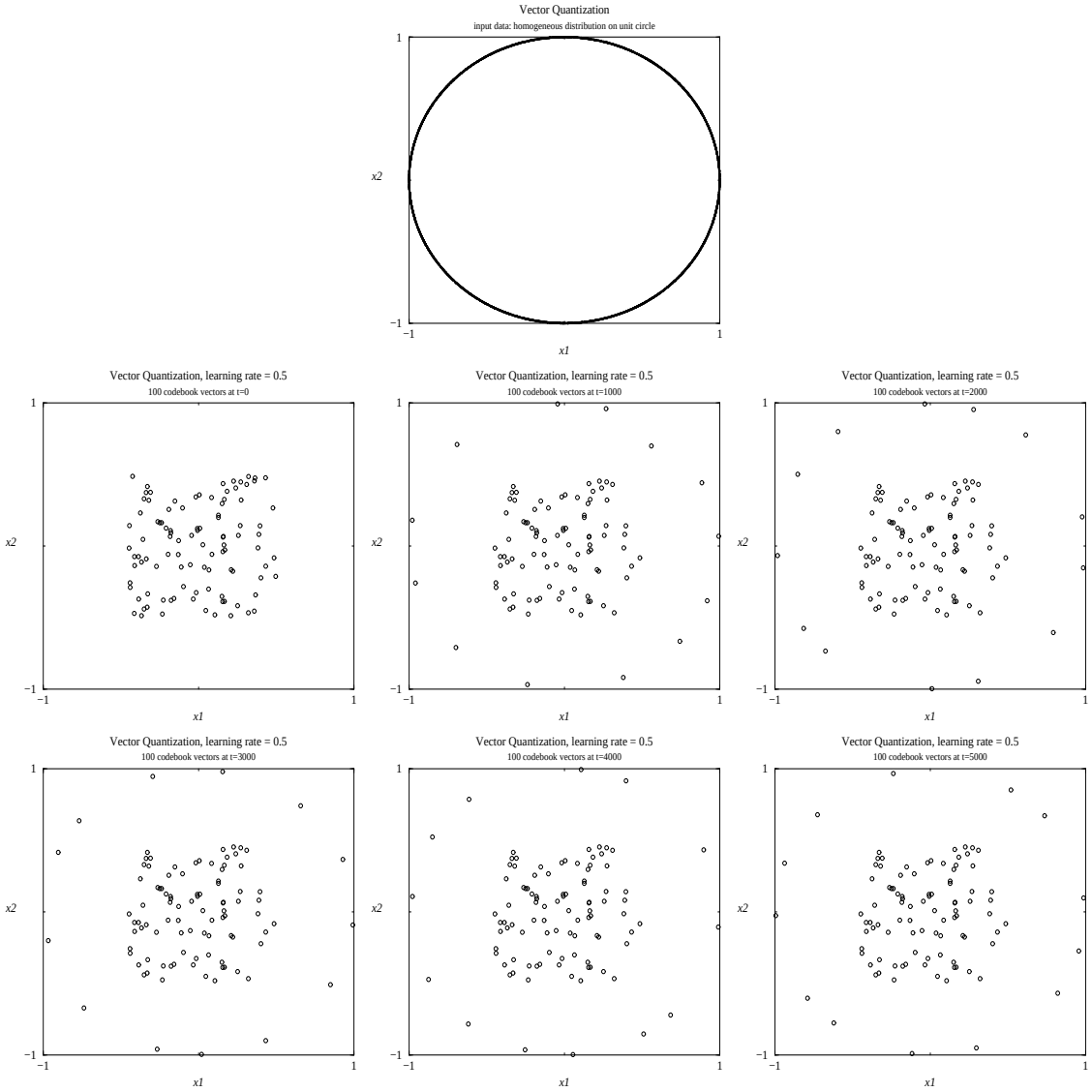


Figure 2.9: Example: numerical simulation of Vector Quantization, with $n = 2$, $\eta = 0.5$ and $N = 100$. Top graph: data distribution $p(\boldsymbol{x})$, homogeneously distributed over the circle $|\boldsymbol{x}| = 1$. Bottom six graphs: locations of the code-book vectors $\boldsymbol{m}_i$ during the course of the process ($i = 1, \ldots, 100$), at times $t = 0$ (middle row, left), $t = 1000$ (middle row, center), $t = 2000$ (middle row, right), $t = 3000$ (bottom row, left), $t = 4000$ (bottom row, middle) and $t = 5000$ (bottom row, right). Times are measures in the number of iterations. Initialisation of code-book vector allocations: randomly in the square $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$. Comparison with figure 2.6 shows that code-book vectors initialized in data regions where $p(\boldsymbol{x})$ is zero tend to get 'stuck'.

2.2 Soft Vector Quantization

The SVQ Algorithm and its Properties. The Soft Vector Quantization (SVQ) algorithm, which can be regarded as a 'smooth' version of VQ, is defined as follows. First we initialize the N code-book vectors $\boldsymbol{m}_i \in \Re^n$ randomly (in contrast to VQ, random initialization poses no problem for SVQ as we will see). Then we iterate the following stochastic process:

step 1: pick a data point $\boldsymbol{x}$ at random, according to the probability density $p(\boldsymbol{x})$

step 2: calculate, for *all* i:
$$F_i[\boldsymbol{x}, \{\boldsymbol{m}\}] = \frac{e^{-\beta(\boldsymbol{x}-\boldsymbol{m}_i)^2}}{\sum_{j=1}^{N} e^{-\beta(\boldsymbol{x}-\boldsymbol{m}_j)^2}} \tag{2.3}$$

step 3: move *all* $\boldsymbol{m}_i$ towards $\boldsymbol{x}$: $\boldsymbol{m}_i \rightarrow \boldsymbol{m}_i + \eta(\boldsymbol{x} - \boldsymbol{m}_i)F_i[\boldsymbol{x}, \{\boldsymbol{m}\}]$

step 4: return to 1

The parameter $0 < \eta \ll 1$ (learning rate) again controls the overall magnitude of the changes. Note that, by construction, $\sum_i F_i[\boldsymbol{x}, \{\boldsymbol{m}\}] = 1$ and that always $0 \leq F_i[\boldsymbol{x}, \{\boldsymbol{m}\}] \leq 1$.

The SVQ algorithm can be seen to have the following general properties:

(i) All code-book vectors are moved towards $\boldsymbol{x}$ at every iteration step, but those which are closest to $\boldsymbol{x}$ are moved most.

(ii) Unless we reduce η during the process, the code-book vectors will continue to move stochastically, although the *density* of code-book vectors in any given region should become stationary.

(iii) SVQ is still simple, but it has more parameter more than VQ. This extra parameter, β, defines a characteristic distance in data space: code-book vectors with $|\boldsymbol{x} - \boldsymbol{m}_i| > \beta^{-\frac{1}{2}}$ will move only weakly, in contrast to those with $|\boldsymbol{x} - \boldsymbol{m}_i| < \beta^{-\frac{1}{2}}$. Hence $1/\sqrt{\beta}$ defines the distance over which code-book vectors tend to exert an influence. Since the characteristic distance between the code-book vectors will also strongly depend on N, one should also expect the optimal value for β to depend on N.

(iv) SVQ has an advantage over VQ when data distributions $p(\boldsymbol{x})$ can (slowly) change over time. Although we can ensure in VQ, by appropriate initialization, that the code-book vectors cannot get stuck in data-poor regions, this will no longer be guaranteed if $p(\boldsymbol{x})$ can change over time: regions which are data-rich now, and which attract code-book vectors, might become data-poor later. In SVQ the system will always be able to adapt to the new data environment, since all code-book vectors are updated all the time.

Let us next investigate the action of the SVQ algorithm for the two extreme choices to be made for the new parameter β: $\beta = \infty$ and $\beta = 0$.

Fact 1: $\lim_{\beta\rightarrow\infty}$ SVQ = VQ

Proof: Define the Voronoi tessalation of data space induced by the code-book vectors $\{\boldsymbol{m}\}$. Consider an SVQ iteration step, where we pick data point $\boldsymbol{x}$. Assume $\boldsymbol{x} \in V_k[\{\boldsymbol{m}\}]$, i.e. $\boldsymbol{x}$ is found to be in the Voronoi cell of code-book vector k (we again rule out the

pathological cases where $\boldsymbol{x}$ is at the boundary of Voronoi cells; see the section on VQ). Multiply numerator and denominator of (2.3) by $e^{\beta(\boldsymbol{x}-\boldsymbol{m}_k)^2}$, and use the property that $|\boldsymbol{x}-\boldsymbol{m}_i| > |\boldsymbol{x}-\boldsymbol{m}_k|$ for all $i \neq k$:

$$\begin{aligned}\lim_{\beta\to\infty} F_i[\boldsymbol{x},\{\boldsymbol{m}\}] &= \lim_{\beta\to\infty} \frac{e^{-\beta[(\boldsymbol{x}-\boldsymbol{m}_i)^2-(\boldsymbol{x}-\boldsymbol{m}_k)^2]}}{\sum_{j=1}^{N} e^{-\beta[(\boldsymbol{x}-\boldsymbol{m}_j)^2-(\boldsymbol{x}-\boldsymbol{m}_k)^2]}} \\ &= \lim_{\beta\to\infty} \frac{e^{-\beta[(\boldsymbol{x}-\boldsymbol{m}_i)^2-(\boldsymbol{x}-\boldsymbol{m}_k)^2]}}{1+\sum_{j\neq k} e^{-\beta[(\boldsymbol{x}-\boldsymbol{m}_j)^2-(\boldsymbol{x}-\boldsymbol{m}_k)^2]}} = \begin{cases} 1 & \text{for } i=k \\ 0 & \text{for } i\neq k \end{cases}\end{aligned}$$

Hence for $\beta \to \infty$ the SVQ modification of the code-book vectors in a single iteration reduces to

$$\begin{aligned}&\boldsymbol{m}_k \to \boldsymbol{m}_k + \eta(\boldsymbol{x}-\boldsymbol{m}_k) \\ &\boldsymbol{m}_i \to \boldsymbol{m}_i \qquad\qquad \text{for all } i\neq k\end{aligned}$$

which is identical to that of VQ. □

Fact 2: For $\beta \to 0$ all code-book vectors $\boldsymbol{m}_i$ will ultimately collapse to a single point, which will fluctuate around the average data-point $\langle \boldsymbol{x}\rangle = \int d\boldsymbol{x}\ \boldsymbol{x}\ p(\boldsymbol{x})$.

Proof: For $\beta \to 0$ we find $F_i[\boldsymbol{x},\{\boldsymbol{m}\}] = N^{-1}$ (for any $\boldsymbol{x}$, any i and any $\{\boldsymbol{m}\}$) and the SVQ modifications simply reduce to

$$\boldsymbol{m}_i \to \boldsymbol{m}_i + \frac{\eta}{N}(\boldsymbol{x}-\boldsymbol{m}_i) \qquad \text{for all } i$$

Now consider the difference between any two code-book vectors. If we label the iterations by $\ell = 0, 1, 2, \ldots$, we find

$$\boldsymbol{m}_i(\ell+1) - \boldsymbol{m}_j(\ell+1) = (1-\frac{\eta}{N})[\boldsymbol{m}_i(\ell)-\boldsymbol{m}_j(\ell)]$$

$$\text{hence} \qquad \boldsymbol{m}_i(\ell)-\boldsymbol{m}_j(\ell) = (1-\frac{\eta}{N})^{\ell}[\boldsymbol{m}_i(0)-\boldsymbol{m}_j(0)]$$

so $\lim_{\ell\to\infty}[\boldsymbol{m}_i(\ell)-\boldsymbol{m}_j(\ell)] = \boldsymbol{0}$ □

To find out where all the code-book vectors will go, we only need to inspect the dynamics of the average $\boldsymbol{m} = N^{-1}\sum_i \boldsymbol{m}_i$ (since $\lim_{\ell\to\infty}[\boldsymbol{m}(\ell)-\boldsymbol{m}_i(\ell)] = \boldsymbol{0}$). Upon writing the data point drawn at iteration ℓ as $\boldsymbol{x}(\ell)$, we get $\boldsymbol{m}(\ell+1) = (1-\frac{\eta}{N})\boldsymbol{m}(\ell) + \frac{\eta}{N}\boldsymbol{x}(\ell)$. Hence

$$\begin{aligned}\boldsymbol{m}(1) &= (1-\frac{\eta}{N})\boldsymbol{m}(0) + \frac{\eta}{N}\boldsymbol{x}(0) \\ \boldsymbol{m}(2) &= (1-\frac{\eta}{N})\left[(1-\frac{\eta}{N})\boldsymbol{m}(0) + \frac{\eta}{N}\boldsymbol{x}(0)\right] + \frac{\eta}{N}\boldsymbol{x}(1) \\ &= (1-\frac{\eta}{N})^2\boldsymbol{m}(0) + \frac{\eta}{N}\left[(1-\frac{\eta}{N})\boldsymbol{x}(0) + \boldsymbol{x}(1)\right] \\ \boldsymbol{m}(3) &= (1-\frac{\eta}{N})\left[(1-\frac{\eta}{N})^2\boldsymbol{m}(0) + \frac{\eta}{N}(1-\frac{\eta}{N})\boldsymbol{x}(0) + \frac{\eta}{N}\boldsymbol{x}(1)\right] + \frac{\eta}{N}\boldsymbol{x}(2) \\ &= (1-\frac{\eta}{N})^3\boldsymbol{m}(0) + \frac{\eta}{N}\left[(1-\frac{\eta}{N})^2\boldsymbol{x}(0) + (1-\frac{\eta}{N})\boldsymbol{x}(1) + \boldsymbol{x}(2)\right]\end{aligned}$$

$$\begin{aligned} \vdots &= \vdots \\ \boldsymbol{m}(\ell) &= (1-\frac{\eta}{N})^{\ell}\boldsymbol{m}(0) + \frac{\eta}{N}\sum_{k=0}^{\ell-1}(1-\frac{\eta}{N})^{\ell-1-k}\boldsymbol{x}(k) \end{aligned}$$

Averaging over the possible choices of data, using $\langle \boldsymbol{x}(k)\rangle = \langle \boldsymbol{x}\rangle$, the average of the distribution $p(\boldsymbol{x})$ (assuming this average to be finite), and using $\sum_{k\geq 0} z^k = 1/(1-z)$ (for $|z|<1$) then gives us

$$\begin{aligned} \lim_{\ell\to\infty}\langle \boldsymbol{m}(\ell)\rangle &= \lim_{\ell\to\infty}\left\{(1-\frac{\eta}{N})^{\ell}\boldsymbol{m}(0) + \frac{\eta}{N}\langle \boldsymbol{x}\rangle\sum_{k\geq 0}^{\ell-1}(1-\frac{\eta}{N})^{k}\right\} \\ &= \frac{\eta}{N}[1-(1-\frac{\eta}{N})]^{-1}\langle \boldsymbol{x}\rangle = \langle \boldsymbol{x}\rangle \end{aligned}$$

As claimed. □

One can similarly show that the fluctuations of $\boldsymbol{m}$ around $\langle \boldsymbol{x}\rangle$ remain finite, provided the width of the data distribution is finite, i.e. $\langle \boldsymbol{x}^2\rangle = \int d\boldsymbol{x}\; p(\boldsymbol{x})\boldsymbol{x}^2 < \infty$.

Thus VQ can be seen as a special case of SVQ, obtained upon putting $\beta\to\infty$. Secondly we infer from inspection of the extreme case $\beta = 0$ that one of the effects of the smoothening of SVQ (relative to VQ) is for code-vectors to 'drag one another along'.

A Lyapunov Function for Small Learning Rates. Our understanding of SVQ (and thus also of VQ) would greatly improve if we could recognise the process as the minimisation of some error measure. For finite η this is not possible, but for for $\eta\to 0$ it is. Upon labeling the different iterations with $\ell = 0,1,2,\ldots$, and upon writing the data point drawn at stage ℓ as $\boldsymbol{x}(\ell)$, we can write both VQ and SVQ in the following compact form:

$$\boldsymbol{m}_i(\ell+1) = \boldsymbol{m}_i(\ell) + \eta F_i[\boldsymbol{x}(\ell),\{\boldsymbol{m}(\ell)\}][\boldsymbol{x}(\ell)-\boldsymbol{m}_i(\ell)] \tag{2.4}$$

with

$$F_i^{\rm SVQ}[\boldsymbol{x},\{\boldsymbol{m}\}] = \frac{e^{-\beta(\boldsymbol{x}-\boldsymbol{m}_i)^2}}{\sum_{j=1}^{N} e^{-\beta(\boldsymbol{x}-\boldsymbol{m}_j)^2}} \tag{2.5}$$

$$F_i^{\rm VQ}[\boldsymbol{x},\{\boldsymbol{m}\}] = \left\{\begin{array}{ll} 1 & \text{if } \boldsymbol{x}\in V_i[\{\boldsymbol{m}\}] \\ 0 & \text{otherwise}\end{array}\right. \tag{2.6}$$

For small learning rates we follow the procedure introduced to derive deterministic equations for on-line learning in layered neural networks (see lecture notes 'Neural Networks'), and define a new time unit $t=\eta\ell$. We then take the $\eta\to 0$ limit and find the stochastic process (2.4) being replaced by the deterministic equation

$$\frac{d}{dt}\boldsymbol{m}_i = \langle (\boldsymbol{x}-\boldsymbol{m}_i)F_i[\boldsymbol{x},\{\boldsymbol{m}\}]\rangle \tag{2.7}$$

(with $\langle f(\boldsymbol{x})\rangle = \int d\boldsymbol{x}\; p(\boldsymbol{x})f(\boldsymbol{x})$). Let us now choose $F_i^{\rm SVQ}[\boldsymbol{x},\{\boldsymbol{m}\}]$, and let us define the following average of Gaussian distributions, each centred at one of the code-book vectors:

$$q(\boldsymbol{x}|\{\boldsymbol{m}\}) = \frac{1}{N}\sum_i \frac{e^{-\beta(\boldsymbol{x}-\boldsymbol{m}_i)^2}}{(\sqrt{\pi/\beta})^n} \tag{2.8}$$

We can now prove the following:

Fact 3: For $F_i[\boldsymbol{x},\{\boldsymbol{m}\}] = F_i^{\mathrm{SVQ}}[\boldsymbol{x},\{\boldsymbol{m}\}]$, equation (2.7) defines gradient descent:

$$\frac{d}{dt}\boldsymbol{m}_i = -\frac{1}{2\beta}\boldsymbol{\nabla}_{\boldsymbol{m}_i}E[\{\boldsymbol{m}\}] \qquad E[\{\boldsymbol{m}\}] = \int d\boldsymbol{x}\; p(\boldsymbol{x})\log\left[\frac{p(\boldsymbol{x})}{q(\boldsymbol{x}|\{\boldsymbol{m}\})}\right] \tag{2.9}$$

Proof: Just work out the relevant partial derivatives of $E[\{\boldsymbol{m}\}]$:

$$\begin{aligned}
-\frac{1}{2\beta}\boldsymbol{\nabla}_{\boldsymbol{m}_i}E[\{\boldsymbol{m}\}] &= \frac{1}{2\beta}\int d\boldsymbol{x}\; p(\boldsymbol{x})\boldsymbol{\nabla}_{\boldsymbol{m}_i}\log\; q(\boldsymbol{x}|\{\boldsymbol{m}\}) \\
&= \frac{1}{2\beta}\int d\boldsymbol{x}\; p(\boldsymbol{x})\boldsymbol{\nabla}_{\boldsymbol{m}_i}\log\sum_j e^{-\beta(\boldsymbol{x}-\boldsymbol{m}_j)^2} \\
&= \frac{1}{2\beta}\int d\boldsymbol{x}\; p(\boldsymbol{x})\left\{\frac{\boldsymbol{\nabla}_{\boldsymbol{m}_i}e^{-\beta(\boldsymbol{x}-\boldsymbol{m}_i)^2}}{\sum_j e^{-\beta(\boldsymbol{x}-\boldsymbol{m}_j)^2}}\right\} \\
&= \int d\boldsymbol{x}\; p(\boldsymbol{x})\left\{\frac{(\boldsymbol{x}-\boldsymbol{m}_i)e^{-\beta(\boldsymbol{x}-\boldsymbol{m}_i)^2}}{\sum_j e^{-\beta(\boldsymbol{x}-\boldsymbol{m}_j)^2}}\right\} = \langle\; (\boldsymbol{x}-\boldsymbol{m}_i)F_i^{\mathrm{SVQ}}[\boldsymbol{x},\{\boldsymbol{m}\}]\;\rangle
\end{aligned}$$

Hence the desired result. □

The object in equation (2.9) is a familiar function in information theory, where it would have been written as $D(p||q)$, and where it goes under the name Kullback-Leibler distance. It obeys $D(p||q) \geq 0$ for any two distributions $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$, with equality only if $p(\boldsymbol{x}) = q(\boldsymbol{x})$ (in a distributional sense). It is an information-theoretic measure of the deviation between the two distributions $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$.

From the general inequality $D(p||q) \geq 0$ (and hence $E[\{\boldsymbol{m}\}] \geq 0$), in combination with the gradient descent equation in (2.9) (which ensures that $\frac{d}{dt}E \leq 0$), we may now conclude that the function in (2.9) is a Lyapunov function for the SVQ process (2.4). Hence we can interpret SVQ, at least for small η, as approximating the data distribution $p(\boldsymbol{x})$ optimally by a mixture of Gaussians of the form (2.8), via adaptation of the centres $\boldsymbol{m}_i$ of the Gaussians. Similarly, since $\lim_{\beta\to\infty}\mathrm{SVQ} = VQ$ and since $\lim_{\beta\to\infty}[\beta/\pi]^{\frac{n}{2}}e^{-\beta(\boldsymbol{x}-\boldsymbol{m}_i)^2} = \delta[\boldsymbol{x}-\boldsymbol{m}_i]$, we may for small η interpret VQ as approximating the data distribution $p(\boldsymbol{x})$ optimally by a mixture of delta-distributions, via adaptation of the centres of the delta-distributions:

$$\eta \ll 1,\quad \mathrm{SVQ}: \qquad \text{finds } \{\boldsymbol{m}\} \text{ such that} \qquad p(\boldsymbol{x}) \;\approx\; \frac{1}{N}\sum_i \frac{e^{-\beta(\boldsymbol{x}-\boldsymbol{m}_i)^2}}{(\sqrt{\pi/\beta})^n}$$

$$\eta \ll 1,\quad \mathrm{VQ}: \qquad \text{finds } \{\boldsymbol{m}\} \text{ such that} \qquad p(\boldsymbol{x}) \;\approx\; \frac{1}{N}\sum_i \delta[\boldsymbol{x}-\boldsymbol{m}_i]$$

The characteristic distance $\sigma = 1/\sqrt{2\beta}$ which we already identified before, is now giving the widths of the individual Gaussians with which SVQ aims to copy the data distribution $p(\boldsymbol{x})$. For finite η (provided not too large) we may regard SVQ and VQ as noisy versions of the above gradient descent processes.

Finally we briefly note that the stationary state of the $\eta \to 0$ equation (2.7), given by $\langle(\boldsymbol{x}-\boldsymbol{m}_i)F_i[\boldsymbol{x},\{\boldsymbol{m}\}]\rangle = \boldsymbol{0}$, translates into

$$\boldsymbol{m}_i = \frac{\int d\boldsymbol{x}\; \boldsymbol{x}F_i[\boldsymbol{x},\{\boldsymbol{m}\}]p(\boldsymbol{x})}{\int d\boldsymbol{x}\; F_i[\boldsymbol{x},\{\boldsymbol{m}\}]p(\boldsymbol{x})} \qquad \text{so for VQ}: \qquad \boldsymbol{m}_i = \frac{\int_{V_i[\{\boldsymbol{m}\}]}d\boldsymbol{x}\; \boldsymbol{x}\; p(\boldsymbol{x})}{\int_{V_i[\{\boldsymbol{m}\}]}d\boldsymbol{x}\; p(\boldsymbol{x})} \tag{2.10}$$

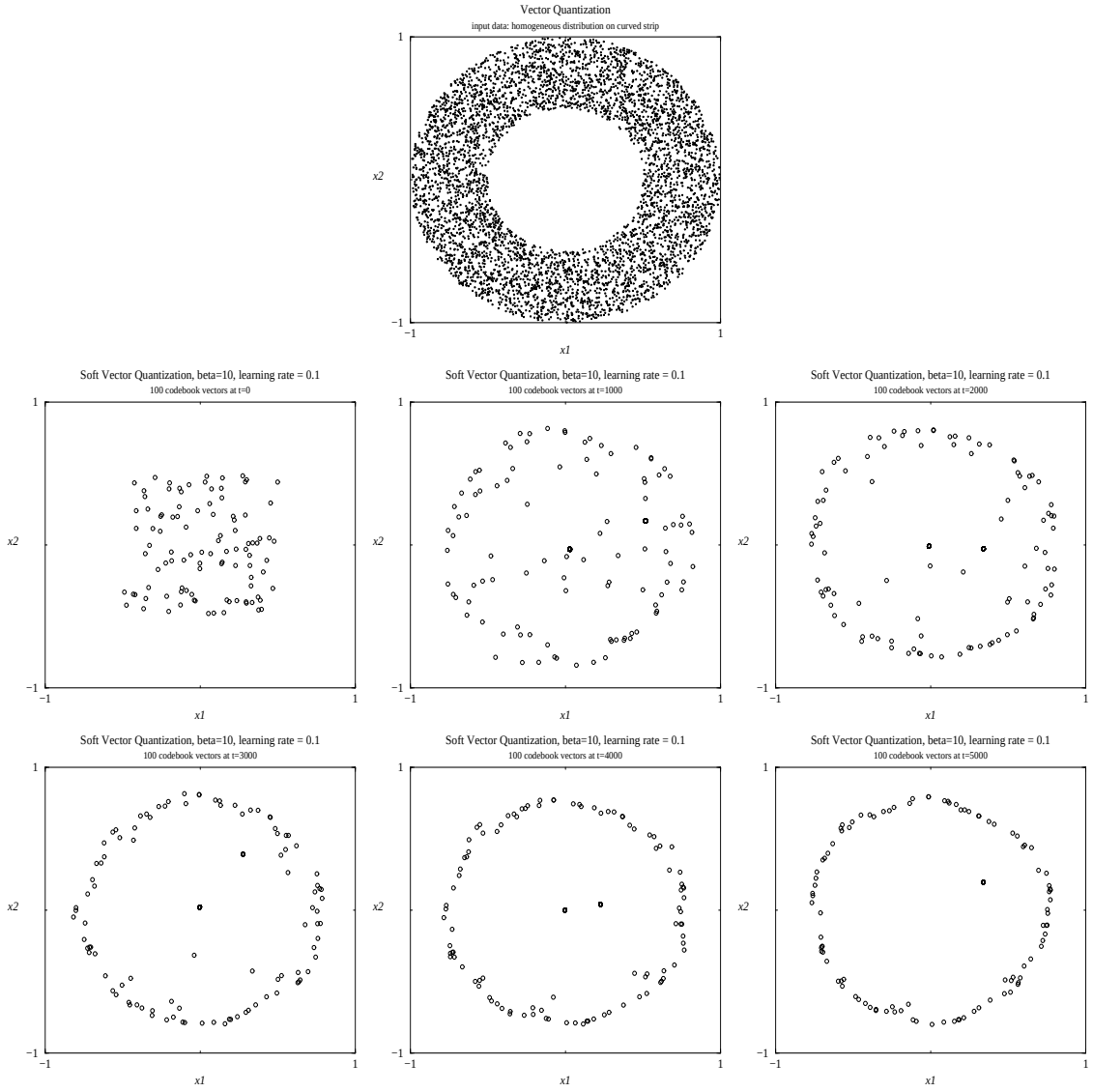


Figure 2.10: Example: numerical simulation of Soft Vector Quantization, with $n = 2$, $\eta = 0.1$, $\beta = 10$ and $N = 100$. Top graph: data distribution $p(\boldsymbol{x})$, homogeneously distributed over the region $\frac{1}{2} < |\boldsymbol{x}| < 1$. Bottom six graphs: locations of the code-book vectors $\boldsymbol{m}_i$ during the course of the process ($i = 1, \ldots, 100$), at times $t = 0$ (middle row, left), $t = 1000$ (middle row, center), $t = 2000$ (middle row, right), $t = 3000$ (bottom row, left), $t = 4000$ (bottom row, middle) and $t = 5000$ (bottom row, right). Times are measures in the number of iterations. Initialisation of code-book vector allocations: randomly in the square $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$.

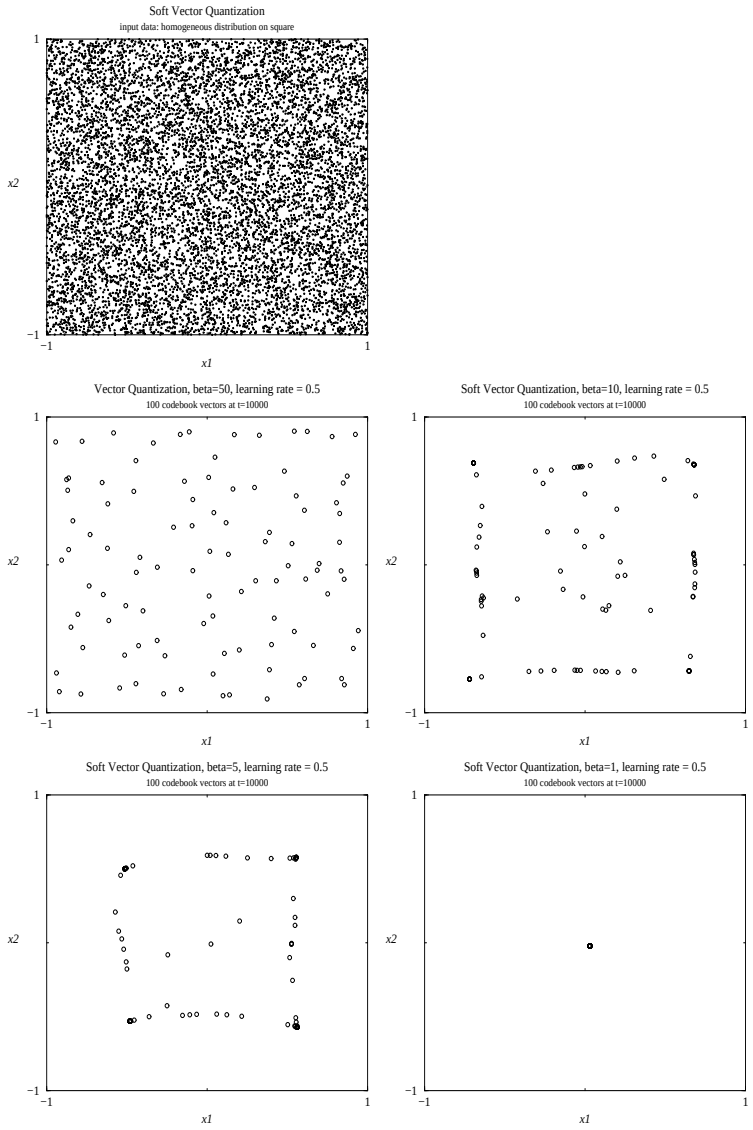


Figure 2.11: Example: numerical simulation of Soft Vector Quantization, with $n = 2$, $\eta = 0.5$ and $N = 100$. Top graph: data distribution $p(\boldsymbol{x})$, homogeneously distributed over the square $[-1, 1] \times [-1, 1]$. Bottom four graphs: locations of the code-book vectors $\boldsymbol{m}_i$ after 10,000 iteration steps ($i = 1, \ldots, 100$), for different choices of the parameter β: $\beta = 50$ (middle row, left), $\beta = 10$ (middle row, right), $\beta = 5$ (bottom row, left) and $\beta = 1$ (bottom row, right). Initialization of code-book vector allocations: randomly in the square $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$.

For VQ we thus have the transparent result that, in equilibrium and for η sufficiently small, the location of code-book vector $\boldsymbol{m}_i$ will be exactly the centre of gravity of the data distribution within its associated Voronoi cell.

Examples of SVQ in Action. The figures on the previous two pages illustrate the functioning of the SVQ algorithm for a number of simple examples with $n = 2$ (i.e. where one has data points $\boldsymbol{x} = (x_1, x_2)$ in a plane) and $N = 100$ (i.e. a population of 100 code-book vectors), but for different choices of the data distribution $p(\boldsymbol{x})$. Figure 2.10 shows the locations of the code-book vectors at different times, for $\beta = 10$, where the characteristic width of the data covering Gaussians is $\sigma = 1/\sqrt{2\beta} \approx 0.22$. We observe that code-book vectors indeed do not get 'stuck' in data-poor regions, and that the code-book vectors indeed keep a distance of the order of σ from the boundaries of the data region (due to the width of the data strip being $\frac{1}{2}$, this more or less sends all code-book vectors to the circle $|\boldsymbol{x}| = 3/4$). Figure 2.11 shows the asymptotic locations (after 10,000 iterations) of the code-book vectors for four different values of β. The corresponding values of σ are: $\sigma = 0.1$ ($\beta = 50$), $\sigma \approx 0.22$ ($\beta = 10$), $\sigma \approx 0.32$ ($\beta = 5$), and $\sigma \approx 0.71$ ($\beta = 1$). With these values one can understand perfectly the observed clustering properties of the code-book vectors; this underlines the power of results such as those above (i.e. fact 3).

2.3 Time-Dependent Learning Rates

We have seen that for finite η the processes of the type (2.4) can be regarded as 'noisy' versions of the deterministic equation (2.7) (for SVQ and VQ, the latter, in turn, minimise a sensible error measure). Hence one would in the initial stages of these processes prefer to have a finite η (since the added randomness may prevent the system from going to a sub-optimal local minimum of the error measure), but one would asymptotically prefer a fluctuation-free and unique (reproducible) final state, i.e. $\eta \to 0$. It would therefore appear natural to choose a slowly but monotonically decreasing time-dependent learning rate $\eta(\ell)$, where ℓ labels the iterations of the algorithm (2.4). The question is how to determine the optimal rate of decay for $\eta(\ell)$. Too fast a reduction of $\eta(\ell)$ might cause evolution to local minima, or might prevent the code-book vectors from traveling over sufficiently large distances to take up their optimal positions in data space (note that we do not know beforehand the size of data space). Too slow a reduction of $\eta(\ell)$ might cause fluctuations and non-uniqueness of the final state to persist too long for us to achieve the desired stationary state within the time-scales of our experiments. Below we show that sensible criteria to be met by $\eta(\ell)$ are

$$\text{not too fast}: \quad \sum_{\ell=0}^{\infty} \eta(\ell) = \infty \qquad \text{not too slow}: \quad \sum_{\ell=0}^{\infty} \eta^2(\ell) < \infty \tag{2.11}$$

Fact 1: The learning rate must obey $\sum_{\ell=0}^{\infty} \eta(\ell) = \infty$, otherwise there will be an a priori bound on the possible distance to be covered by the code-book vectors.

Proof: From $\boldsymbol{m}_i(\ell+1) = \boldsymbol{m}_i(\ell)\,[1 - \eta F_i[\boldsymbol{x}(\ell), \{\boldsymbol{m}(\ell)\}]] + \eta F_i[\boldsymbol{x}(\ell), \{\boldsymbol{m}(\ell)\}]\boldsymbol{x}(\ell)$, which is just an alternative way to write equation (2.4), together with $F_i[\boldsymbol{x}, \{\boldsymbol{m}\}] \in [0, 1]$ and $0 \leq \eta \leq 1$, we deduce for time dependent learning rates:

$$|\boldsymbol{m}_i(\ell+1)| \leq |\boldsymbol{m}_i(\ell)| + \eta(\ell)|\boldsymbol{x}(\ell)|$$

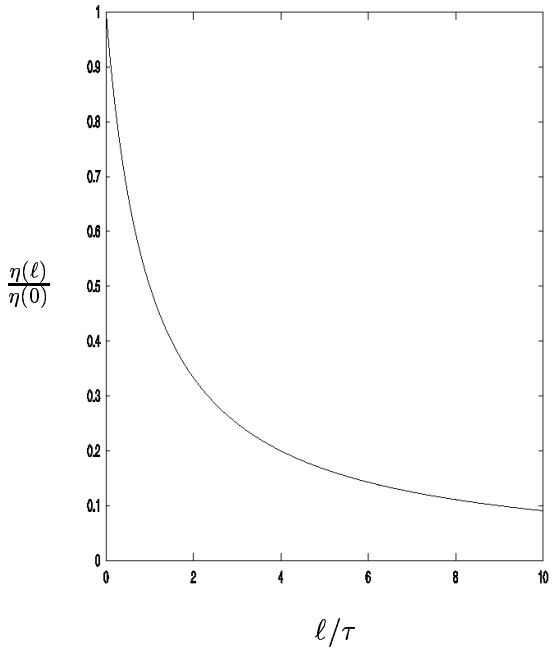


Figure 2.12: The time-dependent learning rate defined in (2.12), which meets the conditions (2.11) on the asymptotic rate of decay to zero.

Hence, upon repeated iteration:

$$|\boldsymbol{m}_i(\ell)| \leq |\boldsymbol{m}_i(0)| + \sum_{k=0}^{\ell-1} \eta(k)|\boldsymbol{x}(k)| \qquad \text{so} \qquad \lim_{\ell\to\infty} \langle |\boldsymbol{m}_i(\ell)| \rangle \leq |\boldsymbol{m}_i(0)| + \langle |\boldsymbol{x}| \rangle \sum_{k=0}^{\infty} \eta(k)$$

Thus, unless $\sum_{k=0}^{\infty} \eta(k) = \infty$, we have an a priori bound on the distance which can be traveled by any code-book vector, which could prevent outliers of the data distribution from being reached. □

Fact 2: The learning rate must obey $\sum_{\ell=0}^{\infty} \eta^2(\ell) < \infty$, otherwise even for the simplest process of the class (2.4) the uncertainty in the location of the average code-book vector will diverge.

