Advances in Probabilistic Modelling

Martin John Oldfield
mjo@mrao.cam.ac.uk

Trinity College
Cambridge

A dissertation submitted in candidature for the degree of Doctor of Philosophy in the University of Cambridge.
This is the second edition which corrects about a dozen minor typographical errors present in the original. The style and size have also been changed.

First edition: March 1995.
Second edition: May 1995.

Acknowledgements

Over the past years I have become greatly indebted to many of the people in MRAO; to paraphrase a rather more gifted Trinity scientist, if I have seen anything it is because I not so much stood on but was hoisted by others. To this end I thank my supervisor, Steve Gull, not only for his copious supply of rope but also his skill which has kept me from hanging myself. Great thanks are also due to David Robinson, Mark Charter, Anton Garrett and David MacKay for many interesting and enjoyable discussions and helpful comments on this manuscript. In the course of number plate reading Mark and I have also found ourselves in many crazy situations which I hope we both remember with amusement. On the academic front I would also like to thank those unfortunate enough to be saddled with me as a physics supervisor; I can only hope that I taught them as much as they managed to teach me!

Dave Titterington deserves special mention for two things: for keeping the MRAO Sun cluster alive and letting me run a frightening number of jobs on it, and for keeping the tea-room stocked with both coffee and biscuits!

There are also many people who have kept me both sane and happy over some or all of the duration of this research. To choose some people is to not choose others and so this is an inevitably invidious task, but I think it would be unfair to not mention Isabel, Simon and Kevin all of whom have made my life much happier than it would otherwise have been.

Finally I would like to thank Kate for tolerating both me and my musical tastes.

Preface

This dissertation describes work carried out between October 1990 and March 1995 at the Mullard Radio Astronomy Observatory in the Cavendish Laboratory, Cambridge.

With the exception of chapter four, which has been done in collaboration with David J. C. MacKay and where explicit reference is made to the work of others, this dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration. It has not, nor has any similar dissertation, been submitted for a degree, diploma or any other qualification at this or any other university. This dissertation does not exceed sixty thousand words in length.

Martin Oldfield.

To my family

Contents

List of Figures

List of Tables

Chapter 1

Probability for engineers

1.1 Probability

> probability: n. [((O)Fr. *probabilité* f.) L *probabilitas*, f. *probabilis*] 1. The quantity or fact of being probable; degree of likelihood; the appearance of truth, or likelihood of being realized, which a statement or event bears in the light of current evidence.
>
> *The Shorter Oxford English Dictionary*

Probability is a common enough word. Any competent dictionary will furnish a definition, often in terms of `likelihood' or `frequency of occurrence'. Such definitions are inadequate for our needs; the former is an essentially circular definition, the latter precludes us from discussing the probability of an unique event. The Oxford dictionary is better than most: in particular this definition makes it clear that `current evidence' must be considered when assigning a probability. To be fair to dictionary compilers, the definition of probability is still the subject of debate in some academic circles, but we have an intuitive notion of probability that transcends such semantic arguments. I believe the `Bayesian' viewpoint best matches our intuition, and will adopt this in the rest of this thesis without further discussion.

Definition 1.1 *A probability is a real number, between zero and unity, which encodes our strength of belief in a proposition.*

In less precise language a probability is a container for information; by assigning a probability we are encoding some piece of information in a mathematically precise way.

Suppose I am standing upright and holding a ball in my hand. The statement `The probability of this ball hitting the ground within a second of my releasing it is unity' implies we have a unshakeable faith in gravity; when released the ball *shall* hit the ground before a second has elapsed. Similarly the statement `The probability of the ball rising out of my hand is zero' discounts such a possibility ever happening. Superficially these propositions are true but suppose I am standing on a cliff-edge or in a particularly strong updraft. I might then reasonably assign different probabilities, thus illustrating that the probability of a proposition depends on the background to the assignment. It is important to understand that there are no stone-tablets inscribed with the `correct' probabilities;

rather we always assign a probability in some context, which will often modify our assignment. If a calculation involves multiple contexts we can make our assumptions explicit by discussing `the probability of a proposition assuming other propositions'. Conversely we will usually leave some assumptions implicit; a more detailed analysis of the falling ball will almost certainly assume that Newtonian mechanics is valid, but little would be gained from stating this explicitly at every step. This is analogous to the variables used in a computer subroutine; some are sufficiently constant that we choose to make them global, whilst others we pass to the subroutine as arguments.

To facilitate discussion and manipulation, I will introduce the following notation:

Definition 1.2 $\mathrm{p}(A_1,\ldots,A_m|B_1,\ldots,B_n)$ *is the probability that the propositions* $A_1,\ldots,A_m$ *are all true, assuming that the propositions* $B_1,\ldots,B_n$ *are all true.*

This may appear more general than the examples of the preceding paragraph above but we can always use Boolean algebra to reduce the multiple propositions into a single one,

$$\mathrm{p}(A_1,A_2,\ldots,A_m|B_1,B_2,\ldots,B_n) = \mathrm{p}(A|B), \tag{1.1}$$

where

$$A = \prod_{i=1}^{m} A_i, \tag{1.2}$$

$$B = \prod_{i=1}^{n} B_i. \tag{1.3}$$

The product used here may be more familiar to computer programmers as the Boolean AND operator.

1.2 A probability calculus

A single probability is not very useful. Since the probability merely encodes what we already know it can tell us nothing new. The power of this formalism only appears when we have multiple probabilities and manipulate them to calculate the probability of new propositions. The rules for manipulation form a calculus wholly derivable from simple consistency requirements. It is important we appreciate that the rules are unique in this way; it implies that any manipulations at odds with our rules will lead to inconsistency. This is perhaps the most compelling argument in favour of the Bayesian framework. The derivation, whilst elegant mathematically, is not otherwise instructive and so is not reproduced here. Instead we refer the reader to the classic paper by Cox (1946) and quote his results. It would be unfair not to cite Aczél (1966) also which removes Cox's assumption of differentiability.

Theorem 1.1 (The sum rule)

$$\mathrm{p}(A|C) + \mathrm{p}(\overline{A}|C) = 1. \tag{1.4}$$

The sum rule relates the probability that A is true to the probability that $\overline{A}$ is true i.e. A is false, both probabilities being evaluated under the same set of assumptions C. We find that these probabilities sum to unity.

Theorem 1.2 (The product rule)

$$\mathrm{p}(A,B|C) = \mathrm{p}(A|B,C)\mathrm{p}(B|C) = \mathrm{p}(B|A,C)\mathrm{p}(A|C). \tag{1.5}$$

Just as the sum rule allows us to relate the probability of a proposition to the probability of its logical opposite, the product rule allows us to relate the probability of the logical product of propositions to the probability of the individual propositions themselves.

Every electronics engineer knows that NOT and AND gates are sufficient to build any digital circuit; equivalently equations 1.4 and 1.5 will suffice to evaluate the probability of any compound proposition.

Consider for example the probability of the logical sum of propositions i.e. the probability that A OR B is true. We shall have recourse to De Morgan's theorem,

$$\overline{A+B} = \overline{A}\,\overline{B}. \tag{1.6}$$

Now

$$\begin{aligned}
\mathrm{p}(A+B|C) &= 1-\mathrm{p}(\overline{A+B}|C), && (1.7)\\
&= 1-\mathrm{p}(\overline{A},\overline{B}|C), && (1.8)\\
&= 1-\mathrm{p}(\overline{A}|C)\mathrm{p}(\overline{B}|\overline{A},C), && (1.9)\\
&= 1-\mathrm{p}(\overline{A}|C)(1-\mathrm{p}(B|\overline{A},C)), && (1.10)\\
&= \mathrm{p}(A|C)+\mathrm{p}(B|C)\mathrm{p}(\overline{A}|B,C), && (1.11)\\
&= \mathrm{p}(A|C)+\mathrm{p}(B|C)(1-\mathrm{p}(A|B,C)), && (1.12)\\
&= \mathrm{p}(A|C)+\mathrm{p}(B|C)-\mathrm{p}(A,B|C). && (1.13)
\end{aligned}$$

If A and B are mutually exclusive then $\mathrm{p}(A,B|C) = 0$ and equation 1.13 takes the familiar form,

$$\mathrm{p}(A+B|C) = \mathrm{p}(A|C)+\mathrm{p}(B|C). \tag{1.14}$$

We can extend this argument by induction to the case of n mutually exclusive propositions, A_i. Furthermore, if the propositions are exhaustive one of them must be true and equation 1.14 becomes

$$\sum_{i=1}^{n} \mathrm{p}(A_i|C) = 1, \tag{1.15}$$

demonstrating that the traditional normalization result is a consequence of our axioms and need not be added as an additional assumption.

Our brief tour of fundamental results would be incomplete without one further result. Sometimes we possess a set of probabilities $\{\mathrm{p}(A,B_i|C) : 1 \le i \le n\}$, but are only interested in A; in such circumstances the B_i are called `nuisance propositions'. Assuming the B_i are exhaustive and applying first the product rule and then equation 1.15 on the preceding page we find:

$$\sum_{i=1}^{n} \mathrm{p}(A, B_i|C) = \sum_{i=1}^{n} \left(\mathrm{p}(B_i|A,C)\mathrm{p}(A|C)\right), \tag{1.16}$$

$$= \mathrm{p}(A|C) \sum_{i=1}^{n} \mathrm{p}(B_i|A,C), \tag{1.17}$$

$$= \mathrm{p}(A|C). \tag{1.18}$$

Discarding the unwanted information about B_i in this way is called *marginalizing over B*. Expanding the joint probability in a different way is also instructive *viz.*,

$$\sum_{i=1}^{n} \mathrm{p}(A, B_i|C) = \sum_{i=1}^{n} \left(\mathrm{p}(A|B_i,C)\mathrm{p}(B_i|C)\right). \tag{1.19}$$

We see that the marginalized probability of A is just the sum of the probability of A assuming B_i to be true, weighted by the probability that B_i really is true.

Up to this point, the A have all been propositions i.e. simple binary choices, each either true or false. An obvious generalization is to allow the A to be integers in some range. If we introduce the exclusive and exhaustive propositions $\{A_r : X = r\}$ it is clear that we can encode $A = N$ as a compound proposition and apply all our previous results. We can play a similar trick on the real numbers with the set of propositions $\{A_r : X \in [\lambda r, \lambda(r+1)[\}$. By making λ small we can achieve arbitrary accuracy in our statements about X. In the limit $\lambda \to 0$, $\mathrm{p}(X)$ becomes a *probability distribution*. Although the mathematics of distributions is well understood, this process is mathematically risky; once we have taken the limit we have lost the details of the limiting process and travelled into a world populated by the dragons of paradox. Often our cavalier approach will save time and avoid technical verbiage - if it fails we must heed Gauss' words, `Infinity is merely a figure of speech, the true meaning being a limit', and defer the continuum until a later point in the calculation.

More generally, we can decompose almost any information into a set of simple propositions and then bring the machinery of probability theory to bear. This process is often easier to describe than perform, and indeed much current research is concerned with encoding ever more subtle and complicated information.