Proof: The simplest process of the class (2.4) is the $\beta \to 0$ limit of SVQ: $F_i[\boldsymbol{x}, \{\boldsymbol{m}\}] = 1/N$, so $\boldsymbol{m}(\ell+1) = \boldsymbol{m}(\ell) + (\eta(\ell)/N)[\boldsymbol{x}(\ell) - \boldsymbol{m}(\ell)]$, where $\boldsymbol{m}(\ell) = \frac{1}{N}\sum_i \boldsymbol{m}_i(\ell)$. We now write $\boldsymbol{m}(\ell) = \langle \boldsymbol{m}(\ell) \rangle + \boldsymbol{v}(\ell)$, so that $\langle \boldsymbol{v}(\ell) \rangle = \boldsymbol{0}$ and

$$\boldsymbol{v}(\ell+1) = [1 - \frac{\eta(\ell)}{N}]\boldsymbol{v}(\ell) + \frac{\eta(\ell)}{N}[\boldsymbol{x}(\ell) - \langle \boldsymbol{x} \rangle]$$

Note that $\boldsymbol{v}(\ell)$ is statistically independent of all $\boldsymbol{x}(\ell')$ with $\ell' \geq \ell$. Note also that, since at $\ell = 0$ there are no fluctuations yet: $\boldsymbol{v}(0) = \boldsymbol{0}$. Hence

$$\begin{aligned} \boldsymbol{v}^2(\ell+1) &= [1-\frac{\eta(\ell)}{N}]^2 \boldsymbol{v}^2(\ell) + \frac{\eta^2(\ell)}{N^2}[\boldsymbol{x}(\ell) - \langle \boldsymbol{x} \rangle]^2 + \frac{2\eta(\ell)}{N}[1-\frac{\eta(\ell)}{N}]\boldsymbol{v}(\ell) \cdot [\boldsymbol{x}(\ell) - \langle \boldsymbol{x} \rangle] \\ \langle \boldsymbol{v}^2(\ell+1) \rangle &= [1-\frac{\eta(\ell)}{N}]^2 \langle \boldsymbol{v}^2(\ell) \rangle + \frac{\eta^2(\ell)}{N^2} \langle [\boldsymbol{x} - \langle \boldsymbol{x} \rangle]^2 \rangle \end{aligned}$$

Further iteration gives

$$\begin{aligned}
\langle \boldsymbol{v}^2(\ell+2)\rangle &= [1-\frac{\eta(\ell+1)}{N}]^2[1-\frac{\eta(\ell)}{N}]^2\langle \boldsymbol{v}^2(\ell)\rangle \\
&\quad +[1-\frac{\eta(\ell+1)}{N}]^2\frac{\eta^2(\ell)}{N^2}\langle[\boldsymbol{x}-\langle\boldsymbol{x}\rangle]^2\rangle + \frac{\eta^2(\ell+1)}{N^2}\langle[\boldsymbol{x}-\langle\boldsymbol{x}\rangle]^2\rangle \\
&= [1-\frac{\eta(\ell+1)}{N}]^2[1-\frac{\eta(\ell)}{N}]^2\langle \boldsymbol{v}^2(\ell)\rangle \\
&\quad +[\frac{\eta^2(\ell+1)}{N^2}+\frac{\eta^2(\ell)}{N^2}+\mathcal{O}(\eta^3)]\langle[\boldsymbol{x}-\langle\boldsymbol{x}\rangle]^2\rangle \\
&\vdots \qquad \vdots \\
\langle \boldsymbol{v}^2(\ell)\rangle &= \langle \boldsymbol{v}^2(0)\rangle\prod_{k=0}^{\ell-1}[1-\frac{\eta(k)}{N}]^2 + \langle[\boldsymbol{x}-\langle\boldsymbol{x}\rangle]^2\rangle\sum_{k=0}^{\ell-1}[\frac{\eta^2(k)}{N^2}+\mathcal{O}(\eta^3)] \\
&= \langle[\boldsymbol{x}-\langle\boldsymbol{x}\rangle]^2\rangle\sum_{k=0}^{\ell-1}[\frac{\eta^2(k)}{N^2}+\mathcal{O}(\eta^3)]
\end{aligned}$$

We conclude that the fluctuations $\boldsymbol{v}$ diverge, i.e. $\lim_{\ell\to\infty}\langle \boldsymbol{v}^2(\ell)\rangle = \infty$, as soon as $\sum_{\ell=0}^{\infty}\eta^2(\ell) = \infty$. As claimed. □

The most commonly used asymptotic form of decay for $\eta(\ell)$, which meets the above two requirements, is $\eta(\ell) \sim \ell^{-1}$ as $\ell \to \infty$. For instance:

$$\eta(\ell) = \frac{\eta(0)}{1+\ell/\tau} \qquad \text{which obeys} \qquad \begin{array}{ll} \ell \ll \tau: & \eta(\ell) \approx \eta(0)[1-\ell/\tau] \\ \ell \approx \tau: & \eta(\ell) \approx \frac{1}{2}\eta(0) \\ \ell \gg \tau & \eta(\ell) \approx \eta(0)\tau/\ell \end{array} \tag{2.12}$$

This dependence is drawn in figure 2.12.

2.4 Self-Organizing Maps

Detour: Biological Inspiration. Any flexible and robust autonomous system (whether living or robotic) will have to be able to create, or at least update, an internal 'map' or representation of its environment. Information on its environment, however, is usually obtained in an indirect manner, through a redundant set of usually highly non-linear sensors which each provide only partial and indirect information. The system responsible for forming this map needs to be adaptive, as both environment and sensors can change their characteristics during the system's life-time. Our brain performs recallibration of sensors all the time; e.g. simply because we grow will the neuronal information about limb positions (generated by sensors which measure the stretch of muscles) have to be reinterpreted continually. Anatomic changes, and even learning new skills (like playing an instrument), are found to induce modifications of internal maps. At a more abstract level, the neural system responsible for building the internal representation is confronted with a complicated non-linear mapping from a relatively low-dimensional and more or less flat space (the 'physical world' W, with D dimensions) into a high-dimensional one (the space of sensory signals, of which there are $n \gg D$), and the aim is to extract the properties of the original space W from the sensory signals alone, i.e.

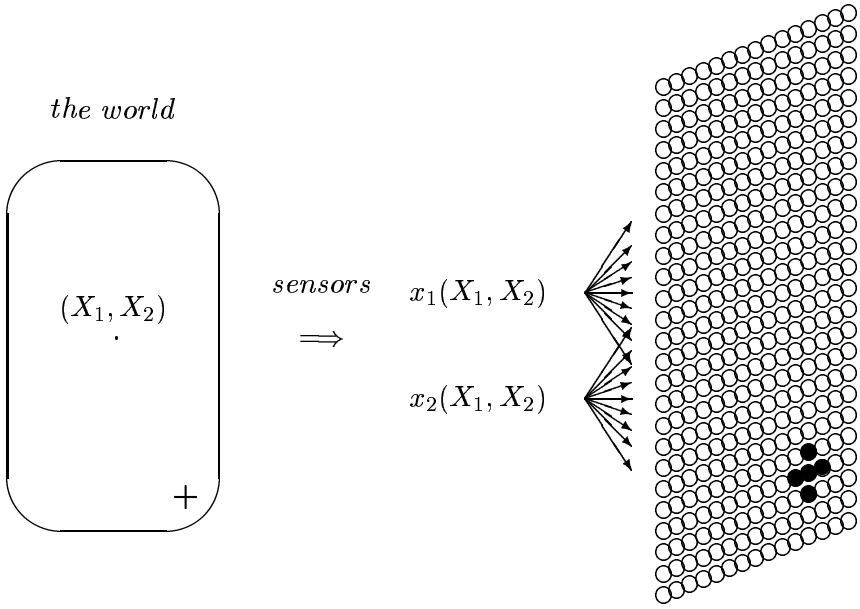


Figure 2.13: Simple 2-dim. example of the internal representation ('map') of the environment as built in biological systems. The brain has no direct access to the world coordinates (X_1, X_2), it is only given sensory signals $x_1(X_1, X_2)$ and $x_2(X_1, X_2)$. Many representations of the coordinates (X_1, X_2) would have been possible; in the specific one aimed for here, each (indirectly) observed position (X_1, X_2) gives rise to a specific localised area of firing activity in a sheet of neurons. Compare this to driving a car with blinded windows, but with electronic sensors observing the environment. The sensory information is to be converted back into world coordinates, and to make an light flash up in a road map inside the car, exactly at the car's current location. This is the information used by the driver. The question discussed in this section is how to *construct* the road map and the conversion of sensory to world coordinates, on the basis of structure in the sensory signals $x_1(X_1, X_2)$ and $x_2(X_1, X_2)$.

to 'invert' the operation which maps world states to sensory signals.

world coordinates :		*sensory signals* :		*internal representation* :
$\boldsymbol{X} = (X_1, \ldots, X_D) \in W$	$\longrightarrow$	$\boldsymbol{x} = (x_1, \ldots, x_n)$	$\longrightarrow$	reconstruction of W ?

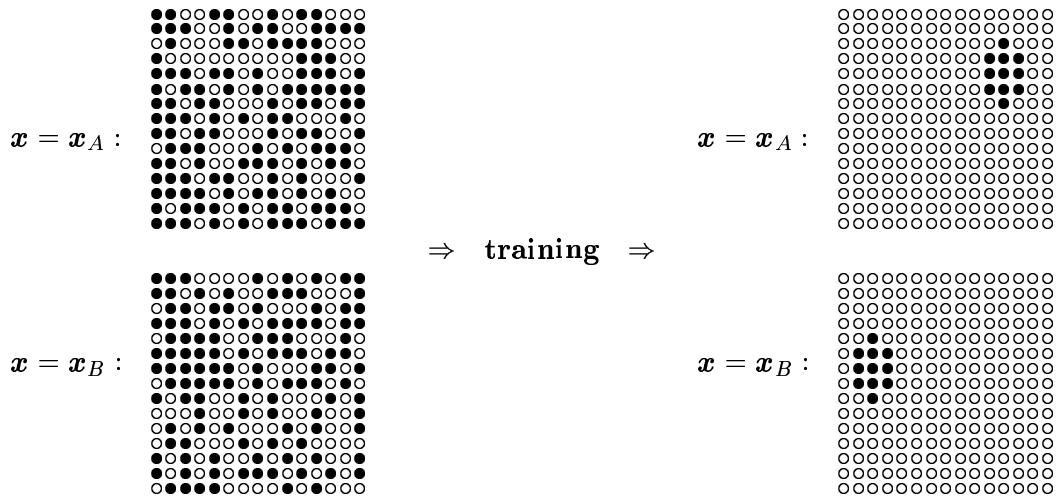
The definition of the 'world' depends on the specific degrees of freedom of the system at hand, e.g. D need not be three. The sensory signals are usually non-linear, indirect and noisy, but also highly redundant to compensate for their individual inadequacies. The key to achieving the objective is to exploit continuity and correlations in sensory signals, assuming similar sensory signals to represent similar positions in the environment, which therefore must correspond to similar positions in the internal map: if $\boldsymbol{X'} = \boldsymbol{X} + \Delta\boldsymbol{X}$ then also $\boldsymbol{x}(\boldsymbol{X'}) \approx \boldsymbol{x}(\boldsymbol{X}) + \boldsymbol{A}(\boldsymbol{X}).\Delta\boldsymbol{X}$, for some $\boldsymbol{A}(\boldsymbol{X})$ (the Jacobian matrix of the mapping $\boldsymbol{X} \to \boldsymbol{x}$).

Before turning to the learning process which is to carry out the construction task, we first explain in more detail the type of representation one aims for (which is inspired by biological systems). Let us give a simple example (see also figure 2.13). Image a system operating in

a simple two-dimensional world, where positions are represented by Cartesian coordinates $\boldsymbol{X} = (X_1, X_2)$, observed by sensors and fed into a neural network as input signals. For simplicity we will consider trivial sensors, and just put $x_i = X_i$. Each neuron i receives information on the input signals (x_1, x_2) in the usual way, via modifiable synapses $\{m_{ij}\}$: $y_i(\boldsymbol{x}) = g[m_{i1}x_1 + m_{i2}x_2]$, for some monotonic non-linear function $g[z]$ (e.g. $g[z] = \mathrm{erf}[z]$ or $g[z] = \frac{1}{2}(1 + \tanh[z])$). We indicate the (physical) location of neuron i in the array as $\boldsymbol{r}_i$. If this network is to become an internal coordinate system, faithfully reflecting the events $\boldsymbol{x} = (x_1, x_2)$ observed in the outside world (in the present example its topology must accordingly be that of a two-dimensional array), the following objectives are to be met

1. Each neuron y_i is more or less 'tuned' to a specific type of signal $\boldsymbol{x}_i$, i.e. $y_i(\boldsymbol{x}) > 0$ only when $|\boldsymbol{x} - \boldsymbol{x}_i|$ is sufficiently small.
2. Neighbouring neurons in the array are tuned to similar signals, i.e. if $|\boldsymbol{r}_i - \boldsymbol{r}_j|$ is small, than $|\boldsymbol{x}_i - \boldsymbol{x}_j|$ is small.
3. External 'distance' is monotonically related to internal 'distance', i.e. if $|\boldsymbol{r}_i - \boldsymbol{r}_j| < |\boldsymbol{r}_i - \boldsymbol{r}_k|$, than $|\boldsymbol{x}_i - \boldsymbol{x}_j| < |\boldsymbol{x}_i - \boldsymbol{x}_k|$.

How to Learn A Topologically Correct Map. It turns out that in order to achieve these objectives one needs learning rules where neurons effectively enter a competition for having input signals $\boldsymbol{x}_i$ 'allocated' to them, whereby neighbouring neurons stimulate one another to develop similar synaptic interactions and distant neurons are prevented from developing similar interactions. Let us try to construct the simplest such learning rule.



Initially, before learning has taken place, input signals just evoke random firing patterns in the neuronal array. After learning we wish to see localised firing patterns, as shown above. Our equations take their simplest form in the case where the input signals are normalised, so we define $(x_1, x_2) \in [-1, 1]^2$ and add a dummy variable $x_3 = \sqrt{1 - x_1^2 - x_2^2}$, together with a dummy synaptic interaction m_{i3} for every neuron. If we write the synapses of neuron i as $\boldsymbol{m}_i = (m_{i1}, m_{i2}, m_{i3})$, and normalise them according to $|\boldsymbol{m}_i| = 1$, we simply get

$$y(\boldsymbol{x}) = g[\boldsymbol{m}_i \cdot \boldsymbol{x}], \qquad |\boldsymbol{m}_i| = |\boldsymbol{x}| = 1 \tag{2.13}$$

Learning implies rotating the vectors $\boldsymbol{m}_i$. By construction (since $g[z]$ is a monotonically increasing function), a given neuron is triggered most when $\boldsymbol{m}_i \cdot \boldsymbol{x}$ is maximal, which is for the input signal $\boldsymbol{x} = \boldsymbol{m}_i$. This has two important implications: one can now interpret $\boldsymbol{m}_i$ as the input signal to which neuron i is 'tuned', and 'tuning' neuron i to a given signal $\boldsymbol{x}$ is found to be equivalent to rotating $\boldsymbol{m}_i$ towards $\boldsymbol{x}$. As a result one finds that a learning rule with the desired effect is, starting from random (normalised) synaptic vectors $\boldsymbol{m}_i$, to iterate the following recipe until a (more or less) stable situation is reached:

step 1: pick a data point $\boldsymbol{x}$ with $|\boldsymbol{x}| = 1$ at random

step 2: find the most active neuron, i.e. the i such that $|\boldsymbol{m}_i \cdot \boldsymbol{x}| > |\boldsymbol{m}_i \cdot \boldsymbol{x}|$ for all $j \neq i$

step 3: rotate the synaptic vector of neuron i *and its neighbours in the array* towards $\boldsymbol{x}$, rotate the synaptic vector of all other neurons slightly away from $\boldsymbol{x}$

step 4: return to 1

The neuron that was already the one most responsive to the signal $\boldsymbol{x}$ will be made even more so (together with its neighbours); the other neurons are made less responsive to $\boldsymbol{x}$. In biology the above is achieved via Hebbian-type learning rules, and by having short-range excitatory synapses which ensure that neighbouring neurons tend to be simultaneously active, and 'team up' in the synaptic adaptation.

We note that in the above formulation the normalisation of synaptic vectors and input vectors was purely introduced in order to retain the connection with the biological picture of firing neurons, via expressions like (2.13). For the purpose of computation and learning in synthetic systems, on the other hand, we can move away from the firing of neurons and work directly in terms of the $\{\boldsymbol{m}_i\}$ only. Henceforth we will define $\boldsymbol{m}_i$ simply as the signal in input space to which neuron i is tuned; we forget about how this is actually realised, and we forget about normalisation.

Visualization of the Learning Process: Fishing Nets. The above formulation of the self-organizing maps in terms of the $\{\boldsymbol{m}_i\}$ has brought us (deliberately) very close the VQ and SVQ algorithms. However, there is an important difference, which is the insistence that the various code-book vectors $\boldsymbol{m}_i$ are no longer inter-changable (in VQ and SVQ we only cared about the overall distribution of code-book vectors in input space), but now each $\boldsymbol{m}_i$ is associated with a specific location $\boldsymbol{r}_i$ in an array, with the condition that those $\{i, j\}$ which belong to nearby locations $\{\boldsymbol{r}_i, \boldsymbol{r}_j\}$ in the array must have code-book vectors $\{\boldsymbol{m}_i, \boldsymbol{m}_j\}$ which are very similar. Hence, simply drawing the code-book vectors in input space (as we did for VQ and SVQ) is no longer adequate, because it will not inform us about the topological features of the state.

In the case of SOM's the standard visual representation of a state $\{\boldsymbol{m}_i\}$ is the following:

- Draw each $\boldsymbol{m}_i$ as a point ('knot') in input space (as in VQ and SVQ)
- Connect with line segments all those points $\{i, j\}$ which correspond to neighbours in in the array of the $\{\boldsymbol{r}_i\}$

We then end up with a graphical representation of the synaptic structure of a network in the form of a 'fishing net', with the positions of the knots representing the signals in the world

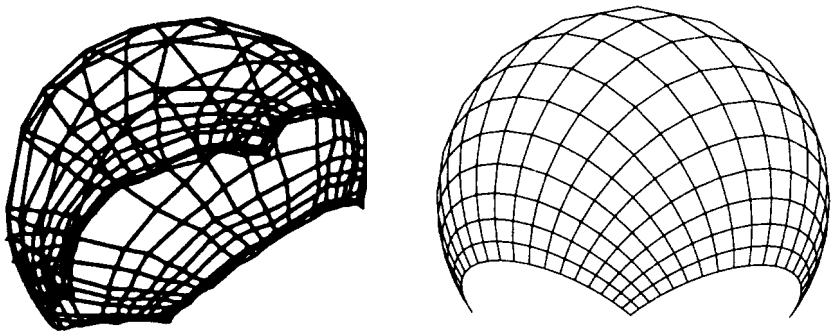


Figure 2.14: Graphical representation of the state of a self-organizing map (SOM) in the form of a 'fishing net'. Here $\boldsymbol{r}_i \in [0,1]^2$ (i.e. the neurons live in a square 2-dimensional array) and $\boldsymbol{x} \in \Re^3$, with $|\boldsymbol{x}| = 1$ (i.e. the data live on the surface of a sphere). The positions of the knots represent the signals $\boldsymbol{m}_i$ to which the neurons are 'tuned', and the cords connect the knots of neighbouring neurons in the array. Left: equilibrium result of the numerical simulation of a particular version on the SOM algorithm. Right: corresponding theoretical prediction.

to which the neurons are tuned and with the cords indicating neighbourship, see figures 2.14. The three objectives of map formation set out at the beginning of this section thereby translate into the following desired properties of the graph:

1. all knots in the graph are separated
2. all cords of the graph are similarly stretched
3. there are no regions with overlapping pieces of graph

Note that this representation is in practice useful only for two-dimensional arrays (i.e. when creating two-dimensional internal representations).

The SOM Algorithm and its Properties. The Self-Organising Map (SOM) algorithm is an abstract realisation of the processes underlying the creation of topologically correct internal representations in the higher brain regions of humans and animals. It is defined in terms of 'code-book' vectors $\boldsymbol{m}_i$, which represent the specific signals in data space to which neurons are 'tuned'. It aims to create a topologically correct internal representation of low-dimensional structure (potentially non-linear) hidden in (possibly high-dimensional) data.

Each neuron is located in a physical array, the dimension of which will become the dimension of the internal map created; the vector $\boldsymbol{r}_i$ indicates the location of neuron i. We define a neighbourhood function h_{ij} as follows:

$$h_{ij} = h\left(\frac{|\boldsymbol{r}_i - \boldsymbol{r}_j|}{\sigma}\right), \qquad \sigma > 0 \tag{2.14}$$

in which $h(z)$ is a monotonically decreasing function, with $h(0) = 1$, $h(\infty) = 0$, and with a width of order 1. For example:

$$h(z) = e^{-\frac{1}{2}z^2} : \qquad h_{ij} = e^{-\frac{1}{2}(\boldsymbol{r}_i - \boldsymbol{r}_j)^2/\sigma^2}$$

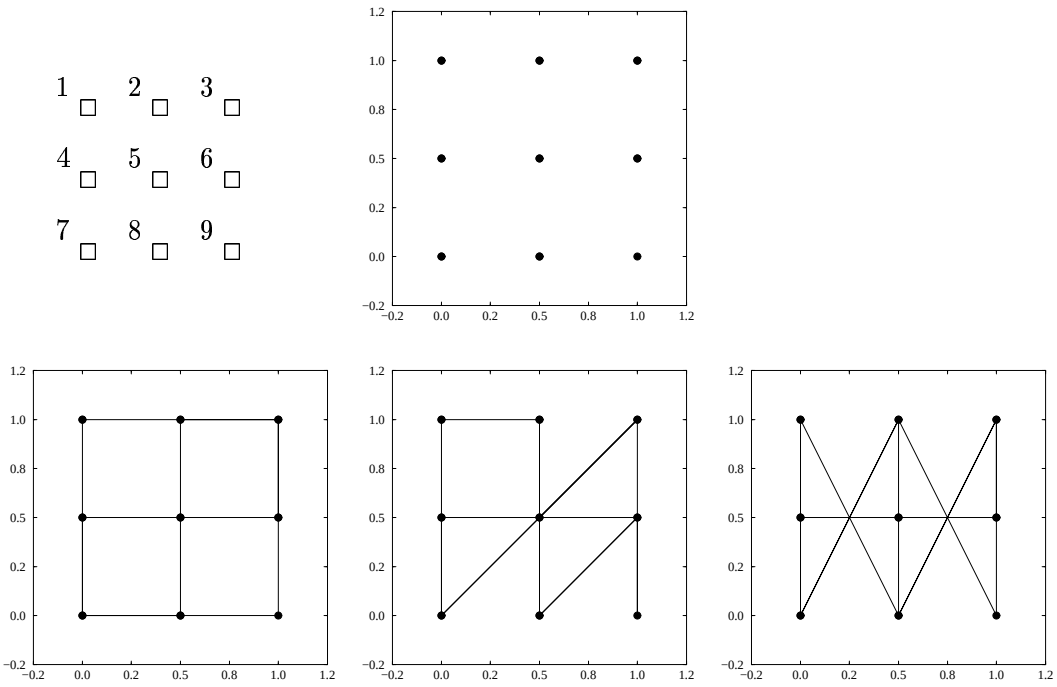


Figure 2.15: Example of the 'fishing net' representation of the state of a self-organizing map system. Upper left: the arrangement of 9 neurons in a physical array (defining the $\{\boldsymbol{r}_i\}$, and hence the notion of neighbours). Upper middle: the positioning of the nine code-book vectors $\boldsymbol{m}_i$ in data space, corresponding to the state $\{\boldsymbol{m}_1 = (0,1),\ \boldsymbol{m}_2 = (\frac{1}{2},1),\ \boldsymbol{m}_3 = (1,1),\ \boldsymbol{m}_4 = (0,\frac{1}{2}),\ \boldsymbol{m}_5 = (\frac{1}{2},\frac{1}{2}),\ \boldsymbol{m}_6 = (1,\frac{1}{2}),\ \boldsymbol{m}_7 = (0,0),\ \boldsymbol{m}_8 = (\frac{1}{2},0),\ \boldsymbol{m}_9 = (1,0)\}$. Bottom graph left: the fishing net graph corresponding to the aforementioned state. Bottom graph middle: the fishing net graph corresponding to the state one obtains by exchanging code-book vectors $\boldsymbol{m}_3$ and $\boldsymbol{m}_8$. Bottom graph right: the fishing net graph corresponding to the state one obtains by exchanging code-book vectors $\boldsymbol{m}_2$ and $\boldsymbol{m}_8$. Note that all cases would give rise to exactly the same distribution of code-book vectors, so that the incorrect topology of the last two states can indeed only be inferred from the fishing net graph.

$$h(z) = \theta[1-z]: \qquad \begin{cases} h_{ij} = 1 & \text{if } |\boldsymbol{r}_i - \boldsymbol{r}_j| < \sigma \\ h_{ij} = 0 & \text{if } |\boldsymbol{r}_i - \boldsymbol{r}_j| > \sigma \end{cases}$$

in which $\theta[z]$ is the step function ($\theta[z>0]=1$, $\theta[z<0]=0$). The SOM algorithm can now be defined as follows. First we initialize the N code-book vectors $\boldsymbol{m}_i \in \Re^n$ randomly Then we iterate the following stochastic process:

step 1: pick a data point $\boldsymbol{x}$ at random, according to the probability density $p(\boldsymbol{x})$

step 2: find the Voronoi cell containing $\boldsymbol{x}$, i.e. find i such that $|\boldsymbol{x}-\boldsymbol{m}_i| < |\boldsymbol{x}-\boldsymbol{m}_j|\ \forall j \neq i$

step 3: move *all* $\boldsymbol{m}_j$ towards $\boldsymbol{x}$: $\boldsymbol{m}_j \to \boldsymbol{m}_j + \eta(\boldsymbol{x}-\boldsymbol{m}_j)F_j[\boldsymbol{x},\{\boldsymbol{m}(\ell)\}]$, where $F_j[\ldots] = h_{ij}$

step 4: return to 1

The learning rate $0 < \eta \ll 1$ controls, as always, the overall magnitude of the changes.

The SOM algorithm can be seen to have the following general properties:

(*i*) All code-book vectors are moved towards $\boldsymbol{x}$ at every iteration step, but those which are closest to the code-book vector in whose Voronoi cell the data point is (i.e. that of the neuron which is triggered most) are moved most.

(*ii*) Unless we reduce η during the process, the code-book vectors will continue to move stochastically, although the *density* of code-book vectors in any given region should become stationary.

(*iii*) The properties of the $\{h_{ij}\}$ guarantee that neighbouring neurons 'team up', and develop similar code-book vectors.

(*iv*) SOM has one parameter more than VQ. This extra parameter, σ, defines a characteristic distance in the original physical array of the neurons: code-book vectors with $|\boldsymbol{r}_i - \boldsymbol{r}_j| < \sigma$ will be the ones to feel one another's influence.

(*v*) $\lim_{\sigma \to 0}$ SOM = VQ (the proof is trivial). Hence we can see the SOM as VQ plus enforced spatial continuity.

We observe that the SOM is again of the general form (2.4), hence we can apply our results on time-dependent learning rates also to the SOM. The relation between the three processes discussed so far is

$$\lim_{\beta \to \infty} \text{SVQ} \;=\; \text{VQ} \;=\; \lim_{\sigma \to 0} \text{SOM}$$

Applications of the SOM include sensor fusion (in robotics), data visualisation and data-base mining (finding hidden regularities in messy data), data preprocessing, medical diagnostics (finding causal relationships in medical data), etc.

It will be clear that the determination of the right value for σ is delicate. If σ is too small, the system will behave like VQ, and the correct topology will not emerge (or be correct only locally). If σ is too large, all neurons will drag one another along, and all code-book vectors will become identical (see also the simulation examples below). The new parameter σ also introduces boundary effects. Since neurons 'drag along' one another's code-book vectors, it will be clear that those neurons which are at the boundary of the array will feel an effective force dragging their code-book vectors towards those of the inner region of the array (since there are no neurons pulling from the outside). This shows itself in a tendency for the boundaries of the 'fishing net' to keep a certain σ-dependent distance from the boundaries of the data region. In practice one therefore chooses a time-dependent σ, similar to the time-dependent learning rate, i.e.

$$\eta(t) = \frac{\eta(0)}{1 + t/\tau_\eta} \qquad \sigma(t) = \frac{\sigma(0)}{1 + t/\tau_\sigma}$$

Examples of SOM in Action. The figures below illustrate the functioning of the SOM algorithm for a number of simple examples with $n = 2$ (i.e. where one has data points $\boldsymbol{x} = (x_1, x_2)$ in a plane) and a 10×10 array of neurons (i.e. a population of 100 code-book vectors), but for different choices of the distance parameter σ and the time-dependence of the learning rate.

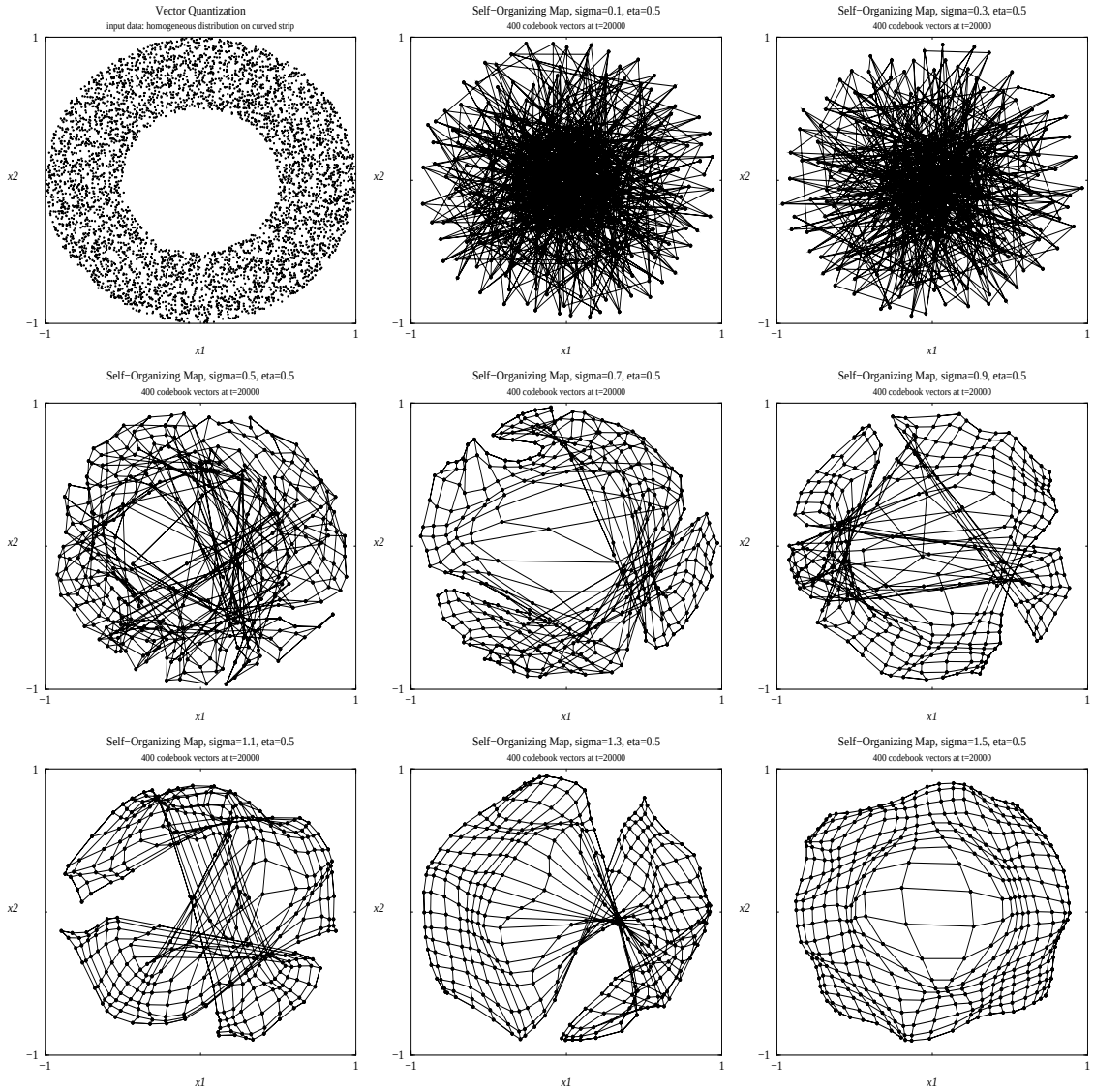


Figure 2.16: Example: numerical simulation of a Self-Organising Map, with $n = 2$, $\eta = 0.5$ and $N = 100$. Top left graph: data distribution $p(\boldsymbol{x})$, homogeneously distributed over the strip $\frac{1}{2} < |\boldsymbol{x}| < 1$. Other eight graphs: system state after 20000 iteration steps, for different values of the range parameter σ ($\sigma \in \{0.1,\ 0.3,\ 0.5,\ 0.7,\ 0.9,\ 1.1,\ 1.3,\ 1.5\}$, from top middle to bottom right). One clearly observes the two effects of loosing the correct topology for σ too small, with only local correctness for intermediate σ, and of the code-book vectors of boundary neurons keeping a σ-dependent distance from the data boundary.

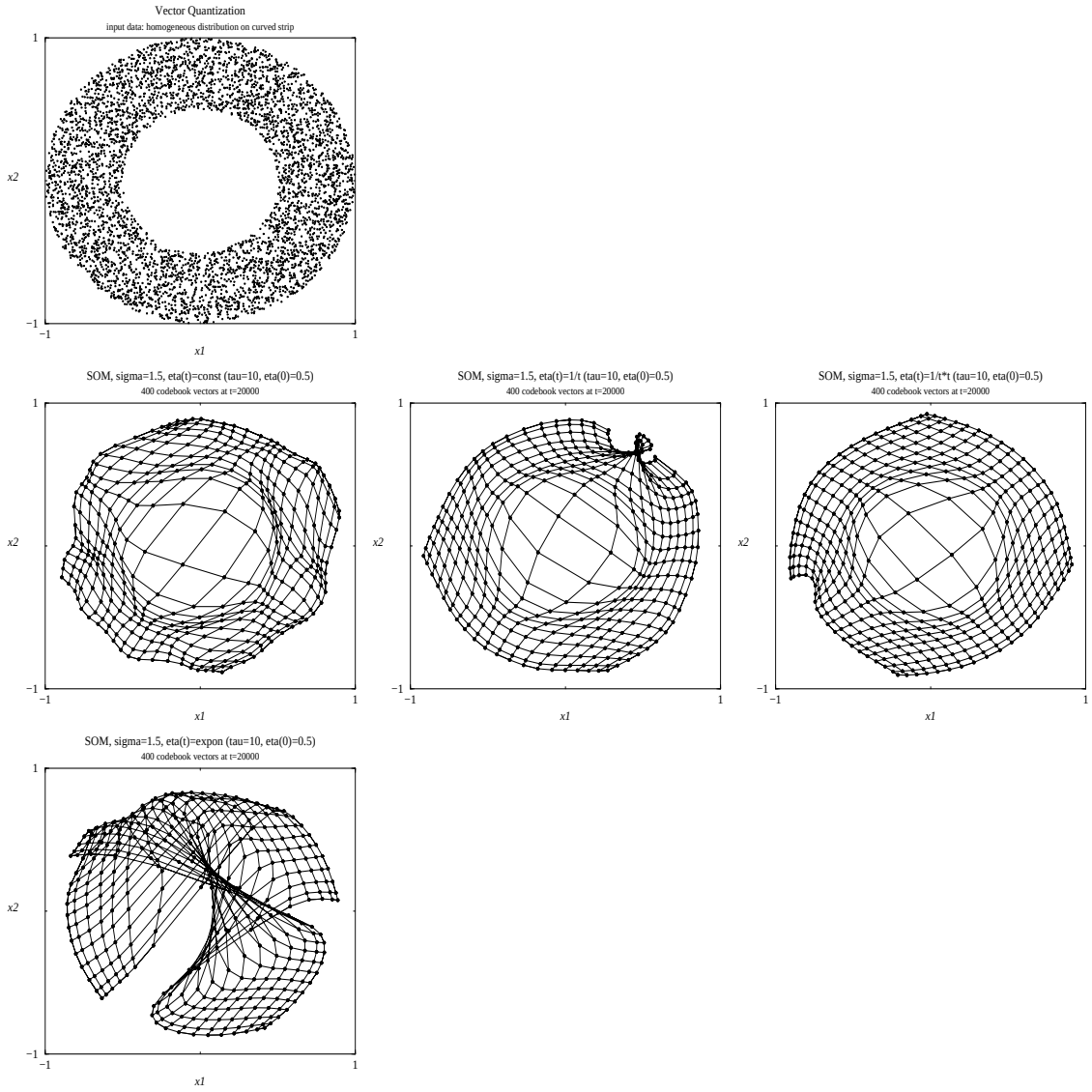


Figure 2.17: Example: numerical simulation of a Self-Organising Map, with $n = 2$, $\sigma = 1.5$, $\eta(0) = 0.5$, and $N = 100$. Top left graph: data distribution $p(\boldsymbol{x})$, homogeneously distributed over the strip $\frac{1}{2} < |\boldsymbol{x}| < 1$. Other eight graphs: system state after 20000 iteration steps, for different choices of the time-dependence of the learning rate η. Second row left: $\eta = 0.5$ (constant). Second row, middle: $\eta(t) = \eta(0)/[1 + t/\tau]$, with $\tau = 10$. Second row, right: $\eta(t) = \eta(0)/[1 + t^2/\tau^2]$, with $\tau = 10$. Bottom: $\eta(t) = \eta(0)\exp[-t/\tau]$, with $\tau = 10$. One clearly observes that the system can get 'stuck' in a suboptimal state, where the topology is only locally correct.

Chapter 3

Bayesian Techniques in Supervised Learning

In this chapter we discuss the Bayesian approach to supervised learning from examples. Its introduction in the 1990's has been a very welcome development, in that it contributed significantly to converting synthetic neural computation, especially learning in multi-layer networks, from a collection of interesting but ad hoc algorithms ('invent a rule, play with parameters, draw performance graphs, and hope for the best') to a sound, well grounded, and systematic scientific procedure. It also contributed to the area becoming more mathematical in nature.

3.1 Preliminaries and Introduction

Supervised Learning from Examples. Let us first set the scene and define our terminology. In supervised learning from examples we are confronted with a 'task', defined by a collection of questions $\boldsymbol{\xi}^\mu$, drawn randomly from some set $D \subseteq \Re^N$ (with probabilities, or probability density, $p(\boldsymbol{\xi})$, with corresponding answers t^μ:

$$\text{the task :} \qquad \begin{array}{lll} \text{'questions'} : & \{\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p\}, & \boldsymbol{\xi}^\mu \in D \subseteq \Re^N \\ \text{'answers'} : & \{t^1, \ldots, t^p\}, & t^\mu \in \Re \end{array}$$

This task, assumed to have been generated by a 'teacher', is to be learned by a 'student' (the neural network). The student executes a parametrized operation $S : \Re^N \to \Re$, where $S(\boldsymbol{\xi}) = f(\boldsymbol{\xi}; \boldsymbol{w})$. The parameters $\boldsymbol{w}$ determine the details of the operation, and thus represent the 'program'; in multi-layer neural networks they would be the synaptic weights and the neuronal thresholds. If the outputs (or 'targets') are binary, e.g. $t^\mu \in \{0, 1\}$ or $t^\mu \in \{-1, 1\}$ we would call the task *binary classification*. If the outputs can truly take values from a continuous set, we would speak about *regression*.

It is assumed that the data (or 'answers') were not assigned randomly by the teacher (otherwise there would be no point in learning), but that they were generated according to some function $T : \Re^N \to \Re$. The student has no direct information on the function T (the teacher is a 'black box', or 'oracle'), but has to infer T from the p input-output pairs $(\boldsymbol{\xi}^\mu, t^\mu)$; this is the objective of the learning process. The problem faced by the student is made more difficult by the fact (which is quite realistic in the real world) that the data are not perfect,

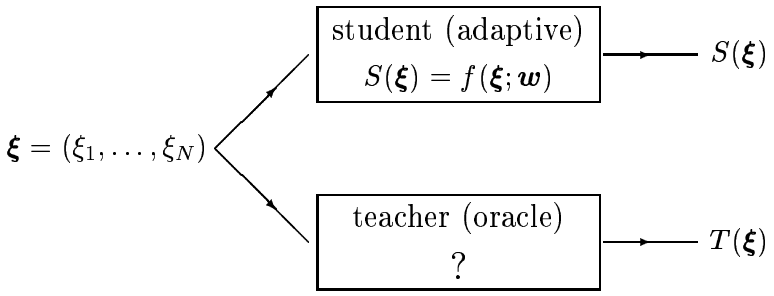


Figure 3.1: The 'teacher-student' scenario of on-line supervised learning. The student can be any type of parametrised operation (not necessarily neural or neural-inspired).

i.e. the teacher can make mistakes, or is subject to noise. If, for simplicity, we assume that the inaccuracy or noise is independent for the different outputs (i.e. the teacher is sloppy, rather than consistently using the wrong rule), we have

$$\begin{array}{lll} \text{binary classification :} & \text{Prob}[t^\mu = T(\boldsymbol{\xi}^\mu)] = 1 - \epsilon & 0 \leq \epsilon \leq 1 \\ \text{regression :} & t^\mu = T(\boldsymbol{\xi}^\mu) + z_\mu & z_\mu \in \Re \quad \text{drawn randomly according to } P(z) \end{array} \tag{3.1}$$

In the case of binary classification, the parameter ϵ measures the amount of noise in the data, with $\epsilon = 0$ corresponding to perfect data. In the case of regression we may without loss of generality assume $\langle z \rangle = \int dz\ zP(z) = 0$ (since a non-zero average would be equivalent to a structural modification of the underlying rule T, and could be absorbed in its definition); hence the amount of noise in the data is here measured by the variance of $P(z)$, with perfect data given by $P[z] = \delta[z]$.