1.3 Inference - learning by numbers

So far we have discussed the meaning of probability and stated the rules which by we are allowed to manipulate probabilities. Our next task is to apply our rules to inference: how should our belief in a proposition or propositions change when we are presented with new data ? We will use f to denote the quantities of interest, D to denote new data drawn from some experiment and H to denote the context of our discussion. More explicitly H is a shorthand way of writing everything that is not included in D or f; sometimes it is best thought of as nebulous background information, whilst at other times H will be a specific model. In this latter case f can be seen as representing parameters of the model.

The standard path through the labyrinth of possible expressions starts at the joint probability of all relevant quantities and then proceeds via exhaustive expansion using the results of the previous section. Treating the H as a fixed conditioning expression, we find

$$\mathrm{p}(f,D|H) = \mathrm{p}(f|D,H)\mathrm{p}(D|H), \tag{1.20}$$
$$= \mathrm{p}(D|f,H)\mathrm{p}(f|H), \tag{1.21}$$

and thus

$$\mathrm{p}(f|D,H) = \frac{\mathrm{p}(D|f,H)\mathrm{p}(f|H)}{\mathrm{p}(D|H)}. \tag{1.22}$$

This is Bayes' theorem. Its four terms are sufficiently important that they are all given names:

$\mathrm{p}(f|H)$: The prior on f assuming H. This term encodes our knowledge of f *before* we have sight of the data, D.

$\mathrm{p}(D|f,H)$: The likelihood of D assuming H. This term encodes our knowledge of the experiment. If we assume a particular f the likelihood tells us how likely we consider each result D.

$\mathrm{p}(D|H)$: The Evidence for D assuming H. This is most useful when H represents a specific model. The Evidence then quantifies how likely we consider the data if we assume the model to be true.

$\mathrm{p}(f|D,H)$: The posterior probability of f assuming both D and H. This is our goal; it encodes our knowledge of f *after* we have incorporated the information gleaned from the experiment.

1.3.1 Model comparison

When we only consider a single model, the Evidence is nothing more than a normalizing constant. However, if we broaden our horizons and consider a number of different models it becomes a rather more interesting and important quantity. If we have several models for some data it is natural to ask which of the models is `better'. Quite what better means in this context would normally be a subjective matter, but probability theory gives us an objective measure of model quality based upon the Evidence for the different models.

Suppose we have a set of models, H_i. To assess their quality, probability theory would have us evaluate the probability of the model (being true) given the data i.e. $\mathrm{p}(H_i|D,\mathcal{H})$ where $\mathcal{H}$ denotes some wider context encompassing all the H_i. Bayes' theorem (equation 1.22) lets us write this as,

$$\mathrm{p}(H_i|D,\mathcal{H}) = \frac{\mathrm{p}(D|H_i,\mathcal{H})\mathrm{p}(H_i|\mathcal{H})}{\mathrm{p}(D|\mathcal{H})}. \tag{1.23}$$

This result, although nothing more than equation 1.22 with different symbols, allows us to objectively compare many different models for our data. This is in contrast to the body of hypothesis tests in conventional statistics, which typically compute a function of the data and compare it with a pre-calculated significance level; the result of this comparison allows us to pronounce the model `valid at the 90% level' or similar. Sometimes we can translate these standard tests into our framework; in such circumstances the traditional test implicitly compares the hypothesis being tested to some kind of `flat' model.

The probabilistic framework does not provide a way to test models in isolation, but merely to compare several alternative models for a given data-set. Logically this is sensible: how can one comment on the *absolute* performance of model, which implicitly compares the model with every other model, without first listing those other models ?

Bayes' theorem tells us how to combine new information with our initial state of knowledge; in simple terms it provides a quantitative rule for consistent learning. We can apply the theorem both to model fitting problems, i.e. inferring f for a given H, and model selection problems, i.e. inferring which H_i is the best. This is in contrast to conventional treatments where we have to learn two different sets of techniques.

It is difficult to proceed further without becoming more specific and so we will consider two simple `toy' problems; the first involving a rather unusual deck of cards, the second a simple measuring problem in the laboratory.

1.4 A guessing game

Many card games involve guessing the identity of a hidden card from some information. Sometimes the information will be obfuscated and the inference presented as magic; at others times the inference will be a trivial part of a game of speed. Our toy problem considers inferring the suit and value of a hidden card given only noisy pictures of the card. Our deck is a non-standard one containing six suits; a,b,c,d,e and f, each suit containing the cards A,2,3,4,5 and 6. The motif on each card is an ellipse whose orientation corresponds to the suit and whose eccentricity to the value. We will display our state of knowledge in figures, e.g. figure 1.1 on the next page. Each of the squares in the grid corresponds to a card, the degree of shading showing how strongly we believe that the square matches the hidden card. The marginalized probabilities for the suit and the value of the card are displayed to the top and left of the grid.

Since knowledge of the suit and value uniquely determine the card we can treat the problem as inferring the suit and value from the data. Following the notation introduced in the previous section, f is a two dimensional vector of integers, f_s and f_v, denoting the suit and value of the hidden card respectively; D is a 256-dimension vector of bits which we regard as an image, and H contains the assumptions about the deck and any image degradation.

1.4.1 The prior

The first step in our calculation is to find the prior for f. In this case we know only that both hidden suit and the hidden value are one of the six in the deck. By symmetry we assign each suit and value a prior probability of one sixth. This leads to a uniform prior for each tile,

$$\begin{aligned} p(f) &= p(f_s|f_v)p(f_v), && (1.24)\\ &= p(f_s)p(f_v), && (1.25)\\ &= \frac{1}{6}\frac{1}{6}, && (1.26)\\ &= \frac{1}{36}. && (1.27) \end{aligned}$$

Figure 1.1 illustrates this by shading each motif a uniform shade of grey.

It is important to note that this assignment is not a statement about the process by which the hidden card was selected, but a statement about our *knowledge* of the process. We would always like to include all of the information available to us but sometimes this will prove impractical or even impossible. Jaynes (1994) uses the example of drawing a ball from a shaken urn in which we know both the starting position of each ball and the precise shake applied. We could *in principle* calculate where each ball lies immediately prior to the draw but the computation would be ferociously difficult. Instead we approximate and assume that each card, or ball, is as likely as any other to attract the hand of fate.

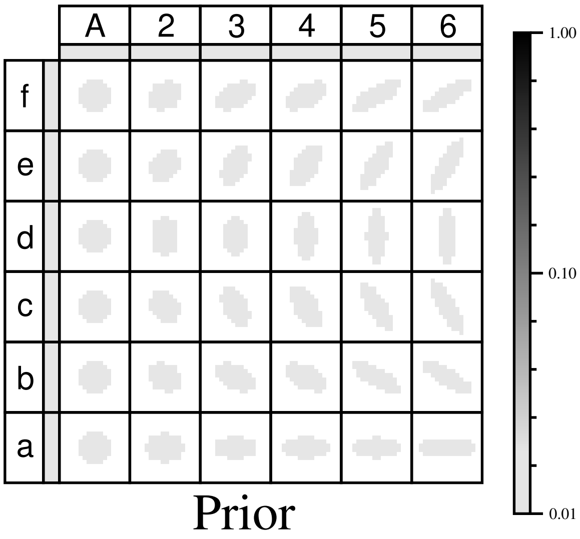


Figure 1.1: The featureless prior probabilities of each card.

1.4.2 A simple likelihood

Our next task is to assign the likelihood, $p(D|f)$. To calculate this we assume that a particular f is true (e.g. c5) and then calculate the pattern that we expect to see. Here this is simple; we just draw an ellipse of the desired orientation and eccentricity on a 16×16 grid. Our manufactured grid resembles the data from our camera — mathematically it lives in the same space as the results from the experiment — and so it is often called the *mock-data*. To calculate the likelihood, we compare each pixel in the mock-data with the corresponding pixel in the data from the experiment. In the noise-free regime the likelihood becomes a simple binary choice,

$$p(D|f) = \begin{cases} 1 & \text{if all the pixels match,} \\ 0 & \text{otherwise.} \end{cases} \tag{1.28}$$

Given both the prior and the likelihood we can evaluate the joint distribution $p(D, f)$ for each possible f and the single data-set D that we happened to observe. Bayesian analysis considers only one data-set and does not indulge in speculations based upon data that were not observed. As a consequence, Bayesian calculations never sum over the space of possible D.

The joint distribution is often the summit of a Bayesian calculation; it encodes all of the information we have about the problem, and subsequent downhill steps usually involve marginalizing over a variable to look at a particular facet of the information. Our first marginalization will be over f to derive the Evidence,

$$p(D) = \sum_f p(D, f). \tag{1.29}$$

We can now substitute equations 1.27, 1.28 and 1.29 into 1.22 on page 5 and calculate the posterior probabilities for f. To do this we consider each f in turn, calculate the joint probability of $p(D, f)$ and then normalize by the Evidence. The resulting posterior distribution is shown in figure 1.2.

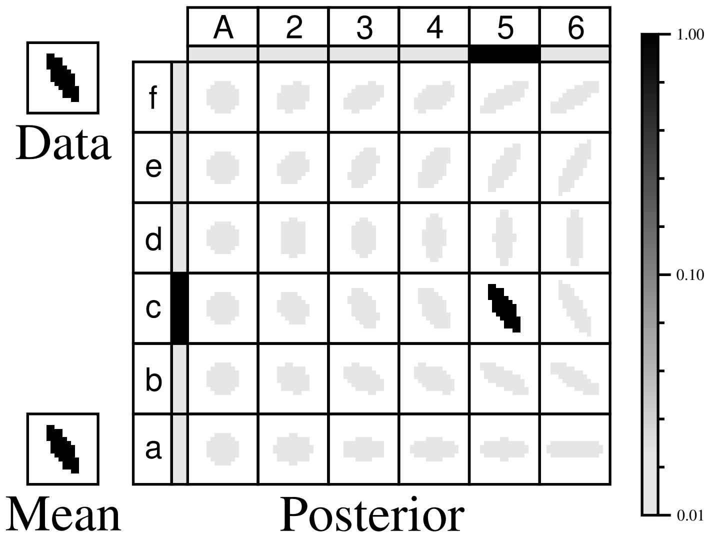


Figure 1.2: The posterior probabilities when faced with a single noise-free measurement.

It would be reasonable to say that we could have drawn figure 1.2 without any reference to probability theory. We could have measured the major and minor axes of the pixelized ellipse and thus deduced its eccentricity and inclination. The data will determine these uniquely and so we can deduce a single card without any ambiguity.

1.4.3 An uninformative experiment

Probability theory doesn't reveal any great truths for the example above but this misses the point. Our goal is a general theory of information handling and inference that is applicable to all problems, trivial or complex. For example, suppose the hidden card had been dA and not c5. Our probabilistic approach is precisely the same, indeed figure 1.3 was produced by simply feeding different data into the program that produced its predecessor.

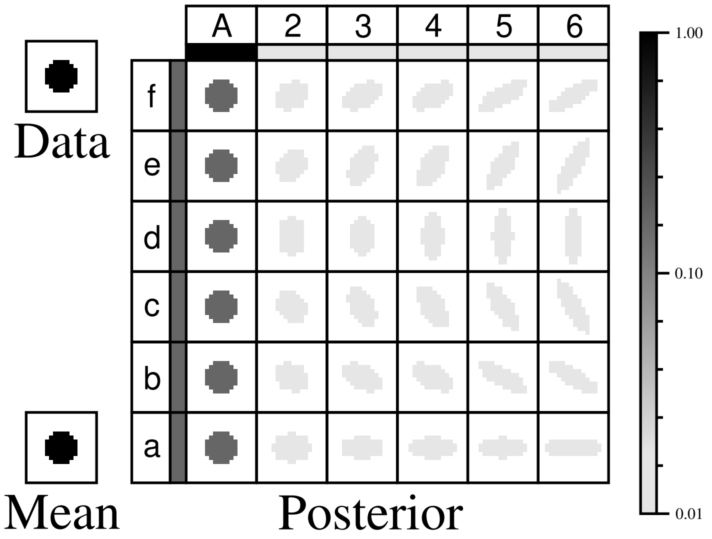


Figure 1.3: The posterior probabilities when faced with ambiguous data.

Since the ellipse has zero eccentricity (i.e. a circle), the suit of the card is not constrained by an image of it. This is reflected in the posterior, in which all the cards with value A are equally probable. Suppose instead that we tried to deduce the card's identity from the measured eccentricity and inclination. The inclination isn't defined for a circle and so our analysis must either explicitly include this as a special case or produce nonsense. More generally how might any deductive process, such as one starting from the data and trying to directly calculate parameters, cope when the data don't constrain a parameter? In this simple problem we can graft hand-written rules onto the basic structure to handle these cases but this is both ugly and impractical for real applications. In contrast, the probabilistic approach does not suffer from the problem; by starting from a set of plausible parameters and using the data to reject a sub-set of them, poor data simply leads to fewer rejections.

1.4.4 The effect of noise

Let us now consider a more difficult set of data, one containing noise on the image. Noise alters our experiment and accordingly we must change the likelihood in our analysis. Even in the presence of noise the mock-data are still useful. To calculate the likelihood we start from a set of parameters, calculate the mock-data and then compare them with those from our image. To do the comparison we need a noise model; our data are digital and so we suppose that the noise randomly flips pixels in the image. To keep the calculation simple we assume that the pixels are independent and the probability of a flip is θ. Therefore we can write the single pixel likelihood as a table:

		Mock-data	
		Black	White
Pixel	Black	θ	$1-\theta$
colour	White	$1-\theta$	θ

The likelihood for the whole picture then becomes

$$\mathrm{p}(D|f) = \prod_i \mathrm{p}(D_i|f), \tag{1.30}$$

$$= \theta^n(1-\theta)^m. \tag{1.31}$$

where n and m are the number of pixels in the data matching and mis-matching their counterparts in the mock-data. In the following demonstrations $\theta = 0.6$.

Figure 1.4 on the facing page shows a single picture and the corresponding posterior probabilities. There is clearly less information in the presence of noise, and a single data-set reduces the number of plausible cards to a dozen, i.e to a third of the deck. A second measurement (see figure 1.5 on the next page) constrains the posterior still further; only three shapes are considered plausible, although since one of these is the circle eight cards are still in contention. Enlarging the data-set still further to six pictures removes any ambiguity (see figure 1.6 on page 12). Interpreting multiple black and white images is difficult for humans; to facilitate a fairer comparison with the computer's inference, the mean data-set is also displayed in the bottom left corner of the figures.

We hope that the data from experiments will reduce the width of our posterior. Sometimes the data will determine the parameters exactly and our posterior will approach a delta function; in others the data will only be informative of some aspects of the parameter space and our posterior will have a more complicated geometry, perhaps a step or an arête. In all cases the method outlined above will work, consistently combining our results into a coherent statement of our knowledge.

1.4.5 A second deck

Suppose that we have two different decks of cards, one as described above, the other rather simpler and containing only two different cards. Suppose further that one card is decorated with a circle and the other with a square. If we do not know from which deck our hidden card was drawn, we must consider this second deck in our analysis. This is simple; introduce H_1 to denote the first deck and H_2 the second. All our previous results hold if we recognize an implicit conditioning on H_1. To consider H_2 we simply repeat the calculation with the new parameters and mock-data. Typical results are shown in figure 1.7 on page 12. From this we still make a strong inference: if the card belongs to the second pack then it is a circle, demonstrating that a sharp peak in the posterior is not necessarily a good indication that our model is true. Rather, we should look at the Evidences for the two models, which have been tabulated below. The final column in the table shows the probability of the first model assuming a flat prior i.e. that both decks were equally likely *a priori*.

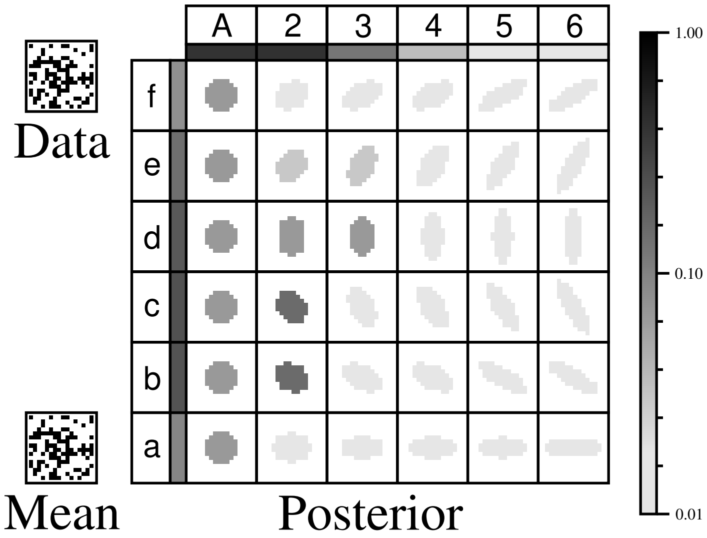


Figure 1.4: The posterior probabilities for a single noisy data-set.

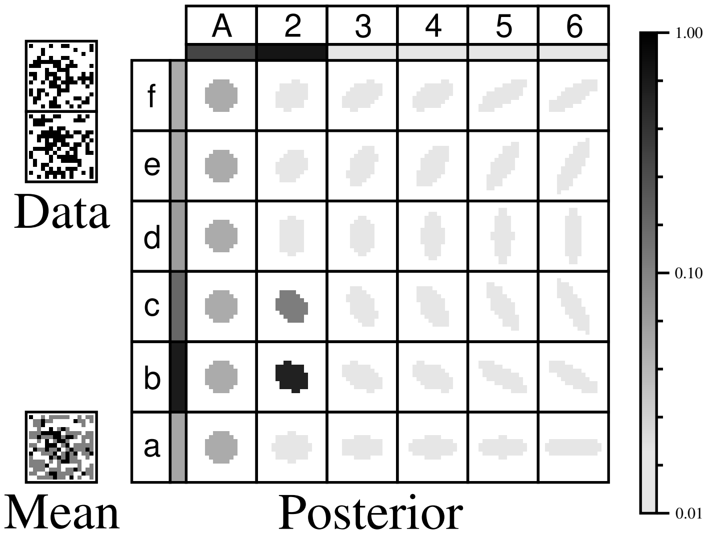


Figure 1.5: The posterior probabilities for a noisy data-set.

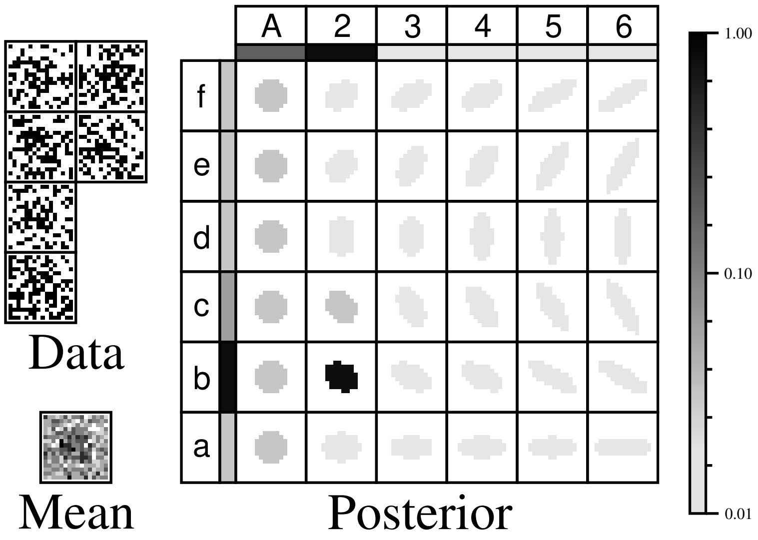


Figure 1.6: The posterior probabilities for a large noisy data-set; the multiple images have almost uniquely identified the cards.

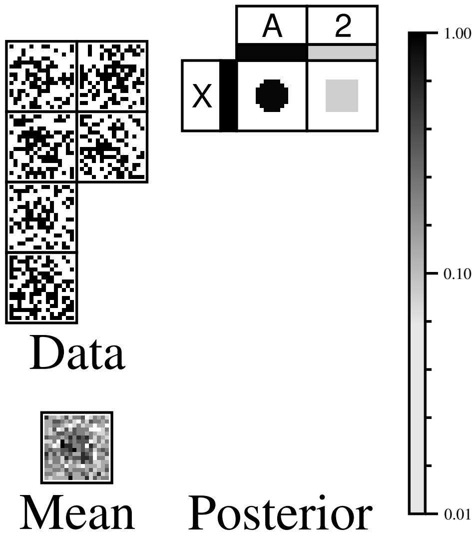


Figure 1.7: The posterior probabilities for an old data-set but a different deck of cards.

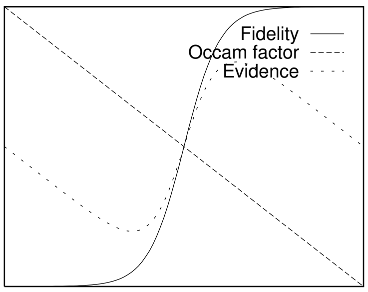
Number of images	Log Evidence H_1	Log Evidence H_2	p(H_1|D)
1	-166.13	-165.75	0.41
2	-335.10	-334.53	0.36
3	-499.99	-501.09	0.75
4	-677.80	-678.99	0.77
5	-847.97	-847.92	0.49
6	-1019.5	-1020.0	0.62
10	-1717.52	-1719.46	0.87

Although the data are rather noisy, two trends are apparent. As we get more data the probability of the first (true) model increases — as expected. Perhaps more surprising is the preference for the second model when we have little data. This too is explicable and is a desirable property; it leads us to prefer *simpler* models unless the data prefer more complicated ones. Our analyses will thus adhere to the `KISS' (Keep It Simple, Stupid!) philosophy. To understand the preference consider

$$\mathrm{p}(D) = \sum_f \mathrm{p}(f)\mathrm{p}(D|f), \tag{1.32}$$

and regard the likelihood as a vetting function which either accepts or rejects bits of the prior. Suppose that just one of the N possible sets of parameters match the model, and the prior on those parameters is flat. Then the Evidence will be $1/N$ and the simpler model (that with fewer parameter settings) will be preferred. If we consider a more general likelihood which is either zero or unity we see the Evidence is just the fraction of parameter space consistent with the data. As our model gets more complicated we have to `tweak' the parameters more carefully to match the data; the fraction of parameters consistent with the data falls and with it the Evidence for the model.