Due to the possibility of data noise and the difference between our finite sample $\{\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p\}$ and the full set of possible questions D, there are several performance measures one could define. Here we concentrate on two: the *training error* E_t (measuring how well the student reproduces the given answers t^μ), and the *generalization error* E_g (measuring how well the student has learned the underlying rule T):

$$E_t = \frac{1}{p} \sum_{\mu=1}^{p} \mathcal{E}[t^\mu, S(\boldsymbol{\xi}^\mu)] \qquad E_g = \int_D d\boldsymbol{\xi}\ p(\boldsymbol{\xi})\ \mathcal{E}[T(\boldsymbol{\xi}), S(\boldsymbol{\xi})]$$

where $\mathcal{E}[t, s]$ is some, as yet unspecified, measure of the difference between t and s, such as $\mathcal{E}[t, s] = |t - s|^\gamma$ (with $\gamma > 0$). Given the dependence of the student's operation on the parameters $\boldsymbol{w}$, via $S(\boldsymbol{\xi}) = f(\boldsymbol{\xi}, \boldsymbol{w})$, this becomes

$$E_t[\boldsymbol{w}] = \frac{1}{p} \sum_{\mu=1}^{p} \mathcal{E}[t^\mu, f(\boldsymbol{\xi}^\mu; \boldsymbol{w})] \qquad E_g[\boldsymbol{w}] = \int_D d\boldsymbol{\xi}\ p(\boldsymbol{\xi})\ \mathcal{E}[T(\boldsymbol{\xi}), f(\boldsymbol{\xi}; \boldsymbol{w})] \tag{3.2}$$

There are three important observations to make at this stage:

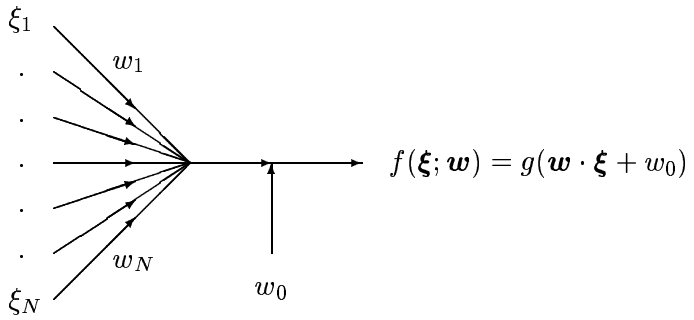
- The above two error measures $E_t[\boldsymbol{w}]$ and $E_g[\boldsymbol{w}]$ differ in two aspects: (i) E_t is an average only over the p available questions $\{\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p\}$, whereas E_g involves *all* questions in D. (ii) E_t checks performance relative to the (noisy) answers $\{t^\mu\}$ given, whereas E_g checks performance relative to the correct underlying rule T.

- The real objective of learning is to minimise $E_g[\boldsymbol{w}]$, via suitable modification of the parameters $\boldsymbol{w}$, since we ultimately wish to use our system for predicting the answers to novel questions, rather than just parrot the answers to those of the training stage.
- In the learning process, however, we have no access to $E_g[\boldsymbol{w}]$, since we know neither the full set D nor the true rule T; we only have the data $\{(\boldsymbol{\xi}^1, t^1), \ldots, (\boldsymbol{\xi}^p, t^p)\}$ ...

Many of the obstacles and problems in neural network learning in the past (i.e. before, say, 1990) had their origin in the fact that one wished to minimize one object (E_g), but one could only measure the other (E_t), and one tried to get away (out of necessity, it seemed) with pretending that the differences between the two were not too important.

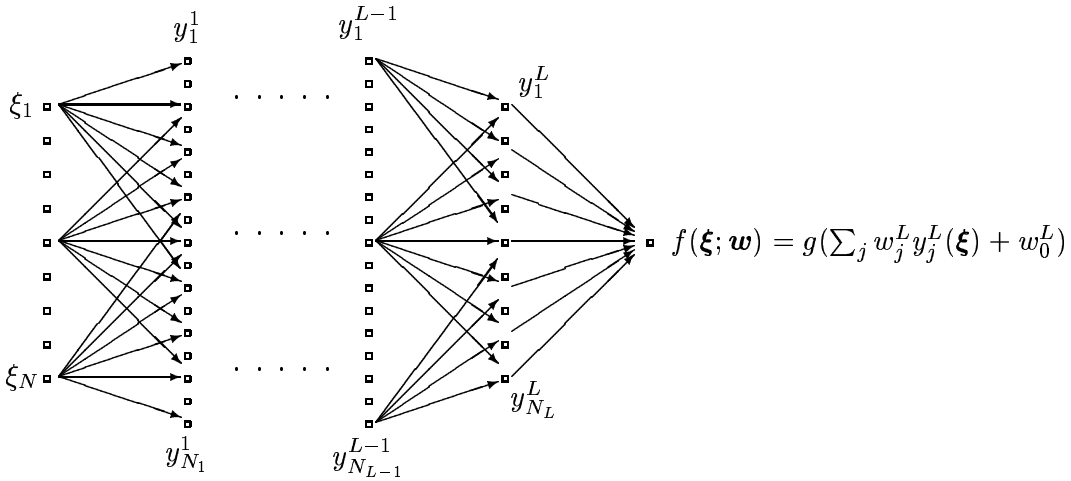
Conventional Network Types. Let us next briefly review the most common types of neural network $S(\boldsymbol{\xi}) = f(\boldsymbol{\xi}, \boldsymbol{w})$ which are being used in practice for the above purpose. We first consider regression tasks, where the outputs are supposed to be continuous:

- Single layer neural networks (perceptrons):



These are basically single neurons, carrying out a (soft) linear separation in the space D of the input vectors $\boldsymbol{\xi}$, of the form $S(\boldsymbol{\xi}) = g(\boldsymbol{w} \cdot \boldsymbol{\xi} + w_0)$. Here $g(z)$ is a monotonically increasing sigmoidal function, which saturates for $z \to \pm\infty$. The modifiable parameters (i.e. the 'program') are the synaptic weights $(w_1, \ldots, w_N)$ and the threshold w_0.

- Multi-layer neural networks (MLP's):

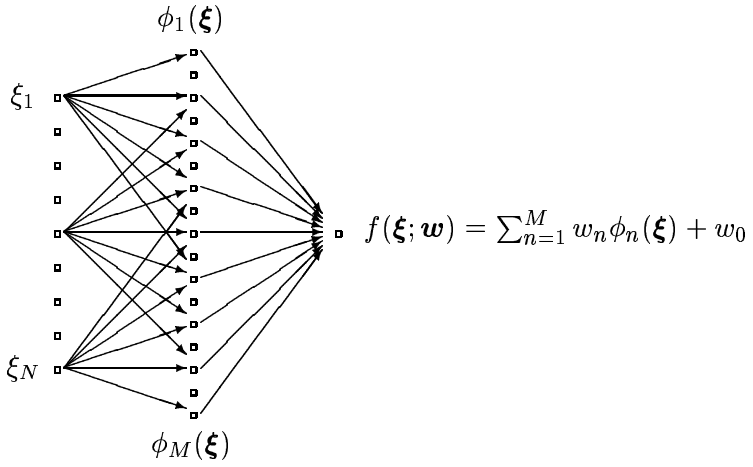


These are feed-forward processing networks of neurons of the type described above (soft linear separators), with monotonically increasing and saturating sigmoidal functions $g(z)$. The overall output is calculated iteratively from the outputs of the preceding layer, according to

$$y_i^1 = g[\sum_{j=1}^{N} w_{ij}^0 \xi_j + w_0^0] \qquad y_i^{\ell+1} = g[\sum_{j=1}^{N_\ell} w_{ij}^\ell y_j^\ell + w_0^\ell] \qquad f(\boldsymbol{\xi}; \boldsymbol{w}) = g[\sum_{j=1}^{N_L} w_j^L y_j^L + w_0^L] \quad (3.3)$$

and the modifiable parameters (i.e. the 'program') are the synaptic weights and thresholds of all L layers, i.e. $\boldsymbol{w} = \{w_{ij}^\ell,\ w_i^L,\ w_0^\ell\}$. Given a sufficient number of layers and units within each layer (dependent on the problem to be solved), these networks have been proven to be 'universal approximators': any sufficiently well-behaved continuous function $f : \Re^N \to \Re$ can be approximated to arbitrary accuracy by MLP's (we will not discuss the precise conditions or the proof here).

- Radial basis function networks (RBF):



The motivation behind these networks is also the principle underlying the previous multilayer networks: one converts a non-linearly separable problem into a linearly separable one, by suitable pre-processing. In the previous network the final neuron does the linear separation, and the preceding L layers do the pre-processing. In radial basis function networks one chooses specific functions $\phi_n(\boldsymbol{\xi})$ to do a specific type of pre-processing:

$$f(\boldsymbol{\xi}; \boldsymbol{w}) = \sum_{n=1}^{M} w_n \phi_n(\boldsymbol{\xi}) + w_0 \qquad \phi_n(\boldsymbol{\xi}) = \phi(|\boldsymbol{\xi} - \boldsymbol{x}^n|) \quad (3.4)$$

The $\phi_n(\boldsymbol{\xi})$ (the 'radial basis functions') are allowed to depend only on the distance $|\boldsymbol{\xi} - \boldsymbol{x}^n|$ between the input vector and a code-book vector $\boldsymbol{x}^n$. For the remaining scalar function ϕ to be specified several choices have been proposed; which one to pick is crucial for performance,

but (unfortunately) highly dependent on the problem at hand[1]. Popular choices are:

$$\text{localised basis functions :} \qquad \begin{array}{l} \phi(u) = e^{-u^2/2\sigma^2} \\ \phi(u) = (u^2+\sigma^2)^{-\alpha} \quad (\alpha > 0) \end{array}$$

$$\text{non}-\text{localised basis functions :} \qquad \begin{array}{l} \phi(u) = u \\ \phi(u) = (u^2+\sigma^2)^{\beta} \quad (0 < \beta < 1) \end{array}$$

The modifiable parameters are the weights $\boldsymbol{w}$ (and sometimes one also allows the code-book vectors $\boldsymbol{x}^n$ to adapt). Note, finally, that in RBF networks the output does not undergo a non-linear (sigmoidal) operation, hence the output depends linearly on the parameters $\boldsymbol{w}$.

Model Complexity and Overfitting. Systems such as those discussed above have been used (and are still being used) extensively as 'student networks' $S(\boldsymbol{\xi}) = f(\boldsymbol{\xi}; \boldsymbol{w})$ in the sense of figure 3.1. The strategy for finding the best parameters $\boldsymbol{w}$ was, in the early stages of neural computation, based on performing gradient descent on the surface defined by the training error $E_t[\boldsymbol{w}]$ as given in (3.2), with a quadratic error measure $\mathcal{E}[z] = \frac{1}{2}z^2$:

$$\frac{d}{dt}\boldsymbol{w} = -\eta \boldsymbol{\nabla}_{\boldsymbol{w}} E_t[\boldsymbol{w}] \qquad E_t[\boldsymbol{w}] = \frac{1}{2p}\sum_{\mu=1}^{p}[t^\mu - f(\boldsymbol{\xi}^\mu; \boldsymbol{w})]^2 \tag{3.5}$$

with a learning rate $\eta > 0$. The objective is to find $\boldsymbol{w}^\star$ such that $\min_{\boldsymbol{w}} E_t[\boldsymbol{w}] = E_t[\boldsymbol{w}^\star]$; the above process would at least be guaranteed to lead us to a local minimum of $E_t[\boldsymbol{w}]$, since

$$\frac{d}{dt}E_t[\boldsymbol{w}] = \boldsymbol{\nabla}_{\boldsymbol{w}} E_t[\boldsymbol{w}] \cdot \frac{d}{dt}\boldsymbol{w} = -\eta \left[\boldsymbol{\nabla} E_t[\boldsymbol{w}]\right]^2 \leq 0$$

$$\frac{d}{dt}\boldsymbol{w} = \boldsymbol{0} \quad \Longleftrightarrow \quad \boldsymbol{\nabla}_{\boldsymbol{w}} E_t[\boldsymbol{w}] = \boldsymbol{0}$$

One problem with this approach has been clear from the beginning: one can end in a local rather than a global minimum. A second problem was appreciated only some ten years later: gradient descent is not the optimal local process to find minima on an error surface (see the chapter on information geometry in the Information Theory course). Here we concentrate on a third problem, called inappropriate model complexity, or 'overfitting'.

This problem is best explained using a very simple example. Suppose we have a task $T : \Re \to \Re$ defined by a teacher which carries out some function $g(x)$. The student $S : \Re \to \Re$ tries to emulate the teacher by some parametrized function $f(x; \boldsymbol{w})$, for which we choose a finite polynomial (i.e. truncated Taylor expansion):

$$f(x; \boldsymbol{w}) = w_0 + w_1 x + \ldots + w_M x^M \qquad \text{or} \qquad f(x; \boldsymbol{w}) = \boldsymbol{w} \cdot \boldsymbol{\phi}(x), \quad \phi_\ell(x) = x^\ell \tag{3.6}$$

Learning means adapting $\boldsymbol{w} = \{w_0, \ldots, w_M\}$. The data consist of examples of p inputs x_μ and corresponding outputs t^μ (with $p \geq M+1$), however, the teacher is not perfect (there is a

[1] This illustrates how in RBF networks the learning problem is, in a way, simply moved under the carpet, for how do we now choose the function $\phi(u)$, the code-book vectors $\boldsymbol{x}^n$, or the number M ? And under which conditions can we be sure that the ad-hoc restriction to RBF units does not render the problem unlearnable ? At least in image processing, however, one can set up a rigorous framework, and show that all non-pathological images can be decomposed in terms of a bundle of local expansions (derivatives of Gaussian functions), which can (in turn) be constructed by adding and subtracting Gaussian functions of the type in (3.4).

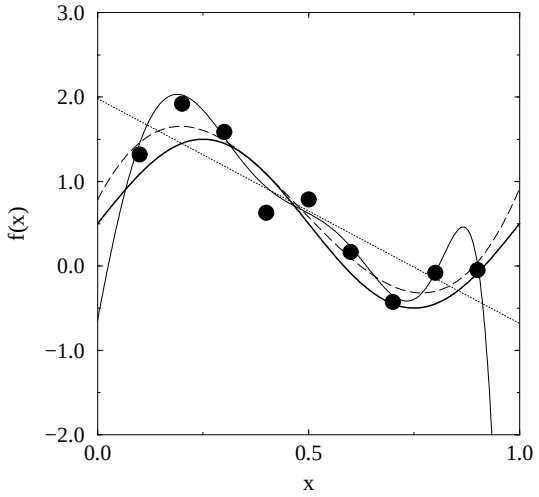


Figure 3.2: Example of a simple learning process where a system 'learns' via gradient descent on a quadratic error measure how to approximate a function $g(x) = \frac{1}{2} + \sin(2\pi x)$ (thick solid line) with by an order-M polynomial $f(x; \boldsymbol{w}) = w_0 + w_1 x + \ldots + w_M x^M$, on the basis of nine noisy sample points of this function (circles). The resulting polynomials are shown for $M - 1$ (dotted), $M = 3$ (dashed) and $M = 9$ (thin solid).

noise source in the data generation), so $t_\mu = g(x_\mu) + z_\mu$ where the z_μ are drawn independently according to a zero-average distribution $P(z)$. The training error is given by

$$E_t[\boldsymbol{w}] = \frac{1}{2} \sum_{\mu=1}^{p} [t^\mu - \boldsymbol{w} \cdot \boldsymbol{\phi}(x_\mu)]^2$$

Gradient descent learning then boils down to

$$\frac{1}{\eta} \frac{d}{dt} w_i = \sum_{\mu=1}^{p} [t^\mu - \boldsymbol{w} \cdot \boldsymbol{\phi}(x_\mu)] \, \phi_i(x_\mu) = \sum_{\mu=1}^{p} t^\mu \phi_i(x_\mu) - \sum_{j=1}^{M} \left[\sum_{\mu=1}^{p} \phi_i(x_\mu) \phi_j(x_\mu) \right] w_j$$

This can be written as $\eta^{-1} \frac{d}{dt} \boldsymbol{w} = \boldsymbol{u} - \boldsymbol{A}\boldsymbol{w}$, with $u_i = \sum_{\mu=1}^{p} t^\mu (x_\mu)^i$ and $A_{ij} = \sum_{\mu=1}^{p} (x_\mu)^{i+j}$, with solution (provided $\boldsymbol{A}$ is invertible)

$$\boldsymbol{w}(t) = \boldsymbol{A}^{-1}\boldsymbol{u} + e^{-\eta t \boldsymbol{A}} [\boldsymbol{w}(0) - \boldsymbol{A}^{-1}\boldsymbol{u}]$$

The final outcome of the learning process is the weight vector $\boldsymbol{w}^\star = \boldsymbol{w}(\infty) = \boldsymbol{A}^{-1}\boldsymbol{u}$. Insertion into (3.6) gives the associated function extracted by the student from the data. In figure 3.2 we show the result for the example $g(x) = \frac{1}{2} + \sin(2\pi x)$, with $0 \leq x \leq 1$. For small M (e.g. $M = 1$ in the figure), the complexity of the student is insufficient for reducing either training or generalization error. Increasing the complexity of the student (i.e. M) initially improves both (e.g. $M = 3$ in the figure). However, for large M the system increasingly succeeds in reducing the training error by getting the exact locations of the data points right (including the noise), with the side effect of a deterioration of generalization (i.e. $M = 9$ in the figure). Plotting the values of E_t and E_g as functions of time for a student with excessive complexity (M too large) would lead to the behaviour illustrated in figure 3.3.

The most commonly used solutions to the overfitting problem are the following:

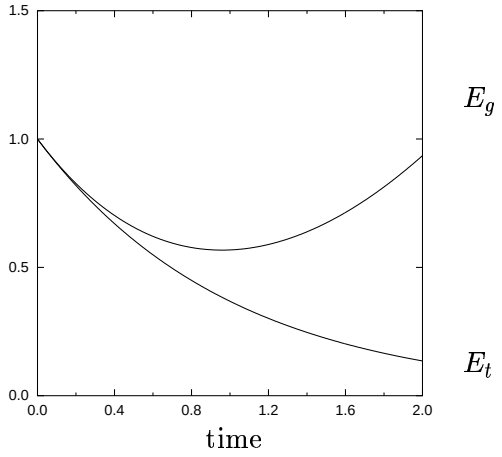


Figure 3.3: Evolution of training and generalization errors during a learning process defined as gradient descent on an error surface, with noisy data, in the regime where the student is too complex ('overfitting') and starts reproducing also the noise in the data.

(i) Cross-validation.

The idea is to split the p available data points into two sub-sets: a training set $\{(\boldsymbol{\xi}^1, t^1), \ldots, (\boldsymbol{\xi}^k, t^k)\}$ and a validation set $\{(\boldsymbol{\xi}^{k+1}, t^{k+1}), \ldots, (\boldsymbol{\xi}^p, t^p)\}$. One then uses the training set for learning, and the validation set to estimate the generalization error:

$$E_t[\boldsymbol{w}] = \frac{1}{k} \sum_{\mu=1}^{k} \mathcal{E}[t^\mu, f(\boldsymbol{\xi}^\mu; \boldsymbol{w})] \qquad E_g[\boldsymbol{w}] \approx \frac{1}{p-k} \sum_{\mu=k+1}^{p} \mathcal{E}[t^\mu, f(\boldsymbol{\xi}^\mu; \boldsymbol{w})] \tag{3.7}$$

This latter estimate can be used in two possible ways:

(a) Controlling model complexity (or: finding optimal complexity).

In terms of the example of figure 3.2: start learning by minimizing the training error E_t with a small value of M, measure the estimate of the generalization error in (3.7), and repeat this procedure for increased values of M until E_g is seen to increase.

(b) Controlling learning times (or: 'early stopping').

In terms of figure 3.2 (see also figure 3.3): learn by minimizing the training error E_t with a large value of M (a complex student) and monitor the estimate of the generalization error in (3.7) as a function of time. Terminate learning as soon as E_g is seen to increase.

(ii) Regularization.

This method is based on the assumption that both the function to be learned and the basis functions of the student are in principle smooth, and that any observed irregularity and discontinuity of data must have been due to noise. Since the only way for the student to reproduce irregular or discontinuous functions is by having large (possibly diverging) components of the parameter vector $\boldsymbol{w}$, the idea behind regularization is to prevent $\boldsymbol{w}$ from becoming too large by adding a suitable 'penalty term' to the function

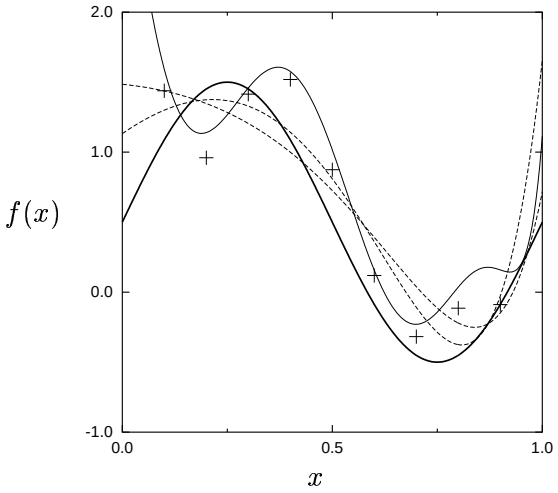


Figure 3.4: Regularization with a quadratic penalty term, as in eqn (3.8), in the example of figure 3.2. Thick solid line: the function to be learned. Crosses: nine noisy sample points of this function. Thin solid line: approximation by order-9 polynomial, without regularization. Dashed lines: similar, but with regularization term ($\lambda = 0.001$ and $\lambda = 0.02$, respectively, from bottom to top at the left of the figure). The $\lambda = 0.001$ curve improves generalization compared to the un-regularized one (i.e. it resembles the true function more). However, for $\lambda = 0.02$ we observe over-regularization: the approximation is becoming too smooth.

to be minimized, such as:

$$\frac{d}{dt}\boldsymbol{w} = -\eta \boldsymbol{\nabla}_{\boldsymbol{w}} \left\{ E_t[\boldsymbol{w}] + \frac{1}{2}\lambda \boldsymbol{w}^2 \right\} \qquad E_t[\boldsymbol{w}] = \frac{1}{2p}\sum_{\mu=1}^{p} [t^\mu - f(\boldsymbol{\xi}^\mu; \boldsymbol{w})]^2 \tag{3.8}$$

Neither cross-validation nor regularization are ideal. In the first case we waste data and CPU time which could have been used for learning. In the second we must tune or guess the form of the penalty term (and its parameter λ); see also figure 3.4. In either case we cannot be sure that we arrive at the lowest possible generalization error. The origin of the problem is clear: we continue to minimize an object which we can easily measure (E_t) which differs from the object which we would really like to minimize (E_g), but which we cannot measure.

3.2 Bayesian Learning of Network Weights

Ideas, Definitions and Benefits. The Bayesian approach deals in a systematic and exact way with the general problem of learning in neural networks and other parametrized information processing systems of the general form $S(\boldsymbol{\xi}) = f(\boldsymbol{\xi}; \boldsymbol{w}) +$ noise, given the observation of (possibly noisy) data $D = \{(\boldsymbol{\xi}^1, t^1), \ldots, (\boldsymbol{\xi}^p, t^p)\}$. In its simplest form, i.e. for a given model, it is based on the following three ideas:

- Consider not just a single parameter vector $\boldsymbol{w}$, but an *ensemble* of parameter vectors, characterized by a probability distribution $p(\boldsymbol{w})$ which evolves during learning.

- Assume that the data D were generated by a system of the form $S(\boldsymbol{\xi}) = f(\boldsymbol{\xi}; \boldsymbol{w}) +$ noise. Try to calculate the probability $p(\boldsymbol{w}|D)$ of the parameter vectors, given the data.
- Use the general Bayesian relation $P(A|B)P(B) = P(B|A)P(A)$ to express the desired object $p(\boldsymbol{w}|D)$ in terms of $p(D|\boldsymbol{w})$ (the latter can be calculated quite easily).

Learning is regarded as a process during which the arrival of data reduces our uncertainty about the 'right' parameter vector $\boldsymbol{w}$ from a *prior distribution* $p(\boldsymbol{w})$ (reflecting prior knowledge or prejudices about the problem) to a *posterior distribution* $p(\boldsymbol{w}|D)$ (reflecting both prior assumptions and the data-generated evidence). Note that we can combine the general statistical relations $p(D|\boldsymbol{w})p(\boldsymbol{w}) = p(\boldsymbol{w}|D)P(D)$ and $p(D) = \int d\boldsymbol{w}\ p(\boldsymbol{w}, D)$, to obtain

$$p(\boldsymbol{w}|D) = \frac{p(D|\boldsymbol{w})p(\boldsymbol{w})}{\int d\boldsymbol{w}'\ p(\boldsymbol{w}')p(D|\boldsymbol{w}')} \tag{3.9}$$

This is the core identity.

We now put the above ideas in practice. In a nutshell, Bayesian learning works like this:

Stage 1: definitions
Define (i) the parametrized model, assumed responsible for the data, (ii) the prior distribution $p(\boldsymbol{w})$ of its parameters, and (iii) the data $D = \{(\boldsymbol{\xi}^1, t^1), \ldots, (\boldsymbol{\xi}^p, t^p)\}$.

Stage 2: model translation
Convert the model definition into a probabilistic standard form: specify the likelihood of finding output t upon presentation of input $\boldsymbol{\xi}$, given the parameters $\boldsymbol{w}$:

$$\text{model definition in standard form} \qquad p(t|\boldsymbol{\xi}, \boldsymbol{w}) \tag{3.10}$$

Stage 3: the posterior distribution
Calculate the *data* likelihood, given the model, $p(D|\boldsymbol{w}) = \prod_{\mu=1}^{p} p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w})$. From this, together with identity (3.9), follows the desired posterior parameter distribution

$$p(\boldsymbol{w}|D) = \frac{p(\boldsymbol{w}) \prod_{\mu=1}^{p} p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w})}{\int d\boldsymbol{w}'\ p(\boldsymbol{w}') \prod_{\mu=1}^{p} p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w}')} \tag{3.11}$$

Stage 4: prediction
The residual uncertainty in the parameters $\boldsymbol{w}$ generates uncertainty in subsequent data prediction. Prediction of the output t corresponding to a new input $\boldsymbol{\xi}$, given our observation of the data D and our choice of model, thus takes the probabilistic form:

$$p(t|\boldsymbol{\xi}, D) = \int d\boldsymbol{w}\ p(t|\boldsymbol{\xi}, \boldsymbol{w})p(\boldsymbol{w}|D) \tag{3.12}$$

This concludes the description of Bayesian learning for a single model; the problem has been reduced to doing integrals. The remainder of this section deals with implementation and understanding, and with the generalization of the ideas to model selection.

The Bayesian approach to learning has become very popular within a relatively short period of time. The reason for this can be appreciated by simply listing some of its main appeals and benefits (to be derived on subsequent pages in these notes):

- It provides an interpretation of regularizers and their associated parameters.
- It provides an interpretation of single-point error measures.
- There is no need for cross-validation, so all data can be used for learning.
- It allows for the selection of the optimal complexity within a given model class, and for the selection of the optimal model class from a given set of candidates.
- It provides not only predictions, but also specific confidence estimates (i.e. error bars) corresponding to these predictions.
- Traditional learning via gradient descent training error minimization, with regularization, is recovered as a particular approximation within the Bayesian framework.

Let us gain intuition on how in the Bayesian picture the arrival of data is converted into probabilistic statements about model parameters, by working out the steps (3.10,3.11) for some simple examples. We will write inputs as $\xi \in \Re$ or $\boldsymbol{\xi} \in \Re^K$, adjustable parameters as $w \in \Re$ or $\boldsymbol{w} \in \Re^N$, outputs as $t \in \Re$, and data as $D = \{(\boldsymbol{\xi}^1, t^1), \ldots, (\boldsymbol{\xi}^p, t^p)\}$.

Example 1. Our first example describes a deterministic model with one real input and one real output. Stage one in the process is the definition of model, prior parameter distribution[2] and data, for which we take:

$$t(\xi) = \tanh(w\xi) \qquad p(w) = (2\pi)^{-\frac{1}{2}} e^{-\frac{1}{2}w^2} \qquad D = \{(1, -\frac{1}{2})\}$$

(one data point). Next we convert the model into the probabilistic standard form (3.10)[3]:

$$p(t|\xi, w) = \delta[t - \tanh(w\xi)]$$

We can now calculate the posterior distribution (3.11):

$$p(w|D) = \frac{p(w)p(t^1|\xi^1, w)}{\int dw'\ p(w')p(t^1|\xi^1, w')} = \frac{\delta[-\frac{1}{2} - \tanh(w)]e^{-\frac{1}{2}w^2}}{\int dw'\ \delta[-\frac{1}{2} - \tanh(w')]e^{-\frac{1}{2}w'^2}}$$

$$= \delta[w - \tanh^{\rm inv}(-\frac{1}{2})] = \delta[w + \log\sqrt{3}] \tag{3.13}$$

Here we have used the two identities $\delta[f(w)] = |f'(f^{\rm inv}(0))|^{-1}\delta[w - f^{\rm inv}(0)]$ (see Appendix C) and $\tanh^{\rm inv}(z) = \frac{1}{2}\log[(1+z)/(1-z)]$.

Before we had observed any data our uncertainty about the parameter w was described by the Gaussian prior $p(w)$. The arrival of the data point $(1, -\frac{1}{2})$ induced a collapse of this prior to the δ-distributed posterior $p(w|D) = \delta[w + \log\sqrt{3}]$ (without any uncertainty). This is illustrated in figure 3.5.

Example 2. Let us now make matters slightly more interesting and replace the one-parameter function of the first example by a two-parameter function:

$$t(\xi) = \tanh(w_0 + w_1\xi) \qquad p(\boldsymbol{w}) = (2\pi)^{-1} e^{-\frac{1}{2}\boldsymbol{w}^2} \qquad D = \{(1, -\frac{1}{2})\}$$

[2]Note that, given a choice made for average and variance, the Gaussian distribution is the maximum entropy (i.e. maximum uncertainty) distribution for real values random variables.

[3]See appendix C on distributions for the definition and properties of the distribution $\delta[z]$.

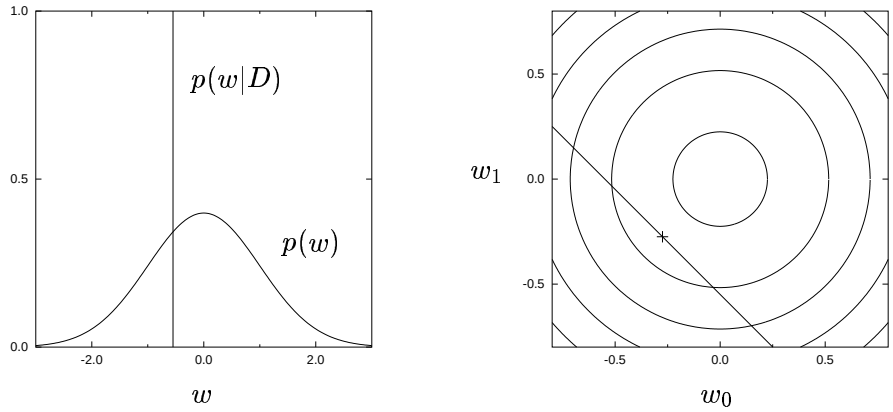


Figure 3.5: Left: reduction of parameter uncertainty due to arrival of data in example 1, from a Gaussian prior distribution $p(w)$ to a δ-peaked posterior distribution $p(w|D)$. Right: reduction of parameter uncertainty in example 2, from a Gaussian prior distribution $p(w_0, w_1)$ (contour lines) to a posterior distribution $p(w_0, w_1|D)$ with support only along the line $w_0 + w_1 = -\log\sqrt{3}$ (with its maximum at the point indicated by $+$ in the figure).

The probabilistic standard form becomes

$$p(t|\xi, \boldsymbol{w}) = \delta[t - \tanh(w_0 + w_1\xi)]$$

The posterior distribution becomes:

$$p(\boldsymbol{w}|D) = \frac{p(\boldsymbol{w})p(t^1|\xi^1, \boldsymbol{w})}{\int d\boldsymbol{w}'\ p(\boldsymbol{w}')p(t^1|\xi^1, \boldsymbol{w}')} = \frac{\delta[-\frac{1}{2} - \tanh(w_0 + w_1)]\ e^{-\frac{1}{2}\boldsymbol{w}^2}}{\int d\boldsymbol{w}'\ \delta[-\frac{1}{2} - \tanh(w_0' + w_1')]\ e^{-\frac{1}{2}\boldsymbol{w}'^2}}$$

$$= \frac{\delta[w_0 + w_1 + \tanh^{\rm inv}(\frac{1}{2})]\ e^{-\frac{1}{2}\boldsymbol{w}^2}}{\int d\boldsymbol{w}'\ \delta[w_0' + w_1' + \tanh^{\rm inv}(\frac{1}{2})]\ e^{-\frac{1}{2}\boldsymbol{w}'^2}} = \frac{\delta[w_0 + w_1 + \log\sqrt{3}]\ e^{-\frac{1}{2}\boldsymbol{w}^2}}{\int d\boldsymbol{w}'\ \delta[w_0' + w_1' + \log\sqrt{3}]\ e^{-\frac{1}{2}\boldsymbol{w}'^2}} \qquad (3.14)$$

(again using the identities $\delta[f(w)] = |f'(f^{\rm inv}(0))|^{-1}\delta[w - f^{\rm inv}(0)]$ and $\tanh^{\rm inv}(z) = \frac{1}{2}\log[(1+z)/(1-z)]$). Note that the posterior (3.14) is nonzero only along the line $w_0 + w_1 = -\log\sqrt{3}$ in the parameter plane, and that along this line it is maximal for the choice $w_0 = w_1 = -\frac{1}{2}\log\sqrt{3}$. This latter result simply follows upon calculating the minimum of $\boldsymbol{w}^2$ along the line $w_0 + w_1 = -\log\sqrt{3}$ (see figure 3.5).

Before we observed any data our uncertainty about the parameters $\boldsymbol{w} = (w_0, w_1)$ was described by the Gaussian prior $p(\boldsymbol{w})$. The arrival of the data point $(1, -\frac{1}{2})$ induced a collapse of this prior to an infinitely narrow distribution along the line $w_0 + w_1 = -\log\sqrt{3}$ (a 'slice' of the two-dimensional Gaussian prior in the parameter plane). The most likely parameter vector is $w_0 = w_1 = -\frac{1}{2}\log\sqrt{3}$, but we note that in (3.14) we also have exact quantitative information regarding the uncertainty in (w_0, w_1).

Note, finally, that (in contrast to the above two simple examples) we will generally have to allow our data generating models to incorporate noise, in order to retain a non-zero probability for having generated the (generally noisy) observed data.

The Link Between Bayesian Learning and Traditional Learning. At first sight it appears that Bayesian learning is quite remote from traditional gradient descent learning on an error surface, possibly with regularization, as in

$$\frac{d}{dt}\boldsymbol{w} = -\eta \boldsymbol{\nabla}_{\boldsymbol{w}} \{E_t[\boldsymbol{w}] + \lambda E_w[\boldsymbol{w}]\} \qquad E_t[\boldsymbol{w}] = \frac{1}{p}\sum_{\mu=1}^{p} \mathcal{E}[t^\mu - f(\boldsymbol{\xi}^\mu; \boldsymbol{w})] \tag{3.15}$$

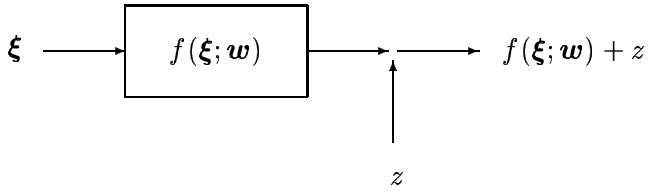
in which $\mathcal{E}[u]$ represents a single-point error measure, the function $E_w[\boldsymbol{w}]$ represents a regularizer, and with $\boldsymbol{w} \in \Re^N$. To reveal the link between the two viewpoints we must turn to (3.11), which we write as

$$\log p^{-1}(\boldsymbol{w}|D) = -\log p(\boldsymbol{w}) - \sum_{\mu=1}^{p} \log p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w}) + \log \int d\boldsymbol{w}'\; p(\boldsymbol{w}') \prod_{\mu=1}^{p} p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w}')$$

Finding the *most probable* parameter vector $\boldsymbol{w}_{\rm MP}$, given the data D, is thus equivalent to minimizing $\log p^{-1}(\boldsymbol{w}|D)$, i.e. to minimizing the quantity $S(\boldsymbol{w}, D)$:

$$S(\boldsymbol{w}, D) = -\log p(\boldsymbol{w}) - \sum_{\mu=1}^{p} \log p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w}) \tag{3.16}$$

Let us work out this expression for the following simple (and natural) class of data-generating model: $t = f(\boldsymbol{\xi}; \boldsymbol{w}) + z$,



where f denotes some parametrized function (such as performed by the standard neural networks, e.g. MLP, RBF, etc, which are traditionally used in gradient descent learning of the type (3.15)) and where z is a zero-average random number, representing additive noise in the data. If the noise variable is distributed according to $P[z]$, one has

$$p(t|\boldsymbol{\xi}; \boldsymbol{w}) = P[t - f(\boldsymbol{\xi}; \boldsymbol{w})] \tag{3.17}$$

Now the function (3.16) (whose minimum gives the most probable parameters) reduces to

$$\frac{1}{p}S(\boldsymbol{w}, D) = -\frac{1}{p}\sum_{\mu=1}^{p} \log P[t^\mu - f(\boldsymbol{\xi}^\mu; \boldsymbol{w})] + \frac{1}{p}\log[1/p(\boldsymbol{w})]$$

Comparison with expression (3.15) reveals the following:

- Learning by finding the minimum on a training error surface, with regularizer, as in (3.15), is equivalent to finding the *most probable* parameter vector $\boldsymbol{w}_{\rm MP}$[4].
- Choosing a specific single-point error measure $\mathcal{E}[u]$ in (3.15) means making the following assumption on the data noise statistics: $P[z] \sim e^{-\mathcal{E}[z]}$.

[4]This is also called the Maximum A-Posteriori Probability (MAP) procedure

- Choosing a specific regularizer $\lambda E_w[\boldsymbol{w}]$ in (3.15) means deciding on the following specific prior distribution for the parameters $\boldsymbol{w}$: $p(\boldsymbol{w}) = e^{-p\lambda E_w[\boldsymbol{w}]}$.

Our previously ad-hoc choices of error measure and regularizer are now replaced by direct interpretations, and hence guides for how to make appropriate choices. We also learn *en passant* that the regularizer strength must scale with the number of examples as $\lambda \sim p^{-1}$.

For example, learning by minimisation of the simplest (quadratic) training error measure $\mathcal{E}[u] = u^2$ and quadratic regularizer, viz.

$$\frac{d}{dt}\boldsymbol{w} = -\eta \boldsymbol{\nabla}_{\boldsymbol{w}} \left\{ \frac{1}{p} \sum_{\mu=1}^{p} [t^\mu - f(\boldsymbol{\xi}^\mu; \boldsymbol{w})]^2 + \frac{1}{2}\lambda \boldsymbol{w}^2 \right\} \tag{3.18}$$

is found to be fully equivalent to finding the most probable parameter vector $\boldsymbol{w}_{\rm MP}$ for the model

$$p(t|\boldsymbol{\xi}; \boldsymbol{w}) = \left[\frac{2\pi}{\beta}\right]^{-\frac{1}{2}} e^{-\frac{1}{2}\beta[t - f(\boldsymbol{\xi};\boldsymbol{w})]^2} \qquad p(\boldsymbol{w}) = \left[\frac{2\pi}{\alpha}\right]^{-\frac{N}{2}} e^{-\frac{1}{2}\alpha \boldsymbol{w}^2} \qquad \lambda = \frac{2\alpha}{\beta p} \tag{3.19}$$

For this choice the surface (3.16) simplifies to (modulo an irrelevant constant):

$$S(\boldsymbol{w}, D) = \frac{1}{2}\beta \sum_{\mu=1}^{p} [t^\mu - f(\boldsymbol{\xi}^\mu; \boldsymbol{w})]^2 + \frac{1}{2}\alpha \boldsymbol{w}^2 \tag{3.20}$$

We need no longer *guess* the value of λ, but can calculate it via $\lambda = 2\alpha/\beta p$ from

$$\alpha = \frac{1}{\langle w_i^2 \rangle_{\rm prior}} \qquad \beta = \frac{1}{\langle t^2 \rangle_{\rm noise} - \langle t \rangle_{\rm noise}^2} \qquad p = \text{ number of data points}$$

Quantities such as α or β in (3.19), which are not themselves adjustable parameters in the sense of $\boldsymbol{w}$, but are more global parameters which reflect prior knowledge of the problem and which will influence the leaning process, are called *hyper-parameters.*

Approximation of the Posterior Parameter Distribution. If we were to only calculate the most probable parameter vector $\boldsymbol{w}_{\rm MP}$ we would gain only interpretations compared to old-fashioned training error minimization. The power of the Bayesian techniques is that they also provide information on the reliability of a learning outcome. This information is embodied in the full posterior distribution $p(\boldsymbol{w}|D)$, which can be written in terms of (3.16) as

$$p(\boldsymbol{w}|D) = \frac{e^{-S(\boldsymbol{w},D)}}{\int d\boldsymbol{w}'\ e^{-S(\boldsymbol{w}',D)}} \tag{3.21}$$

In those cases where $S(\boldsymbol{w}, D)$ has only a single relevant local (and thus also global) minimum $\boldsymbol{w}_{\rm MP}$, the reliability of the learning outcome is mainly coded in the local curvature (i.e. 'width') of $S(\boldsymbol{w}, D)$ around $\boldsymbol{w}_{\rm MP}$. We now expand $S(\boldsymbol{w}, D)$ in the vicinity of $\boldsymbol{w}_{\rm MP}$:

$$S(\boldsymbol{w}, D) = S(\boldsymbol{w}_{\rm MP}, D) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_{\rm MP}) \cdot \boldsymbol{A}(\boldsymbol{w} - \boldsymbol{w}_{\rm MP}) + \mathcal{O}(|\boldsymbol{w} - \boldsymbol{w}_{\rm MP}|^3) \tag{3.22}$$

$$A_{ij} = \left.\frac{\partial^2 S}{\partial w_i \partial w_j}\right|_{\boldsymbol{w}_{\rm MP}} \tag{3.23}$$

($\boldsymbol{A}$ is called the Hessian matrix of S at the point $\boldsymbol{w}_{\rm MP}$; linear terms are absent in (3.22) due to $\boldsymbol{w}_{\rm MP}$ being a minimum).

Truncation of the expansion (3.22) after the quadratic term leads to a Gaussian approximation of the posterior $p(\boldsymbol{w}|D)$:

$$S(\boldsymbol{w},D) \;\rightarrow\; \tilde{S}(\boldsymbol{w},D) = S(\boldsymbol{w}_{\rm MP},D) + \frac{1}{2}(\boldsymbol{w}-\boldsymbol{w}_{\rm MP})\cdot\boldsymbol{A}(\boldsymbol{w}-\boldsymbol{w}_{\rm MP}) \tag{3.24}$$

$$p(\boldsymbol{w}|D) \;\rightarrow\; \tilde{p}(\boldsymbol{w}|D) = \left[\frac{\det\boldsymbol{A}}{(2\pi)^N}\right]^{\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{w}_{\rm MP})\cdot\boldsymbol{A}(\boldsymbol{w}-\boldsymbol{w}_{\rm MP})} \tag{3.25}$$

Average, variance and co-variance matrix of the approximated posterior distribution $\tilde{p}(\boldsymbol{w}|D)$ are given by the following identities (see appendix B) which involve the inverse of the Hessian (3.23), with the notation $\langle f(\boldsymbol{w})\rangle = \int d\boldsymbol{w}\; f(\boldsymbol{w})\; \tilde{p}(\boldsymbol{w}|D)$:

$$\langle\boldsymbol{w}\rangle = \boldsymbol{w}_{\rm MP} \qquad \langle w_i^2\rangle - \langle w_i\rangle^2 = (\boldsymbol{A}^{-1})_{ii} \qquad \langle w_i w_j\rangle - \langle w_i\rangle\langle w_j\rangle = (\boldsymbol{A}^{-1})_{ij} \tag{3.26}$$

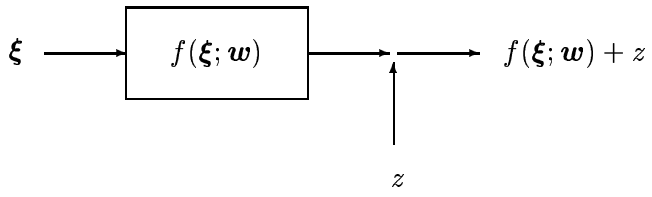
3.3 Predictions with Error Bars: Real-Valued Functions

Learning and predicting the action of continuous real-valued functions from (noisy) data examples is called *regression*. As we have seen in the previous section, a trained network is within the Bayesian framework given by the posterior distribution $p(\boldsymbol{w}|D)$ for the system parameters, as given by (3.11). Prediction then proceeds via formula (3.12), which can also be written in terms of the function $S(\boldsymbol{w},D)$ (3.16) as

$$p(t|\boldsymbol{\xi},D) = \int d\boldsymbol{w}\; p(t|\boldsymbol{\xi},\boldsymbol{w})p(\boldsymbol{w}|D) \qquad p(\boldsymbol{w}|D) = \frac{e^{-S(\boldsymbol{w},D)}}{\int d\boldsymbol{w}'\; e^{-S(\boldsymbol{w}',D)}} \tag{3.27}$$

This formally defines the statistics of the output to be associated with input $\boldsymbol{\xi}$.

Mean and Variance for Gaussian Output Noise and Gaussian Priors. We now work out output average and variance of (3.27) for the simplest class of systems (3.19): a parametrized function $f(\boldsymbol{\xi};\boldsymbol{w})$ with additive zero-average Gaussian data noise, and with a Gaussian prior parameter distribution $p(\boldsymbol{w})$:



The moments $\langle t\rangle = \int dt\; t\; p(t|\boldsymbol{\xi},D)$ and $\langle t^2\rangle = \int dt\; t^2\; p(t|\boldsymbol{\xi},D)$ become (transform integration variables by putting $t = f(\boldsymbol{\xi};\boldsymbol{w}) + z/\sqrt{\beta}$):

$$\begin{aligned}\langle t\rangle &= \left[\frac{2\pi}{\beta}\right]^{\frac{1}{2}} \int dt\; t \int d\boldsymbol{w}\; e^{-\frac{1}{2}\beta[t-f(\boldsymbol{\xi};\boldsymbol{w})]^2} p(\boldsymbol{w}|D) &(3.28)\\ &= \int d\boldsymbol{w} \int \frac{dz}{\sqrt{2\pi}}\; e^{-\frac{1}{2}z^2}\,[f(\boldsymbol{\xi};\boldsymbol{w}) + z/\sqrt{\beta}]\; p(\boldsymbol{w}|D)\\ &= \int d\boldsymbol{w}\; f(\boldsymbol{\xi};\boldsymbol{w})\; p(\boldsymbol{w}|D) &(3.29)\end{aligned}$$

$$\begin{aligned}\langle t^2\rangle &= \left[\frac{2\pi}{\beta}\right]^{\frac{1}{2}} \int dt\ t^2 \int d\boldsymbol{w}\ e^{-\frac{1}{2}\beta[t-f(\boldsymbol{\xi};\boldsymbol{w})]^2} p(\boldsymbol{w}|D) \\ &= \int d\boldsymbol{w} \int \frac{dz}{\sqrt{2\pi}}\ e^{-\frac{1}{2}z^2}[f(\boldsymbol{\xi};\boldsymbol{w}) + z/\sqrt{\beta}]^2\ p(\boldsymbol{w}|D) \\ &= \frac{1}{\beta} + \int d\boldsymbol{w}\ f^2(\boldsymbol{\xi};\boldsymbol{w})\ p(\boldsymbol{w}|D) \end{aligned} \tag{3.30}$$

The prediction variance $\sigma^2 = \langle t^2\rangle - \langle t\rangle^2$ now follows as

$$\sigma^2 = \frac{1}{\beta} + \underbrace{\int d\boldsymbol{w}\ f^2(\boldsymbol{\xi};\boldsymbol{w})\ p(\boldsymbol{w}|D) - \left[\int d\boldsymbol{w}\ f(\boldsymbol{\xi};\boldsymbol{w})\ p(\boldsymbol{w}|D)\right]^2}_{\text{variance due to uncertainty about system parameters}} \tag{3.31}$$

The first term, β^{-1}, represents the intrinsic uncertainty in the data generation model; the remainder in (3.31) reflects our uncertainty about the right parameters after having learned the data. The Bayesian predicted output $t^\star(\boldsymbol{\xi})$ for the input $\boldsymbol{\xi}$, and its associated error margin $\Delta t^\star(\boldsymbol{\xi})$, are now defined as $\langle t\rangle$ and σ, respectively:[5]

$$t^\star(\boldsymbol{\xi}) = \int d\boldsymbol{w}\ f(\boldsymbol{\xi};\boldsymbol{w})\ p(\boldsymbol{w}|D) \tag{3.32}$$

$$\Delta t^\star(\boldsymbol{\xi}) = \sqrt{\frac{1}{\beta} + \int d\boldsymbol{w}\ f^2(\boldsymbol{\xi};\boldsymbol{w})\ p(\boldsymbol{w}|D) - \left[\int d\boldsymbol{w}\ f(\boldsymbol{\xi};\boldsymbol{w})\ p(\boldsymbol{w}|D)\right]^2} \tag{3.33}$$

For Gaussian output noise and priors the function $S(\boldsymbol{w}, D)$ is given by (3.20), hence

$$p(\boldsymbol{w}|D) = \frac{e^{-S(\boldsymbol{w},D)}}{\int d\boldsymbol{w}'\ e^{-S(\boldsymbol{w}',D)}} \qquad S(\boldsymbol{w}, D) = \frac{1}{2}\beta\sum_{\mu=1}^{p}[t^\mu - f(\boldsymbol{\xi}^\mu;\boldsymbol{w})]^2 + \frac{1}{2}\alpha\boldsymbol{w}^2$$

In (3.32) we now have proper predictions *with error bars.*

Simplification for Approximated Posterior Distribution. The above exact result (3.32,3.33) can be simplified using the approximated posterior distribution $\tilde{p}(\boldsymbol{w}|D)$ of (3.25). Upon also putting $\boldsymbol{w} = \boldsymbol{w}_{\rm MP} + \boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{z}$ in the parameter integrations, we then find

$$t^\star(\boldsymbol{\xi}) = \int d\boldsymbol{z}\ \frac{e^{-\frac{1}{2}\boldsymbol{z}^2}}{(2\pi)^{N/2}}\ f(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP} + \boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{z}) \tag{3.34}$$

$$\Delta t^\star(\boldsymbol{\xi}) = \sqrt{\frac{1}{\beta} + \int d\boldsymbol{z}\ \frac{e^{-\frac{1}{2}\boldsymbol{z}^2}}{(2\pi)^{N/2}}\ f^2(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP} + \boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{z}) - [t^\star(\boldsymbol{\xi})]^2} \tag{3.35}$$

We now make one further and final approximation. If the width of the Gaussian approximation (3.25) is not too large, we may approximate/expand $f(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP}+\boldsymbol{u})$ for small $\boldsymbol{u}$:

$$f(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP}+\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{z}) = f(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP}) + \boldsymbol{x}(\boldsymbol{\xi})\cdot\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{z} + \frac{1}{2}(\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{z})\cdot\boldsymbol{B}(\boldsymbol{\xi})(\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{z}) + \mathcal{O}([\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{z}]^3) \tag{3.36}$$

[5]Note: $t^\star(\boldsymbol{\xi})$ need not be identical to $f(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP})$, where $\boldsymbol{w}_{\rm MP}$ denotes the most probable parameters.

where $x_i(\boldsymbol{\xi}) = \partial f(\boldsymbol{\xi};\boldsymbol{w})/\partial w_i|_{\boldsymbol{w}_{\rm MP}}$ and $B_{ij}(\boldsymbol{\xi}) = \partial^2 f(\boldsymbol{\xi};\boldsymbol{w})/\partial w_i \partial w_j|_{\boldsymbol{w}_{\rm MP}}$. Insertion into (3.34,3.35) gives:

$$\begin{aligned}
t^\star(\boldsymbol{\xi}) &= f(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP}) + \frac{1}{2}\sum_{ij}\int d\boldsymbol{z}\ \frac{e^{-\frac{1}{2}\boldsymbol{z}^2}}{(2\pi)^{N/2}}\ z_i[\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{B}(\boldsymbol{\xi})\boldsymbol{A}^{-\frac{1}{2}}]_{ij}z_j + \mathcal{O}(|\boldsymbol{A}^{-2}|) \\
&= f(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP}) + \frac{1}{2}{\rm Tr}[\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{B}(\boldsymbol{\xi})\boldsymbol{A}^{-\frac{1}{2}}] + \mathcal{O}(|\boldsymbol{A}^{-2}|) \\
[\Delta t^\star(\boldsymbol{\xi})]^2 &= \frac{1}{\beta} + f^2(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP}) - [t^\star(\boldsymbol{\xi})]^2 + \mathcal{O}(|\boldsymbol{A}^{-2}|) \\
&\quad + \int d\boldsymbol{z}\ \frac{e^{-\frac{1}{2}\boldsymbol{z}^2}}{(2\pi)^{N/2}}\ \Big[[\boldsymbol{x}(\boldsymbol{\xi})\cdot\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{z}]^2 + f(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP})(\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{z})\cdot\boldsymbol{B}(\boldsymbol{\xi})(\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{z})\Big] \\
&= \frac{1}{\beta} - f(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP}).{\rm Tr}[\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{B}(\boldsymbol{\xi})\boldsymbol{A}^{-\frac{1}{2}}] + \mathcal{O}(|\boldsymbol{A}^{-2}|) \\
&\quad + \sum_{ij}\Big\{[\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{x}(\boldsymbol{\xi})]_i[\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{x}(\boldsymbol{\xi})]_j + f(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP})[\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{B}(\boldsymbol{\xi})\boldsymbol{A}^{-\frac{1}{2}}]_{ij}\Big\}\delta_{ij} \\
&= \frac{1}{\beta} - f(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP}).{\rm Tr}[\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{B}(\boldsymbol{\xi})\boldsymbol{A}^{-\frac{1}{2}}] \\
&\quad + \boldsymbol{x}(\boldsymbol{\xi})\cdot\boldsymbol{A}^{-1}\boldsymbol{x}(\boldsymbol{\xi}) + f(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP}){\rm Tr}[\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{B}(\boldsymbol{\xi})\boldsymbol{A}^{-\frac{1}{2}}] + \mathcal{O}(|\boldsymbol{A}^{-2}|) \\
&= \frac{1}{\beta} + \boldsymbol{x}(\boldsymbol{\xi})\cdot\boldsymbol{A}^{-1}\boldsymbol{x}(\boldsymbol{\xi}) + \mathcal{O}(|\boldsymbol{A}^{-2}|)
\end{aligned}$$

(with the trace of a matrix $\boldsymbol{C}$ defined as usual: ${\rm Tr}\boldsymbol{C} = \sum_i C_{ii}$). Hence we arrive at:

$$t^\star(\boldsymbol{\xi}) = f(\boldsymbol{\xi};\boldsymbol{w}_{\rm MP}) + \frac{1}{2}{\rm Tr}[\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{B}(\boldsymbol{\xi})\boldsymbol{A}^{-\frac{1}{2}}] + \mathcal{O}(|\boldsymbol{A}^{-2}|) \tag{3.37}$$

$$\Delta t^\star(\boldsymbol{\xi}) = \sqrt{\beta^{-1} + \boldsymbol{x}(\boldsymbol{\xi})\cdot\boldsymbol{A}^{-1}\boldsymbol{x}(\boldsymbol{\xi}) + \mathcal{O}(|\boldsymbol{A}^{-2}|)} \tag{3.38}$$

For expressions (3.37,3.38) to be accurate[6] and useful (given our earlier assumptions of Gaussian additive data noise and a Gaussian prior) we need $p(\boldsymbol{w}|D)$ to be (i) approximately Gaussian, and (ii) sufficiently narrow (in view of (3.26), since this would guarantee that $\boldsymbol{A}^{-1}$ can indeed be treated as small).

Models for which Truncation is Exact. In order to appreciate the nature of the approximations one would arrive at by simply neglecting the $\mathcal{O}(|\boldsymbol{A}^{-2}|)$ terms in (3.37,3.38), it is instructive to determine the conditions under which they would have been exact. Given Gaussian data output noise and a Gaussian prior, approximation (3.25) would have been exact only if $S(\boldsymbol{w}, D)$ (3.16) would depend quadratically on $\boldsymbol{w}$, i.e. if $f(\boldsymbol{\xi};\boldsymbol{w})$ would depend linearly on $\boldsymbol{w}$:

$$p(t|\boldsymbol{\xi};\boldsymbol{w}) = \left[\frac{2\pi}{\beta}\right]^{\frac{1}{2}} e^{-\frac{1}{2}\beta[t-f(\boldsymbol{\xi};\boldsymbol{w})]^2} \qquad p(\boldsymbol{w}) = \left[\frac{2\pi}{\alpha}\right]^{\frac{N}{2}} e^{-\frac{1}{2}\alpha\boldsymbol{w}^2} \qquad f(\boldsymbol{\xi};\boldsymbol{w}) = \sum_i w_i\phi_i(\boldsymbol{\xi}) \tag{3.39}$$

(for some set of functions $\{\phi_i(\boldsymbol{\xi})\}$, as in e.g. RBF networks). Here also (3.36) is exact when truncated after the term linear in $\boldsymbol{z}$, so $\boldsymbol{B}(\boldsymbol{\xi}) = 0$ and the $\mathcal{O}(|\boldsymbol{A}^{-2}|)$ terms in (3.37,3.38) are

[6] Note: in many textbooks one finds that the second term in $t^\star(\boldsymbol{\xi})$ is also neglected.

absent. Furthermore, since $S(\boldsymbol{w}, D)$ is now truly Gaussian, we can calculate $\boldsymbol{x}(\boldsymbol{\xi})$ and $\boldsymbol{w}_{\rm MP}$ explicitly. We define $\boldsymbol{\phi}(\boldsymbol{\xi}) = (\phi_1(\boldsymbol{\xi}), \ldots, \phi_N(\boldsymbol{\xi}))$, and get

$$\begin{aligned} S(\boldsymbol{w}, D) &= = \frac{1}{2}\beta \sum_{\mu=1}^{p} [t^\mu - \boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{\xi}^\mu)]^2 + \frac{1}{2}\alpha \boldsymbol{w}^2 \\ &= \frac{1}{2}\beta \sum_{\mu=1}^{p} [t^\mu]^2 - \beta \boldsymbol{w}\cdot\sum_{\mu=1}^{p} t^\mu \boldsymbol{\phi}(\boldsymbol{\xi}^\mu) + \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{A}\boldsymbol{w} \qquad (3.40) \\ A_{ij} &= \alpha\delta_{ij} + \beta \sum_{\mu=1}^{p} \phi_i(\boldsymbol{\xi}^\mu)\phi_j(\boldsymbol{\xi}^\mu) \qquad (3.41) \end{aligned}$$

Since the Hessian matrix $\boldsymbol{A}$ is positive define (provided $\alpha > 0$), the surface $S(\boldsymbol{w}, D)$ is convex and has a unique minimum, which is calculated simply from putting $\partial S(\boldsymbol{w}, D)/\partial w_i = 0$ for all i. This reveals $\boldsymbol{A}\boldsymbol{w}_{\rm MP} = \beta \sum_{\mu=1}^{p} t^\mu \boldsymbol{\phi}(\boldsymbol{\xi}^\mu)$, or

$$\boldsymbol{w}_{\rm MP} = \beta \boldsymbol{A}^{-1} \sum_{\mu=1}^{p} t^\mu \boldsymbol{\phi}(\boldsymbol{\xi}^\mu) \qquad (3.42)$$

Similarly one finds $x_i(\boldsymbol{\xi}) = \phi_i(\boldsymbol{\xi})$. Insertion into (3.37,3.38) then gives the, for the present models (3.39) fully exact, result

$$t^\star(\boldsymbol{\xi}) = \beta \boldsymbol{\phi}(\boldsymbol{\xi})\cdot\boldsymbol{A}^{-1}\sum_{\mu=1}^{p} t^\mu \boldsymbol{\phi}(\boldsymbol{\xi}^\mu) \qquad \Delta t^\star(\boldsymbol{\xi}) = \sqrt{\beta^{-1} + \boldsymbol{\phi}(\boldsymbol{\xi})\cdot\boldsymbol{A}^{-1}\boldsymbol{\phi}(\boldsymbol{\xi})} \qquad (3.43)$$

Estimation of Hyper-Parameters by Bootstrapping. The predictions and error margins in (3.37,3.38) and (3.43) obviously depend on the choices made for the hyper-parameters α and β. We will discuss a proper procedure for this later. A simpler (and generally quicker) method for estimating α and β is the following. Provided the number p if data is not too small, it makes sense to require that the trained system will *on average* (i) predict correctly the outputs for the data points used in training, and (ii) make errors on these training data whose magnitude is of the order of the uncertainty it associates with its own predictions:

$$\frac{1}{p}\sum_{\mu=1}^{p}[t^\mu - t^\star(\boldsymbol{\xi}^\mu)] = 0 \qquad \frac{1}{p}\sum_{\mu=1}^{p}\left\{[t^\mu - t^\star(\boldsymbol{\xi}^\mu)]^2 - [\Delta t^\star(\boldsymbol{\xi}^\mu)]^2\right\} = 0 \qquad (3.44)$$

These two equations can be used to determine α and β.

Example 1. Let us illustrate the above procedures using the example problem of (3.6), which involves a parametrized function of the form (3.39), with a quadratic regularizer term $\frac{1}{2}\lambda\boldsymbol{w}^2$. To make the learning problem slightly more difficult we have now also distributed the data inhomogeneously along the x-axis. The non-Bayesian solution to this simple problem (i.e. minimize the training error plus generalizer), is easily calculated and found to be

$$t^\star(x) = \sum_{ij=0}^{M} \phi_i(x)(\boldsymbol{C}^{-1})_{ij}\sum_{\mu} t^\mu \phi_j(x) \qquad C_{ij} = \lambda\delta_{ij} + \sum_{\mu=1}^{p}\phi_i(x^\mu)\phi_j(x^\mu) \qquad \phi_\ell(x) = x^\ell \quad (3.45)$$

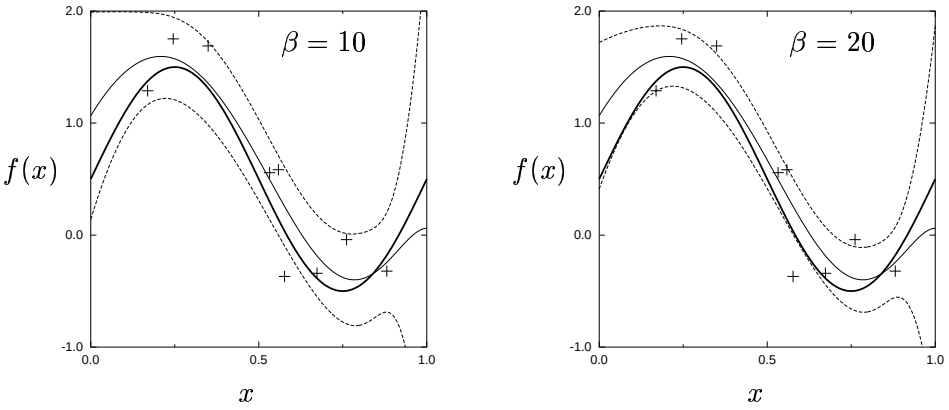


Figure 3.6: Bayesian prediction on the basis of an order-9 polynomial with assumed Gaussian data noise (variance: β^{-1}) and a Gaussian prior (variance: $\alpha^{-1} = (\lambda\beta)^{-1}$, $\lambda = 0.005$). Thick solid line: the actual function $f(x) = \frac{1}{2} + \sin(2\pi x)$ to be learned. Crosses: nine noisy sample points of this function. Solid line: predicted function $t^\star(x)$ after learning. Dashed lines: $t^\star(x) \pm \Delta t^\star(x)$, where $\Delta t^\star(x)$ denotes the Bayesian prediction uncertainty. Note: the actual data variance corresponds to $\beta = 12$.

(without error bars). The Bayesian answer (3.43), upon assuming Gaussian output noise and a Gaussian prior, would be exactly the same prediction $t^\star(x)$ given in (3.45)[7], with $\lambda = \alpha/\beta$, but it would also equip this answer with the following error estimate:

$$\Delta t^\star(x) = \beta^{-\frac{1}{2}} \sqrt{1 + \sum_{ij=0}^{M} x^{i+j} (\boldsymbol{C}^{-1})_{ij}} \tag{3.46}$$

The result is shown in figure 3.6 for $\lambda = \alpha/\beta = 0.005$, $M = 9$, and $\beta \in \{10, 20\}$. The actual value of the variance in the data points (unknown to the network) in this example corresponds to $\beta = 12$. As expected, we see again in the above graphs that it will be very important to have a tool with which to calculate the hyper-parameters α and β.

Example 2. Our second example is also a simple linear system as in (3.39), but now the input vectors are two-dimensional, and we will focus on finding the hyper-parameters via 'bootstrapping':

$$p(t|\boldsymbol{\xi};\boldsymbol{w}) = \left[\frac{2\pi}{\beta}\right]^{-\frac{1}{2}} e^{-\frac{1}{2}\beta[t-f(\boldsymbol{\xi};\boldsymbol{w})]^2} \qquad p(\boldsymbol{w}) = \left[\frac{2\pi}{\alpha}\right]^{-\frac{N}{2}} e^{-\frac{1}{2}\alpha\boldsymbol{w}^2} \qquad f(\boldsymbol{\xi};\boldsymbol{w}) = \boldsymbol{w}\cdot\boldsymbol{\xi} \tag{3.47}$$

with $\boldsymbol{\xi}, \boldsymbol{w} \in \Re^2$. In the notation of (3.39) we here have $\phi_i(\boldsymbol{\xi}) = \xi_i$. The data points to be learned from, four in number, are taken to be the following:

$$(\boldsymbol{\xi}^1, t^1) = ((1,0), \frac{3}{4}), \quad (\boldsymbol{\xi}^2, t^2) = ((0,1), \frac{1}{4}), \quad (\boldsymbol{\xi}^3, t^3) = ((-1,0), -\frac{1}{4}), \quad (\boldsymbol{\xi}^4, t^4) = ((0,-1), -\frac{3}{4})$$

For this choice of data the matrix $\boldsymbol{A}$ (3.41) becomes

$$\boldsymbol{A} = \alpha \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \beta \left\{ \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \right\} = (\alpha + 2\beta)\mathbb{1}$$

[7] This is, of course, a direct consequence of our choice of a simple linear example.

Its inversion is trivial, and hence the predictions and error bars (3.43) can be worked out explicitly. Using $\boldsymbol{A}^{-1} = (\alpha+2\beta)^{-1}\mathbb{I}$ and $\sum_{\mu=1}^{4} t^{\mu}\boldsymbol{\xi}^{\mu} = (1,1)$ one arrives at

$$t^{\star}(\boldsymbol{\xi}) = \frac{\beta(\xi_1+\xi_2)}{\alpha+2\beta} \qquad \Delta t^{\star}(\boldsymbol{\xi}) = \beta^{-\frac{1}{2}}\sqrt{1+\frac{\beta(\xi_1^2+\xi_2^2)}{\alpha+2\beta}} \tag{3.48}$$

In order to work out the 'bootstrapping' relations (3.44) for the present example we next calculate the system's internal output predictions for the four data points:

$$t^{\star}(\boldsymbol{\xi}^1) = t^{\star}(\boldsymbol{\xi}^2) = \frac{\beta}{\alpha+2\beta} \qquad t^{\star}(\boldsymbol{\xi}^3) = t^{\star}(\boldsymbol{\xi}^4) = \frac{-\beta}{\alpha+2\beta}$$

$$\Delta t^{\star}(\boldsymbol{\xi}^1) = \Delta t^{\star}(\boldsymbol{\xi}^2) = \Delta t^{\star}(\boldsymbol{\xi}^3) = \Delta t^{\star}(\boldsymbol{\xi}^4) = \beta^{-\frac{1}{2}}\sqrt{\frac{\alpha+3\beta}{\alpha+2\beta}}$$

Since also $\sum_{\mu=1}^{4} t^{\mu} = 0$, the first condition of (3.44) now reduces to the trivial identity $0 = 0$. The second condition of (3.44), however, gives us

$$\frac{\alpha+3\beta}{\beta(\alpha+2\beta)} = \frac{1}{4}\left\{[\frac{3}{4}-\frac{\beta}{\alpha+2\beta}]^2+[\frac{1}{4}-\frac{\beta}{\alpha+2\beta}]^2+[-\frac{1}{4}+\frac{\beta}{\alpha+2\beta}]^2+[-\frac{3}{4}+\frac{\beta}{\alpha+2\beta}]^2\right\}$$

$$\frac{\alpha+3\beta}{\beta(\alpha+2\beta)} = \frac{1}{32}\left\{[\frac{3\alpha+2\beta}{\alpha+2\beta}]^2+[\frac{\alpha-2\beta}{\alpha+2\beta}]^2\right\}$$

Working out this expression upon putting $\alpha = \lambda\beta$ leads to

$$\beta = 24\left[\frac{1+\frac{5}{6}\lambda+\frac{1}{6}\lambda^2}{1+\lambda+\frac{5}{4}\lambda^2}\right] \tag{3.49}$$

We thus have only one free hyper-parameter left. Upon putting $\alpha \to 0$ (since $p = 4$ and we have just two adjustable parameters there should in principle be little need for regularization via a prior) gives, via (3.49), the prescription $\beta = 24$, and hence the parameter-free predictions

$$t^{\star}(\boldsymbol{\xi}) = \frac{1}{2}(\xi_1+\xi_2) \qquad \Delta t^{\star}(\boldsymbol{\xi}) = \frac{1}{4}\sqrt{\frac{1}{3}(2+\xi_1^2+\xi_2^2)} \tag{3.50}$$

This result is indeed perfectly consistent with the statistics of the training data:

	actual data	$t^{\star}(\boldsymbol{\xi}) \pm \Delta t^{\star}(\boldsymbol{\xi})$
t^1 :	$3/4$	$1/2 \pm 1/4$
t^2 :	$1/4$	$1/2 \pm 1/4$
t^3 :	$-1/4$	$-1/2 \pm 1/4$
t^4 :	$-3/4$	$-1/2 \pm 1/4$

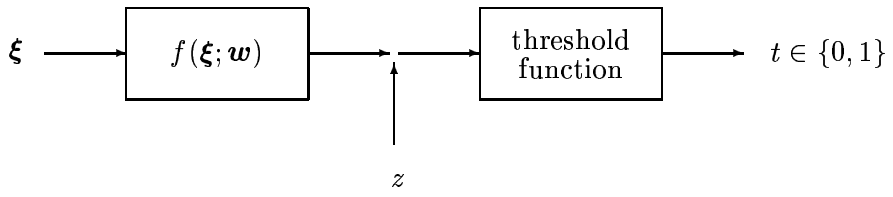
The model is also seen to take the sensible decision to assign increasing error bars when predictions are requested on input vectors which are further away from the origin (around which the data are located).

3.4 Predictions with Error Bars: Binary Classification

Predicting binary classifications on the basis of (noisy) data examples proceeds in a way similar to regression. A trained network is in the Bayesian picture given by the posterior distribution $p(\boldsymbol{w}|D)$ (3.11) for the system parameters, and prediction is again based on (3.12):

$$p(t|\boldsymbol{\xi}, D) = \int d\boldsymbol{w}\; p(t|\boldsymbol{\xi}, \boldsymbol{w}) p(\boldsymbol{w}|D) \qquad p(\boldsymbol{w}|D) = \frac{p(\boldsymbol{w}) \prod_{\mu=1}^{p} p(t^{\mu}|\boldsymbol{\xi}^{\mu}, \boldsymbol{w})}{\int d\boldsymbol{w}'\; p(\boldsymbol{w}') \prod_{\mu=1}^{p} p(t^{\mu}|\boldsymbol{\xi}^{\mu}, \boldsymbol{w}')} \tag{3.51}$$

The only difference with regression is that, in view of the requirement to extract binary outputs from continuous parametrized functions, the output noise can never be Gaussian.



Without output noise we would define the deterministic operation $t(\boldsymbol{\xi}) = \theta[f(\boldsymbol{\xi}; \boldsymbol{w})]$, with the step function (i.e. $\theta[z > 0] = 1$, $\theta[z < 0] = 0$). Adding noise implies allowing for $t = 0$ even when $f(\boldsymbol{\xi}; \boldsymbol{w}) > 0$ and for $t = 1$ even when $f(\boldsymbol{\xi}; \boldsymbol{w}) < 0$. So $p(1|\boldsymbol{\xi}, \boldsymbol{w})$ must be a monotonically increasing function of $f(\boldsymbol{\xi}; \boldsymbol{w})$, such that $p(1|\boldsymbol{\xi}, \boldsymbol{w}) \to 0$ when $f(\boldsymbol{\xi}; \boldsymbol{w}) \to -\infty$ and $p(1|\boldsymbol{\xi}, \boldsymbol{w}) \to 1$ when $f(\boldsymbol{\xi}; \boldsymbol{w}) \to \infty$[8]. A common choice is:

$$\begin{aligned} p(1|\boldsymbol{\xi}, \boldsymbol{w}) &= \tfrac{1}{2} + \tfrac{1}{2} \tanh[f(\boldsymbol{\xi}; \boldsymbol{w})] \\ p(0|\boldsymbol{\xi}, \boldsymbol{w}) &= \tfrac{1}{2} - \tfrac{1}{2} \tanh[f(\boldsymbol{\xi}; \boldsymbol{w})] \end{aligned} \tag{3.52}$$

Decision Boundary and Measure of Uncertainty. Our objective is to classify new inputs $\boldsymbol{\xi}$ into one of two categories. Hence learning from data is here equivalent to deciding on the decision boundary in input space, and to quantify the uncertainty of this decision. If we simply combine (3.51,3.52) we arrive at

$$p(1|\boldsymbol{\xi}, D) = \frac{1}{2} + \frac{1}{2} I(\boldsymbol{\xi}, D) \qquad p(0|\boldsymbol{\xi}, D) = \frac{1}{2} - \frac{1}{2} I(\boldsymbol{\xi}, D) \tag{3.53}$$

$$I(\boldsymbol{\xi}, D) = \frac{\int d\boldsymbol{w}\;\; \tanh[f(\boldsymbol{\xi}; \boldsymbol{w})]\; p(\boldsymbol{w}) \prod_{\mu=1}^{p} p(t^{\mu}|\boldsymbol{\xi}^{\mu}, \boldsymbol{w})}{\int d\boldsymbol{w}\; p(\boldsymbol{w}) \prod_{\mu=1}^{p} p(t^{\mu}|\boldsymbol{\xi}^{\mu}, \boldsymbol{w})} \tag{3.54}$$

We classify $t^{\star}(\boldsymbol{\xi}) = 1$ if $p(1|\boldsymbol{\xi}, D) > p(0|\boldsymbol{\xi}, D)$, and $t^{\star}(\boldsymbol{\xi}) = 0$ if $p(1|\boldsymbol{\xi}, D) < p(0|\boldsymbol{\xi}, D)$. Hence the decision boundary in $\boldsymbol{\xi}$-space is defined by $I(\boldsymbol{\xi}, D) = 0$. Our uncertainty is measured by

$$\Delta t^{\star}(\boldsymbol{\xi}) = \text{ Prob [incorrect classification] } = \begin{cases} p(1|\boldsymbol{\xi}, D) & \text{when } t^{\star}(\boldsymbol{\xi}) = 0 \\ p(0|\boldsymbol{\xi}, D) & \text{when } t^{\star}(\boldsymbol{\xi}) = 1 \end{cases}$$

[8] An implicit requirement is that the selected parametrized function $f(\boldsymbol{\xi}; \boldsymbol{w})$ can actually approach $\pm\infty$ by boosting suitably the parameters $\boldsymbol{w}$. This, however, can always be achieved. For example, if we initially have a bounded function $g(\boldsymbol{\xi}; \boldsymbol{w})$, we can add an extra adjustable parameter w_0 and define $f(\boldsymbol{x}; \boldsymbol{w}) = w_0 g(\boldsymbol{\xi}; \boldsymbol{w})$

These statements can all be compactified, using the definition (3.54), into

$$\begin{aligned} I(\boldsymbol{\xi},D)>0: \quad & t^\star(\boldsymbol{\xi})=1 \\ I(\boldsymbol{\xi},D)<0: \quad & t^\star(\boldsymbol{\xi})=0 \end{aligned} \qquad\qquad \Delta t^\star(\boldsymbol{\xi}) = \frac{1}{2}-\frac{1}{2}|I(\boldsymbol{\xi},D)| \tag{3.55}$$

At the decision boundary $I(\boldsymbol{\xi},D)=0$ we have $\Delta t^\star(\boldsymbol{\xi})=\frac{1}{2}$, i.e. a mis-classification probability equivalent to random guessing, as it should.

Example. Let us consider a simple task involving the binary classification of two-dimensional vectors, with a linear parametrized function and a Gausian prior:

$$\boldsymbol{\xi},\boldsymbol{w}\in\Re^2 \qquad f(\boldsymbol{\xi};\boldsymbol{w})=\boldsymbol{w}\cdot\boldsymbol{\xi} \qquad p(\boldsymbol{w})=\frac{\alpha}{2\pi}e^{-\frac{1}{2}\alpha\boldsymbol{w}^2} \tag{3.56}$$

and with the following two data points

$$D=\{((0,-1),1),\ ((0,1),0)\} \tag{3.57}$$

Working out the key function $I(\boldsymbol{\xi},D)$ of (3.54) for this example gives

$$\begin{aligned} I(\boldsymbol{\xi},D) &= \frac{\int d\boldsymbol{w}\ \tanh[\boldsymbol{w}\cdot\boldsymbol{\xi}]\ e^{-\frac{1}{2}\alpha\boldsymbol{w}^2}\prod_{\mu=1}^2 p(t^\mu|\boldsymbol{\xi}^\mu,\boldsymbol{w})}{\int d\boldsymbol{w}\ e^{-\frac{1}{2}\alpha\boldsymbol{w}^2}\prod_{\mu=1}^2 p(t^\mu|\boldsymbol{\xi}^\mu,\boldsymbol{w})} \\ &= \frac{\int d\boldsymbol{w}\ \tanh[w_1\xi_1+w_2\xi_2]\ e^{-\frac{1}{2}\alpha\boldsymbol{w}^2}\left[1-\tanh(w_2)\right]^2}{\int d\boldsymbol{w}\ e^{-\frac{1}{2}\alpha\boldsymbol{w}^2}\left[1-\tanh(w_2)\right]^2} \end{aligned} \tag{3.58}$$

The decision boundary in input space is found to be given by the line $\xi_2=0$, for if we substitute this into (3.58) we indeed find $I((\xi_1,0),D)=0$. Verification of the behaviour of (3.58) for $\xi_2\to\pm\infty$ further shows that $I(\boldsymbol{\xi},D)>0$ (and hence $t^\star(\boldsymbol{\xi})=1$) for $\xi_2<0$, and $I(\boldsymbol{\xi},D)<0$ (and hence $t^\star(\boldsymbol{\xi})=0$) for $\xi_2>0$.