In a real calculation the likelihood will not be a binary function; we expect complicated models to match the data with more fidelity and will accordingly be awarded a higher likelihood. However beyond a certain point the complexity does not improve our fit and the Evidence decreases. The terms in the Evidence favouring simple models are often called Occam factors after William of Occam whose famous statement *`entia non sunt multiplicanda præter necessitatem'* (which roughly translates as `no more things should be presumed to exist than are absolutely necessary') should in fact be attributed to Ponce's much later study of William's work (see Garrett, 1990). The graph below illustrates how these quantities scale with complexity.



Complexity (Arbitrary Units)

In the spectrum of models there is an optimum complexity above which the models are too complicated and below which they fail to predict the data. If we are uncertain of the model, probability theory tells us to marginalize over all models but in practice we often approximate and consider only the most probable.

1.5 A measuring problem

Having considered a discrete problem in the previous section we now consider something rather closer to those found in the laboratory. We follow many elementary physics courses and discuss measuring the length of a bench or table with a tape-measure. For this example our data are a set of lengths $\{D_1, D_2, \ldots, D_N\}$, each measurement subject to uncorrelated Gaussian noise with a standard deviation σ of 3mm.

We begin by finding the mock-data, $\hat{D}_i$. Here they are trivial; there is only one parameter and each datum measures it directly. Thus $\hat{D}_i = f$. From the mock-data it is now a short step to the likelihood, $\mathrm{p}(D|f)$. For so-called `random' uncertainties of a known size, the Gaussian function is appropriate. As we're assuming that the measurements are independent, the total likelihood is just the product of the likelihoods for each individual measurement i.e.

$$\mathrm{p}(D|f) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(D_i - f)^2\right). \tag{1.33}$$

We also need a prior probability that encodes our knowledge of tables in general. We choose another Gaussian primarily because it leads to a simple result, but it is not unreasonable to assume that tables are $\overline{f} \pm \tau$ long. We might choose $\overline{f} = 1.5$m and $\tau = 1$m.

$$\mathrm{p}(f) = \frac{1}{\sqrt{2\pi\tau^2}} \exp\left(-\frac{1}{2\tau^2}(f - \overline{f})^2)\right). \tag{1.34}$$

So, applying equation 1.33, we derive the full joint distribution

$$\mathrm{p}(f, D) = \mathrm{p}(D|f)\mathrm{p}(f), \tag{1.35}$$

$$= \frac{1}{(2\pi)^{(N+1)/2}\tau\sigma^N} \exp -\frac{1}{2}\left(\frac{(f-\overline{f})^2}{\tau^2} + \sum_{i=1}^{N} \frac{(f - D_i)^2}{\sigma^2}\right), \tag{1.36}$$

From the joint distribution, a simple marginalization over f yields the Evidence $\mathrm{p}(D)$,

$$\mathrm{p}(D) = \int \mathrm{p}(D, f)\, \mathrm{d}f, \tag{1.37}$$

$$= (2\pi\sigma^2)^{-\frac{N-1}{2}} \left(2\pi(\sigma^2 + N\tau^2)\right)^{-\frac{1}{2}} \exp -\frac{1}{2}\left(\frac{\sum_i (D_i - \overline{D})^2}{\sigma^2} + \frac{N(\overline{f} - \overline{D})^2}{\sigma^2 + N\tau^2}\right). \tag{1.38}$$

where $\overline{D}$ is the mean of the data, $\frac{1}{N}\sum_i D_i$.

The Evidence contains two terms, which can be seen as misfits within the data, and the misfit between the model and data. The first, $\sum_i (D_i - \overline{D})^2$, is essentially the variance of the data; if we really are directly measuring a single quantity we expect all the measurements to have roughly the same value. The scale of the misfit is, unsurprisingly, just the

Data	Inference
-	1.5 ± 1.0
1.20071	1.20072 ± 0.003
1.20071, 1.19775, 1.20086, 1.19456, 1.19966 1.19911, 1.19971, 1.19762, 1.20357, 1.20342	1.1997 ± 0.00095

Table 1.1: The table shows simulated data and corresponding inferences for the table measuring problem. For this example, the table length was 1.2m.

measurement error, σ. The second term, $(\overline{f} - \overline{D})^2$, measures the difference between our prior information and the data; our model is judged better if $\overline{f}$ and the mean of the data are similar. Here the misfit is scaled by $\sqrt{(\tau^2 + \sigma^2/N)}$. The τ^2 term reflects the uncertainty in our prior, the σ^2/N reflects the expected difference between the mean of the data and the true value.

We now apply Bayes' theorem and calculate the posterior distribution,

$$\mathrm{p}(f|D) = \frac{\mathrm{p}(f,D)}{\mathrm{p}(D)}, \tag{1.39}$$

$$= \frac{\tau\sigma}{\sqrt{2\pi(\sigma^2 + N\tau^2)}} \exp\left(-\frac{\sigma^2 + N\tau^2}{2\tau^2\sigma^2}(f - \hat{f})^2\right), \tag{1.40}$$

where the $\hat{f}$ is given by,

$$\hat{f} = \left(\frac{\overline{f}}{\tau^2} + \frac{N\overline{D}}{\sigma^2}\right) / \left(\frac{1}{\tau^2} + \frac{N}{\sigma^2}\right). \tag{1.41}$$

This is a Gaussian distribution and so we can characterize it by the mean and variance, $\hat{f}$ and $\tau^2\sigma^2/(N\tau^2 + \sigma^2)$ respectively. Without data these reflect our prior conceptions of tables. As we add data, the peak of the distribution moves towards the mean of the data and our uncertainty falls. The prior ceases to affect the data significantly when $N \gg (\sigma/\tau)^2$; if $\sigma \ll \tau$ a single measurement will overwhelm our `weak' prior. In this regime the distribution becomes a Gaussian centred on the mean of the data with a width $1/\sqrt{N}$ times that of an individual measurement. This is a standard result familiar to anyone who has read Squires (1985) or attended a course in practical physics.

Table 1.1 shows simulated data and the corresponding inferences when the true length of the table l was 1.2m:

1.5.1 Binned data

Suppose now that our data come from a digital ruler. We assume that, after the inevitable Gaussian error, the ruler bins the measurement and returns the index of the bin. If our bin size is λ the likelihood becomes

$$\mathrm{p}(m|f) = \int_{\lambda m}^{\lambda(m+1)} \mathrm{p}(m|D,f)\mathrm{p}(D|f)\mathrm{d}D, \tag{1.42}$$
$$= \int_{\lambda m}^{\lambda(m+1)} \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \left(\frac{(D-f)^2}{2\sigma^2}\right) \mathrm{d}D, \tag{1.43}$$
$$= \frac{1}{2}\left[\mathrm{erf}\left(\frac{\lambda(m+1)-f}{\sigma\sqrt{2}}\right) - \mathrm{erf}\left(\frac{\lambda m - f}{\sigma\sqrt{2}}\right)\right], \tag{1.44}$$
$$\equiv \phi_m. \tag{1.45}$$

Each measurement can be thought of as the role of a biased die where the $\mathrm{p}(m|f)$ are the probabilities of rolling an m at each stage. If n_m denotes the number of times m was observed, the likelihood is

$$\mathrm{p}(\{n_m\}|f) = N! \prod_{m=-\infty}^{\infty} \frac{(\phi_m)^{n_m}}{n_m!}. \tag{1.46}$$

Since each term in the product becomes unity if $n_m = 0$, we need only include in the product the bins containing data.

Introducing the same prior as before and writing the integrals in terms of ϕ_m, we derive the full joint distribution,

$$\mathrm{p}(\{n_m\},f) = \frac{N!}{2^N\sqrt{2\pi\tau^2}} \exp - \frac{(\bar{f}-f)^2}{2\tau^2} \prod_{m:n_m\neq 0} \frac{(\phi_m)^{n_m}}{n_m!}. \tag{1.47}$$

We can not in general proceed any further analytically, but a numerical method is relatively simple; we need look no further than Press *et al.* (1992) for suitable algorithms. As ever, the first step is to integrate 1.47 over f to calculate the Evidence. This is a one-dimensional integral and may be efficiently evaluated by the standard Romberg technique. Our posterior is not Gaussian and so it is more appropriate to distil the information into the median and n^{th} percentiles. These are less sensitive to outlying samples, and measure more robustly where the bulk of the probability lies. Simulated data and the corresponding posterior distribution for a number of true values l and noise-levels σ are shown in figure 1.8 and table 1.2. Throughout these demonstrations, the bin-size λ is set to 0.01m. Comparing the graphs shows that a single datum localizes the probability to the relevant bin. As more data arrive the uncertainty decreases but the distribution remains symmetric about the middle of the bin until we begin to fill multiple bins with data. Since this point is determined by the position of the true value within the bin, it isn't easy to decide in advance how many measurements must be taken to achieve a particular accuracy.

In the large number limit we can obtain a useful approximation to the likelihood. In terms of the log of the probability we have

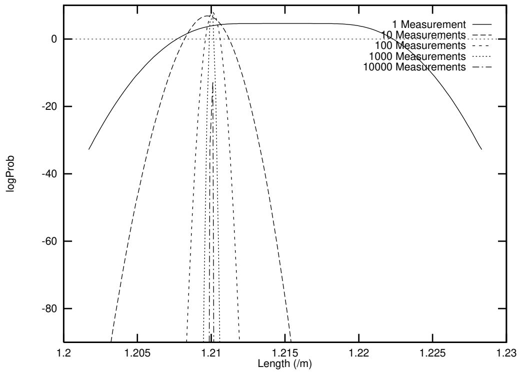
$$\log \mathrm{p}(\{n_m\}|f) = \log N! + \sum_m (n_m \log\phi_m - \log n_m!)\,. \tag{1.48}$$

If the n_m are large we can use Stirling's approximation,

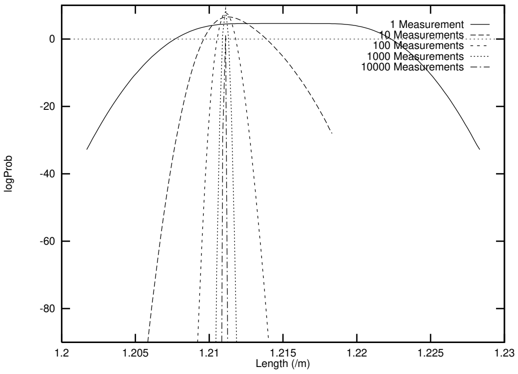
$$\log n! \approx n\log n - n. \tag{1.49}$$