The more interesting (and less trivial) results concern the dependence of the error measure $\Delta t^\star(\boldsymbol{\xi})$ in (3.55) on $\boldsymbol{\xi}$. In figure 3.7 (left panel) we show a contour plot of the mis-classification probability $\Delta t^\star(\boldsymbol{\xi})$ in the input plane, for $\alpha=1$. The model (3.56) assumes a linear separation (a line through the origin, and data noise), the slope of the separating line is extracted from the data. Clearly, the further away from the data points, the larger the possible impact of an uncertainty in this slope, which is seen to be reflected in the contours of $\Delta t^\star(\boldsymbol{\xi})$.

We continue with the example model (3.56), but now we add one more data point, namely $(\boldsymbol{\xi}^3,t^3)=((1,\frac{1}{2}),0)$ and study its effect on the decision boundary and the mis-classification probability:

$$D=\{((0,-1),1),\ ((0,1),0),\ ((1,\frac{1}{2}),0)\} \tag{3.59}$$

Working out $I(\boldsymbol{\xi},D)$ of (3.54) now gives

$$\begin{aligned} I(\boldsymbol{\xi},D) &= \frac{\int d\boldsymbol{w}\ \tanh[\boldsymbol{w}\cdot\boldsymbol{\xi}]\ e^{-\frac{1}{2}\alpha\boldsymbol{w}^2}\prod_{\mu=1}^3 p(t^\mu|\boldsymbol{\xi}^\mu,\boldsymbol{w})}{\int d\boldsymbol{w}\ e^{-\frac{1}{2}\alpha\boldsymbol{w}^2}\prod_{\mu=1}^3 p(t^\mu|\boldsymbol{\xi}^\mu,\boldsymbol{w})} \\ &= \frac{\int d\boldsymbol{w}\ \tanh[w_1\xi_1+w_2\xi_2]\ e^{-\frac{1}{2}\alpha\boldsymbol{w}^2}\left[1-\tanh(w_2)\right]^2\left[1-\tanh(w_1+\frac{1}{2}w_2)\right]}{\int d\boldsymbol{w}\ e^{-\frac{1}{2}\alpha\boldsymbol{w}^2}\left[1-\tanh(w_2)\right]^2\left[1-\tanh(w_1+\frac{1}{2}w_2)\right]} \end{aligned} \tag{3.60}$$

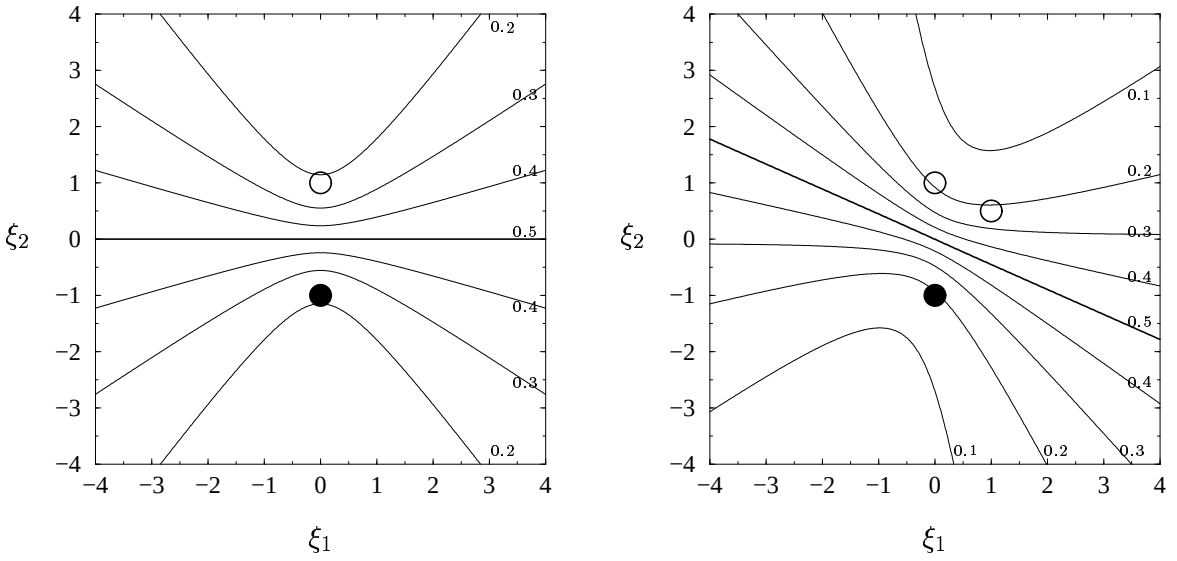


Figure 3.7: Bayesian binary classification of two-dimensional input vectors $\boldsymbol{\xi} = (\xi_1, \xi_2)$ on the basis of a linear parametrized function with (non-Gaussian) data noise and a Gaussian prior (variance: $\alpha^{-1} = 1$). Thick solid line: the decision boundary, where $p(1|\boldsymbol{\xi}, D) = p(0|\boldsymbol{\xi}, D) = \frac{1}{2}$ (equivalently: where $\Delta t^\star(\boldsymbol{\xi}) = \frac{1}{2}$). Below this line all points are classified as $t^\star(\boldsymbol{\xi}) = 1$, above it as $t^\star(\boldsymbol{\xi}) = 0$. Circles: data points $\boldsymbol{\xi}^\mu$ (filled: $t^\mu = 1$, open: $t^\mu = 0$). Thin continuous curves: contour lines of the error probability $\Delta t^\star(\boldsymbol{\xi})$. Left panel: $p = 2$. Right panel: $p = 3$ (i.e. one further data point added to those of the left panel; see the main text for details).

The decision boundary is now no longer given by $\xi_2 = 0$; the balance of evidence has changed. In figure 3.7 (right panel) we show a contour plot of the mis-classification probability $\Delta t^\star(\boldsymbol{\xi})$ in the input plane for the new situation with the extra data point. Still our model assumes a 'noisy' linear separation, but compared to the previous $p = 2$ case we observe both a change in the location of the decision boundary and an overall reduction of the mis-classification probability. Note that, had the new data point been less compatible with the first two, one could also have found an *increase* in the mis-classification probability.

3.5 Bayesian Model Selection

Bayesian Model Comparison. In the formalism described so far one chooses (beforehand) a model assumed responsible for having generated the data, followed by an analysis of the likelihood of parameters for this model. This picture can be generalized to include multiple candidate models. Instead of working with the joint distribution $p(D, \boldsymbol{w})$ of data and model parameters, we must switch to the joint distribution $p(D, \boldsymbol{w}, H)$ to find data D, model H and parameters $\boldsymbol{w}$ for model H[9]. The generalized picture then becomes

- Consider an *ensemble* of models H with associated parameter vectors $\boldsymbol{w}$, characterized by a probability distribution $p(H, \boldsymbol{w})$ which evolves during learning.

[9]Note that different models will generally have parameter vectors $\boldsymbol{w}$ with different dimensionality

- Assume that the data D were generated by a system of the form $S(\boldsymbol{\xi}) = f(\boldsymbol{\xi}; \boldsymbol{w}) +$ noise. Calculate the likelihood $p(\boldsymbol{w}, H|D)$ of models and their parameters, given the data.
- Express the desired objects $p(\boldsymbol{w}|D, H)$ in terms of $p(D|\boldsymbol{w}, H)$ (as before) and $p(H|D)$ in terms of $p(D|H)$, where $p(D|H) = \int d\boldsymbol{w}\ p(D|\boldsymbol{w}, H)p(\boldsymbol{w}|H)$.

Learning is a process during which the arrival of data reduces our uncertainty about the 'right' model H and its 'right' parameters $\boldsymbol{w}$ from the *prior distributions* $p(H)$ and $p(\boldsymbol{w}|H)$ to the *posterior distributions* $p(H|D)$ and $p(\boldsymbol{w}|D, H)$. Note that

$$p(\boldsymbol{w}|D, H) = \frac{p(D|\boldsymbol{w}, H)p(\boldsymbol{w}|H)}{\int d\boldsymbol{w}'\ p(\boldsymbol{w}'|H)p(D|\boldsymbol{w}', H)} \tag{3.61}$$

$$p(H|D) = \frac{p(D|H)p(H)}{\sum_{H'} p(D|H')p(H')} \tag{3.62}$$

Generalized Bayesian learning, with multiple models, now works like this:

Stage 1: definitions

Define (i) the parametrized models H, assumed responsible for the data, (ii) the prior distribution $p(H)$ for these models, (iii) the prior distributions $p(\boldsymbol{w}|H)$ of their parameters, and (iv) the data $D = \{(\boldsymbol{\xi}^1, t^1), \ldots, (\boldsymbol{\xi}^p, t^p)\}$.

Stage 2: model translation

Convert the model definition into a probabilistic standard form: specify the likelihood of finding output t upon presentation of input $\boldsymbol{\xi}$, given model H and parameters $\boldsymbol{w}$:

$$\text{model definition in standard form} \qquad p(t|\boldsymbol{\xi}, \boldsymbol{w}, H) \tag{3.63}$$

Stage 3: the posterior distribution

Calculate the *data* likelihood, given model H, $p(D|\boldsymbol{w}, H) = \prod_{\mu=1}^{p} p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w}, H)$. From this follow the desired posterior parameter and model distributions

$$p(\boldsymbol{w}|D, H) = \frac{p(\boldsymbol{w}|H) \prod_{\mu=1}^{p} p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w}, H)}{\int d\boldsymbol{w}'\ p(\boldsymbol{w}'|H) \prod_{\mu=1}^{p} p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w}', H)} \tag{3.64}$$

$$p(H|D) = \frac{p(H) \int d\boldsymbol{w}\ \prod_{\mu=1}^{p} p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w}, H)p(\boldsymbol{w}|H)}{\sum_{H'} p(H') \int d\boldsymbol{w}\ \prod_{\mu=1}^{p} p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w}, H')p(\boldsymbol{w}|H')} \tag{3.65}$$

Stage 4: prediction

The residual uncertainty in the choice of model H and parameters $\boldsymbol{w}$ generates the uncertainty in predictions. Prediction of the output t corresponding to input $\boldsymbol{\xi}$, given our observation of the data D and our choice of model set, takes the probabilistic form:

$$p(t|\boldsymbol{\xi}, D) = \sum_{H} p(H|D) \int d\boldsymbol{w}\ p(t|\boldsymbol{\xi}, \boldsymbol{w}, H)p(\boldsymbol{w}|D, H) \tag{3.66}$$

As an alternative to stage 4, which describes the final output statistics as a weighted average over all models under consideration, one could also simply select the most probable model $H^\star$, defined as $p(H^\star|D) = \max_H p(H|D)$. This boils down to finding

$$\max_H \{p(H)p(D|H)\} = \max_H \left\{ p(H) \int d\boldsymbol{w} \prod_{\mu=1}^{p} p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w}, H) p(\boldsymbol{w}|H) \right\} \tag{3.67}$$

or, equivalently, can be done by comparing models pair-wise via the ratios

$$\frac{p(H|D)}{p(H'|D)} = \frac{p(D|H)}{p(D|H')} \frac{p(H)}{p(H')} \tag{3.68}$$

This can then be followed by the ordinary Bayesian parameter analysis for a *single* model $H^\star$, as described in the previous sections.

Application to Hyper-parameter Selection. We note that the above reasoning can also be applied to the hyper-parameter selection problem, since systems which differ in the choice of hyper-parameters (rather than in architecture) can be regarded as different candidate models in the sense above. For instance, to describe the family of systems (3.19)

$$p(t|\boldsymbol{\xi}, \boldsymbol{w}, \beta) = \left[\frac{2\pi}{\beta}\right]^{-\frac{1}{2}} e^{-\frac{1}{2}\beta[t-f(\boldsymbol{\xi};\boldsymbol{w})]^2} \qquad p(\boldsymbol{w}|\alpha) = \left[\frac{2\pi}{\alpha}\right]^{-\frac{N}{2}} e^{-\frac{1}{2}\alpha \boldsymbol{w}^2} \tag{3.69}$$

with two hyper-parameters (α, β), we can simply make the replacement $H \to (\alpha, \beta)$ in our above formulae. The specific form of (3.69), for which e.g. $p(t|\boldsymbol{\xi}, \boldsymbol{w}, \alpha, \beta)$ is independent of α, and $p(\boldsymbol{w}|\alpha, \beta)$ is independent of β, also generates several simplifications:

- Consider an *ensemble* of models of the form (3.69), characterized by a probability distribution $p(\alpha, \beta, \boldsymbol{w})$ which evolves during learning.
- Assume that the data D were generated by a system of the form (3.69). Calculate the likelihood $p(\boldsymbol{w}, \alpha, \beta|D)$ of its parameters and hyper-parameters, given the data.
- Express the desired objects $p(\boldsymbol{w}|D, \alpha, \beta)$ in terms of $p(D|\boldsymbol{w}, \alpha, \beta)$, and $p(\alpha, \beta|D)$ in terms of $p(D|\alpha, \beta)$, where $p(D|\alpha, \beta) = \int d\boldsymbol{w}\ p(D|\boldsymbol{w}, \beta) p(\boldsymbol{w}|\alpha)$.

Learning is a process during which the arrival of data reduces our uncertainty about the 'right' hyper-parameters (α, β) and the 'right' parameters $\boldsymbol{w}$ from the *prior distributions* $p(\alpha, \beta)$ and $p(\boldsymbol{w}|\alpha, \beta)$ to the *posterior distributions* $p(\alpha, \beta|D)$ and $p(\boldsymbol{w}|D, \alpha, \beta)$, according to

$$p(\boldsymbol{w}|D, \alpha, \beta) = \frac{p(D|\boldsymbol{w}, \beta) p(\boldsymbol{w}|\alpha)}{\int d\boldsymbol{w}'\ ) p(D|\boldsymbol{w}', \beta) p(\boldsymbol{w}'|\alpha)} \tag{3.70}$$

$$p(\alpha, \beta|D) = \frac{p(D|\alpha, \beta) p(\alpha, \beta)}{\int d\alpha' d\beta'\ p(D|\alpha', \beta') p(\alpha', \beta')} \tag{3.71}$$

Generalized Bayesian learning, including learning of hyper-parameters, now works like this :

Stage 1: definitions

Define (i) the function $f(\boldsymbol{\xi}; \boldsymbol{w})$ in the parametrized model (3.69), (ii) the prior distribution $p(\alpha, \beta)$ for its hyper-parameters, and (iii) the data $D = \{(\boldsymbol{\xi}^1, t^1), \ldots, (\boldsymbol{\xi}^p, t^p)\}$.

Stage 2: the posterior distribution

Calculate the *data* likelihood, given (α, β) and $\boldsymbol{w}$: $p(D|\boldsymbol{w}, \beta) = \prod_{\mu=1}^{p} p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w}, \beta)$. From this follow the desired posterior distributions

$$p(\boldsymbol{w}|D, \alpha, \beta) = \frac{e^{-\frac{1}{2}\alpha \boldsymbol{w}^2 - \frac{1}{2}\beta \sum_{\mu=1}^{p} [t^\mu - f(\boldsymbol{\xi}^\mu; \boldsymbol{w})]^2}}{\int d\boldsymbol{w}' \; e^{-\frac{1}{2}\alpha \boldsymbol{w}'^2 - \frac{1}{2}\sum_{\mu=1}^{p} \beta [t^\mu - f(\boldsymbol{\xi}^\mu; \boldsymbol{w}')]^2}} \tag{3.72}$$

$$p(\alpha, \beta|D) = \frac{\frac{p(\alpha,\beta)}{\sqrt{\alpha^N \beta^p}} \int d\boldsymbol{w} \; e^{-\frac{1}{2}\alpha \boldsymbol{w}^2 - \frac{1}{2}\beta \sum_{\mu=1}^{p} [t^\mu - f(\boldsymbol{\xi}^\mu; \boldsymbol{w})]^2}}{\int d\alpha' d\beta' \; \frac{p(\alpha',\beta')}{\sqrt{\alpha'^N \beta'^p}} \int d\boldsymbol{w} \; e^{-\frac{1}{2}\alpha' \boldsymbol{w}^2 - \frac{1}{2}\beta' \sum_{\mu=1}^{p} [t^\mu - f(\boldsymbol{\xi}^\mu; \boldsymbol{w})]^2}} \tag{3.73}$$

Stage 3: prediction

Prediction of the output t corresponding to a new input $\boldsymbol{\xi}$, given our observation of the data D and our choice of model family (3.69), takes the probabilistic form:

$$p(t|\boldsymbol{\xi}, D) = \int d\alpha d\beta \; p(\alpha, \beta|D) \left[\frac{(2\pi)^{N+1}}{\alpha^N \beta} \right]^{-\frac{1}{2}} \int d\boldsymbol{w} \; p(\boldsymbol{w}|D, \alpha, \beta) \; e^{-\frac{1}{2}\alpha \boldsymbol{w}^2 - \frac{1}{2}\beta [t - f(\boldsymbol{\xi}; \boldsymbol{w})]^2} \tag{3.74}$$

Again the learning problem has in principle been reduced to calculating integrals. However, one will now have to think about how to choose the hyper-parameter prior $p(\alpha, \beta)$.

Model Selection – Occam's Razor. The Bayesian model comparison procedure, followed by model selection according to (3.67) or (3.68), will automatically lead to the selection of the simplest possible model $H^\star$ which can account for the data D (modulo prejudices embodied in the prior $p(H)$). This desirable action is referred to as implementing 'Occam's Razor'[10] To see how it comes about we return to (3.68), and remove the effect of prejudice by putting $p(H) = p(H')$ for all models $\{H, H'\}$ under consideration. This gives

$$\frac{p(H|D)}{p(H'|D)} = \frac{p(D|H)}{p(D|H')} \tag{3.75}$$

To simplify the argument we will take the collection of possible data sets D to be discrete and countable. Let us compare the following three models:

$$\begin{aligned}
&\text{model H}_1: \quad && p(D|H_1) = 0 \\
&\text{model H}_2: \quad && p(D|H_2) > 0, \qquad p(D'|H_2) = 0 \;\; \text{for all} \;\; D' \neq D \\
&\text{model H}_3: \quad && p(D|H_3) > 0, \qquad p(D'|H_3) > 0 \;\; \text{for some} \;\; D' \neq D
\end{aligned}$$

Model 1 cannot explain the data D. Both model 2 and model 3 can explain these data, but 3 is more complex than 2 because it can explain (by a suitable choice of parameters) a greater variety of possible data (not just D). Hence model 2 is the *simplest* model which can account for data D. Insertion into (3.75) reveals

$$\frac{p(H_1|D)}{p(H_2|D)} = \frac{p(H_1|D)}{p(H_3|D)} = 0 \qquad \frac{p(H_2|D)}{p(H_3|D)} = \frac{p(D|H_2)}{p(D|H_3)} \tag{3.76}$$

[10] After a monk, William of Occam, who is claimed to have first proposed the general philosophical principle that one should always select the *simplest* possible explanation for an observed phenomenon.

One now always has $\sum_{D'} p(D'|H) = 1$ (marginal distributions are normalized, for any model H). From this it follows for the above examples that $p(D|H_2) = 1 - \sum_{D' \neq D} p(D'|H_2) = 1$ and $p(D|H_3) = 1 - \sum_{D' \neq D} p(D'|H_3) < 1$. Hence we find in (3.76) that $p(H_2|D) > p(H_3|D) > p(H_1|D)$, and that the Bayesian procedure instructs us to select model 2.

Example. We close with a simple example, to illustrate the action of 'Occam's Razor'. The task is to learn a binary classification of the inputs $\xi \in \{0,1\}$ to the targets $t \in \{0,1\}$. There are four such classifications possible, i.e. four possible data sets with $p = 2$:

$$D_A = \{(0,1),(1,0)\} \qquad D_B = \{(0,0),(1,0)\} \qquad D_C = \{(0,1),(1,1)\} \qquad D_D = \{(0,0),(1,1)\}$$

Given one of these data sets, we have to choose between two deterministic parametrized candidate models H_1 and H_2, without initial prejudice, i.e. $p(H_1) = p(H_2) = \frac{1}{2}$:

$$H_1: \qquad t(\xi) = \theta[w_1] \tag{3.77}$$

$$H_2: \qquad t(\xi) = \theta[w_1 + w_2 \xi] \tag{3.78}$$

with $\boldsymbol{w} = (w_1, w_2) \in \Re^2$ and $\theta[z]$ denoting the step function. As a prior parameter distribution we take $p(\boldsymbol{w}) = (2\pi)^{-1} e^{-\frac{1}{2}\boldsymbol{w}^2}$. Clearly model 2 is more complex than 1. We will select our model by working out the ratio (3.68), this requires calculating $p(D|H_1) = \int dw_1\ p(w_1) p(D|w_1, H_1)$ and $p(D|H_2) = \int d\boldsymbol{w}\ p(\boldsymbol{w}) p(D|\boldsymbol{w}, H_2)$.

First we focus on model H_1. Here we have $t(\xi) = 1$ for all ξ if $w_1 > 0$, and $t(\xi) = 0$ for all ξ if $w_1 < 0$. Hence we simply find

$$p(D_A|w_1, H_1) = p(D_D|w_1, H_1) = 0, \qquad p(D_B|w_1, H_1) = \theta[-w_1], \qquad p(D_C|w_1, H_1) = \theta[w_1]$$

This gives

$$\begin{array}{l} p(D_A|H_1) = 0 \\ p(D_B|H_1) = \int_{-\infty}^{0} dw_1\ p(w_1) = \frac{1}{2} \\ p(D_C|H_1) = \int_{0}^{\infty} dw_1\ p(w_1) = \frac{1}{2} \\ p(D_A|H_1) = 0 \end{array} \tag{3.79}$$

Next we turn to the two-parameter model H_2. Here we have $t(0) = \theta[w_1]$ and $t(1) = \theta[w_1 + w_2]$. Hence we find

$$p(D_A|\boldsymbol{w}, H_2) = \theta[w_1]\ \theta[-(w_1+w_2)] \qquad p(D_B|\boldsymbol{w}, H_2) = \theta[-w_1]\ \theta[-(w_1+w_2)]$$

$$p(D_C|\boldsymbol{w}, H_2) = \theta[w_1]\ \theta[w_1+w_2] \qquad p(D_D|\boldsymbol{w}, H_2) = \theta[-w_1]\ \theta[w_1+w_2]$$

This leads to, with the short-hand $\chi = (2\pi)^{-1} \int_0^\infty dw_1 e^{-\frac{1}{2}w_1^2} \int_{w_1}^\infty dw_2\ e^{-\frac{1}{2}w_2^2} > 0$ and upon using the symmetries of the prior $p(\boldsymbol{w})$:

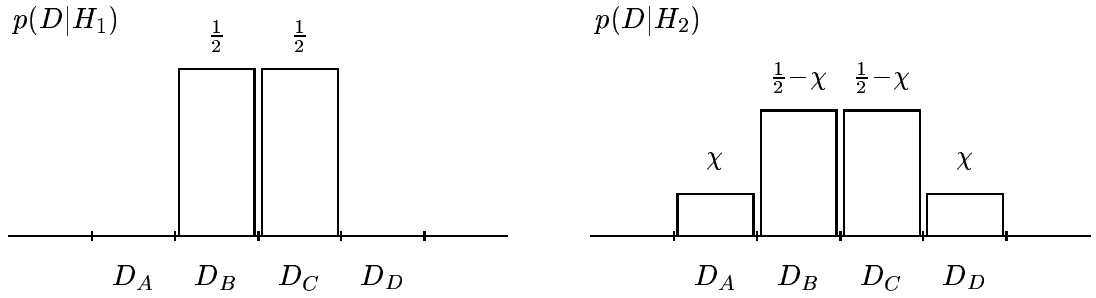
$$p(D_A|H_2) = \int_0^\infty dw_1 \int_{-\infty}^{-w_1} dw_2\ p(\boldsymbol{w}) = \chi \tag{3.80}$$

$$p(D_B|H_2) = \int_{-\infty}^0 dw_1 \int_{-\infty}^{-w_1} dw_2\ p(\boldsymbol{w}) = \int_0^\infty dw_1 \int_{-\infty}^{w_1} dw_2\ p(\boldsymbol{w}) = \frac{1}{2} - \chi \tag{3.81}$$

$$p(D_C|H_2) = \int_0^\infty dw_1 \int_{-w_1}^\infty dw_2\ p(\boldsymbol{w}) = \frac{1}{2} - \chi \tag{3.82}$$

$$p(D_D|H_2) = \int_{-\infty}^0 dw_1 \int_{-w_1}^\infty dw_2\ p(\boldsymbol{w}) = \int_0^\infty dw_1 \int_{w_1}^\infty dw_2\ p(\boldsymbol{w}) = \chi \tag{3.83}$$

Finally we combine the results (3.79) and (3.80-3.83) into the following picture:



It follows from (3.68) that

$$\frac{p(H_1|D_A)}{p(H_2|D_A)} = \frac{p(H_1|D_D)}{p(H_2|D_D)} = 0, \qquad \frac{p(H_1|D_B)}{p(H_2|D_B)} = \frac{p(H_1|D_C)}{p(H_2|D_C)} = \frac{1/2}{1/2-\chi} > 1$$

We conclude that when observing data D_A or D_D we must select model H_2 (which would be the only candidate to explain these data), but that when observing data D_B or D_C we must select model H_1 (both models can explain these data, but H_1 wins simply because it is the simpler explanation).

3.6 Practicalities: Measuring Curvature

When learning by gradient descent on the surface $S(\boldsymbol{w}, D)$ (3.16), and using simple expressions such as (3.38) to assign error bars to neural network predictions, one needs to know the curvature matrix $\boldsymbol{A}$ as given in (3.23) at the minimum $\boldsymbol{w}_{\rm MP}$ of $S(\boldsymbol{w}, D)$. In principle this matrix can be calculated directly from the model definitions, but for many-parameter and/or multi-layer networks this is difficult. Here we discuss an alternative, based on the idea that the degree of curvature around the minimum will have a direct effect on the gradient descent dynamics at the minimum when such dynamics is equipped with additive noise. As a result the curvature matrix can be extracted from a measurement of the fluctuations.

Let us assume we have arrived by gradient descent at the minimum $\boldsymbol{w}_{\rm MP}$ of $S(\boldsymbol{w}, D)$. We now replace the (discretized) gradient descent dynamics by the following noisy version:

$$w_i(t+\delta) = w_i(t) - \delta . \frac{\partial S(\boldsymbol{w}, D)}{\partial w_i} + \sqrt{2\delta}\ \eta_i(t) \tag{3.84}$$

where the $\eta_i(t)$ are independently distributed zero-average Gaussian random variables, with $\langle \eta_i^2(t) \rangle = 1$, and with $0 < \delta \ll 1$. The parameter dynamics is now a stochastic process. Let

us define averages and covariances of this process as

$$\overline{w}_i(t) = \langle w_i(t) \rangle \qquad C_{ij}(t) = \langle [w_i(t) - \overline{w}_i(t)][w_j(t) - \overline{w}_j(t)] \rangle \tag{3.85}$$

Their dynamics follow directly from (3.84):

$$\begin{aligned}
\overline{w}_i(t+\delta) &= \overline{w}_i(t) - \delta \, \langle \frac{\partial S(\boldsymbol{w}, D)}{\partial w_i} \rangle \\
C_{ij}(t+\delta) &= \langle w_i(t+\delta) w_j(t+\delta) \rangle - \overline{w}_i(t+\delta) \overline{w}_j(t+\delta) \\
&= \langle \left[w_i(t) - \delta . \frac{\partial S(\boldsymbol{w}, D)}{\partial w_i} \right] \left[w_j(t) - \delta . \frac{\partial S(\boldsymbol{w}, D)}{\partial w_j} \right] \rangle + 2\delta . \delta_{ij} - \overline{w}_i(t+\delta) \overline{w}_j(t+\delta) \\
&= C_{ij}(t) + \delta \left\{ 2\delta_{ij} - \langle w_i(t) \frac{\partial S(\boldsymbol{w}, D)}{\partial w_j} \rangle + \overline{w}_i(t) \langle \frac{\partial S(\boldsymbol{w}, D)}{\partial w_j} \rangle \right. \\
&\qquad \left. - \langle w_j(t) \frac{\partial S(\boldsymbol{w}, D)}{\partial w_i} \rangle + \overline{w}_j(t) \langle \frac{\partial S(\boldsymbol{w}, D)}{\partial w_i} \rangle \right\} + \mathcal{O}(\delta^2)
\end{aligned}$$

Since $\delta \ll 1$, the system can only make infinitesimal excursions $\boldsymbol{w} = \boldsymbol{w}_{\rm MP} + \mathcal{O}(\sqrt{\delta})$ away from the minimum $\boldsymbol{w}_{\rm MP}$, hence we may approximate $S(\boldsymbol{w}, D)$ safely by the quadratic approximation (3.22)

$$S(\boldsymbol{w}, D) = S(\boldsymbol{w}_{\rm MP}, D) + \frac{1}{2} (\boldsymbol{w} - \boldsymbol{w}_{\rm MP}) \cdot \boldsymbol{A} (\boldsymbol{w} - \boldsymbol{w}_{\rm MP}) + \mathcal{O}(\delta^{3/2})$$

This implies that

$$\frac{\partial S(\boldsymbol{w}, D)}{\partial w_i} = \sum_j A_{ij} (w_j - w_{{\rm MP}, j}) + \mathcal{O}(\delta^{3/2})$$

Thus:

$$\begin{aligned}
\overline{\boldsymbol{w}}(t+\delta) &= \overline{\boldsymbol{w}}(t) - \delta \; \boldsymbol{A} (\overline{\boldsymbol{w}} - \boldsymbol{w}_{\rm MP}) + \mathcal{O}(\delta^{3/2}) \\
C_{ij}(t+\delta) &= C_{ij}(t) + \delta \left\{ 2\delta_{ij} - \langle w_i(t) \sum_k A_{jk} (w_k(t) - w_{{\rm MP},k}) \rangle + \overline{w}_i(t) \sum_k A_{jk} (\overline{w}_k - w_{{\rm MP},k}) \right. \\
&\qquad \left. - \langle w_j(t) \sum_k A_{ik} (w_k(t) - w_{{\rm MP},k}) \rangle + \overline{w}_j(t) \sum_k A_{ik} (\overline{w}_k - w_{{\rm MP},k}) \right\} + \mathcal{O}(\delta^{3/2})
\end{aligned}$$

We now arrive at the simple result (using the symmetry of both $\boldsymbol{A}$ and $\boldsymbol{C}$):

$$\boldsymbol{C}(t+\delta) = \boldsymbol{C}(t) + \delta \, [2\mathbf{I} - \boldsymbol{C}\boldsymbol{A} - \boldsymbol{A}\boldsymbol{C}] + \mathcal{O}(\delta^{3/2}) \tag{3.86}$$

One can show that, provided $\delta \ll 1$ (more specifically: provided $\delta < 2/a_i$ for all eigenvalues a_i of the matrix $\boldsymbol{A}$), the leading order of (3.86) will evolve towards the stationary state $\boldsymbol{C} = \boldsymbol{A}^{-1} + \mathcal{O}(\sqrt{\delta})$, and the vector $\overline{\boldsymbol{w}}$ will evolve towards $\boldsymbol{w}_{\rm MP}$. Hence the desired curvature matrix $\boldsymbol{A}$ can be measured in equilibrium, by choosing δ sufficiently small, according to

$$\delta \to 0 : \qquad \boldsymbol{A} = \boldsymbol{C}^{-1} \qquad C_{ij} = \lim_{t \to \infty} \langle \langle [w_i - w_{{\rm MP},i}][w_j - w_{{\rm MP},j}] \rangle \tag{3.87}$$

Chapter 4

Gaussian Processes

We return to the general Bayesian formalism for a single model, characterized by the likelihood $p(t|\boldsymbol{\xi}, \boldsymbol{w})$ to find output t, given input $\boldsymbol{\xi}$ and parameters $\boldsymbol{w}$. The data are, as always, given by p pairs of example inputs $\boldsymbol{\xi}^\mu$ with corresponding (possibly noisy) outputs t^μ:

$$D = \{(\boldsymbol{\xi}^1, t^1), \ldots, (\boldsymbol{\xi}^p, t^p)\}$$

For reasons to become clear below we will now write the data set D in expressions such as $p(t|\boldsymbol{\xi}, D) = \int d\boldsymbol{w}\ p(t|\boldsymbol{\xi}, \boldsymbol{w}) p(\boldsymbol{w}|D)$ as the full specification of the constituent pairs $(\boldsymbol{\xi}^\mu, t^\mu)$:

$$p(t|\boldsymbol{\xi}, \boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, t^1, \ldots, t^p) = \int d\boldsymbol{w}\ p(t|\boldsymbol{\xi}, \boldsymbol{w}) p(\boldsymbol{w}|\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, t^1, \ldots, t^p) \tag{4.1}$$

Multiplication of this expression by $p(t^1, \ldots, t^p|\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p)$ gives:

$$\begin{aligned} p(t, t^1, \ldots, t^p|\boldsymbol{\xi}, \boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p) &= \int d\boldsymbol{w}\ p(t|\boldsymbol{\xi}, \boldsymbol{w}) p(\boldsymbol{w}, t^1, \ldots, t^p|\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p) \\ &= \int d\boldsymbol{w}\ p(\boldsymbol{w}) p(t|\boldsymbol{\xi}, \boldsymbol{w}) p(t^1, \ldots, t^p|\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{w}) \end{aligned}$$

Upon renaming the target input and output as $(\boldsymbol{\xi}^0, t^0) = (\boldsymbol{\xi}, t)$, we arrive at the simple and transparent expression

$$p(t^0, \ldots, t^p|\boldsymbol{\xi}^0, \ldots, \boldsymbol{\xi}^p) = \int d\boldsymbol{w}\ p(\boldsymbol{w}) \prod_{\mu=0}^{p} p(t^\mu|\boldsymbol{\xi}^\mu, \boldsymbol{w}) \tag{4.2}$$

We now see that the desired prediction output distribution (4.1) can be simply obtained as the fraction

$$p(t^0|\boldsymbol{\xi}^0, \ldots, \boldsymbol{\xi}^p, t^1 \ldots, t^p) = \frac{p(t^0, \ldots, t^p|\boldsymbol{\xi}^0, \ldots, \boldsymbol{\xi}^p)}{p(t^1, \ldots, t^p|\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p)} \tag{4.3}$$

or, written slightly differently:

$$p(t^0|\boldsymbol{\xi}^0, \ldots, \boldsymbol{\xi}^p, t^1, \ldots, t^p) = \frac{p(t^0, t^1, \ldots, t^p|\boldsymbol{\xi}^0, \ldots, \boldsymbol{\xi}^p)}{\int dt\ p(t, t^1, \ldots, t^p|\boldsymbol{\xi}^0, \ldots, \boldsymbol{\xi}^p)}$$

Gaussian processes are information processing systems of the above form $p(t|\boldsymbol{\xi}, \boldsymbol{w})$ such that the key object (4.2) is a (multivariate) Gaussian distribution of $\{t^0, \ldots, t^p\}$, with moments generally depending on $\{\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p\}$ in a non-trivial and model-specific way[1].

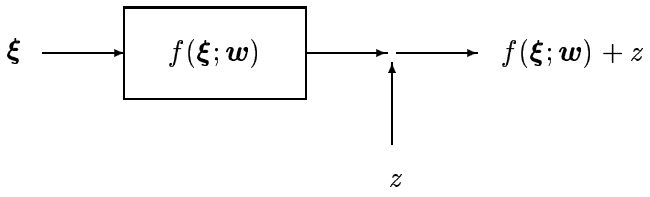
[1]Note: in literature one can sometimes find slightly more broad definitions, where upon adding non-Gaussian output noise to Gaussian-distributed targets one would still speak about a Gaussian process.

4.1 Examples of Networks Reducing to Gaussian Processes

In order to show that Gaussian processes are not just found upon choosing far-fetched and/or trivial models, we will first discuss two examples of systems reducing to Gaussian processes. Both are of the form

$$p(t|\boldsymbol{\xi},\boldsymbol{w}) = \left[\frac{\beta}{2\pi}\right]^{\frac{1}{2}} e^{-\frac{1}{2}\beta[t-f(\boldsymbol{\xi};\boldsymbol{w})]^2}, \qquad p(\boldsymbol{w}) = \left[\frac{\alpha}{2\pi}\right]^{\frac{N}{2}} e^{-\frac{1}{2}\alpha\boldsymbol{w}^2} \tag{4.4}$$

i.e. composed of a deterministic parametrized function $f(\boldsymbol{w};\boldsymbol{\xi})$, with a Gaussian prior and Gaussian output noise.



Radial Basis Function Networks. RBF systems with Gaussian output noise and priors, which are members of the class (4.4), are described by a deterministic part of the form $f(\boldsymbol{w};\boldsymbol{\xi}) = \boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{\xi})$, with $\boldsymbol{w}\in\Re^N$ and with N functions $\phi_i(\boldsymbol{\xi})$ to be specified[2]. We now work out the joint output distribution (4.2) for data and target. With a modest amount of foresight we define

$$A_{ij} = \alpha\delta_{ij} + \beta\sum_{\mu=0}^{p}\phi_i(\boldsymbol{\xi}^\mu)\phi_j(\boldsymbol{\xi}^\mu) \qquad b_i = \beta\sum_{\mu=0}^{p} t^\mu\phi_i(\boldsymbol{\xi}^\mu) \tag{4.5}$$

Insertion of (4.4) into (4.2), with $f(\boldsymbol{\xi};\boldsymbol{w}) = \boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{\xi})$, and subsequent usage of the short-hands (4.5) and the Gaussian integrals in appendix B, gives

$$\begin{aligned}
p(t^0,\ldots,t^p|\boldsymbol{\xi}^0,\ldots,\boldsymbol{\xi}^p) &= \left[\frac{\alpha}{2\pi}\right]^{\frac{N}{2}}\left[\frac{\beta}{2\pi}\right]^{\frac{p+1}{2}}\int d\boldsymbol{w}\; e^{-\frac{1}{2}\alpha\boldsymbol{w}^2-\frac{1}{2}\beta\sum_{\mu=0}^{p}[t^\mu-\boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{\xi}^\mu)]^2}\\
&= \left[\frac{\alpha}{2\pi}\right]^{\frac{N}{2}}\left[\frac{\beta}{2\pi}\right]^{\frac{p+1}{2}} e^{-\frac{1}{2}\beta\sum_{\mu=0}^{p}[t^\mu]^2}\int d\boldsymbol{w}\; e^{-\frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{A}\boldsymbol{w}+\boldsymbol{w}\cdot\boldsymbol{b}}\\
&= \left[\frac{\alpha}{2\pi}\right]^{\frac{N}{2}}\left[\frac{\beta}{2\pi}\right]^{\frac{p+1}{2}} e^{-\frac{1}{2}\beta\sum_{\mu=0}^{p}[t^\mu]^2}\;\frac{(2\pi)^{N/2}}{\sqrt{\det\boldsymbol{A}}}e^{\frac{1}{2}\boldsymbol{b}\cdot\boldsymbol{A}^{-1}\boldsymbol{b}}\\
&= \left[\frac{\alpha^N\beta^{p+1}}{(2\pi)^{p+1}\det\boldsymbol{A}}\right]^{\frac{1}{2}} e^{-\frac{1}{2}\beta\sum_{\mu=0}^{p}[t^\mu]^2+\frac{1}{2}\beta^2[\sum_{\mu=0}^{p}t^\mu\boldsymbol{\phi}(\boldsymbol{\xi}^\mu)]\cdot\boldsymbol{A}^{-1}[\sum_{\mu=0}^{p}t^\mu\boldsymbol{\phi}(\boldsymbol{\xi}^\mu)]}\\
&= \left[\frac{\alpha^N\beta^{p+1}}{(2\pi)^{p+1}\det\boldsymbol{A}}\right]^{\frac{1}{2}} e^{-\frac{1}{2}\sum_{\mu\nu=0}^{p}t^\mu t^\nu\left[\beta\delta_{\mu\nu}-\beta^2\boldsymbol{\phi}(\boldsymbol{\xi}^\mu)\cdot\boldsymbol{A}^{-1}\boldsymbol{\phi}(\boldsymbol{\xi}^\nu)\right]}
\end{aligned} \tag{4.6}$$

The output distribution (4.6) is clearly a (zero-average) Gaussian one, characterized fully by the covariance matrix $\boldsymbol{C}[\boldsymbol{\xi}^0,\ldots,\boldsymbol{\xi}^p]$ which is defined as:

$$C_{\mu\nu}[\boldsymbol{\xi}^0,\ldots,\boldsymbol{\xi}^p] = \int dt^0\ldots dt^p\; t^\mu t^\nu\; p(t^0,\ldots,t^p|\boldsymbol{\xi}^0,\ldots,\boldsymbol{\xi}^p)$$

[2]Note: the model family (4.4) with $f(\boldsymbol{\xi};\boldsymbol{w}) = \boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{\xi})$ is in fact more general than just the set of all RBF networks, since we need not put RBF-type restrictions on the form of the functions $\phi_i(\boldsymbol{\xi})$.