Truth	Number of data	Percentile 5	Percentile 50	Percentile 95	Data (bin, count)
$l = 1.211$m, $\sigma = 0.001$m, $\lambda = 0.010$m	1	1.2087m	1.2150m	1.2213m	(121, 1)
	10	1.2087m	1.2097m	1.2108m	(121, 4) (120, 6)
	100	1.2098m	1.2101m	1.2104m	(121, 54) (120, 46)
	1000	1.2099m	1.2100m	1.2101m	(121, 510) (120, 490)
	10000	1.2100m	1.2100m	1.2101m	(121, 5101) (120, 4899)
$l = 1.212$m, $\sigma = 0.001$m, $\lambda = 0.010$m	1	1.2087m	1.2150m	1.2213m	(121, 1)
	10	1.2101m	1.2114m	1.2131m	(121, 9) (120, 1)
	100	1.2107m	1.2111m	1.2116m	(121, 87) (120, 13)
	1000	1.2110m	1.2111m	1.2112m	(121, 864) (120, 136)
	10000	1.2110m	1.2110m	1.2111m	(121, 8520) (120, 1480)
$l = 1.213$m, $\sigma = 0.001$m, $\lambda = 0.010$m	1	1.2087m	1.2150m	1.2213m	(121, 1)
	10	1.2109m	1.2150m	1.2191m	(121, 10)
	100	1.2121m	1.2150m	1.2179m	(121, 100)
	1000	1.2119m	1.2121m	1.2124m	(121, 983) (120, 17)
	10000	1.2119m	1.2120m	1.2121m	(121, 9766) (120, 234)
$l = 1.214$m, $\sigma = 0.001$m, $\lambda = 0.010$m	1	1.2087m	1.2150m	1.2213m	(121, 1)
	10	1.2109m	1.2150m	1.2191m	(121, 10)
	100	1.2121m	1.2150m	1.2179m	(121, 100)
	1000	1.2129m	1.2150m	1.2171m	(121, 1000)
	10000	1.2128m	1.2130m	1.2133m	(121, 9988) (120, 12)
$l = 1.214$m, $\sigma = 0.001$m, $\lambda = 0.010$m	1	1.2087m	1.2150m	1.2213m	(121, 1)
	10	1.2109m	1.2150m	1.2191m	(121, 10)
	100	1.2121m	1.2150m	1.2179m	(121, 100)
	1000	1.2129m	1.2150m	1.2171m	(121, 1000)
	10000	1.2133m	1.2138m	1.2148m	(121, 9999) (120, 1)
$l = 1.211$m, $\sigma = 0.010$m, $\lambda = 0.010$m	1	1.1882m	1.2150m	1.2418m	(121, 1)
	10	1.2025m	1.2110m	1.2195m	(119, 1) (120, 4) (121, 3) (122, 2)
	100	1.2101m	1.2128m	1.2155m	(119, 10) (120, 29) (121, 37) (122, 21) (123, 3)
	1000	1.2103m	1.2112m	1.2120m	(118, 14) (119, 92) (120, 352) (121, 364) (122, 159) (123, 19)
	10000	1.2110m	1.2113m	1.2116m	(117, 10) (118, 179) (119, 1060) (120, 3266) (121, 3560) (122,1623) (123, 279) (124, 23)

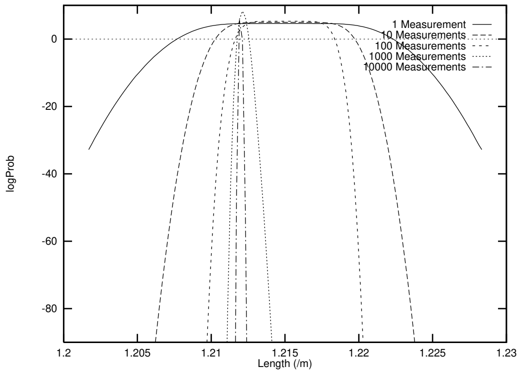
Table 1.2: The simulated data and inferences for different values of the true length l measurement uncertainty σ and bin-size λ.



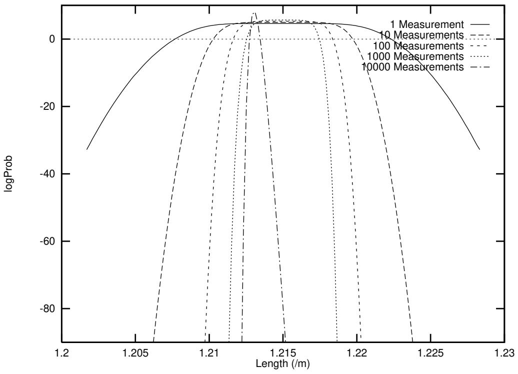
(a) $l = 1.210$m, $\sigma = 0.001$m



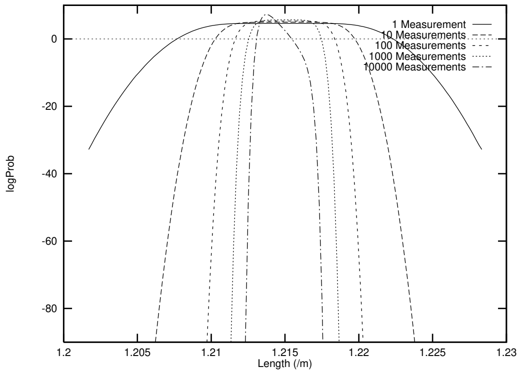
(b) $l = 1.211$m, $\sigma = 0.001$m



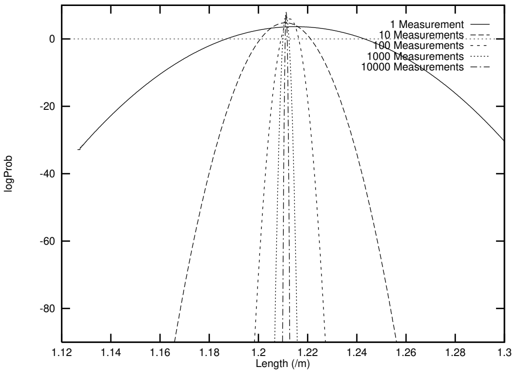
(c) $l = 1.212$m, $\sigma = 0.001$m



(d) $l = 1.213$m, $\sigma = 0.001$m



(e) $l = 1.214$m, $\sigma = 0.001$m



(f) $l = 1.211$m, $\sigma = 0.010$m

Figure 1.8: The posterior distributions for different values of the true length l measurement uncertainty σ and bin-size λ. Each graph shows how the posterior changes as more measurements are taken.

After some re-arrangement this gives

$$\log \mathrm{p}(\{n_m\}|f) \approx -N \sum_m \theta_m \log \frac{\theta_m}{\phi_m}, \tag{1.50}$$

where $\theta_m = n_m/N$, the fraction of ms observed. This expression, the cross-entropy of θ and ϕ, often arises when comparing two sets of probabilities; it can be thought of as the natural measure of distance in probability space. The function has a maximum value of zero, attained when θ and ϕ match perfectly; here this means that the most likely set of data is one in which all the measurements occur with a relative frequency equal to the measurement's probability. If we ignore the prior density for f this is the same as saying that the best f is the one whose expectations most closely match the observed frequencies. Imagine sliding a Gaussian distribution along the x-axis, binning the probability and then comparing these values with the fraction of measurements falling into that bin. The optimum match, using the entropic metric, reveals the best f.

1.5.2 Model comparison

The next part of this demonstration will be to compare two models. A plausible error in the hardware of our digital ruler is a transposition of the two least significant bits. To test this hypothesis we compute the Evidence assuming the wiring is correct, $\mathrm{p}(D|H_c)$, and compare it to the Evidence assuming the transposition, $\mathrm{p}(D|H_t)$. If the noise on the data is small we can distinguish visually — transposing bits will often lead to gaps in the data — but consider the two sets of data shown in figure 1.9 on the following page. Which of these is a better Gaussian ? In contrast to intuition the Evidence is unambiguous; the transposition makes the data nine times more likely and accordingly the probability of the wires being transposed is 89%. If we take more measurements, for example 10000 instead of 100, the improvement becomes visible and the probability rises to $1-10^{-41}$.

1.5.3 An unknown noise level

All our previous analysis has assumed that we know the noise level on our data. Often this is not the case. It is not difficult to extend our analysis to handle unknown noise; we marginalize over the unknown noise level σ,

$$\mathrm{p}(f|D) = \int \mathrm{p}(f,\sigma|D)\,d\sigma, \tag{1.51}$$

$$= \int \frac{\mathrm{p}(f|D,\sigma)\mathrm{p}(\sigma|D)}{\mathrm{p}(f|D)}\,d\sigma. \tag{1.52}$$

We could do this numerically but, since we expect the noise level to be well determined by the data, we sometimes approximate $\mathrm{p}(\sigma|D)$ by a delta-function. Then,

$$\mathrm{p}(f|D) = \mathrm{p}(f|D,\hat{\sigma}), \tag{1.53}$$

where $\hat{\sigma}$ maximizes $\mathrm{p}(\sigma|D)$. If the prior on σ is weak this will be very close to the value which maximizes $\mathrm{p}(D|\sigma)$, i.e. the Evidence for D. For this reason, this approach is often called the `Evidence approximation'.

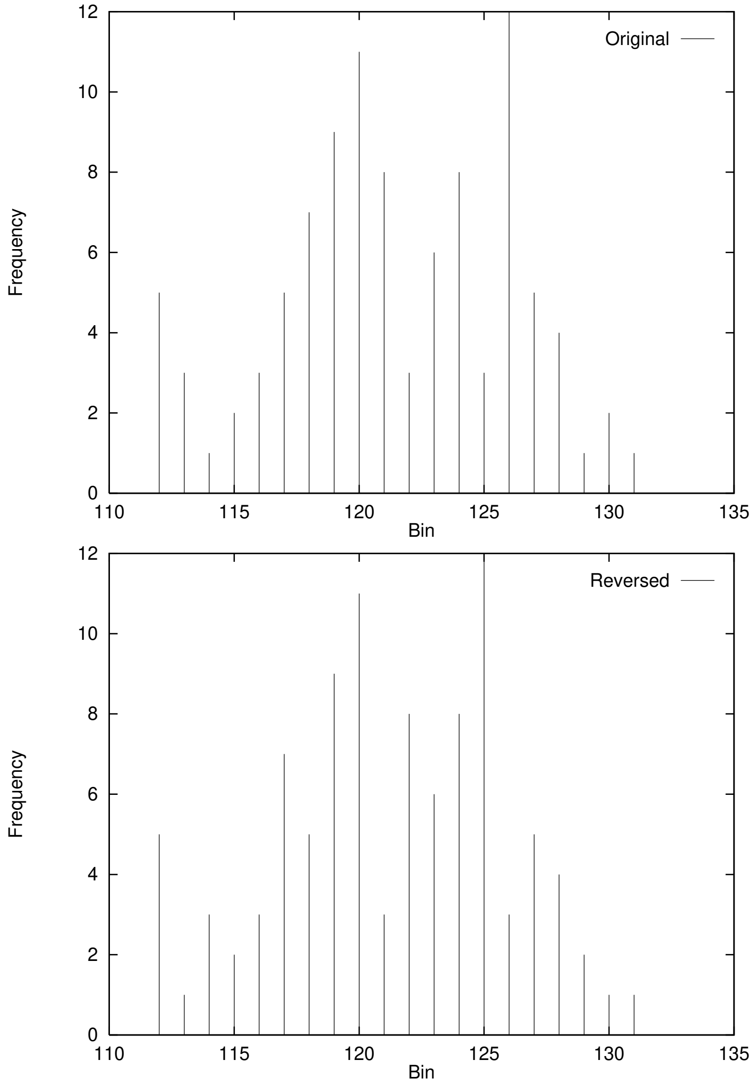


Figure 1.9: *Upper* : A noisy data-set. *Lower* : The same data set assuming that the lowest two-bits have been transposed.

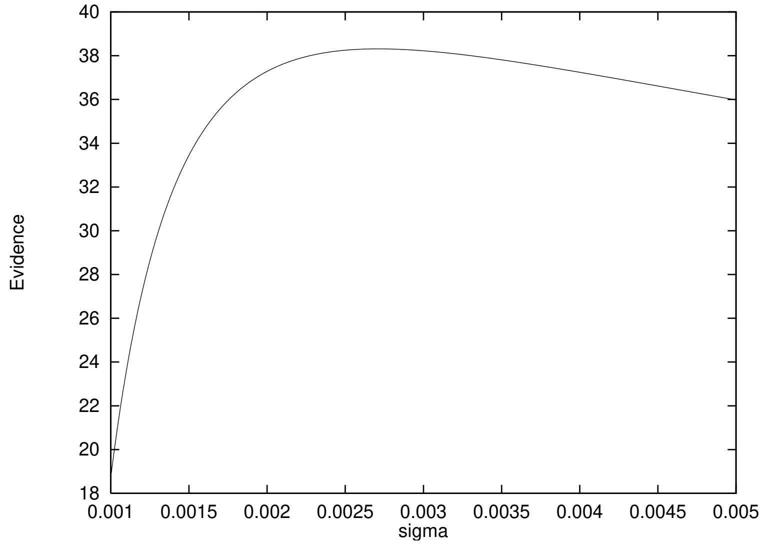


Figure 1.10: A plot of Evidence vs. noise level σ for repeated measurements with Gaussian noise.

Figure 1.10 plots Evidence as a function of σ. We can regard σ as controlling the model's complexity; a small value expects the data to lie in a small area which, albeit in a rather forced sense, can be called simple behaviour. Thus we expect the graph to follow the general form sketched in section 1.4.5 i.e. a sharp rise in Evidence as the model becomes a better fit to the data, followed by a slow decline as it is penalized by increasing complexity.

We can proceed slightly further analytically: taking logs of equation 1.38 on page 14 gives

$$\log \mathrm{p}(D|\sigma) = -\frac{1}{2}\left(N\log(2\pi) + (N-1)\log(\sigma^2) + \frac{\sum_i (D_i - \overline{D})^2}{\sigma^2} \right.$$
$$\left. + \log(\sigma^2 + N\tau^2) + \frac{N(\overline{f} - \overline{D})^2}{\sigma^2 + N\tau^2}\right). \tag{1.54}$$

The last two terms have negligible dependence on σ in the presence of good data ($\sigma^2 \ll N\tau^2$) and in these conditions we find,

$$\hat{\sigma}^2 = \frac{\sum_i (D_i - \overline{D})^2}{N-1}. \tag{1.55}$$

This is another standard result from sampling theory but the derivation above is, I believe, conceptually clearer. The factor of $N-1$ in the denominator refers to the number of *good degrees of freedom* in the problem, a term first used by Gull (1989). In this simple problem we start with N measurements worth of information but lose one because we are ignorant of the true value, l. In more complicated cases where the data are not independent `Good' will have some value between zero and the number of measurements; those measurements hidden in the noise, or which duplicate previous data will not contribute to the total. Some results from conventional statistics can be seen as approximations to the probabilistic formulae in the case where Good is equal to the number of measurements.

1.6 Summary

The most important lesson in this chapter is that probability theory is the best way to analyse data. The approach is rooted in the desire that all our analyses, regardless of the quality of the data, should be consistent. Another advantage is a certain simplicity; once we have assigned probabilities the problem is reduced to mathematics. Ignoring any technical issues such as approximations, all the assumptions in our analysis are explicitly included once we assign the probabilities. Accordingly, if our analysis leads to surprising results we know where to look for the explanation. Similarly, if two analysts reach different conclusions from the same data it will be clear why they disagree; they have different prior information.

In general every analysis will follow the same four-step process:

1. Encode everything that is known as a set of probabilities.
2. Decide upon the quantity of interest; encode this also in terms of a probability.
3. Use the axioms in section 1.2 to express the latter probability in terms of the former ones.
4. Decode the posterior distribution and interpret it.

So the most important things to learn are the axioms. They have the same status as $F = ma$ in Newtonian mechanics and little can be done without them. Beyond that experience counts a great deal but reading the literature can help. The best single reference is the *magnum opus* of Jaynes (1994) which is presently available only in electronic form. Beyond this, the proceedings of the annual workshop `Maximum Entropy and Bayesian Methods' (most recently Skilling and Sibisi, 1994) document the current state of the art.

Chapter 2

Fishing for patterns in data: Probabilistic networks

> Frequently, complexity takes the form of a hierarchy, whereby a complex system is composed of interrelated subsystems that have in turn their own subsystems and so on, until some lowest level of elementary components is reached.
>
> *On Time and Space Decomposition of Complex Structures*, P. Courtois.

2.1 General Purpose Probabilistic Models

In the previous chapter we discussed the basic framework of probabilistic inference. In particular, we saw how probability distributions could act as information containers. Some of the time we are content to use simple functions to encode simple information. This often means that we have cogent information about the problem; perhaps we have a theoretical model for the likelihood or the prior. For example, Newton's law of gravitation contains a single parameter, G, which has a global significance far beyond any specific experiment or set of data. Given observations of the planets we are interested in both their subsequent motion and the implied value of G.

However we are not limited to such models. Indeed, one of the great advantages of the probabilistic framework is the freedom it gives us to consider a set of models and then objectively decide between them; put another way, we let the data be the arbiter of the models' fitness. We are free to consider any model in our analysis, be it a simple one with a strong theoretical justification, a more general model or an idle daydream. Finding good models in the first of these classes is a task best solved by experts in the field; we note that as physicists we are equipped to build models which capture salient features whilst remaining tractable. That just leaves general purpose models which we will now consider.

If we were trying to predict the motion of Jupiter given some previous observations there would be two quantities of immediate interest; the position of the the planet and the time of observation or prediction. Were we trying to solve this problem in the traditional Bayesian manner we would start with Newton's Laws and write down the likelihood function for the position at a given time. To do that we would need to introduce some new parameters: the constant G, the solar mass, the angular momentum of Jupiter and so forth. Of course there is a range of possible models, the simplest assuming a circular orbit, the most complicated modelling the entire solar-system; each model will introduce a different set of parameters and each must be assigned a prior.

If our task is to infer the planet's position at some later time from a set of previous observations, the Bayesian methodology has us write down the probability distribution for the position given these data, as is given by a marginalization over parameter space. If X are the parameters, $\mathbf{r}_i$ and t_i the previous positions and observation times and $\mathbf{r}$ the prediction for time t,

$$\mathrm{p}(\mathbf{r}|t,\mathbf{r}_i,t_i) = \int dX \mathrm{p}(\mathbf{r}|t,X)\mathrm{p}(X|\mathbf{r}_i,t_i). \tag{2.1}$$

There is nothing wrong with this approach; indeed if the process can be modelled there is none better — but suppose we were ignorant of Newton's laws. Quite reasonably we might believe that there is some function which would tell us where Jupiter could be found at a given time, though it would be an impressive feat to simply write it down. However if we measured Jupiter's position on successive nights we could build a `free-form' model for its subsequent motion. We believe that such an approach should also be cast in a probabilistic framework.

Just as we have divided our observations into the observation time and the position of Jupiter, we will generally divide our data into `inputs' and `outputs' and seek to infer a mapping between them. This inference process, often called training, is a prerequisite for the prediction phase of our analyis in which we will predict new outputs for a set of given inputs.

This form of analysis is quite a general one; we could use the same model to predict the strength of a weld given some information about the welding process, or even to predict the future behaviour of a financial instrument given a trading history. Such utility is an advantage but it makes the analysis opaque; although we may be able to infer Jupiter's motion we can not honestly say that we *understand* why it moves thus, nor would it become any easier to solve the motion of Saturn.

Nevertheless this non-specific approach to modelling has its advantages; in particular we can apply it easily to quite complicated scenarios. The question is then which model we should use for an analysis. There is no right answer, but we should choose from models which have the following properties:

- **Computability.** Perhaps the most important of our *desiderata* is a pragmatic one. Our model is useless if its consequences can not be computed. The steady increase in computer speed obviously facilitates more ambitious models, but we are still precluded from directly using many naïve algorithms which require $O(N!)$ operations. Algorithms which are expected to take exponential time often arise from considering all possible configurations of some set; such a calculation may be necessary to calculate the Evidence exactly for a discrete model.

- **Complex behaviour.** A good general purpose model should provide a moderately general mapping between its parameters (inputs) and outputs. Let us explain this statement by way of an (admittedly artificial) example. We could conceive of a model with two inputs, A and B, and a single output, X. If $X \approx A+B$ when B is roughly 1, and $X \approx B\sin A$ when B is roughly 2, the mapping would be complex in the sense that local plots of $X(A,B)$ vary quite significantly as the parameters change on a larger scale. If the model is general enough, we hope to approximate any local mapping by choosing the relevant microcosm.

- **Smoothness.** One of our long-term aims is to store information about some process in our model. If our model is not smooth, it will be no more use than the list of the data that we already have; to be useful we would like our model to interpolate (in some loose sense) and give sensible predictions for all data. There is a strong parallel here to a discrete topology; although mathematicians find such things interesting, pragmatists prefer a space with more structure.

2.2 Traditional models

Before we consider more complex models we survey the more important traditional ones. Whilst these are the building blocks of many models I present them not as a main course, but rather a contrasting apéritif.

2.2.1 Entropic models

If our data are probabilities then a sensible choice for the prior is one which favours models close to some underlying, featureless, map. As we noted in the previous chapter, the cross entropy between two distributions is the natural way to measure their similarity. So when inferring a probability distribution defined on M cells it is reasonable to use an entropic prior,

$$\log \mathrm{p}(f|m) = \alpha \left(\sum_i^M f_i \log \frac{f_i}{m_i} - f_i - m_i \right) + const. \tag{2.2}$$

Were this prior only useful for probabilities it would be interesting enough, but it has also been used for many *positive-additive quantities*, for example the flux from a radio-galaxy or the oscillator strengths underlying a chemical spectra. MaxEnt priors, as they are called, have been used to tackle a wide variety of problems. Robinson (1992) gives both an exposition of the theory and pointers to further work.

2.2.2 The Gaussian model

Undoubtedly the most famous and ubiquitous probability distributions is the Gaussian,

$$\mathrm{p}(f|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(f-\mu)^2}{2\sigma^2} \right), \tag{2.3}$$

where μ and σ are the mean and standard deviation respectively.

This distribution is useful to encode the notion that a quantity lies roughly within $\pm\sigma$ of μ. We also find that in many cases this distribution is a good approximation to the exact solution of our equations, particularly in the `large N' limit. Furthermore, the approximation is often a useful one because Gaussians are easy to manipulate algebraically. In particular,

- The product of two Gaussians is a Gaussian.
- The convolution of two Gaussians is a Gaussian.
- The Fourier transform of a Gaussian is a Gaussian.

There are many derivations of this distribution — given its ubiquity it would be surprising if there were not — but one of the nicest stems from an entropic argument. Suppose we have a set of data with known mean, μ, and standard deviation, σ. Let us now seek the distribution closest — in the entropic sense — to a constant, yet consistent with this mean and variance. Formally we seek to maximize

$$Q = \int dx \exp - \left(f(x) \log f(x) - f(x) \right), \tag{2.4}$$

subject to the constraints

$$\int f(x) dx = 1, \tag{2.5}$$

$$\int x f(x) dx = \mu, \tag{2.6}$$

$$\int (x-\mu)^2 f(x) dx = \sigma^2. \tag{2.7}$$

Introducing Lagrange multipliers for the constraints, we find that the optimum distribution satisfies

$$\log f = \lambda + \kappa x + \nu (x-\mu)^2. \tag{2.8}$$

Using the constraint equations to eliminate λ, κ and ν in favour of μ and σ gives the familiar Gaussian form.