According to (4.6) we have (see also appendix B):

$$(\boldsymbol{C}^{-1})_{\mu\nu}[\boldsymbol{\xi}^0,\ldots,\boldsymbol{\xi}^p] = \beta\delta_{\mu\nu} - \beta^2\boldsymbol{\phi}(\boldsymbol{\xi}^\mu)\cdot\boldsymbol{A}^{-1}\boldsymbol{\phi}(\boldsymbol{\xi}^\nu)$$

Inversion of the above matrix gives

$$C_{\mu\nu}[\boldsymbol{\xi}^0,\ldots,\boldsymbol{\xi}^p] = \frac{1}{\beta}\delta_{\mu\nu} + \frac{1}{\alpha}\boldsymbol{\phi}(\boldsymbol{\xi}^\mu)\cdot\boldsymbol{\phi}(\boldsymbol{\xi}^\nu) \tag{4.7}$$

This can be verified by insertion:

$$\beta\sum_\rho C_{\mu\rho}[\ldots]\left[\delta_{\rho\nu}-\beta\boldsymbol{\phi}(\boldsymbol{\xi}^\rho)\cdot\boldsymbol{A}^{-1}\boldsymbol{\phi}(\boldsymbol{\xi}^\nu)\right] = \sum_\rho\left[\delta_{\mu\rho}+\frac{\beta}{\alpha}\boldsymbol{\phi}(\boldsymbol{\xi}^\mu)\cdot\boldsymbol{\phi}(\boldsymbol{\xi}^\rho)\right]\left[\delta_{\rho\nu}-\beta\boldsymbol{\phi}(\boldsymbol{\xi}^\rho)\cdot\boldsymbol{A}^{-1}\boldsymbol{\phi}(\boldsymbol{\xi}^\nu)\right]$$

$$= \delta_{\mu\nu} - \beta\boldsymbol{\phi}(\boldsymbol{\xi}^\mu)\cdot\boldsymbol{A}^{-1}\boldsymbol{\phi}(\boldsymbol{\xi}^\nu) + \frac{\beta}{\alpha}\boldsymbol{\phi}(\boldsymbol{\xi}^\mu)\cdot\boldsymbol{\phi}(\boldsymbol{\xi}^\nu) - \frac{\beta^2}{\alpha}\sum_\rho[\boldsymbol{\phi}(\boldsymbol{\xi}^\mu)\cdot\boldsymbol{\phi}(\boldsymbol{\xi}^\rho)][\boldsymbol{\phi}(\boldsymbol{\xi}^\rho)\cdot\boldsymbol{A}^{-1}\boldsymbol{\phi}(\boldsymbol{\xi}^\nu)]$$

$$= \delta_{\mu\nu} - \frac{\beta}{\alpha}\sum_{ij}\phi_i(\boldsymbol{\xi}^\mu)\phi_j(\boldsymbol{\xi}^\mu)\left[\alpha(\boldsymbol{A}^{-1})_{ij} - \delta_{ij} + \beta\sum_k\sum_\rho\phi_i(\boldsymbol{\xi}^\rho)\phi_k(\boldsymbol{\xi}^\rho)(\boldsymbol{A}^{-1})_{kj}\right]$$

$$= \delta_{\mu\nu} - \frac{\beta}{\alpha}\sum_{ij}\phi_i(\boldsymbol{\xi}^\mu)\phi_j(\boldsymbol{\xi}^\mu)\left[\alpha(\boldsymbol{A}^{-1})_{ij} - \delta_{ij} + \sum_k[A_{ik}-\alpha\delta_{ik}](\boldsymbol{A}^{-1})_{kj}\right] = \delta_{\mu\nu}$$

This confirms (4.7).

Linear-Output Two-Layer Perceptrons. Let us next turn a less trivial model example, still of the form (4.4), which is also found to reduce to a Gaussian process in a specific limit. We take a two-layer network with sigmoidal transfer functions $g(u)$ in the hidden layer (of size L) and a linear output neuron, i.e. $f(\boldsymbol{\xi};\boldsymbol{w},\boldsymbol{J}) = \sum_{i=1}^L w_i\ g(\sum_{j=1}^N J_{ij}\xi_j)$:

$$p(t|\boldsymbol{\xi},\boldsymbol{w},\boldsymbol{J}) = \left[\frac{\beta}{2\pi}\right]^{\frac{1}{2}} e^{-\frac{1}{2}\beta[t-\sum_{i=1}^L w_i g(\sum_{j=1}^N J_{ij}\xi_j)]^2} \tag{4.8}$$

$$p(\boldsymbol{w}) = \left[\frac{\alpha_w L}{2\pi}\right]^{\frac{L}{2}} e^{-\frac{1}{2}\alpha_w L\boldsymbol{w}^2}, \qquad p(\boldsymbol{J}) = \left[\frac{\alpha_J}{2\pi}\right]^{\frac{NL}{2}} e^{-\frac{1}{2}\alpha_J\sum_{i=1}^L\sum_{j=1}^N J_{ij}^2} \tag{4.9}$$

The width of the prior for the L hidden-to-output weights $\{w_i\}$ has been re-scaled by a factor $1/\sqrt{L}$, in order to find a well-defined limit $L\to\infty$ (corresponding to an infinitely large hidden layer) later. This model has three hyper-parameters: α_w, α_J and β. Insertion of the above definitions into (4.2) is now found to give

$$p(t^0,\ldots,t^p|\boldsymbol{\xi}^0,\ldots,\boldsymbol{\xi}^p) = \left[\frac{\alpha_w L}{2\pi}\right]^{\frac{L}{2}}\left[\frac{\alpha_J}{2\pi}\right]^{\frac{NL}{2}}\left[\frac{\beta}{2\pi}\right]^{\frac{p+1}{2}}$$

$$\times\int d\boldsymbol{w}d\boldsymbol{J}\ e^{-\frac{1}{2}\alpha_w L\boldsymbol{w}^2-\frac{1}{2}\alpha_J\sum_{i=1}^L\sum_{j=1}^N J_{ij}^2-\frac{1}{2}\beta\sum_{\mu=0}^p[t^\mu-\sum_{i=1}^L w_i g(\sum_{j=1}^N J_{ij}\xi_j^\mu)]^2}$$

It will be advantageous to introduce $p+1$ additional integrals to get rid of the squares in the exponent, via the identity

$$e^{-\frac{1}{2}\sum_{\mu=0}^p K_\mu^2} = (2\pi)^{-\frac{p+1}{2}}\int d\boldsymbol{k}\ e^{-\frac{1}{2}\boldsymbol{k}^2+i\sum_{\mu=0}^p k_\mu K_\mu}$$

with $\boldsymbol{k} = (k_0, \ldots, k_p)$. This allows us to write

$$\begin{aligned} p(t^0, \ldots, t^p | \boldsymbol{\xi}^0, \ldots, \boldsymbol{\xi}^p) &= \left[\frac{\alpha_w L}{2\pi}\right]^{\frac{L}{2}} \left[\frac{\alpha_J}{2\pi}\right]^{\frac{NL}{2}} \left[\frac{\beta}{(2\pi)^2}\right]^{\frac{p+1}{2}} \int d\boldsymbol{k}\ e^{-\frac{1}{2}\boldsymbol{k}^2 + i\sqrt{\beta}\sum_{\mu=0}^{p} k_\mu t_\mu} \\ &\times \int d\boldsymbol{J}\ e^{-\frac{1}{2}\alpha_J \sum_{i=1}^{L}\sum_{j=1}^{N} J_{ij}^2} \prod_{i=1}^{L} \left[\int dw_i\ e^{-\frac{1}{2}\alpha_w L w_i^2 - i w_i \sqrt{\beta}\sum_{\mu=0}^{p} k_\mu g(\sum_{j=1}^{N} J_{ij}\xi_j^\mu)}\right] \\ = \left[\frac{\alpha_w L}{2\pi}\right]^{\frac{L}{2}} &\left[\frac{\alpha_J}{2\pi}\right]^{\frac{NL}{2}} \left[\frac{\beta}{(2\pi)^2}\right]^{\frac{p+1}{2}} \int d\boldsymbol{k}\ e^{-\frac{1}{2}\boldsymbol{k}^2 + i\sqrt{\beta}\sum_{\mu=0}^{p} k_\mu t_\mu} \int d\boldsymbol{J}\ e^{-\frac{1}{2}\alpha_J \sum_{i=1}^{L}\sum_{j=1}^{N} J_{ij}^2} \\ &\times \prod_{i=1}^{L} \left[\int dw\ e^{-\frac{1}{2}\alpha_w L[w + \frac{i\sqrt{\beta}}{\alpha_w L}\sum_{\mu=0}^{p} k_\mu g(\sum_{j=1}^{N} J_{ij}\xi_j^\mu)]^2 - \frac{\beta}{2\alpha_w L}[\sum_{\mu=0}^{p} k_\mu g(\sum_{j=1}^{N} J_{ij}\xi_j^\mu)]^2}\right] \\ = \left[\frac{\alpha_J}{2\pi}\right]^{\frac{NL}{2}} &\left[\frac{\beta}{(2\pi)^2}\right]^{\frac{p+1}{2}} \int d\boldsymbol{k}\ e^{-\frac{1}{2}\boldsymbol{k}^2 + i\sqrt{\beta}\sum_{\mu=0}^{p} k_\mu t_\mu} \\ &\times \int d\boldsymbol{J}\ e^{-\frac{1}{2}\alpha_J \sum_{i=1}^{L}\sum_{j=1}^{N} J_{ij}^2}\ e^{-\frac{\beta}{2\alpha_w}\sum_{\mu\nu=0}^{p} k_\mu k_\nu \left[\frac{1}{L}\sum_{i=1}^{L} g(\sum_{j=1}^{N} J_{ij}\xi_j^\mu) g(\sum_{j=1}^{N} J_{ij}\xi_j^\nu)\right]} \end{aligned}$$

We re-organize the weights $\{J_{ij}\}$ into L vectors $\boldsymbol{v}_i$ of dimension N each, $\boldsymbol{v}_i = (J_{i1}, J_{i2}, \ldots, J_{iN})$:

$$\begin{aligned} p(t^0, \ldots, t^p | \boldsymbol{\xi}^0, \ldots, \boldsymbol{\xi}^p) &= \left[\frac{\alpha_J}{2\pi}\right]^{\frac{NL}{2}} \left[\frac{\beta}{(2\pi)^2}\right]^{\frac{p+1}{2}} \int d\boldsymbol{k}\ e^{-\frac{1}{2}\boldsymbol{k}^2 + i\sqrt{\beta}\sum_{\mu=0}^{p} k_\mu t_\mu} \\ &\times \int \prod_{i=1}^{L} \left[d\boldsymbol{v}_i\ e^{-\frac{1}{2}\alpha_J \boldsymbol{v}_i^2}\right]\ e^{-\frac{\beta}{2\alpha_w}\sum_{\mu\nu=0}^{p} k_\mu k_\nu \left[\frac{1}{L}\sum_{i=1}^{L} g(\boldsymbol{v}_i\cdot\boldsymbol{\xi}^\mu) g(\boldsymbol{v}_i\cdot\boldsymbol{\xi}^\nu)\right]} \end{aligned}$$

If now we take the limit $L \to \infty$ it is clear that, due to the fact that all Gaussian random vectors $\boldsymbol{v}_i$ are distributed identically but also *independently*, we may use the general identity

$$\lim_{L\to\infty} \frac{1}{L}\sum_{i=1}^{L} F(\boldsymbol{v}_i) = \langle F(\boldsymbol{v})\rangle_{\boldsymbol{v}} = \int d\boldsymbol{v}\ p(\boldsymbol{v}) F(\boldsymbol{v})$$

where $p(\boldsymbol{v})$ denotes the distribution of an individual $\boldsymbol{v}_i$, i.e. $p(\boldsymbol{v}) = (\alpha_J/2\pi)^{N/2} e^{-\frac{1}{2}\alpha_J \boldsymbol{v}^2}$. Hence, after a further simplifying transformation $\boldsymbol{k} \to \boldsymbol{k}/\sqrt{\beta}$ we find

$$\begin{aligned} \lim_{L\to\infty} p(t^0, \ldots, t^p | \boldsymbol{\xi}^0, \ldots, \boldsymbol{\xi}^p) &= \int \frac{d\boldsymbol{k}}{(2\pi)^{p+1}}\ e^{i\sum_{\mu=0}^{p} k_\mu t_\mu - \frac{1}{2}\sum_{\mu\nu=0}^{p} k_\mu k_\nu \left[\frac{1}{\beta}\delta_{\mu\nu} + \frac{1}{\alpha_w}\langle g(\boldsymbol{v}\cdot\boldsymbol{\xi}^\mu) g(\boldsymbol{v}\cdot\boldsymbol{\xi}^\nu)\rangle_{\boldsymbol{v}}\right]} \\ &= \frac{e^{-\frac{1}{2}\sum_{\mu\nu=0}^{p} t^\mu (\boldsymbol{C}[\boldsymbol{\xi}^0, \ldots, \boldsymbol{\xi}^p])^{-1}_{\mu\nu} t^\nu}}{\sqrt{(2\pi)^{p+1} \det \boldsymbol{C}[\boldsymbol{\xi}^0, \ldots, \boldsymbol{\xi}^p]}} \end{aligned} \tag{4.10}$$

Again we find a Gaussian distribution (4.10), here characterized by the covariance matrix

$$C_{\mu\nu}[\boldsymbol{\xi}^0, \ldots, \boldsymbol{\xi}^p] = \frac{1}{\beta}\delta_{\mu\nu} + \frac{1}{\alpha_w}\int \frac{d\boldsymbol{z}}{(2\pi)^{N/2}}\ e^{-\frac{1}{2}\boldsymbol{z}^2} g(\frac{\boldsymbol{z}\cdot\boldsymbol{\xi}^\mu}{\sqrt{\alpha_J}})\ g(\frac{\boldsymbol{z}\cdot\boldsymbol{\xi}^\nu}{\sqrt{\alpha_J}}) \tag{4.11}$$

Thus, also two-layer feed-forward networks with sigmoidal transfer functions in the hidden layer and a linear output neuron, and suitably scaled Gaussian priors, are found to become Gaussian processes in the limit of an infinitely large hidden layer.

4.2 Learning and Prediction with Gaussian Processes

Definitions and Rationale. The association of outputs $t(\boldsymbol{\xi}) \in \Re$ to inputs $\boldsymbol{\xi} \in \Re^N$ is said to be governed by a Gaussian process if for any finite selection of p inputs $\{\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p\}$ the conditional joint output distribution $p(t^1, \ldots, t^p | \boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p)$ is Gaussian:

$$p(t^1, \ldots, t^p | \boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p) = \frac{e^{-\frac{1}{2}(\boldsymbol{t}-\boldsymbol{a}[\boldsymbol{\xi}^1,\ldots,\boldsymbol{\xi}^p])\cdot \boldsymbol{C}^{-1}[\boldsymbol{\xi}^1,\ldots,\boldsymbol{\xi}^p](\boldsymbol{t}-\boldsymbol{a}[\boldsymbol{\xi}^1,\ldots,\boldsymbol{\xi}^p])}}{\sqrt{(2\pi)^p \ \det \boldsymbol{C}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p]}} \tag{4.12}$$

with the short-hands $\boldsymbol{t} = (t^1, \ldots, t^p)$, $\boldsymbol{a}[\ldots] = (a_1[\ldots], \ldots, a_p[\ldots])$, and with

$$a_\mu[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p] = \int d\boldsymbol{t} \ p(t^1, \ldots, t^p | \boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p) \ t_\mu \tag{4.13}$$

$$C_{\mu\nu}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p] = \int d\boldsymbol{t} \ p(t^1, \ldots, t^p | \boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p) \ (t_\mu - a_\mu[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p])(t_\nu - a_\nu[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p]) \tag{4.14}$$

We note:

- Gaussian processes are fully characterized by the functions in (4.13,4.14). The detailed dependence of these averages $a_\mu[\ldots]$ and covariances $C_{\mu\nu}[\ldots]$ on the arguments $\{\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p\}$ reflects both structural properties and prior parameter assumptions of the underlying models (see e.g. the explicit derivations in the previous section).
- The rationale behind using Gaussian processes is: (i) it allows us to move away from model specific details (e.g. network architecture and weights) and deal with a reasonably large class of systems in one go, and (ii) we will be able to derive surprisingly simple (and hence CPU efficient) expressions for predicted outputs and associated error bars.

Our formulae hold (by definition) for any value of p. To keep notation simple, it will turn out helpful to refer to the new input $\boldsymbol{\xi}$ and its output t to be predicted (which in the previous section we called $\boldsymbol{\xi}^0$ and t^0) as $\boldsymbol{\xi}^{p+1}$ and t^{p+1}. With the same objective, and since the $\{\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p\}$ occur only as arguments of $\boldsymbol{C}[\ldots]$ and $\boldsymbol{a}[\ldots]$, we will also abbreviate

$$\boldsymbol{C}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p] = \boldsymbol{C}[p] \quad (p \times p \text{ matrix}), \qquad \boldsymbol{a}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p] = \boldsymbol{a}[p] \quad (p - \text{dim vector})$$

Prediction means finding (4.3), which upon inserting (4.12) becomes (in our new notation):

$$p(t^{p+1} | \boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^{p+1}, t^1, \ldots, t^p) = \frac{p(t^1, \ldots, t^{p+1} | \boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^{p+1})}{p(t^1, \ldots, t^p | \boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p)}$$

$$= \frac{e^{-\frac{1}{2}\begin{pmatrix} t^1 - a_1[p+1] \\ \vdots \\ t^{p+1} - a_p[p+1] \end{pmatrix} \cdot \boldsymbol{C}^{-1}[p+1] \begin{pmatrix} t^1 - a_1[p+1] \\ \vdots \\ t^{p+1} - a_p[p+1] \end{pmatrix} + \frac{1}{2}\begin{pmatrix} t^1 - a_1[p] \\ \vdots \\ t^p - a_p[p] \end{pmatrix} \cdot \boldsymbol{C}^{-1}[p] \begin{pmatrix} t^1 - a_1[p] \\ \vdots \\ t^p - a_p[p] \end{pmatrix}}}{\sqrt{2\pi \ \det \boldsymbol{C}[p+1] / \det \boldsymbol{C}[p]}} \tag{4.15}$$

Explicit Expressions for Predictions and Error Bars. The output probability distribution (4.15) for t^{p+1}, given the new input $\boldsymbol{\xi}^{p+1}$ and the data $D = \{(\boldsymbol{\xi}^1, t^1), \ldots, (\boldsymbol{\xi}^p, t^p)\}$, must also be Gaussian, since it is a marginal distribution of a multivariate Gaussian distribution. This property allows us to concentrate in the exponent of (4.15) only on terms involving t^{p+1}; the others simply provide an appropriate data-dependent normalization constant:

$$p(t^{p+1}|\boldsymbol{\xi}^{p+1}, D) \sim e^{-\frac{1}{2}(t^{p+1}-a_{p+1}[p+1])^2 \boldsymbol{C}^{-1}[p+1]_{p+1,p+1} - (t^{p+1}-a_{p+1}[p+1]) \sum_{\nu=1}^{p} \boldsymbol{C}^{-1}[p+1]_{p+1,\nu}(t^\nu - a_\nu[p+1])}$$

or, equivalently,

$$p(t^{p+1}|\boldsymbol{\xi}^{p+1}, D) \;\sim\; e^{-\frac{1}{2}\boldsymbol{C}^{-1}[p+1]_{p+1,p+1}\left[t^{p+1}-a_{p+1}[p+1]+\frac{\sum_{\nu=1}^{p}\boldsymbol{C}^{-1}{}_{[p+1]_{p+1,\nu}(t^\nu - a_\nu[p+1])}}{\boldsymbol{C}^{-1}{}_{[p+1]_{p+1,p+1}}}\right]^2}$$

We can now simply read off average and variance of the predicted output. Upon restoring our original notation the resulting prediction and its associated uncertainty read:

$$t^\star(\boldsymbol{\xi}) = a_{p+1}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{\xi}] - \frac{\sum_{\nu=1}^{p} \boldsymbol{C}^{-1}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{\xi}]_{p+1,\nu}(t^\nu - a_\nu[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{\xi}])}{\boldsymbol{C}^{-1}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{\xi}]_{p+1,p+1}} \tag{4.16}$$

$$\Delta t^\star(\boldsymbol{\xi}) = \frac{1}{\sqrt{\boldsymbol{C}^{-1}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{\xi}]_{p+1,p+1}}} \tag{4.17}$$

A non-zero choice for the $a_\mu[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^{p+1}]$ is equivalent to adding a fixed non-parametrized function of the joint inputs to the joint outputs of the underlying model. This is allowed mathematically, but requires rather specific knowledge of the function underlying the observed data, which is usually lacking. The two examples of Gaussian processes discussed earlier, RBF-type systems and two-layer feedforward networks with an infinitely large hidden layer, both had $a_\mu[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^{p+1}] = 0$ for all μ. In the latter cases one finds (4.16) being simplified to

$$t^\star(\boldsymbol{\xi}) = -\frac{\sum_{\nu=1}^{p} \boldsymbol{C}^{-1}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{\xi}]_{p+1,\nu} t^\nu}{\boldsymbol{C}^{-1}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{\xi}]_{p+1,p+1}} \tag{4.18}$$

Given a data set $D = \{(\boldsymbol{\xi}^1, t^1), \ldots, (\boldsymbol{\xi}^p, t^p)\}$ and a choice of model, i.e. of the functions $C_{\mu\nu}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^{p+1}]$ and $a[\boldsymbol{\xi}]$, we can now simply insert a novel input $\boldsymbol{\xi}$ into equations (4.16,4.17). More specifically, there is no (iterative) learning process; the computational effort is reduced to the inversion of a symmetric and positive definite $(p+1) \times (p+1)$ matrix.

Simplifications for On-Line Learning. We will next establish a relation between the matrices $\boldsymbol{C}^{-1}[p+1]$ and $\boldsymbol{C}^{-1}[p]$ (in our short-hand notation). In those cases where the data $(\boldsymbol{\xi}^\mu, t^\mu)$ arrive sequentially, it will reduce drastically the computational cost generated by the need to invert $\boldsymbol{C}[p+1]$. The idea is to exploit the property that, by definition, all entries of the covariance matrix $\boldsymbol{C}[p]$ also occur as the entries in the top-left $p \times p$ corner of $\boldsymbol{C}[p+1]$:

$$\boldsymbol{C}[p+1] = \begin{pmatrix} & & & k_1 \\ & \boldsymbol{C}[p] & & k_2 \\ & & & \vdots \\ k_1 & k_2 & \cdots & k_{p+1} \end{pmatrix} \qquad k_\mu = C_{\mu,p+1}[p+1]$$

We write the inverse $\boldsymbol{C}^{-1}[p+1]$ to be calculated as

$$\boldsymbol{C}^{-1}[p+1] = \begin{pmatrix} & & & u_1 \\ & \boldsymbol{D} & & u_2 \\ & & & \vdots \\ u_1 & u_2 & \cdots & u_{p+1} \end{pmatrix}$$

with as yet unknown numbers $\{u_\mu\}$ and an as yet unknown (symmetric) $p \times p$ matrix $\boldsymbol{D}$. Working out the matrix condition $\mathbb{I} = \boldsymbol{C}^{-1}[p+1].\boldsymbol{C}[p+1]$ leads us to the expression

$$\mathbb{I} = \begin{pmatrix} (\boldsymbol{DC}[p])_{11} + u_1 k_1 & \cdots & (\boldsymbol{DC})_{1p} + u_1 k_p & \sum_{\mu=1}^{p} D_{1\mu} k_\mu + u_1 k_{p+1} \\ (\boldsymbol{DC}[p])_{21} + u_2 k_1 & \cdots & (\boldsymbol{DC})_{2p} + u_2 k_p & \sum_{\mu=1}^{p} D_{2\mu} k_\mu + u_2 k_{p+1} \\ \vdots & & & \vdots \\ (\boldsymbol{DC}[p])_{p1} + u_p k_1 & \cdots & (\boldsymbol{DC})_{pp} + u_p k_p & \sum_{\mu=1}^{p} D_{p\mu} k_\mu + u_p k_{p+1} \\ \sum_{\mu=1}^{p} C_{1\mu} u_\mu + u_{p+1} k_1 & \cdots & \sum_{\mu=1}^{p} C_{p\mu} u_\mu + u_{p+1} k_p & \sum_{\mu=1}^{p+1} u_\mu k_\mu \end{pmatrix}$$

Left and right-hand side are identical if and only if the following hold:

$$(\boldsymbol{DC}[p])_{\mu\nu} + u_\mu k_\nu = \delta_{\mu\nu} \quad \text{for all } \mu, \nu = 1, \ldots, p$$

$$\sum_{\nu=1}^{p} D_{\mu\nu} k_\nu + u_\mu k_{p+1} = 0 \qquad \sum_{\nu=1}^{p} C_{\mu\nu}[p] u_\nu + u_{p+1} k_\mu = 0 \qquad \text{for all } \mu = 1, \ldots, p$$

These equations are easily solved, giving

$$D_{\mu\nu} = (\boldsymbol{C}^{-1}[p])_{\mu\nu} + \frac{[\sum_{\rho\leq p}(\boldsymbol{C}^{-1}[p])_{\mu\rho} k_\rho]\ [\sum_{\rho\leq p}(\boldsymbol{C}^{-1}[p])_{\nu\rho} k_\rho]}{k_{p+1} - \sum_{\lambda\rho\leq p} k_\lambda (\boldsymbol{C}^{-1}[p])_{\lambda\rho} k_\rho} \tag{4.19}$$

$$u_\mu = -\frac{\sum_{\rho\leq p}(\boldsymbol{C}^{-1}[p])_{\mu\rho} k_\rho}{k_{p+1} - \sum_{\lambda\rho\leq p} k_\lambda (\boldsymbol{C}^{-1}[p])_{\lambda\rho} k_\rho} \qquad (\mu \leq p) \tag{4.20}$$

$$u_{p+1} = \frac{1}{k_{p+1} - \sum_{\lambda\rho\leq p} k_\lambda (\boldsymbol{C}^{-1}[p])_{\lambda\rho} k_\rho} \tag{4.21}$$

These results allow us, firstly, to replace expressions (4.16,4.17) by

$$t^\star(\boldsymbol{\xi}) = a_{p+1}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{\xi}] + \sum_{\mu\nu\leq p} C_{p+1,\mu}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{\xi}] \boldsymbol{C}^{-1}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p]_{\mu\nu} (t^\nu - a_\nu[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{\xi}]) \tag{4.22}$$

$$\Delta t^\star(\boldsymbol{\xi}) =$$

$$\sqrt{C_{p+1,p+1}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{\xi}] - \sum_{\mu\nu\leq p} C_{p+1,\mu}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{\xi}] (\boldsymbol{C}^{-1}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p])_{\mu\nu} C_{\nu,p+1}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p, \boldsymbol{\xi}]} \tag{4.23}$$

More importantly, however (since the new equations (4.22,4.23) are not simpler than the previous (4.16,4.17)), the identities (4.19,4.20,4.21) ensure that the inverse of the $p \times p$ matrix $\boldsymbol{C}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^p]$ can be calculated *iteratively* from the inverse of the $(p-1) \times (p-1)$ matrix $\boldsymbol{C}[\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^{p-1}]$, i.e. upon adding the data points to our data set one by one.

4.3 The Choice of Covariance Matrix

Structure of Covariance Matrices. The form chosen for the covariance matrix, equivalent to the selection of data-generating model assumed responsible for the data, is crucial for the type of predictions that will be made by Gaussian processes. The two neural network examples of section 4.1, which reduced to Gaussian processes, were both found to be characterized not only by $a_\mu[\boldsymbol{\xi}^1,\ldots,\boldsymbol{\xi}^p]=0$, but also by specific simple forms of the covariance matrices $C_{\mu\nu}[\boldsymbol{\xi}^1,\ldots,\boldsymbol{\xi}^p]$, viz. (4.7) and (4.11):

$$\text{RBF}: \qquad C_{\mu\nu}[\boldsymbol{\xi}^1,\ldots,\boldsymbol{\xi}^p]=\frac{1}{\beta}\delta_{\mu\nu}+\frac{1}{\alpha}\sum_{i=1}^{L}\phi_i(\boldsymbol{\xi}^\mu)\phi_i(\boldsymbol{\xi}^\nu) \tag{4.24}$$

$$\text{MLP}: \qquad C_{\mu\nu}[\boldsymbol{\xi}^1,\ldots,\boldsymbol{\xi}^p]=\frac{1}{\beta}\delta_{\mu\nu}+\frac{1}{\alpha_w}\int\frac{d\boldsymbol{z}}{(2\pi)^{N/2}}\;e^{-\frac{1}{2}\boldsymbol{z}^2}g(\frac{\boldsymbol{z}\cdot\boldsymbol{\xi}^\mu}{\sqrt{\alpha_J}})\;g(\frac{\boldsymbol{z}\cdot\boldsymbol{\xi}^\nu}{\sqrt{\alpha_J}}) \tag{4.25}$$

(in both cases $\boldsymbol{\xi}\in\Re^N$; the RBF-type system has L basis functions, the MLP has an infinitely large hidden layer). Both are of the following general form:

$$C_{\mu\nu}[\boldsymbol{\xi}^1,\ldots,\boldsymbol{\xi}^p]=\frac{1}{\beta}\delta_{\mu\nu}+C[\boldsymbol{\xi}^\mu,\boldsymbol{\xi}^\nu] \tag{4.26}$$

It is in fact easy to show that *all* data-generating models of the general form (3.19) (not necessarily with Gaussian priors), i.e.

$$p(t|\boldsymbol{\xi};\boldsymbol{w})=e^{-\frac{1}{2}\beta[t-f(\boldsymbol{\xi};\boldsymbol{w})]^2} \tag{4.27}$$

have the property that, if they reduce to a Gaussian process, then this process must have a covariance matrix of the form (4.26), and in addition $a_\mu[\boldsymbol{\xi}^1,\ldots,\boldsymbol{\xi}^p]=a[\boldsymbol{\xi}^\mu]$ for some function $a[\boldsymbol{\xi}]$ (note: the two examples of (4.7) and (4.11) both had $a[\boldsymbol{\xi}]=0$). The joint output distribution for the models (4.27) is

$$p(t^1,\ldots,t^p|\boldsymbol{\xi}^1,\ldots,\boldsymbol{\xi}^p)=\left[\frac{2\pi}{\beta}\right]^{\frac{p}{2}}\int d\boldsymbol{w}\;p(\boldsymbol{w})e^{-\frac{1}{2}\beta\sum_{\mu=1}^{p}[t^\mu-f(\boldsymbol{\xi}^\mu;\boldsymbol{w})]^2}$$

Integration over the outputs then immediately leads to

$$a_\mu[\boldsymbol{\xi}^1,\ldots,\boldsymbol{\xi}^p] \;=\; a(\boldsymbol{\xi}^\mu)=\int d\boldsymbol{w}\;p(\boldsymbol{w})f(\boldsymbol{\xi}^\mu;\boldsymbol{w}) \tag{4.28}$$

$$C_{\mu\nu}[\boldsymbol{\xi}^1,\ldots,\boldsymbol{\xi}^p] \;=\; \frac{1}{\beta}\delta_{\mu\nu}+\int d\boldsymbol{w}\;p(\boldsymbol{w})[f(\boldsymbol{\xi}^\mu;\boldsymbol{w})-a(\boldsymbol{\xi}^\mu)][f(\boldsymbol{\xi}^\nu;\boldsymbol{w})-a(\boldsymbol{\xi}^\nu)] \tag{4.29}$$

This is indeed of the form (4.26), as claimed.

Examples – Homogeneous Covariance Matrices. Let us now work out further the covariance matrix (4.7) of RBF networks, upon choosing Gaussian basis functions of width σ, i.e. $\phi_k(\boldsymbol{\xi})=(2\pi\sigma^2)^{-N/2}e^{-\frac{1}{2}[\boldsymbol{\xi}-\boldsymbol{r}_k]^2/\sigma^2}$. It corresponds to the following function $C[\boldsymbol{\xi},\boldsymbol{\xi}']$, in the representation (4.26):

$$C[\boldsymbol{\xi},\boldsymbol{\xi}']=\frac{1}{\alpha(2\pi\sigma^2)^N}\sum_{k=1}^{L}e^{-\frac{1}{2}[(\boldsymbol{\xi}-\boldsymbol{r}_k)^2+(\boldsymbol{\xi}'-\boldsymbol{r}_k)^2]/\sigma^2}$$

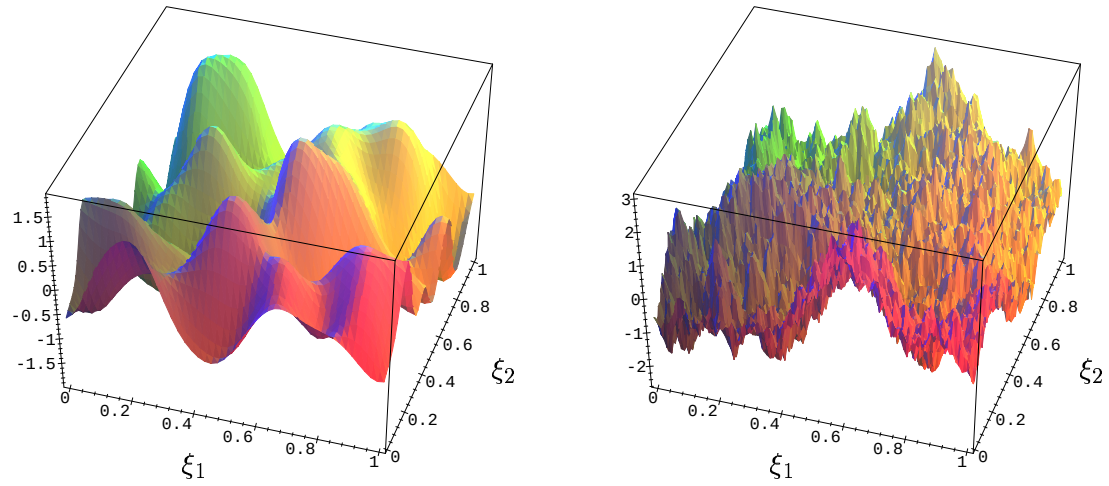


Figure 4.1: Examples of data surfaces, with $\boldsymbol{\xi}, \boldsymbol{\xi}' \in [0,1]^2$, generated by Gaussian processes with homogeneous covariance matrices and $a[\boldsymbol{\xi}] = 0$. Left: $C[\boldsymbol{\xi}, \boldsymbol{\xi}'] = K_0 e^{-\frac{1}{2}|\boldsymbol{\xi}-\boldsymbol{\xi}'|^2/\sigma^2}$, with $K_0 = 10$ and $\sigma = 0.1$. Right: $C[\boldsymbol{\xi}, \boldsymbol{\xi}'] = K_0 e^{-|\boldsymbol{\xi}-\boldsymbol{\xi}'|/\sigma}$, with $K_0 = 10$ and $\sigma = 0.1$.

We now imagine having a very large number $L \to \infty$ of basis functions, distributed homogeneously over $\Re^N$, and we re-scale our hyper-parameter α accordingly as $\alpha = \tilde{\alpha}/\Delta_L$. Here Δ_L denotes a small volume element in $\Re^N$ such that $\lim_{L\to\infty} \sum_{k=1}^{L} \Delta_L f[\boldsymbol{r}_k] = \int d\boldsymbol{r}\ f[\boldsymbol{r}]$ for any sufficiently smooth function f. We now find

$$\begin{aligned} \lim_{L\to\infty} C[\boldsymbol{\xi}, \boldsymbol{\xi}'] &= \frac{1}{\tilde{\alpha}(2\pi\sigma^2)^N} \int d\boldsymbol{r}\ e^{-\frac{1}{2}[(\boldsymbol{\xi}-\boldsymbol{r})^2 + (\boldsymbol{\xi}'-\boldsymbol{r})^2]/\sigma^2} \\ &= \frac{e^{-\frac{1}{2\sigma^2}[\boldsymbol{\xi}^2 + (\boldsymbol{\xi}')^2]}}{\tilde{\alpha}(2\pi\sigma^2)^N} \int d\boldsymbol{r}\ e^{-[\boldsymbol{r}^2 + \boldsymbol{r}\cdot(\boldsymbol{\xi}+\boldsymbol{\xi}')]/\sigma^2} = \frac{e^{-\frac{1}{4\sigma^2}(\boldsymbol{\xi}-\boldsymbol{\xi}')^2}}{\tilde{\alpha}(2\sigma\sqrt{\pi})^N} \end{aligned} \tag{4.30}$$

Working out the covariance matrix of example (4.11) for arbitrary $g[u]$ is hard. For data vectors $\boldsymbol{\xi}$ close to the origin (alternatively: for large values of the hyper-parameter α_J), however, we may expand typical choices such as $g[u] = \tanh[u]$ as $g[u] = u + \mathcal{O}(u^3)$ and find

$$\begin{aligned} C[\boldsymbol{\xi}, \boldsymbol{\xi}'] &= \frac{1}{\alpha_w \alpha_J} \int \frac{d\boldsymbol{z}}{(2\pi)^{N/2}}\ e^{-\frac{1}{2}\boldsymbol{z}^2} (\boldsymbol{z}\cdot\boldsymbol{\xi})(\boldsymbol{z}\cdot\boldsymbol{\xi}') + \mathcal{O}(|\boldsymbol{\xi}||\boldsymbol{\xi}'|^3, |\boldsymbol{\xi}|^3|\boldsymbol{\xi}'|) \\ &= \frac{\boldsymbol{\xi}\cdot\boldsymbol{\xi}'}{\alpha_w \alpha_J} + \mathcal{O}(|\boldsymbol{\xi}||\boldsymbol{\xi}'|^3, |\boldsymbol{\xi}|^3|\boldsymbol{\xi}'|) \end{aligned} \tag{4.31}$$

(truncation after the first term is obviously exact only for $g[u] = u$).

Homogeneous covariance matrices are the subclass of (4.26) where the function $C[\boldsymbol{\xi}, \boldsymbol{\xi}']$ is of the form $C[\boldsymbol{\xi}, \boldsymbol{\xi}'] = C[|\boldsymbol{\xi} - \boldsymbol{\xi}'|]$ (like, for instance, the RBF expression (4.30), but unlike the MLP expression (4.31)). Here the output statistics are fully isotropic in input space (i.e. invariant under arbitrary translations and/or rotations of both $\boldsymbol{\xi}$ and $\boldsymbol{\xi}'$). Within this subclass the only remaining freedom we have regarding covariances is the choice of a scalar function $C[u]$ (with $u \geq 0$); however, still much variation is found to be possible in the types

of data that can be generated and predicted. Figure 4.1 shows examples of data surfaces, with $\boldsymbol{\xi}, \boldsymbol{\xi}' \in [0,1]^2$, generated by Gaussian processes with the following homogeneous covariance matrices:

$$\text{'smooth' functions :} \qquad C[\boldsymbol{\xi}, \boldsymbol{\xi}'] = K_0 e^{-\frac{1}{2}|\boldsymbol{\xi}-\boldsymbol{\xi}'|^2/\sigma^2} \tag{4.32}$$

$$\text{'rough' functions :} \qquad C[\boldsymbol{\xi}, \boldsymbol{\xi}'] = K_0 e^{-|\boldsymbol{\xi}-\boldsymbol{\xi}'|/\sigma} \tag{4.33}$$

The qualitative difference between the two can be appreciated upon realizing that smoothness is defined in terms of the behaviour of the kernels $C[\boldsymbol{\xi}, \boldsymbol{\xi}']$ for $\boldsymbol{\xi}' \to \boldsymbol{\xi}$. Let is choose $\boldsymbol{\xi}' - \boldsymbol{\xi} = \boldsymbol{\epsilon}$, with $|\boldsymbol{\epsilon}| \ll 1$. For the above two examples one finds, respectively:

$$e^{-\frac{1}{2}|\boldsymbol{\xi}-\boldsymbol{\xi}'|^2/\sigma^2} = 1 - \frac{\boldsymbol{\epsilon}^2}{2\sigma^2} + \mathcal{O}(|\boldsymbol{\epsilon}|^4|) \qquad\qquad e^{-|\boldsymbol{\xi}-\boldsymbol{\xi}'|/\sigma} = 1 - \frac{|\boldsymbol{\epsilon}|}{\sigma} + \mathcal{O}(|\boldsymbol{\epsilon}|^2)$$

Hence the second kernel indeed describes much faster de-correlation of outputs for small differences in the inputs. In a similar fashion one could define a more general family of covariance matrices, $C[\boldsymbol{\xi}, \boldsymbol{\xi}'] = K_0 e^{-|\boldsymbol{\xi}-\boldsymbol{\xi}'|^\gamma/\sigma^\gamma}$, in which the degree of smoothness is controlled monotonically by a parameter $\gamma > 0$.

Chapter 5

Support Vector Machines for Binary Classification

Support vector machines carry out binary classifications. They involve a preprocessing stage where non-linearly separable data are converted (hopefully) into linearly separable ones. First, however, we will have to return to the properties of linearly separable problems (for reasons which will become clear); the pre-processing will be introduced later.

5.1 Optimal Separating Plane for Linearly Separable Tasks

Linear Separations & Stability Parameters. Consider a linearly separable binary classification task, with a set of data of the usual form $D = \{(\boldsymbol{\xi}^1, t^1), \ldots, (\boldsymbol{\xi}^p, t^p)\}$, where $\boldsymbol{\xi} \in \Re^N$ and $t^\mu \in \{-1, 1\}$. If this problem is linearly separable, we know that

$$(\exists \boldsymbol{w} \in \Re^N)(\exists w_0 \in \Re): \quad t^\mu = \text{sgn}[\boldsymbol{w} \cdot \boldsymbol{\xi}^\mu + w_0] \quad \text{for all } (\boldsymbol{\xi}^\mu, t^\mu) \in D \tag{5.1}$$

We even have algorithms which are guaranteed to converge towards a solution $(\boldsymbol{w}, w_0)$ (a 'separating plane') of the type (5.1), e.g. the so-called perceptron or AdaTron rules. However, it will be clear that in general there will exist an infinite number of separating planes $(\boldsymbol{w}, w_0)$ that will meet the requirements (5.1). See e.g. figure 5.1 (left panel) for $N = 2$; here a separating plane is any line separating all points $\boldsymbol{\xi}^\mu$ with $t^\mu = 1$ (marked $+$) from those with $t^\mu = -1$ (marked $\times$).

A natural question to ask is whether one can quantify the quality of the various solutions of (5.1). This can be done on the basis of generalization performance: a good plane $(\boldsymbol{w}, w_0)$ is one which keeps a safe distance (if possible) from the data points, so that detrimental effects of noise (which might change the locations of these points slightly) are minimal. Hence we wish to measure not only whether points are correctly classified by a candidate separating plane, but also the *distance* of such a plane to the data points which it aims to separate. Both are given by the so-called *stability parameters* γ_μ (there is one for every data point):

$$\gamma_\mu(\boldsymbol{w}, w_0) = t^\mu(\boldsymbol{w} \cdot \boldsymbol{\xi}^\mu + w_0)/|\boldsymbol{w}| \tag{5.2}$$

They have the properties:

- The plane $(\boldsymbol{w}, w_0)$ correctly classifies point $\boldsymbol{\xi}^\mu$ if and only if: $\gamma_\mu(\boldsymbol{w}, w_0) > 0$
- The distance of the plane $(\boldsymbol{w}, w_0)$ to point $\boldsymbol{\xi}^\mu$ is: $|\gamma_\mu(\boldsymbol{w}, w_0)|$

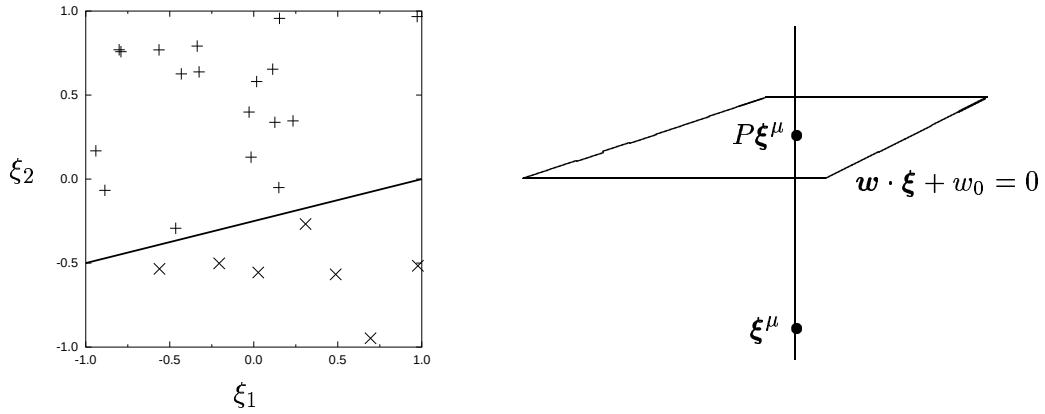


Figure 5.1: Left: illustration of linearly separable tasks for $N = 2$. A separating plane is any line $\boldsymbol{w}\cdot\boldsymbol{\xi} + w_0 = 0$ separating all points $\boldsymbol{\xi}^\mu$ with $t^\mu = 1$ (marked +) from those with $t^\mu = -1$ (marked $\times$). Note that there will generally be infinitely many such planes. Right: measuring separation quality on the basis of the distance of data points $\boldsymbol{\xi}^\mu$ from the separating plane (as calculated via projection); see the main text.

The first property is immediately obvious from (5.1). Demonstrating the second property requires only simple high-school geometry. We first construct a parametrization of the line in $\Re^N$ which goes through $\boldsymbol{\xi}^\mu$ and is orthogonal to our plane (see figure 5.1, right panel): $\boldsymbol{\xi}(\rho) = \boldsymbol{\xi}^\mu + \rho\boldsymbol{w}$ ($\rho \in \Re$). Insertion into the plane's equation $\boldsymbol{w}\cdot\boldsymbol{\xi} + w_0 = 0$ gives the solution $\rho = -(\boldsymbol{w}\cdot\boldsymbol{\xi} + w_0)/\boldsymbol{w}^2$. Hence the projection $P\boldsymbol{\xi}^\mu$ of $\boldsymbol{\xi}^\mu$ onto the plane $(\boldsymbol{w}, w_0)$ is given by

$$P\boldsymbol{\xi}^\mu = \boldsymbol{\xi}^\mu - \boldsymbol{w}\left[\frac{\boldsymbol{w}\cdot\boldsymbol{\xi}^\mu + w_0}{\boldsymbol{w}^2}\right]$$

And thus the desired distance $|\boldsymbol{\xi}^\mu - P\boldsymbol{\xi}^\mu|$ indeed equals (using $|t^\mu| = 1$ for all μ):

$$|\boldsymbol{\xi}^\mu - P\boldsymbol{\xi}^\mu| = \left|\boldsymbol{w}\left[\frac{\boldsymbol{w}\cdot\boldsymbol{\xi}^\mu + w_0}{\boldsymbol{w}^2}\right]\right| = |\gamma_\mu(\boldsymbol{w}, w_0)|$$

As claimed. As a consequence of the properties of the stability parameters we may now sharpen our requirements of a separating plane:

$$\textit{all data points separated correctly} \;\;\&\;\; \textit{all data points at distance} \geq \Delta \textit{ from the plane} \quad \Longleftrightarrow \quad \gamma_\mu(\boldsymbol{w}, w_0) \geq \Delta > 0 \;\text{ for all }\; \mu \tag{5.3}$$

There will be an upper limit to the values of the distance $\Delta > 0$ which are compatible with correct separation (and it is a non-trivial task to determine this limit), since clearly

$$\Delta \leq \frac{1}{2}\Delta^\pm \qquad \Delta^\pm = \min_{\mu\in I_+,\ \nu\in I_-} |\boldsymbol{\xi}^\mu - \boldsymbol{\xi}^\nu| \qquad I_\pm = \{\mu \in \{1,\ldots,p\} |\ t^\mu = \pm 1\}$$

However, if we choose a value of Δ such that a solution $(\boldsymbol{w}, w_0)$ of (5.3) exists, then there is an algorithm which is guaranteed to achieve the objective (a straightforward generalization of the classic Perceptron learning rule, which is indeed recovered for $\Delta \to 0$) :

Generalized Perceptron Learning Rule:

1. drawn at random a $\mu \in \{1, \ldots, p\}$
2. calculate $\gamma_\mu(\boldsymbol{w}, w_0)$ as given in (5.2)
3. if $\gamma_\mu(\boldsymbol{w}, w_0) < \Delta$, modify parameters: $\boldsymbol{w} \to \boldsymbol{w} + t^\mu \boldsymbol{\xi}^\mu$
   $w_0 \to w_0 + t^\mu$
4. return to 1.

Generalized Perceptron Convergence Theorem (E. Gardner):

If $(\exists \boldsymbol{w}^\star \in \Re^N)(\exists w_0^\star \in \Re) : \ \gamma_\mu(\boldsymbol{w}^\star, w_0^\star) > \Delta$ for all $\mu \in \{1, \ldots, p\}$, then the above algorithm will converge towards a solution $(\boldsymbol{w}, w_0)$ with $\gamma_\mu(\boldsymbol{w}, w_0) \geq \Delta$ in a finite number of iteration steps.

The proof will not be given here. For $w_0 = 0$ it is a simple generalization of the original perceptron convergence proof. For $w_0 \neq 0$, however, the situation is more complicated; the reason is that the threshold w_0 does not occur in the denominator of the stability parameters (5.2), so that it cannot simply be absorbed into the classic proof by adding a 'dummy' extra component $\xi_0^\mu = 1$ to all input vectors $\boldsymbol{\xi}^\mu$.

The Optimal Separating Plane. We are now in a position to define the optimal separating plane $(\boldsymbol{w}^\star, w_0^\star)$ as the plane for which the *smallest* of the $\{\gamma^\mu(\boldsymbol{w}, w_0)\}$ (i.e. the stability parameter of the data point which is most in danger of being mis-classified when data are subject to noise) is as large as possible:

$$\text{optimal separating plane} \quad (\boldsymbol{w}^\star, w_0^\star) = \arg \max_{(\boldsymbol{w}, w_0)} \left[\min_\mu \gamma_\mu(\boldsymbol{w}, w_0) \right] \tag{5.4}$$

Note that:

- If the problem is linearly separable, definition (5.4) will produce a separating plane with the largest distance to the closest data point. If the problem is not linearly separable, definition (5.4) will produce a plane which does not separate the data (since that is impossible), but will have the minimum distance to the plane of those mis-classified points which are furthest away from the plane.
- By construction, there must be at least two μ such that $\gamma_\mu(\boldsymbol{w}^\star, w_0^\star) = \min_\rho \gamma_\rho(\boldsymbol{w}^\star, w_0^\star)$ for the optimal separating plane (5.4) (see e.g. figure 5.1).

Finding the optimal plane has now been reduced to a relatively simple optimization exercise (5.4). The only problem with it is that its solution $(\boldsymbol{w}^\star, w_0^\star)$ is still not unique, since there remains the trivial degree of freedom relating to overall scaling: if we multiply $(\boldsymbol{w}, w_0) \to (\rho \boldsymbol{w}, \rho w_0)$ with $\rho > 0$ we simply get the same plane. To eliminate this freedom we nail down ρ by insisting that

$$\min_\mu t^\mu (\boldsymbol{w} \cdot \boldsymbol{\xi}^\mu + w_0) = 1$$

(given the assumption of linear separability this is always possible). This implies that $\min_\mu \gamma_\mu(\boldsymbol{w}, w_0) = 1/|\boldsymbol{w}|$, and that $\max_{(\boldsymbol{w}, w_0)} [\min_\mu \gamma_\mu(\boldsymbol{w}, w_0)] = 1/\min_{(\boldsymbol{w}, w_0)} |\boldsymbol{w}|$. Hence we have now converted the initial optimization problem (5.4) into

$$\text{find :} \qquad \min_{(\boldsymbol{w}, w_0)} \frac{1}{2} \boldsymbol{w}^2 \qquad \text{under constraints} \qquad (\forall \mu) : \ t^\mu (\boldsymbol{w} \cdot \boldsymbol{\xi}^\mu + w_0) \geq 1 \tag{5.5}$$

5.2 Representation in terms of Support Vectors

Solution of the Optimization Problem. We now solve the constrained optimization problem (5.5) using the standard method of Lagrange multipliers. This leads to the following coupled equations:

$$\frac{\partial}{\partial w_i}\frac{1}{2}\boldsymbol{w}^2 = \sum_{\mu=1}^{p}\lambda_\mu(\boldsymbol{w},w_0)\frac{\partial}{\partial w_i}\left[t^\mu(\boldsymbol{w}\cdot\boldsymbol{\xi}^\mu+w_0)-1\right] \qquad i=0,\ldots,N$$

$$\begin{aligned}\text{with}: \quad & \lambda_\mu(\boldsymbol{w},w_0)=0 \quad \text{if extremum has} \quad t_\mu(\boldsymbol{w}\cdot\boldsymbol{\xi}^\mu+w_0)>1\\ & \lambda_\mu(\boldsymbol{w},w_0)\neq 0 \quad \text{if extremum has} \quad t_\mu(\boldsymbol{w}\cdot\boldsymbol{\xi}^\mu+w_0)=1\end{aligned}$$

We define the index set S (dependent on $\boldsymbol{w}$ and w_0) as follows:

$$\begin{aligned}\mu\in S: \quad & t^\mu(\boldsymbol{w}\cdot\boldsymbol{\xi}^\mu+w_0)=1 \quad (\text{so } \lambda_\mu(\boldsymbol{w},w_0)\neq 0)\\ \mu\notin S: \quad & t^\mu(\boldsymbol{w}\cdot\boldsymbol{\xi}^\mu+w_0)>1 \quad (\text{so } \lambda_\mu(\boldsymbol{w},w_0)=0)\end{aligned} \tag{5.6}$$

with $|S|\leq p$ elements. Without danger of confusion we will henceforth drop the arguments of the Lagrange parameters and write them simply as λ_μ. Working out the above partial derivatives gives

$$\boldsymbol{w}=\sum_{\mu\in S}\lambda_\mu t^\mu\boldsymbol{\xi}^\mu \qquad \sum_{\mu\in S}\lambda_\mu t^\mu=0 \tag{5.7}$$

$$\mu\in S: \quad t_\mu\left[\sum_{\nu\in S}\lambda_\nu t^\nu(\boldsymbol{\xi}^\mu\cdot\boldsymbol{\xi}^\nu)+w_0\right]=1 \qquad \mu\neq S: \quad t_\mu\left[\sum_{\nu\in S}\lambda_\nu t^\nu(\boldsymbol{\xi}^\mu\cdot\boldsymbol{\xi}^\nu)+w_0\right]>1 \tag{5.8}$$

We will now show that, given knowledge of the index set S, the solution of (5.7,5.8) follows directly. In doing so we will need the following (symmetric and non-negative definite) $|S|\times|S|$ matrix $\boldsymbol{C}$:

$$C_{\mu\nu}=t^\mu(\boldsymbol{\xi}^\mu\cdot\boldsymbol{\xi}^\nu)t^\nu \qquad \text{for} \quad \mu,\nu\in S \tag{5.9}$$

Firstly, after insertion of the first expression of (5.7) (to eliminate $\boldsymbol{w}$) the first equation of (5.8) allows us to write the non-zero Lagrange parameters as

$$\mu\in S: \qquad \lambda_\mu=\sum_{\nu\in S}(\boldsymbol{C}^{-1})_{\mu\nu}(1-w_0t^\nu) \tag{5.10}$$

Inserting (5.10) into the second equation of (5.7) gives us the threshold w_0:

$$w_0=\frac{\sum_{\mu\nu\in S}t^\mu(\boldsymbol{C}^{-1})_{\mu\nu}}{\sum_{\mu\nu\in S}t^\mu(\boldsymbol{C}^{-1})_{\mu\nu}t^\nu} \tag{5.11}$$

Our problem of finding the solution $(\boldsymbol{w}^\star,w_0^\star)$ of (5.7,5.8), i.e. the optimal separating plane, has thus been reduced to finding the index set S[1]. We note in the above equations that only those inputs play a role which correspond to indices $\mu\in S$. This then leads us to the definition of *Support Vectors*:

[1] If the matrix $\boldsymbol{C}$ has zero eigenvalues (it can never have negative ones) we must replace $\boldsymbol{C}^{-1}$ in the above equations by the pseudo-inverse of $\boldsymbol{C}$.

- Support vectors are those input vectors $\boldsymbol{\xi}^\mu$ in the training set which, in the construction of the optimal separating plane as given above, correspond to indices $\mu \in S$.
- Support vectors are those input vectors which are closest to the separating plane, since the distance for pattern μ equals $\gamma_\mu = t^\mu(\boldsymbol{w}\cdot\boldsymbol{\xi}^\mu + w_0)/|\boldsymbol{w}|$ and (5.8) tells us that

$$\mu \in S: \quad \gamma_\mu = 1/|\boldsymbol{w}| \qquad\qquad \mu \notin S: \quad \gamma_\mu > 1/|\boldsymbol{w}|$$

- Only support vectors and their correlations occur in the construction of the optimal separating plane. All other inputs are classified correctly once the support vectors are.

Let us at this stage derive two further properties of the solution, for later use. Firstly:

$$\sum_{\mu\in S} \lambda_\mu \geq 0 \tag{5.12}$$

Proof: we use the fact that $\boldsymbol{C}$ is non-negative definite and insert the first equation of (5.8), i.e. $\sum_{\nu\in S}\lambda_\nu t^\mu(\boldsymbol{\xi}^\mu\cdot\boldsymbol{\xi}^\nu)t^\nu = 1 - w_0 t^\mu$, followed by the second equation of (5.7), giving

$$0 \leq \sum_{\mu\nu\in S}\lambda_\mu C_{\mu\nu}\lambda_\nu = \sum_{\mu\in S}\lambda_\mu(1-w_0 t^\mu) = \sum_{\mu\in S}\lambda_\mu$$

Secondly, given the parameter vector $\boldsymbol{w}$, the threshold w_0 can be written as

$$w_0 = -\frac{1}{2}\boldsymbol{w}\cdot(\boldsymbol{\xi}^\mu + \boldsymbol{\xi}^\nu) \qquad \text{for any} \quad \mu,\nu\in S \quad \text{with} \quad t^\mu = 1,\ t^\nu = -1 \tag{5.13}$$

Proof: this follows immediately from the definitions (5.6) in combination with $t^\mu \in \{-1,1\}$, according to which $\boldsymbol{w}\cdot\boldsymbol{\xi}^\mu = t^\mu - w_0 =$ for $\mu \in S$.

Transformation of the Optimization Problem. Knowing the structure of the solution, we can transform the problem to a set of equations in the space of the Lagrange multipliers $\{\lambda_\mu\}$, which makes sense particularly if $p < N$. Insertion of $\boldsymbol{w} = \sum_{\mu=1}^p \lambda_\mu t^\mu \boldsymbol{\xi}^\mu$, following (5.7), into the original problem (5.5) gives with $\boldsymbol{\lambda} = (\lambda_1,\ldots,\lambda_p)$:

$$\text{find:} \quad \min_{(\boldsymbol{\lambda},w_0)} \frac{1}{2}\sum_{\mu\nu=1}^p \lambda_\mu t^\mu(\boldsymbol{\xi}^\mu\cdot\boldsymbol{\xi}^\nu)t^\nu\lambda_\nu \quad \text{under constraints} \quad \forall\mu: \ t^\mu\sum_{\nu=1}^p(\boldsymbol{\xi}^\mu\cdot\boldsymbol{\xi}^\nu)t^\nu\lambda_\nu \geq 1 - w_0 t^\mu \tag{5.14}$$

Support vectors correspond to those μ for which the last inequality becomes an equality. In preparation of a statement to be proven below, we note that the equations from which to solve the solution of (5.5) and of (5.14) can be summarized as

$$W_\mu = \sum_\nu t^\mu(\boldsymbol{\xi}^\mu\cdot\boldsymbol{\xi}^\nu)t^\nu\lambda_\nu + w_0 t^\mu - 1, \tag{5.15}$$

$$\sum_{\mu=1}^p \lambda_\mu t^\mu = 0, \qquad \forall\mu: \quad \{\lambda_\mu = 0,\ W_\mu > 0\} \ \text{ or } \ \{\lambda_\mu \neq 0,\ W_\mu = 0\} \tag{5.16}$$

(these are indeed $2p+1$ equations for the $2p+1$ unknowns $\{W_\mu,\lambda_\mu\}$ and w_0). In the case of multiple solutions we must select the one for which the quantity $\frac{1}{2}\sum_{\mu\nu=1}^p \lambda_\mu t^\mu(\boldsymbol{\xi}^\mu\cdot\boldsymbol{\xi}^\nu)t^\nu\lambda_\nu$ is minimal.

A final transformation brings our problem into the form most commonly encountered in textbooks. The statement is that the solution of (5.14) is identical to that of the following similar constrained minimization problem

$$\text{find:} \quad \min_{\boldsymbol{\lambda}} \left\{ \frac{1}{2} \sum_{\mu\nu=1}^{p} \lambda_\mu t^\mu (\boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi}^\nu) t^\nu \lambda_\nu - \sum_\mu \lambda_\mu \right\} \quad \text{constraints:} \quad \sum_{\mu=1}^{p} \lambda_\mu t^\mu = 0, \quad \forall \mu: \ \lambda_\mu \geq 0 \tag{5.17}$$

To appreciate the origin of this statement (we will not give the full elaborate proof here), let us work out the Lagrange equations for (5.17), denoting the Lagrange parameters of the inequality constraints $\lambda_\mu \geq 0$ as W_μ, and the Lagrange parameter of the constraint $\sum_{\mu=1}^{p} \lambda_\mu t^\mu = 0$ as $-W_0$:

$$W_\mu = \sum_\nu t_\mu (\boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi}^\nu) t^\nu \lambda_\nu + W_0 t^\mu - 1 \tag{5.18}$$

$$\sum_{\mu=1}^{p} \lambda_\mu t^\mu = 0, \qquad \forall \mu: \quad \{\lambda_\mu = 0, \ W_\mu \neq 0\} \ \text{ or } \ \{\lambda_\mu > 0, \ W_\mu = 0\} \tag{5.19}$$

Comparison with (5.15,5.16) shows that the two sets of equations and requirements are identical if the following is true:

$$\text{Solution of (5.14):} \quad \lambda_\mu \geq 0 \ \ \forall \mu \qquad\qquad \text{Solution of (5.17):} \quad W_\mu \geq 0 \ \ \forall \mu$$

Both statements are very similar, and can be shown to be true by virtue of the fact that we are only looking for solutions of the Lagrange equations which are *minima* of the quadratic surface to be extremized (the sign of a Lagrange parameter refers to the direction of the associated constraining force, relative to the gradient of the surface). Since the solution of a quadratic minimization problem with linear inequality constraints can also be shown to have a *unique* solution, the two minimization problems are indeed found to be equivalent[2].

A Simple Example. Let us assume, for simplicity, that the input vectors in our data set D are orthogonal and normalized, i.e. $\boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi}^\nu = \delta_{\mu\nu}$ (which implies also that $p \leq N$). We aim to calculate the optimal separating plane $(\boldsymbol{w}^\star, w_0^\star)$ from

$$\boldsymbol{w} = \sum_{\mu \in S} \lambda_\mu t^\mu \boldsymbol{\xi}^\mu \qquad\qquad w_0 = \frac{\sum_{\mu\nu \in S} t^\mu (\boldsymbol{C}^{-1})_{\mu\nu}}{\sum_{\mu\nu \in S} t^\mu (\boldsymbol{C}^{-1})_{\mu\nu} t^\nu} \tag{5.20}$$

with the matrix $\boldsymbol{C}$ as given in (5.9), and with the index set S giving the labels μ of the support vectors (i.e. those with $\lambda_\mu > 0$). We note that in the present example $C_{\mu\nu} = \delta_{\mu\nu}$, so that the second equation of (5.20) becomes

$$w_0 = |S|^{-1} \sum_{\mu \in S} t^\mu \qquad\qquad -1 \leq w_0 \leq 1$$

We calculate the $\{\lambda_\mu\}$ via the original equations (5.10), which reduces to

$$\mu \in S: \qquad \lambda_\mu = 1 - w_0 t^\mu$$

[2] See the specialized literature on convex optimization for full and further details.

Insertion into (5.8) subsequently gives

$$\mu \in S: \quad 1 = 1 \qquad\qquad \mu \notin S: \quad t_\mu w_0 > 1$$

We conclude that, in view of $|w_0| \leq 1$, the only possible solution is the one where $S = \{1, \ldots, p\}$, i.e. all input vectors are support vectors. Hence

$$\boldsymbol{w}^\star = \sum_{\mu=1}^{p} [t^\mu - w_0] \boldsymbol{\xi}^\mu \qquad\qquad w_0^\star = \frac{1}{p} \sum_{\mu=1}^{p} t^\mu \tag{5.21}$$

(which is know as the 'biased Hebb rule'). The corresponding stability parameters $\gamma_\mu = |\boldsymbol{w}|^{-1}$ (since $\mu \in S$ for all μ) are

$$\gamma_\mu = \frac{1}{\sqrt{\sum_{\mu=1}^{p} [t^\mu - w_0]^2}}$$

5.3 Pre-Processing, SVM Kernels

Pre-Processing of Non-Linearly-Separable Data. Let us now turn to problems which are not linearly separable, but could possibly be made so via suitable pre-processing. This implies that we will no longer view the $\boldsymbol{\xi}$ as the input vectors, but see the $\boldsymbol{\xi}$ as the outcome of the pre-processing of true input vectors $\boldsymbol{x} \in \Re^M$. If we write the combined action of our classification machine as $S(\boldsymbol{x}) \in \{-1, 1\}$ we have

$$\boldsymbol{\xi} = \boldsymbol{\phi}(\boldsymbol{x}): \qquad\qquad S(\boldsymbol{x}) = \ \mathrm{sgn}[\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}) + w_0] \tag{5.22}$$

with $\boldsymbol{\phi}(\boldsymbol{x}) = (\phi_1(\boldsymbol{x}), \ldots, \phi_N(\boldsymbol{x}))$ and usually with $N > M$. The data set is now $D = \{(\boldsymbol{x}^1, t^1), \ldots, (\boldsymbol{x}^p, t^p)\}$. The set of pre-processing operations $\phi_i(\boldsymbol{x})$ is typically chosen to contain non-linear functions (otherwise there would be no point in pre-processing), such as

$$\begin{array}{ll} \phi_i(\boldsymbol{x}) = x_i & \text{for } \ i = 1, \ldots, M \\ \phi_i(\boldsymbol{x}) = x_i^2 & \text{for } \ i = M+1, \ldots, 2M \\ \phi_{2M+1}(\boldsymbol{x}) = x_1 x_2 \qquad \phi_{2M+2}(\boldsymbol{x}) = x_1 x_3 \qquad \phi_{2M+3}(\boldsymbol{x}) = x_2 x_3 & \text{etc} \end{array}$$

Our previous results regarding the optimal separating plane $(\boldsymbol{w}^\star, w_0^\star)$ can be immediately applied via the replacement $\boldsymbol{\xi}^\mu \to \boldsymbol{\phi}(\boldsymbol{x}^\mu)$, which leads to

$$\boldsymbol{w}^\star = \sum_{\mu=1}^{p} \lambda_\mu t^\mu \boldsymbol{\phi}(\boldsymbol{x}^\mu) \qquad\qquad w_0^\star = -\frac{1}{2} \boldsymbol{w}^\star \cdot [\boldsymbol{\phi}(\boldsymbol{x}^+) + \boldsymbol{\phi}(\boldsymbol{x}^-)] \tag{5.23}$$

Here the support vectors are those $\boldsymbol{x}^\mu$ for which $\lambda_\mu = 0$, $\boldsymbol{x}^\pm$ are (any) support vectors with $t^\mu = \pm 1$, and the $\{\lambda_\mu\}$ are determined by solving:

$$\min_{\boldsymbol{\lambda}} \left\{ \frac{1}{2} \sum_{\mu\nu=1}^{p} \lambda_\mu \lambda_\nu t^\mu t^\nu \boldsymbol{\phi}(\boldsymbol{x}^\mu) \cdot \boldsymbol{\phi}(\boldsymbol{x}^\nu) - \sum_\mu \lambda_\mu \right\} \quad \text{constraints}: \quad \sum_{\mu=1}^{p} \lambda_\mu t^\mu = 0, \quad \forall \mu: \ \lambda_\mu \geq 0 \tag{5.24}$$

Insertion of the above expressions (5.23) into the machine's operation rule (5.22) gives the final classification recipe

$$S(\boldsymbol{x}) = \text{ sgn}\left[\sum_{\mu=1}^{p} \lambda_\mu t^\mu \left\{\boldsymbol{\phi}(\boldsymbol{x}^\mu)\cdot\boldsymbol{\phi}(\boldsymbol{x}) - \frac{1}{2}\boldsymbol{\phi}(\boldsymbol{x}^\mu)\cdot\boldsymbol{\phi}(\boldsymbol{x}^+) - \frac{1}{2}\boldsymbol{\phi}(\boldsymbol{x}^\mu)\cdot\boldsymbol{\phi}(\boldsymbol{x}^-)\right\}\right] \tag{5.25}$$

At this stage one makes the important observation that neither in (5.25), nor in the equations (5.24) from which to extract the $\{\lambda_\mu\}$ do we retain any 'bare' occurrence of the pre-processing functions $\phi_i(\boldsymbol{x})$. Instead we consistently encounter only the symmetric form

$$K(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{N} \phi_i(\boldsymbol{x})\phi_i(\boldsymbol{y}) \tag{5.26}$$

More specifically, in terms of (5.26) our full equations become

$$S(\boldsymbol{x}) = \text{ sgn}\left[\sum_{\mu=1}^{p} \lambda_\mu t^\mu \left\{K(\boldsymbol{x}^\mu,\boldsymbol{x}) - \frac{1}{2}K(\boldsymbol{x}^\mu,\boldsymbol{x}^+) - \frac{1}{2}K(\boldsymbol{x}^\mu,\boldsymbol{x}^-)\right\}\right] \tag{5.27}$$

$$\min_{\boldsymbol{\lambda}}\left\{\frac{1}{2}\sum_{\mu\nu=1}^{p} \lambda_\mu\lambda_\nu t^\mu t^\nu K(\boldsymbol{x}^\mu,\boldsymbol{x}^\nu) - \sum_\mu \lambda_\mu\right\} \qquad \text{constraints}: \quad \sum_{\mu=1}^{p}\lambda_\mu t^\mu = 0, \quad \forall\mu:\ \lambda_\mu \geq 0 \tag{5.28}$$

We can henceforth work directly with (5.27,5.28), and call the function $K(\boldsymbol{x},\boldsymbol{y})$ in these equations the SVM Kernel. The decision boundary in input space of our machine is defined as the set of all $\boldsymbol{x} \in \Re^M$ for which

$$\sum_{\mu=1}^{p} \lambda_\mu t^\mu \left\{K(\boldsymbol{x}^\mu,\boldsymbol{x}) - \frac{1}{2}K(\boldsymbol{x}^\mu,\boldsymbol{x}^+) - \frac{1}{2}K(\boldsymbol{x}^\mu,\boldsymbol{x}^-)\right\} = 0 \tag{5.29}$$

We can even generalize our classification machine to more general non-negative symmetric kernels $K(\boldsymbol{x},\boldsymbol{y})$ than those of the form (5.26).

SVM Kernels – Properties and Examples. We have seen that, at least for SVM kernels of the form (5.26), making a choice for the $K(\boldsymbol{x},\boldsymbol{y})$ implies choosing a specific type of pre-processing[3]. Not all non-negative and symmetric kernels $K(\boldsymbol{x},\boldsymbol{y})$ can be written in the form (5.26), however; the conditions for this to be possible are given in Mercer's Theorem (which we will not prove here):

> A symmetric kernel $K(\boldsymbol{x},\boldsymbol{y})$ has an expansion of the form $K(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{\infty}\phi_i(\boldsymbol{x})\phi_i(\boldsymbol{y})$ in the function space $L^2(\Re^M)$ if and only if
>
> $$0 < \int d\boldsymbol{x}d\boldsymbol{y}\ g(\boldsymbol{x})K(\boldsymbol{x},\boldsymbol{y})g(\boldsymbol{y}) < \infty \qquad \text{for all functions } \ g \in L^2(\Re^M)$$

Apparently, choosing an acceptable kernel directly, as opposed to via (5.26), will generally be equivalent to working with an *infinite* number of pre-processing functions. To see this more explicitly, we inspect a couple of kernel examples:

[3]Note: choosing bi-linear kernels $K(\boldsymbol{x},\boldsymbol{y}) = \sum_{ij} x_i A_{ij} y_j$ is pointless: we would then assume linear separability of the original data, and would be better of classifying directly via the optimal plane in input space.

- Kernels generating *pre-processing via polynomials*: $K(\boldsymbol{x},\boldsymbol{y}) = [\boldsymbol{x}\cdot\boldsymbol{y}+1]^d$

  Expansion of this type of kernel gives:

$$\begin{aligned} K(\boldsymbol{x},\boldsymbol{y}) &= 1 + d(\boldsymbol{x}\cdot\boldsymbol{y}) + \frac{1}{2}d(d-1)(\boldsymbol{x}\cdot\boldsymbol{y})^2 + \frac{1}{6}d(d-1)(d-2)(\boldsymbol{x}\cdot\boldsymbol{y})^3 + \ldots \\ &= 1 + d\sum_i x_i y_i + \frac{1}{2}d(d-1)\sum_{ij}(x_i x_j)(y_i y_j) \\ &\quad + \frac{1}{6}d(d-1)(d-2)\sum_{ijk}(x_i x_j x_k)(y_i y_j y_k) + \ldots \end{aligned}$$

  This expression is indeed seen to be of the form $K(\boldsymbol{x},\boldsymbol{y}) = \sum_{\ell=1}^{\infty}\phi_\ell(\boldsymbol{\xi})\phi_\ell(\boldsymbol{y})$, describing pre-processing with polynomial functions $\phi_\ell(\boldsymbol{x})$. The series truncates only if and only if d is chosen to be a positive integer.

- Kernels generating *pre-processing via radial basis functions*: $K(\boldsymbol{x},\boldsymbol{y}) = K(|\boldsymbol{x}-\boldsymbol{y}|)$

  Let us consider non-negative definite kernels of the form $K(\boldsymbol{x}-\boldsymbol{y})$, such that the Fourier transform $\hat{K}(\boldsymbol{k})$ of the function $K(\boldsymbol{z})$ exists:

$$\hat{K}(\boldsymbol{k}) = \int d\boldsymbol{x}\; e^{i\boldsymbol{k}\cdot\boldsymbol{x}}K(\boldsymbol{x}) \qquad K(\boldsymbol{x}) = \int \frac{d\boldsymbol{k}}{(2\pi)^M}e^{-i\boldsymbol{k}\cdot\boldsymbol{x}}\hat{K}(\boldsymbol{k})$$

  We first show that $\hat{K}(\boldsymbol{k}) \geq 0$ for all $\boldsymbol{k}\in\Re^M$, due to K being a non-negative kernel. This follows from working out

$$\int d\boldsymbol{y}\; K(\boldsymbol{x},\boldsymbol{y})e^{i\boldsymbol{k}\cdot\boldsymbol{y}} = \int d\boldsymbol{y}\; K(|\boldsymbol{x}-\boldsymbol{y}|))e^{i\boldsymbol{k}\cdot\boldsymbol{y}} = \hat{K}(\boldsymbol{k})e^{i\boldsymbol{k}\cdot\boldsymbol{x}}$$

  Thus the $\hat{K}(\boldsymbol{k})$ are seen to be eigenvalues of the kernel K, and therefore non-negative. We can now construct the following symmetric kernel:

$$L(\boldsymbol{x},\boldsymbol{y}) = L(\boldsymbol{x}-\boldsymbol{y}) \qquad L(\boldsymbol{x}) = \int \frac{d\boldsymbol{k}}{(2\pi)^M}\; e^{-i\boldsymbol{k}\cdot\boldsymbol{x}}\sqrt{\hat{K}(\boldsymbol{k})}$$

  We work out the product

$$\begin{aligned} \int d\boldsymbol{z}\; L(\boldsymbol{x}-\boldsymbol{z})L(\boldsymbol{z}-\boldsymbol{y}) &= \int \frac{d\boldsymbol{u}d\boldsymbol{v}}{(2\pi)^{2M}}\sqrt{\hat{K}(\boldsymbol{u})\hat{K}(\boldsymbol{v})}\int d\boldsymbol{z}\; e^{-i\boldsymbol{u}\cdot(\boldsymbol{x}-\boldsymbol{z})-i\boldsymbol{v}\cdot(\boldsymbol{z}-\boldsymbol{y})} \\ &= \int \frac{d\boldsymbol{u}d\boldsymbol{v}}{(2\pi)^{2M}}\sqrt{\hat{K}(\boldsymbol{u})\hat{K}(\boldsymbol{v})}e^{-i\boldsymbol{u}\cdot\boldsymbol{x}+i\boldsymbol{v}\cdot\boldsymbol{y}}\int d\boldsymbol{z}\; e^{i\boldsymbol{z}\cdot(\boldsymbol{u}-\boldsymbol{v})} \\ &= \int \frac{d\boldsymbol{u}}{(2\pi)^M}\hat{K}(\boldsymbol{u})e^{-i\boldsymbol{u}\cdot(\boldsymbol{x}-\boldsymbol{y})} = K(\boldsymbol{x}-\boldsymbol{y}) \end{aligned}$$

  (where we used the identity $\delta(\boldsymbol{z}) = (2\pi)^{-M}\int d\boldsymbol{u}\; e^{i\boldsymbol{u}\cdot\boldsymbol{z}}$, see also appendix C). If $K(\boldsymbol{x})$ is of the form $K(|\boldsymbol{x}|)$, then also $\hat{K}(\boldsymbol{k}) = \hat{K}(|\boldsymbol{k}|)$ and $L(\boldsymbol{x}) = L(|\boldsymbol{x}|)$, so that

$$K(|\boldsymbol{x}-\boldsymbol{y}|) = \int d\boldsymbol{z}\; \phi_{\boldsymbol{z}}(\boldsymbol{x})\phi_{\boldsymbol{z}}(\boldsymbol{y}), \qquad \phi_{\boldsymbol{z}}(\boldsymbol{x}) = L(|\boldsymbol{x}-\boldsymbol{z}|)$$

  This describes pre-processing with radial basis functions.