This notion of assigning priors with the maximum possible entropy, yet consistent with a set of constraints is a general one. Shannon (1948) justifies it by noting that the entropy measures the information content of the distribution and so the MaxEnt assignment is the least committal one. This insight also explains why the Gaussian distribution is ubiquitous; any process which discards information yet preserves a mean and variance must tend to the Gaussian. The principle is well known in statistics as the Central Limit Theorem.

We can write a Gaussian distribution in larger spaces as well. For example, in N dimensions,

$$\mathrm{p}(\mathbf{f}) = (2\pi)^{-\frac{N}{2}} \det(A)^{\frac{1}{2}} \exp - \frac{1}{2} (\mathbf{f} - \mathbf{f}_0)^T A (\mathbf{f} - \mathbf{f}_0). \tag{2.9}$$

Here $\mathbf{f}_0$ is a vector-valued mean position and A describes the curvature of the probability about that point. If A is diagonal, we can factor the distribution into a product of one-dimensional Gaussians aligned with the co-ordinate axes. The variances of the distributions are the reciprocals of the diagonal elements of A. In the more general case, the factorization must take place along the eigenvectors of A and A's eigenvalues will relate to the spread in these directions. Alternatively A can be regarded as the inverse of the covariance matrix so that

$$\left\langle \Delta\mathbf{f} \Delta\mathbf{f}^T \right\rangle = A^{-1}, \tag{2.10}$$

where $\Delta\mathbf{f} = \mathbf{f} - \mathbf{f}_O$.

2.2.3 Symmetry

Some important priors, particularly for discrete problems, rely on a symmetry in our knowledge of the problem. For example, in section 1.4 symmetry allows us to assign the prior on the card probabilities. More generally we can deploy the apparatus of group theory to relate prior densities in different regions of space. We will not develop these ideas here.

2.3 A hierarchical model - the probabilistic network

As discussed above, conventional models sometimes lack the structure to capture the complicated behaviour of real-world systems. However if we were forced to construct a model for some complicated process we would probably start with a collection of simple models and stick them together. It is possible that each of these models would be quite complicated in its own right, carefully crafted to model some feature of the process. Alternatively each model might be a simple unit, or a collection of simple units; the multiplicity creating a complex, structured model. Such models have been used by numerous people, often in the guise of neural networks though their use as probabilistic models dates from MacKay (1991). As their name suggests these models were originally motivated by the hierarchical structure of animal brains but we will eschew this connection and treat them as abstract mathematical devices. There is a vast literature devoted to synthetic neural networks, including McCulloch and Pitts (1943), Hebb (1949), Hopfield (1982) and Rumelhart *et al.* (1986).

2.3.1 The basic structure of a network

We call the simplest element of a network a node. Each node has n inputs x_i and a single output z. The mapping between inputs and output is parameterized by a set of $n+1$ weights w_i. It is useful to consider the linear sum as a unit and then a subsequent non-linear mapping. Mathematically, we have

$$\left.\begin{aligned} y &= w_0 + \sum_{i=1}^{n} w_i x_i, \\ z &= f(y). \end{aligned}\right\} \tag{2.11}$$

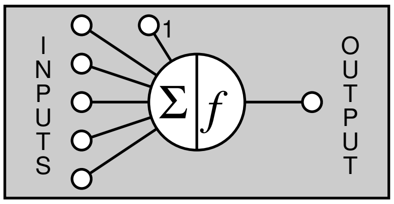


Figure 2.1: A graphical representation of a single node. The inputs, on the left, are weighted and then summed before being passed through the non-linear function, f. The result is passed to the output node on the right.

A typical network consists of many nodes. Although in principle every node might have a different response function or a different set of connections, practical networks will often contain many identical nodes. It makes sense to recognize this symmetry; we call the collection of identical units a group. The groups will frequently be arranged so that all the outputs in one group connect to all the inputs in the next. An example of such a network is shown in figure 2.2.

Having described the arrangement of a network it remains to describe how it makes predictions. We start by assigning values to the inputs on the left of the diagram. We then proceed, a stage at a time, to propagate these values across the network until we reach the outputs. This propagation is simply repeated application of equations 2.11, but it is useful to look at this pictorially; a useful, if poetic, picture is to view the stream of information as flowing left to right across the network. We will also change our notation slightly to reflect the increased scope. The equations which propagate data through the network become:

$$\left.\begin{aligned} y_i^k &= \sum_j w_{ij}^k x_j^k, \\ z_i^k &= f^k(y_i^k), \\ x_i^{k+1} &= z_i^k. \end{aligned}\right\} \tag{2.12}$$

Here the superscripts denote the group to which a node belongs whilst the subscripts index nodes within a group. I have also suppressed the `bias' term on the understanding that it can always be simulated by adding a `virtual' input, x_0, always set to unity. Although this notation looks cumbersome and bristles with indices, other more compact notations obscure the full structure of the equations. Sometimes, when the full structure is not relevant, it will be more useful to write the weights as a single vector, labelled by a single index rather than the three above. Sometimes I will even refer to the vector directly, without any index.

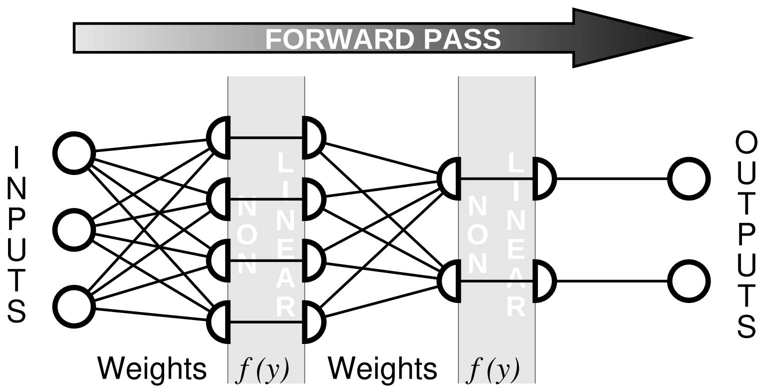


Figure 2.2: A typical small network with three inputs, four `hidden' units and two outputs. In general, the two non-linear blocks (shaded grey) will contain different functions. I will refer to this as a two-layer network because it has two layers of weights.

2.4 The Bayesian viewpoint

In the previous section we have described the basic structure of our model and showed how to calculate predictions given a set of inputs. Our next task is to cast this in a probabilistic framework.

2.4.1 The prior

We consider the prior first. In the network, the unknown parameters that we seek to quantify are the weights, w. We know little about the values the weights should have and so, mainly in the interests of practicality, we assign a Gaussian prior to them. Of course, this means we have also to assign a mean and co-variance; *a priori* we have no reason to suppose the weights are correlated and so we make the covariance matrix diagonal. Similarly there is no reason to prefer a particular sign of weight and so we choose the mean of the distribution to be zero. Thus the prior is:

$$-\log \mathrm{p}(w|\sigma) = \frac{1}{2}\sum_i \frac{w_i^2}{\sigma_i^2} + \frac{N}{2}\log(2\pi) + \sum_i \log\sigma_i. \tag{2.13}$$

The choice of σ_i is more problematical. Our first observation is that there should be rather fewer different σs than ws — otherwise we have achieved nothing. Instead we introduce a set of weight classes, c, each with its own σ_c and assign each weight to a different class. The prior becomes:

$$-\log \mathrm{p}(w|\sigma) = \frac{1}{2}\sum_c \frac{1}{\sigma_c^2}\sum_{i\in C} w_i^2 + \frac{N}{2}\log(2\pi) + \sum_c N_c \log\sigma_c. \tag{2.14}$$

In this summation C refers to the set of N_c weights in class c. To be useful we expect there to be many weights in each class so that $N_c \gg 1$. Sometimes it will be useful to divide the log prior into a weight dependent part Q_W and a weight independent part Ψ_W.

$$Q_W = \frac{1}{2}\sum_c \frac{1}{\sigma_c^2}\sum_{i\in C} w_i^2, \tag{2.15}$$

$$\Psi_W = \frac{N}{2}\log(2\pi) + \sum_c N_c \log\sigma_c. \tag{2.16}$$

The reader should be warned in advance of what is, technically, an abuse of the notation σ_i. Sometimes the suffix will refer to a class of weights (when it will usually be written σ_c) whilst at other times it will refer to the σ appropriate to the i^{th} weight. The context will always be sufficient to remove any ambiguities.

The assignment of weights to classes remains an open question. Since each grouping constitutes a new model, the Bayesian formalism encourages us to try many different possibilities and evaluate the Evidence for each of them. However, we would be foolish to ignore either dimensional information or symmetries between different weights when we choose a grouping. For example, if we assume that the inputs to the network have some dimension associated with them, yet the biases are fed from a dimensionless source, it is clear that the first set of weights should belong to at least two different classes. In

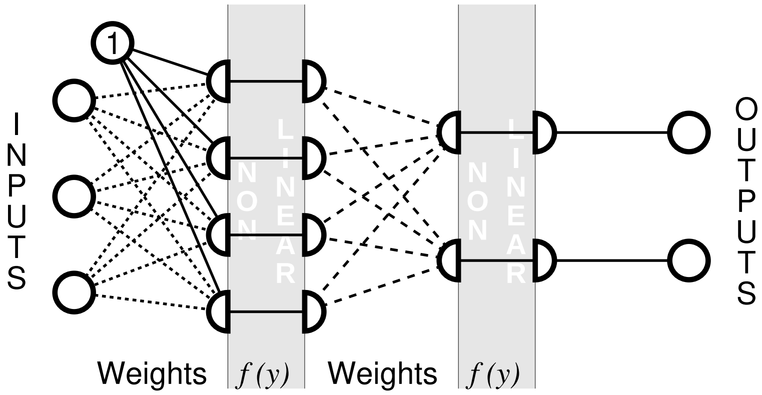


Figure 2.3: A simple weight prior for a network, chosen by dimensional considerations. There are three different classes of weight indicated by different dash patterns; solid lines, dashed lines and dotted lines..

the second group of weights we make no such distinction between inter-node weights and biases and accordingly can place them in the same class. Figure 2.3 illustrates this pictorially.

Of course there is no guarantee that our inputs have the same dimensions. Consider a network with two inputs: a length and a temperature. In this case it would be appropriate to place the weights from different inputs in different classes. This, the so-called `Automatic Relevance Determination' (ARD) model, is due to MacKay and Neal (1994). The name comes not from dimensional consistency; instead once the weights are divided into different classes, probability theory allows us to analyse each class individually. Accordingly we can investigate the relevance of each input to the inference.

2.4.2 The likelihood

Having assigned the prior we turn now to the likelihood. I introduced this in the previous chapter as a measure of the model's fidelity to the data; strictly it is the probability of the data given a set of model parameters. This prescription makes it clear how to calculate the likelihood for our network for a given set of parameters w. We pick a set of parameters (weights), load the inputs with data and propagate them across the network; we can now read the mock-data from the output nodes and compare these with the real data using an appropriate error function. We defer the discussion of different error measures until a later section of this chapter. For now we note that they can all be written in the form:

$$-\log \mathrm{p}(D|w) = Q_D\left(D, \hat{D}(w)\right) + \Psi_D. \tag{2.17}$$

Here Ψ_D is independent of the weights but may include other unknowns such as a measure of the training data's accuracy.