- Kernels generating *pre-processing via sigmoidal neurons*: $K(\boldsymbol{x},\boldsymbol{y}) = g[\boldsymbol{x}\cdot\boldsymbol{y}]$

  Here we choose $g[u]$ to be sigmoidal function, such as $g[u] = \tanh[au+b]$, with constants $a, b \in \Re$. This kernel can not always be written in the separable form $K(\boldsymbol{x},\boldsymbol{y}) = \sum_{\ell\geq 1}\phi_\ell(\boldsymbol{x})\phi_\ell(\boldsymbol{y})$, but by simply working out equation (5.27) it becomes clear that we will be doing pre-processing via a 'hidden' layer of sigmoidal neurons:

$$S(\boldsymbol{x}) = \text{ sgn}\left[\sum_{\mu=1}^{p}\lambda_\mu t^\mu g[\boldsymbol{x}^\mu\cdot\boldsymbol{x}] + w_0\right] \qquad w_0 = -\frac{1}{2}\sum_{\mu=1}^{p}\lambda_\mu t^\mu\left\{g[\boldsymbol{x}^\mu\cdot\boldsymbol{x}^+] + g[\boldsymbol{x}^\mu\cdot\boldsymbol{x}^-]\right\}$$

  The number of support vectors will be the number of hidden neurons, and the input-to-hidden weights are given by the support vectors $\boldsymbol{\xi}^\mu$ themselves.

Application Example. We close this section with an example of the application of Support Vector Machines to a binary classification task which is not linearly separable. We have generated a data set D with $p = 30$ and with data points $\boldsymbol{\xi}^\mu \in \Re^2$; half of these points have $t^\mu = 1$, these are located close to the origin, the other half have $t^\mu = -1$ and are located further away from the origin. We try to separate the two classes with an SVM using the polynomial kernel $K(\boldsymbol{x},\boldsymbol{y}) = [\boldsymbol{x}\cdot\boldsymbol{y}+1]^d$. The resulting decision boundaries (5.29) in the input plane $\Re^2$ are shown, together with the input vectors, in figure 5.2 (after solving numerically the constrained optimization problem (5.28) for the $\{\lambda_\mu\}$) for different values of the parameter $d \in \{2,3,4,5\}$. Note that for the present kernel, the parameter d controls the number of pre-processing functions used. As d is increased (and thus the complexity of the SVM) we observe decision boundaries with increasingly detailed class separation, and also a decrease in the number of support vectors. This example illustrates also one of the unresolved issues with Support Vector Machines (which is in fact being studied intensively at the moment): the method appears to be outside the province of Bayesian analysis, and hence does not yet allow for dealing properly with noisy data, overfitting and decision reliability.

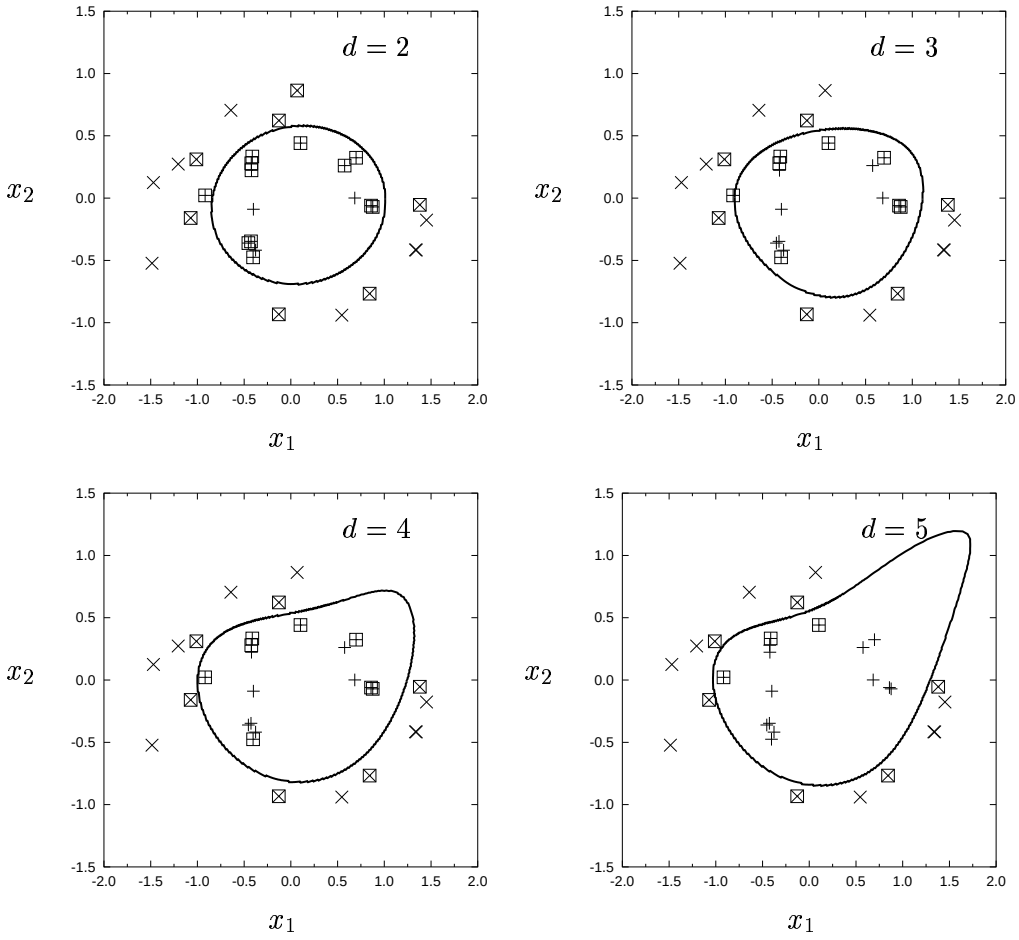


Figure 5.2: Binary classification of $p = 30$ non-linearly-separable input vectors $\boldsymbol{x}^\mu = (x_1^\mu, x_2^\mu)$ (half with $t^\mu = 1$, close to the origin, indicated by '+'; the other half with $t^\mu = -1$, away from the origin, indicated by '×'), by an SVM systems with kernel $K(\boldsymbol{x}, \boldsymbol{y}) = [1 + \boldsymbol{x} \cdot \boldsymbol{y}]^d$. From top left to bottom right: $d \in \{2, 3, 4, 5\}$. Thick curve: the decision boundary as given by (5.29). The support vectors of the solution are those marked with '□'.

Appendix A

Probability in a Nutshell

We define 'events' $\boldsymbol{x}$ as n-dimensional vectors, drawn from some event set $A \subseteq \Re^n$. We associate with each event a real-valued and non-negative probability measure $p(\boldsymbol{x}) \geq 0$.

Discrete Event Sets

Definitions & Conventions. If A is discrete and countable, each component x_i of $\boldsymbol{x}$ can only assume values from a discrete set A_i so $A \subseteq A_1 \otimes A_2 \otimes \ldots \otimes A_n$. We drop explicit mentioning of sets where possible; e.g. $\sum_{x_i}$ will mean $\sum_{x_i \in A_i}$, and $\sum_{\boldsymbol{x}}$ will mean $\sum_{\boldsymbol{x} \in A}$, etc. No problems arise as long as the arguments of $p(\ldots)$ are symbols; only once we evaluate probabilities for explicit values of the arguments we need to indicate to which components of $\boldsymbol{x}$ such values are assigned. The probabilities are normalized according to $\sum_{\boldsymbol{x}} p(\boldsymbol{x}) = 1$.

Interpretation of Probability. Imagine a system which generates events $\boldsymbol{x} \in A$ sequentially, giving the infinite series $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \ldots$. We choose an arbitrary one-to-one index mapping $\pi : \{1, 2, \ldots\} \to \{1, 2, \ldots\}$, and one particular event $\boldsymbol{x} \in A$ (in that order), and calculate for the first M sequence elements $\{\boldsymbol{x}_{\pi(1)}, \ldots, \boldsymbol{x}_{\pi(M)}\}$ the frequency $f_M(\boldsymbol{x})$ with which $\boldsymbol{x}$ occurred:

$$f_M(\boldsymbol{x}) = \frac{1}{M} \sum_{m=1}^{M} \delta_{\boldsymbol{x},\boldsymbol{x}_{\pi(m)}} \qquad \left\{ \begin{array}{ll} \delta_{\boldsymbol{x},\boldsymbol{y}} = 1 & \text{if } \boldsymbol{x} = \boldsymbol{y} \\ \delta_{\boldsymbol{x},\boldsymbol{y}} = 0 & \text{if } \boldsymbol{x} \neq \boldsymbol{y} \end{array} \right.$$

We define *random events* as those generated by a system as above with the property that for each one-to-one index map π, for each event $\boldsymbol{x} \in A$ the frequency of occurrence $f_M(\boldsymbol{x})$ tends to a limit as $M \to \infty$. This limit is then defined as the 'probability' associated with $\boldsymbol{x}$:

$$\forall \boldsymbol{x} \in A : \qquad p(\boldsymbol{x}) = \lim_{M \to \infty} f_M(\boldsymbol{x})$$

Since $f_M(\boldsymbol{x}) \geq 0$ for each $\boldsymbol{x}$, and since $\sum_{\boldsymbol{x}} f_M(\boldsymbol{x}) = 1$ for any M, it follows that $p(\boldsymbol{x}) \geq 0$ and that $\sum_{\boldsymbol{x}} p(\boldsymbol{x}) = 1$ (as it should).

Marginal & Conditional Probabilities, Statistical Independence. The so-called 'marginal probablities' are obtained upon summing over individual components of $\boldsymbol{x} = (x_1, \ldots, x_n)$:

$$p(x_1, \ldots, x_{\ell-1}, x_{\ell+1}, \ldots, x_n) = \sum_{x_\ell} p(x_1, \ldots, x_n) \tag{A.1}$$

In particular we obtain (upon repeating this procedure):

$$p(x_i) = \sum_{x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n} p(x_1,\ldots,x_n) \tag{A.2}$$

Marginal probabilities are normalized. This follows upon combining their definition with the basic normalization $\sum_{\boldsymbol{x}} p(\boldsymbol{x}) = 1$, e.g.

$$\sum_{x_1,\ldots,x_{\ell-1},x_{\ell+1},\ldots,x_n} p(x_1,\ldots,x_{\ell-1},x_{\ell+1},\ldots,x_n) = 1, \qquad \sum_{x_i} p(x_i) = 1$$

For any two disjunct subsets $\{i_1,\ldots,i_k\}$ and $\{j_1,\ldots,j_\ell\}$ of the index set $\{1,\ldots,n\}$ (with necessarily $k+\ell \leq n$) we next define the 'conditional probability'

$$p(x_{i_1},\ldots,x_{i_k}|x_{j_1},\ldots,x_{j_\ell}) = \frac{p(x_{i_1},\ldots,x_{i_k},x_{j_1},\ldots,x_{j_\ell})}{p(x_{j_1},\ldots,x_{j_\ell})} \tag{A.3}$$

(A.3) gives the probability that the k components $\{i_1,\ldots,i_k\}$ of $\boldsymbol{x}$ take the values $\{x_{i_1},\ldots,x_{i_k}\}$, given the knowledge that the ℓ components $\{j_1,\ldots,j_\ell\}$ take the values $\{x_{j_1},\ldots,x_{j_\ell}\}$.

The concept of statistical independence now follows naturally. Loosely speaking: statistical independence means that conditioning in the sense defined above does not affect any of the marginal probabilities. Thus the n events $\{x_1,\ldots,x_n\}$ are said to be statistically independent if for *any* two disjunct subsets $\{i_1,\ldots,i_k\}$ and $\{j_1,\ldots,j_\ell\}$ of $\{1,\ldots,n\}$ we have

$$p(x_{i_1},\ldots,x_{i_k}|x_{j_1},\ldots,x_{j_\ell}) = p(x_{i_1},\ldots,x_{i_k})$$

This can be shown to be equivalent to saying

$$\{x_1,\ldots,x_n\} \text{ are independent}: \qquad \begin{array}{l} p(x_{i_1},\ldots,x_{i_k}) = p(x_{i_1})p(x_{i_2})\ldots p(x_{i_k}) \\ \text{for every subset } \{i_1,\ldots,i_k\} \subseteq \{1,\ldots,n\} \end{array} \tag{A.4}$$

Continuous Event Sets

Definitions & Conventions. Here A is no longer countable. Each component x_i of $\boldsymbol{x}$ can assume values from a continuous set $A_i = \{x \in \Re |\ \exists \boldsymbol{x} \in A \text{ with } x_i = x\}$, and $A \subseteq A_1 \otimes A_2 \otimes \ldots \otimes A_n$. As before we drop explicit mentioning of sets where possible; e.g. $\int dx_i$ will mean $\int_{A_i} dx_i$ and $\int d\boldsymbol{x}$ will mean $\int_A d\boldsymbol{x}$, etc. The function $p(\boldsymbol{x})$ is now to be seen as a *probability density*, which is accordingly normalized via integration: $\int d\boldsymbol{x}\ p(\boldsymbol{x}) = 1$.

Interpretation of Probability. Again we imagine a system which generates events $\boldsymbol{x} \in A$ sequentially, giving $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$. We define 'boxes' (hypercubes) $B(\boldsymbol{x}, \boldsymbol{\Delta})$ in $\Re^n$ as follows:

$$B(\boldsymbol{x}, \boldsymbol{\Delta}) = \{\boldsymbol{y} \in \Re^n |\ x_i \leq y_i < x_i + \Delta_i \text{ for all } i\}$$

in which all $\Delta_i > 0$. The volume of such a box is simply $\prod_{i=1}^n \Delta_i$. We choose an arbitrary one-to-one index mapping $\pi : \{1,2,\ldots\} \to \{1,2,\ldots\}$, and one particular event $\boldsymbol{x} \in A$ (in that order), and calculate for the first M sequence elements $\{\boldsymbol{x}_{\pi(1)},\ldots,\boldsymbol{x}_{\pi(M)}\}$ the frequency $f_M(\boldsymbol{x}, \boldsymbol{\Delta})$ with which $\boldsymbol{x}$ happened to lie in box $B(\boldsymbol{x}, \boldsymbol{\Delta})$:

$$f_M(\boldsymbol{x}, \boldsymbol{\Delta}) = \frac{1}{M} \sum_{m=1}^{M} I(\boldsymbol{x}_{\pi(m)}; \boldsymbol{x}, \boldsymbol{\Delta}) \qquad \left\{ \begin{array}{ll} I(\boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\Delta}) = 1 & \text{if } \boldsymbol{y} \in B(\boldsymbol{x}, \boldsymbol{\Delta}) \\ I(\boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\Delta}) = 0 & \text{if } \boldsymbol{y} \notin B(\boldsymbol{x}, \boldsymbol{\Delta}) \end{array} \right.$$

We define *random events* as those generated by a system as above with the property that for each one-to-one index map π, for each box $B(\boldsymbol{x}, \boldsymbol{\Delta})$ the frequency of occurrence $f_M(\boldsymbol{x}, \boldsymbol{\Delta})$ tends to a limit as $M \to \infty$. This limit is subsequently used to define the 'probability density' $p(\boldsymbol{x})$ associated with $\boldsymbol{x}$:

$$\forall \boldsymbol{x} \in A: \qquad p(\boldsymbol{x}) = \lim_{\boldsymbol{\Delta} \to \boldsymbol{0}} \frac{\lim_{M\to\infty} f_M(\boldsymbol{x}, \boldsymbol{\Delta})}{\prod_{i=1}^n \Delta_i}$$

(provided this limit exists). Since $f_M(\boldsymbol{x}, \boldsymbol{\Delta}) \geq 0$ for each $\boldsymbol{x}$, and since $\int d\boldsymbol{x}\ f_M(\boldsymbol{x}, \boldsymbol{\Delta}) = \prod_{i=1}^n \Delta_i$ for any M, it follows that $p(\boldsymbol{x}) \geq 0$ and that $\int d\boldsymbol{x}\ p(\boldsymbol{x}) = 1$ (as it should).

Marginal & Conditional Probability Densities, Statistical Independence. The marginal probabilities are now obtained upon integrating over individual components of $\boldsymbol{x}$:

$$p(x_1, \ldots, x_{\ell-1}, x_{\ell+1}, \ldots, x_n) = \int dx_\ell\ p(x_1, \ldots, x_n) \tag{A.5}$$

In particular we obtain (upon repeating this procedure):

$$p(x_i) = \int_d x_1 \ldots dx_{i-1} dx_{i+1} \ldots dx_n\ p(x_1, \ldots, x_n) \tag{A.6}$$

Again marginal probabilities are normalized (in integration sense). This follows as before upon combining their definition with the basic normalization $\int d\boldsymbol{x}\ p(\boldsymbol{x}) = 1$, e.g.

$$\int dx_1 \ldots dx_{\ell-1} dx_{\ell+1} \ldots dx_n\ p(x_1, \ldots, x_{\ell-1}, x_{\ell+1}, \ldots, x_n) = 1, \qquad \int dx_i\ p(x_i) = 1$$

For any two disjunct subsets $\{i_1, \ldots, i_k\}$ and $\{j_1, \ldots, j_\ell\}$ of the index set $\{1, \ldots, n\}$ we define the 'conditional probability density'

$$p(x_{i_1}, \ldots, x_{i_k} | x_{j_1}, \ldots, x_{j_\ell}) = \frac{p(x_{i_1}, \ldots, x_{i_k}, x_{j_1}, \ldots, x_{j_\ell})}{p(x_{j_1}, \ldots, x_{j_\ell})} \tag{A.7}$$

It gives the probability density for the k components $\{i_1, \ldots, i_k\}$, given the knowledge that the ℓ components $\{j_1, \ldots, j_\ell\}$ take the values $\{x_{j_1}, \ldots, x_{j_\ell}\}$.

Statistical independence can as before be defined in the following way:

$$\{x_1, \ldots, x_n\} \text{ are independent}: \qquad \begin{array}{l} p(x_{i_1}, \ldots, x_{i_k}) = p(x_{i_1}) p(x_{i_2}) \ldots p(x_{i_k}) \\ \text{for every subset } \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\} \end{array} \tag{A.8}$$

Averages of Specific Random Variables

We define 'random variables' as arbitrary functions $F(\boldsymbol{x})$ of random events $\boldsymbol{x} \in A$. We define *averages* or *expectation values* or *mean values* $\langle F(\boldsymbol{x}) \rangle$ of random variables $F(\boldsymbol{x})$ as follows:

$$\begin{array}{ll} \text{discrete random variables}: & \langle F(\boldsymbol{x}) \rangle = \sum_{\boldsymbol{x}} p(\boldsymbol{x}) F(\boldsymbol{x}) \\ \text{continuous random variables}: & \langle F(\boldsymbol{x}) \rangle = \int d\boldsymbol{x}\ p(\boldsymbol{x}) F(\boldsymbol{x}) \end{array} \tag{A.9}$$

Let us now turn to the definitions and properties of a couple of relevant random variables (discrete or continuous) and averages. Note that in general one cannot be sure beforehand (without explicit proof) that the following averages will actually exist (i.e. are finite):

average: $\mu_i = \langle x_i \rangle$

variance: $\sigma_i^2 = \langle x_i^2 \rangle - \langle x_i \rangle^2$

σ_i^2 is non-negative, since $\langle x_i^2 \rangle - \langle x_i \rangle^2 = \langle [x_i - \langle x_i \rangle]^2 \rangle$. This also shows that $\sigma_i^2 = 0$ implies that $x_i = x_i'$ for any two events $\boldsymbol{x} \in A$ and $\boldsymbol{x}' \in A$ with nonzero probabilities.

covariance matrix: $C_{ij} = \langle x_i x_j \rangle - \langle x_i \rangle \langle x_j \rangle$

Note that $C_{ii} = \sigma_i^2$. The covariance matrix is symmetric so all eigenvalues are real. It is non-negative definite (so all eigenvalues are non-negative), since it can be written as $\langle x_i x_j \rangle - \langle x_i \rangle \langle x_j \rangle = \langle (x_i - \langle x_i \rangle)(x_j - \langle x_j \rangle) \rangle$, from which it follows that

$$\text{for any } \boldsymbol{z} \in \Re^n : \qquad \boldsymbol{z} \cdot \boldsymbol{C} \boldsymbol{z} = \langle \{ \sum_{i=1}^{n} z_i [x_i - \langle x_i \rangle] \}^2 \rangle \geq 0$$

moments: $\langle x_{i_1}^{m_{i_1}} x_{i_2}^{m_{i_2}} \ldots x_{i_k}^{m_{i_k}} \rangle$, with $m_{i_\ell} \in \{0, 1, 2, 3, \ldots\}$

Gaussian (or 'Normal') Distribution:

$$p(x) = \frac{e^{-\frac{1}{2}(x-\mu)^2/\sigma^2}}{\sigma\sqrt{2\pi}}, \qquad x \in \Re \tag{A.10}$$

Normalization is built-in (see also appendix B):

$$\int dx\ p(x) = \int \frac{dx}{\sigma\sqrt{2\pi}}\ e^{-\frac{1}{2}(x-\mu)^2/\sigma^2} = \int \frac{dz}{\sqrt{2\pi}}\ e^{-\frac{1}{2}z^2} = 1$$

For the Gaussian distribution we also obtain:

$$\langle (x-\mu)^n \rangle = \int \frac{dx}{\sigma\sqrt{2\pi}} (x-\mu)^n e^{-\frac{1}{2}(x-\mu)^2/\sigma^2} = \sigma^n \int \frac{dz}{\sqrt{2\pi}}\ z^n e^{-\frac{1}{2}z^2}$$

For n odd the result is zero. For n even we get:

$$\langle (x-\mu)^{2m} \rangle = \sigma^{2m} (-1)^m \lim_{y \to \frac{1}{2}} \frac{d^m}{dy^m} \int \frac{dz}{\sqrt{2\pi}} e^{-yz^2} = \frac{\sigma^{2m}}{\sqrt{2}} (-1)^m \lim_{y \to \frac{1}{2}} \frac{d^m}{dy^m}\ y^{-\frac{1}{2}}$$

$$= \frac{\sigma^{2m}}{\sqrt{2}} \lim_{y \to \frac{1}{2}} \left[\frac{1}{2} . \frac{3}{2} . \cdots . \frac{2m-1}{2} \right] y^{-\frac{1}{2}-m} = \sigma^{2m} \left[1.3. \cdots .(2m-1) \right]$$

From these relations, in turn, follow the moments. For example:

$$\begin{aligned}
\langle (x-\mu) \rangle = 0 \quad &\Rightarrow \quad \langle x \rangle = \mu \\
\langle (x-\mu)^2 \rangle = \sigma^2 \quad &\Rightarrow \quad \langle x^2 \rangle = \mu^2 + \sigma^2 \\
\langle (x-\mu)^3 \rangle = 0 \quad &\Rightarrow \quad \langle x^3 \rangle = \mu^3 + 3\mu\sigma^2 \\
\langle (x-\mu)^4 \rangle = 0 \quad &\Rightarrow \quad \langle x^4 \rangle = 3(\mu^2 + \sigma^2)^2
\end{aligned}$$

Relations such as $\langle x^4 \rangle = 3(\mu^2 + \sigma^2)^2$ can often serve as a quick test to see whether an unknown distribution could be Gaussian.

Appendix B

Gaussian Integrals

Real, Symmetric, Positive Definite Matrices. The symmetric $N \times N$ matrix $\boldsymbol{A}$ is assumed to be positive definite, i.e. $\boldsymbol{x} \cdot \boldsymbol{A}\boldsymbol{x} > 0$ for all $\boldsymbol{x} \in \Re^N$ with $|\boldsymbol{x}| \neq 0$. The eigenvalue polynomial $\det[\boldsymbol{A} - \lambda \mathbb{I}] = 0$ is of order N, so $\boldsymbol{A}$ will have N (possibly complex) solutions λ (some may coincide) of the eigenvalue problem $\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}$ ($\boldsymbol{x} \neq 0$). We denote complex conjugation of complex numbers z as: $z = a + ib$, $z^* = a - ib$ ($a, b \in \Re$), and $|z|^2 = z^* z \in \Re$. We define the unit matrix $\mathbb{I}_{ij} = \delta_{ij}$. We will use the following facts, proofs of which are found in any elementary linear algebra course:

Fact 1: All eigenvalues of the matrix $\boldsymbol{A}$ are real.

Fact 2: All eigenvectors can be chosen real-valued.

Fact 3: All eigenvalues λ are positive.

Fact 4: We can construct a complete orthogonal basis in $\Re^N$ of $\boldsymbol{A}$-eigenvectors.

Thus there exist a set of vectors $\{\hat{\boldsymbol{e}}^i\}$ ($\lambda = 1, \ldots, N$) with the properties:

$$\boldsymbol{A}\hat{\boldsymbol{e}}^i = \lambda_i \hat{\boldsymbol{e}}^i \qquad \lambda_i \in \Re,\ \lambda_i > 0 \qquad \hat{\boldsymbol{e}}^i \in \Re^N,\ \hat{\boldsymbol{e}}^i \cdot \hat{\boldsymbol{e}}^j = \delta_{ij}$$

We can now bring $\boldsymbol{A}$ onto diagonal form by a unitary transformation $\boldsymbol{U}$, which we construct from the components of the normalised eigenvectors $\hat{\boldsymbol{e}}$: $U_{ij} = \hat{e}^j_i$. We denote its transpose by $\boldsymbol{U}^\dagger$, $U^\dagger_{ij} = U_{ji}$, and show that $\boldsymbol{U}$ is indeed unitary, i.e. $\boldsymbol{U}^\dagger \boldsymbol{U} = \boldsymbol{U}\boldsymbol{U}^\dagger = \mathbb{I}$:

$$\sum_j (\boldsymbol{U}^\dagger \boldsymbol{U})_{ij} x_j = \sum_{jk} U^{ki} U_{kj} x_j = \sum_{jk} \hat{e}^i_k \hat{e}^j_k x_j = \sum_j \delta_{ij} x_j = x_i$$

$$\sum_j (\boldsymbol{U}\boldsymbol{U}^\dagger)_{ij} x_j = \sum_{jk} U^{ik} U_{jk} x_j = \sum_{jk} \hat{e}^k_i \hat{e}^k_j x_j = \sum_k \hat{e}^k_i (\hat{\boldsymbol{e}} \cdot \boldsymbol{x}) = x_i$$

(since $\{\hat{\boldsymbol{e}}^\ell\}$ forms a complete orthogonal basis). From $\boldsymbol{U}$ being unitary it follows that $\boldsymbol{U}$ and $\boldsymbol{U}^\dagger$ leave inner products invariant:

$$\boldsymbol{U}\boldsymbol{x} \cdot \boldsymbol{U}\boldsymbol{y} = \boldsymbol{x} \cdot \boldsymbol{U}^\dagger \boldsymbol{U}\boldsymbol{y} = \boldsymbol{x} \cdot \boldsymbol{y} \qquad \boldsymbol{U}^\dagger \boldsymbol{x} \cdot \boldsymbol{U}^\dagger \boldsymbol{y} = \boldsymbol{x} \cdot \boldsymbol{U}\boldsymbol{U}^\dagger \boldsymbol{y} = \boldsymbol{x} \cdot \boldsymbol{y}$$

We can see explicitly that $\boldsymbol{U}$ indeed brings $\boldsymbol{A}$ onto diagonal form:

$$(\boldsymbol{U}^\dagger \boldsymbol{A}\boldsymbol{U})_{ij} = \sum_{kl=1}^N U^\dagger_{ik} A_{kl} U_{lj} = \sum_{kl=1}^N \hat{e}^i_k A_{kl} \hat{e}^j_l = \lambda_j \sum_{k=1}^N \hat{e}^i_k \hat{e}^j_k = \lambda_j \delta_{ij} \tag{B.1}$$

The inverse $\boldsymbol{A}^{-1}$ of the matrix $\boldsymbol{A}$ exists, and can be written as $(\boldsymbol{A}^{-1})_{ij} = \sum_{k=1}^N \lambda_k^{-1} \hat{e}^k_i \hat{e}^k_j$.

Gaussian Integrals. We now turn to the associated Gaussian integrals

$$I = \int d\boldsymbol{x}\ f(\boldsymbol{x}) e^{-\frac{1}{2}\boldsymbol{x}\cdot\boldsymbol{A}\boldsymbol{x}} \tag{B.2}$$

The simplest such integral is $I = \int dx\ e^{-\frac{1}{2}x^2}$. It is calculated by writing its square as an integral in $\Re^2$, and switching to polar coordinates, ($z_1 = r\cos\phi,\ z_2 = r\sin\phi$). The Jacobian of this transformation is r. Hence

$$I^2 = \int dz_1 dz_2\ e^{-\frac{1}{2}\boldsymbol{z}^2} = \int_0^{2\pi} d\phi \int_0^{\infty} dr\ re^{-\frac{1}{2}r^2} = 2\pi\left[-e^{-\frac{1}{2}r^2}\right]_0^{\infty} = 2\pi \qquad \text{so} \qquad I = \sqrt{2\pi}$$

For $f(\boldsymbol{x}) = 1$ we can do the integral (B.2) by using the previous results on the diagonalisability of the matrix $\boldsymbol{A}$. We put $\boldsymbol{x} = \boldsymbol{U}\boldsymbol{z}$ (since $\boldsymbol{U}$ leaves inner products invariant: $d\boldsymbol{x} = d\boldsymbol{z}$):

$$\int d\boldsymbol{x}\ e^{-\frac{1}{2}\boldsymbol{x}\cdot\boldsymbol{A}\boldsymbol{x}} = \int d\boldsymbol{z}\ e^{-\frac{1}{2}\boldsymbol{z}\cdot\boldsymbol{U}^\dagger\boldsymbol{A}\boldsymbol{U}\boldsymbol{z}} = \prod_{\ell=1}^{N}\left[\int dz\ e^{-\frac{1}{2}\lambda_\ell z^2}\right]$$

$$= \left[\prod_{\ell=1}^{N}\frac{1}{\sqrt{\lambda_\ell}}\right]\left[\int dz\ e^{-\frac{1}{2}z^2}\right]^N = \frac{(2\pi)^{N/2}}{\sqrt{\det\boldsymbol{A}}} \tag{B.3}$$

(note: the determinant of $\boldsymbol{A}$ is unvariant under rotations, so it can be evaluated with $\boldsymbol{A}$ on diagonal form, which gives the product of the N eigenvalues).

Due to the symmetry of the integrand in (B.2) under reflection $\boldsymbol{x} \to -\boldsymbol{x}$, the integral reduces to zero for $f(\boldsymbol{x}) = x_i$. For $f(\boldsymbol{x}) = x_i x_j$ we find:

$$\int d\boldsymbol{x}\ x_i x_j e^{-\frac{1}{2}\boldsymbol{x}\cdot\boldsymbol{A}\boldsymbol{x}} = \lim_{\boldsymbol{b}\to 0}\frac{\partial^2}{\partial b_i \partial b_j}\int d\boldsymbol{x}\ e^{-\frac{1}{2}\boldsymbol{x}\cdot\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}\cdot\boldsymbol{x}}$$

$$= \lim_{\boldsymbol{b}\to 0}\frac{\partial^2}{\partial b_i \partial b_j}\int d\boldsymbol{z}\ e^{-\frac{1}{2}\boldsymbol{z}\cdot\boldsymbol{U}^\dagger\boldsymbol{A}\boldsymbol{U}\boldsymbol{z}+\boldsymbol{z}\cdot\boldsymbol{U}^\dagger\boldsymbol{b}} = \lim_{\boldsymbol{b}\to 0}\frac{\partial^2}{\partial b_i \partial b_j}\prod_{\ell=1}^{N}\left[\int dz\ e^{-\frac{1}{2}\lambda_\ell z^2 + z(\boldsymbol{U}^\dagger\boldsymbol{b})_\ell}\right]$$

$$= \lim_{\boldsymbol{b}\to 0}\frac{\partial^2}{\partial b_i \partial b_j}\prod_{\ell=1}^{N}\left[\int dz\ e^{-\frac{1}{2}\lambda_\ell[z-(\boldsymbol{U}^\dagger\boldsymbol{b})_\ell\lambda_\ell^{-1}]^2+\frac{1}{2}\lambda_\ell^{-1}(\boldsymbol{U}^\dagger\boldsymbol{b})_\ell^2}\right]$$

$$= \lim_{\boldsymbol{b}\to 0}\frac{\partial^2}{\partial b_i \partial b_j} e^{\frac{1}{2}\sum_{ij\ell=1}^{N}\lambda_\ell^{-1}U_{i\ell}b_i U_{j\ell}b_j}\prod_{\ell=1}^{N}\left[\int dz\ e^{-\frac{1}{2}\lambda_\ell z^2}\right]$$

$$= \frac{(2\pi)^{N/2}}{\sqrt{\det\boldsymbol{A}}}\lim_{\boldsymbol{b}\to 0}\frac{\partial^2}{\partial b_i \partial b_j} e^{\frac{1}{2}\sum_{ij=1}^{N}b_i b_j\sum_{\ell=1}^{N}\lambda_\ell^{-1}\hat{e}_i^\ell\hat{e}_j^\ell} = (A^{-1})_{ij}\frac{(2\pi)^{N/2}}{\sqrt{\det\boldsymbol{A}}} \tag{B.4}$$

In particular, by combining the last two results, we find a powerful (and completely general) relation for Gaussian probability distributions with zero mean $\boldsymbol{x} = 0$ (the latter one can always achieve by a simple translation):

$$\frac{\int d\boldsymbol{x}\ x_i x_j e^{-\frac{1}{2}\boldsymbol{x}\cdot\boldsymbol{A}\boldsymbol{x}}}{\int d\boldsymbol{x}\ e^{-\frac{1}{2}\boldsymbol{x}\cdot\boldsymbol{A}\boldsymbol{x}}} = (A^{-1})_{ij} \tag{B.5}$$

If we know that a given distribution is Gaussian, with zero average, we apparently only need to calculate the correlations $\langle x_i x_j\rangle$ to know the full distribution:

$$P(\boldsymbol{x})\ \text{Gaussian, with}\ \langle\boldsymbol{x}\rangle = 0 \qquad \Rightarrow \qquad P(\boldsymbol{x}) = \frac{e^{-\frac{1}{2}\boldsymbol{x}\cdot\boldsymbol{A}\boldsymbol{x}}}{(2\pi)^{N/2}\det^{-\frac{1}{2}}\boldsymbol{A}},\ \text{with}\ (A^{-1})_{ij} = \langle x_i x_j\rangle \tag{B.6}$$

Appendix C

The δ-Distribution

Definition. There are several ways of introducing the δ-distribution. Here we will go for an intuitive definition first, and a formal one later. We define the δ-distribution as the probability distribution $\delta(x)$ corresponding to a random variable in the limit where the randomness in the variable vanishes. If x is 'distributed' around zero, this implies

$$\int dx\ f(x)\delta(x) = f(0) \qquad \text{for any function } f$$

The problem arises when we want to actually write down an expression for $\delta(x)$. Intuitively one could think of writing something like

$$\delta(x) = \lim_{\Delta\to 0} G_\Delta(x) \qquad G_\Delta(x) = \frac{1}{\Delta\sqrt{2\pi}} e^{-\frac{1}{2}x^2/\Delta^2} \tag{C.1}$$

This is not a true function in a mathematical sense; $\delta(x)$ is zero for $x \neq 0$ and $\delta(0) = \infty$. The way to interpret and use expressions like (C.1) is to realise that $\delta(x)$ only has a meaning when appearing inside an integration. One then takes the limit $\Delta \to 0$ *after* performing the integration. Upon adopting this convention, we can use (C.1) to derive the following properties (for sufficiently well-behaved and differentiable functions f[1]):

$$\int dx\ \delta(x)f(x) = \lim_{\Delta\to 0}\int dx\ G_\Delta(x)f(x) = \lim_{\Delta\to 0}\int \frac{dx}{\sqrt{2\pi}}\ e^{-\frac{1}{2}x^2} f(\Delta x) = f(0)$$

$$\int dx\ \delta'(x)f(x) = \lim_{\Delta\to 0}\int dx \left\{ \frac{d}{dx}\left[G_\Delta(x)f(x)\right] - G_\Delta(x)f'(x)\right\}$$

$$= \lim_{\Delta\to 0}\left[G_\Delta(x)f(x)\right]_{-\infty}^{\infty} - f'(0) = -f'(0)$$

both can be summarised in and generalised to the single expression:

$$\int dx\ f(x)\frac{d^n}{dx^n}\delta(x) = (-1)^n \lim_{x\to 0}\frac{d^n}{dx^n}f(x) \qquad (n = 0, 1, 2, \ldots) \tag{C.2}$$

[1] The conditions on the so-called 'test-functions' f can be properly formalised; this being not a course on distribution theory, here we just concentrate on the basic ideas and properties

Equivalently we can take the result (C.2) as our definition of the δ-distribution.

Representations, Relations, Generalisations. We can use the definitions of Fourier transforms and inverse Fourier transforms to obtain an integral representation of the δ-distribution:

$$\mathcal{F}: \ f(x) \to \hat{f}(k) \qquad \hat{f}(k) = \int dx \ e^{-2\pi ikx} f(x)$$

$$\mathcal{F}^{-1}: \ \hat{f}(k) \to f(x) \qquad f(x) = \int dk \ e^{2\pi ikx} \hat{f}(k)$$

In combination these relations give the identity:

$$f(x) = \int dk \ e^{2\pi ikx} \int dy \ e^{-2\pi iky} f(y)$$

Application to $f(x) = \delta(x)$ gives:

$$\delta(x) = \int dk \ e^{2\pi ikx} = \int \frac{dk}{2\pi} \ e^{ikx} \tag{C.3}$$

A second useful relation is the following one, which relates the δ-distribution to the step-function:

$$\delta(x) = \frac{d}{dx}\theta(x) \tag{C.4}$$

This we prove by showing that both have the same effect inside an integration (with an arbitrary test-function):

$$\int dx \left[\delta(x) - \frac{d}{dx}\theta(x)\right] \phi(x) = \phi(0) - \lim_{\epsilon \to 0} \int_{-\epsilon}^{\epsilon} dx \left\{ \frac{d}{dx} \left[\theta(x)\phi(x)\right] - \phi'(x)\theta(x) \right\}$$

$$= \phi(0) - \lim_{\epsilon \to 0} \left[\phi(\epsilon) - 0\right] + \lim_{\epsilon \to 0} \int_{0}^{\epsilon} dx \ \phi'(x) = 0$$

Thirdly we can inspect the effect of performing a continuously differentiable and invertible transformation f on a variable that occurs inside a δ-distribution, giving rise to the following identity:

$$\delta\left[f(x) - f(a)\right] = \frac{\delta(x-a)}{f'(a)} \tag{C.5}$$

Again this is proved by showing that both sides have the same effect inside an integration (with an arbitrary test-function):

$$\int dx \ \phi(x) \left\{ \delta\left[f(x) - f(a)\right] - \frac{\delta(x-a)}{f'(a)} \right\} = \int dx \ f'(x) \left[\frac{\phi(x)}{f'(x)}\right] \delta\left[f(x) - f(a)\right] - \frac{\phi(a)}{f'(a)}$$

$$= \int dk \ \frac{\phi(f^{-1}(k))}{f'(f^{-1}(k))} \delta\left[k - f(a)\right] - \frac{\phi(a)}{f'(a)} = \frac{\phi(f^{-1}(f(a)))}{f'(f^{-1}(f(a)))} - \frac{\phi(a)}{f'(a)} = 0$$

Finally, the following generalisation is straightforward:

$$\boldsymbol{x} \in \mathcal{R}^N: \qquad \delta(\boldsymbol{x}) = \prod_{i=1}^{N} \delta(x_i) \tag{C.6}$$