Typically Q_D will be a sum of contributions from several nodes in the network, q_i^k:

$$Q_D(w) = \sum_{i,k} q_i^k(z_i^k). \tag{2.18}$$

We must also sum over each example in the training set. However our symbols are already overloaded with indices and so we leave this summation implicit in all discussions.

2.4.3 The joint distribution and an objective function

Equations 2.14 and 2.17 are sufficient to evaluate the joint distribution:

$$-\log \mathrm{p}(D,w) = \frac{1}{2}\sum_c \frac{1}{\sigma_c^2}\sum_{i\in C} w_i^2 + Q_D\left(D,\hat{D}(w)\right) + \frac{N}{2}\log(2\pi) + \sum_c N_c \log\sigma_c + \Psi_D. \tag{2.19}$$

Given a network, some data, and a set of weights, we can merrily compute the value of the joint distribution $\mathrm{p}(w,D)$ for a particular value of the weights. However, this is almost completely useless! If our space is of even moderate dimension merely calculating pointwise values tells us very little; there are too many `points' to explore the space by mindless tabulation. The non-linear nature of the likelihood also makes it impossible to analytically manipulate the distribution in a useful way. To proceed, we need to decide exactly what we want to know about the joint and hence, posterior distributions—after all it is difficult to provide an answer until we have the question. Certainly we are interested in the weights, but only in the sense that they allow us to calculate outputs for as yet unseen inputs. Denoting the prediction for x' by z', our real goal is to find $\mathrm{p}(z'|x',D)$ which is given by

$$\mathrm{p}(z'|x',D) = \int dw\, \mathrm{p}(z'|w,x')\mathrm{p}(w|D). \tag{2.20}$$

Since our goal is an integrable function, a plausible approximate form for the joint distribution is a multivariate Gaussian centred on the modal value of the real joint distribution. Writing $\log \mathrm{p}(z)$ in full gets rather cumbersome and we introduce a shorthand objective function. This is the weight-dependent part of the log-likelihood and as such is a monotonically decreasing function of the joint probability; the mode of the joint distribution corresponds to the minimum value of the objective function. Explicitly, the objective function is

$$Q(w) = Q_W + Q_D = \frac{1}{2}\sum_c \frac{1}{\sigma_c^2}\sum_{i\in C} w_i^2 + Q_D\left(D,\hat{D}(w)\right). \tag{2.21}$$

If we can minimize Q and find the curvature about that point, our approximation for the joint distribution becomes

$$-\log \mathrm{p}(D,w) \approx Q(\hat{w}) + \frac{1}{2}(w-\hat{w})^T A(w-\hat{w}) + \frac{N}{2}\log(2\pi) + \sum_c N_c \log\sigma_c + \Psi_D. \tag{2.22}$$

where

$$A = \nabla\nabla Q|_{\hat{w}} \tag{2.23}$$

and the gradient is taken with respect to the weight vector.

Approximating Q with a Gaussian might seem a silly thing to do. After all, we were drawn to neural networks as an alternative to simple Gaussian models; to return to them sounds a little foolish. However this misses the point. This Gaussian distribution is on the *weights* and not the *outputs* of the network, and the two are related by a complicated, non-linear relationship. A more serious objection to the approximation is that the objective function will in general be rather complicated and possess many local maxima; it is natural to ask which of these we should choose for $\hat{w}$. Our reply is essentially pragmatic — if many maxima exist, let us treat each one as a separate model and apply the Bayesian machinery in the usual fashion. Once we possess the posterior distribution and evidence for each solution, we can marginalize over solutions thus:

$$\mathrm{p}(z'|x',D) \propto \sum_i \mathrm{p}(D|\hat{w}_i)\mathrm{p}(z'|\hat{w}_i,x',D). \tag{2.24}$$

Of course it is impractical to find all the maxima exhaustively and so we must evaluate the sum in equation 2.24 by a Monte Carlo method. Often this will be nothing more than finding a number of $\hat{w}_i$ and averaging over their predictions.

2.5 The Evidence

We get the Evidence by marginalizing the joint distribution over the weights. Integrating equation 2.22 over weight space gives

$$-\log \mathrm{p}(D) \approx Q(\hat{w}) - \frac{1}{2}\log\det A + \frac{1}{2}\sum_c N_c \log\sigma_c + \Psi_D + \frac{1}{2}\log N_P. \tag{2.25}$$

where N_P is the multiplicity of the solution; that is, how many equivalent networks we could make by just permuting the internal nodes and weights. Care must be taken with this calculation if the network becomes degenerate, for example if two nodes have identical weights.

2.6 Calculating the gradient

Most optimization algorithms require the gradient of the function with respect to the parameter vector. Accordingly we must evaluate the gradient of the objective function with respect to the weight vector. The back-propagation algorithm (Rumelhart *et al.* (1986)) is an elegant solution to this problem.

To evaluate the gradient of the objective function with respect to the weight vector we consider Q_W and Q_D separately. Q_W is much the easier term; we can apply the operator directly to 2.15 to give

$$\nabla Q_W = \sum_c \frac{1}{\sigma_c^2} \sum_{i \in C} w_i. \tag{2.26}$$

If we try the same approach with ∇Q_D we get no further than

$$\nabla Q_D = \sum_{i,k} \dot{q}_i^k(z_i^k)\nabla z_i^k. \tag{2.27}$$

To proceed we must relate the gradient ∇z_i^k to other nodes in the network. In order to do this consider a simpler problem: the derivative of one node's activity with respect to another node in the network. We will give an intuitive explanation first and then some rigorous mathematics.

Figure 2.4 shows a simple network. Suppose we change the value of the input node A, x_1^1, by a small amount, and ask how much this changes the value of the output of node B, z_1^2. Information from node A propagates along the weights into the hidden layer and thence into node B. To calculate the derivative $\partial x_1^1/\partial z_1^2$, sum the contributions from each path between them. Each term in the sum will be the product of contributions from each part of the path. Passing along a weight introduces a factor of that weight, while passing through a non-linear function forces us to multiply by the function's derivative. This sounds remarkably like the passage of data through the network to generate outputs, and indeed we shall find that the two are algebraically almost identical. We need one further piece of insight: when we discuss a partial derivative with respect to x_i^j, we implicitly assume that the other $\{x_k^j : k \neq i\}$ are kept constant.

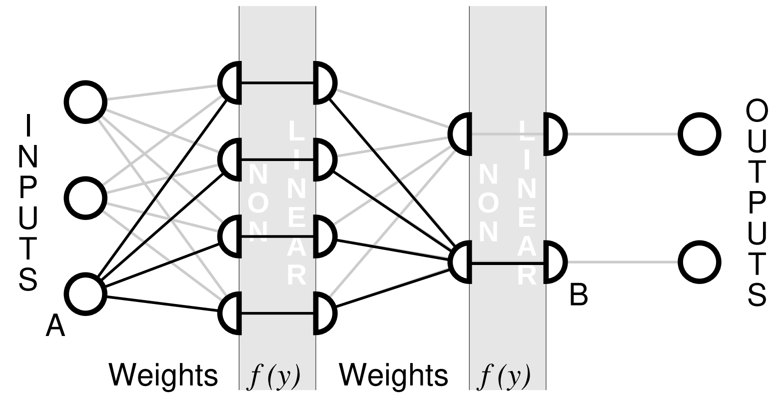


Figure 2.4: A typical small network highlighting the connections relating two nodes. The black lines show the routes by which changes in node A affect node B.

Mathematically we need only differentiate equations 2.12 to derive the factors associated with a single step:

$$\frac{\partial x_j^{k+1}}{\partial z_i^k} = \delta_{ij}, \tag{2.28}$$

$$\frac{\partial z_i^k}{\partial y_j^k} = \dot{f}^k(y_i^k)\delta_{ij}, \tag{2.29}$$

$$\frac{\partial y_i^k}{\partial x_j^k} = w_{ij}^k. \tag{2.30}$$

We can now relate derivatives defined on a layer of the network to those defined on the next layer:

$$\frac{\partial Q_D}{\partial y_i^k} = \sum_j \frac{\partial z_j^k}{\partial y_i^k} \frac{\partial Q_D}{\partial z_j^k} \tag{2.31}$$

$$= \dot{f}^k(y_i^k) \frac{\partial Q_D}{\partial z_i^k} \tag{2.32}$$

$$\frac{\partial Q_D}{\partial x_i^k} = \sum_j \frac{\partial y_j^k}{\partial x_i^k} \frac{\partial Q_D}{\partial y_j^k} \tag{2.33}$$

$$= \sum_j w_{ji}^k \frac{\partial Q_D}{\partial y_j^k} \tag{2.34}$$

$$\frac{\partial Q_D}{\partial z_i^{k-1}} = \sum_j \frac{\partial x_j^k}{\partial z_i^{k-1}} \frac{\partial Q_D}{\partial x_j^k} + \dot{q}^{k-1} \tag{2.35}$$

$$= \frac{\partial Q_D}{\partial x_i^k} + \dot{q}^{k-1} \tag{2.36}$$

In this last equation we have included the $\dot{q}$ term from equation 2.27, reflecting the direct contribution of z to Q_D.

It is messy to have so many partial derivatives in our equations. Accordingly we introduce the following notation:

$$e_i^k = \frac{\partial Q_D}{\partial z_i^k}, \tag{2.37}$$

$$\eta_i^k = \frac{\partial Q_D}{\partial y_i^k}, \tag{2.38}$$

$$\xi_i^k = \frac{\partial Q_D}{\partial x_i^k}, \tag{2.39}$$

which enables us to write the error propagation equations more succinctly as

$$\left.\begin{aligned} e_i^k &= \xi_i^{k+1} + \dot{q}^k, \\ \eta_i^k &= \dot{f} e_i^k, \\ \xi_i^k &= \sum_j w_{ji}^k \eta_j^k. \end{aligned}\right\} \tag{2.40}$$

These equations tell us how sensitive is the objective function to each node in the net. This is just one step away from the gradient with respect to the weights. Consider changing a weight, w_{ji}^k, by a small amount Δw; this will change the y_j^k value of the node to the right of the weight by an amount $\Delta w x_i^k$. Thus,

$$\frac{\partial Q_D}{\partial w_{ij}^k} = x_j^k \eta_i^k. \tag{2.41}$$

The two sets of equations 2.12 and 2.40 constitute the so called `back-propagation' algorithm derived by Rumelhart *et al.* (1986). Although the mathematics above is moderately messy the algorithm has a nice pictorial interpretation. To calculate the derivative with respect to a weight, trace all the paths from the inputs to the node to the left of the weight, propagating values according to equations 2.12. Then trace all the paths going from right to left from the network outputs (strictly all nodes contributing to Q) to the node to the right of the weight. This propagation proceeds right to left and so must use equations 2.40. This is illustrated in figure 2.5. We can also extend the poetic metaphor: by calculating the gradient of the objective function the flow of data becomes tidal, flowing towards the outputs before ebbing back towards the inputs.

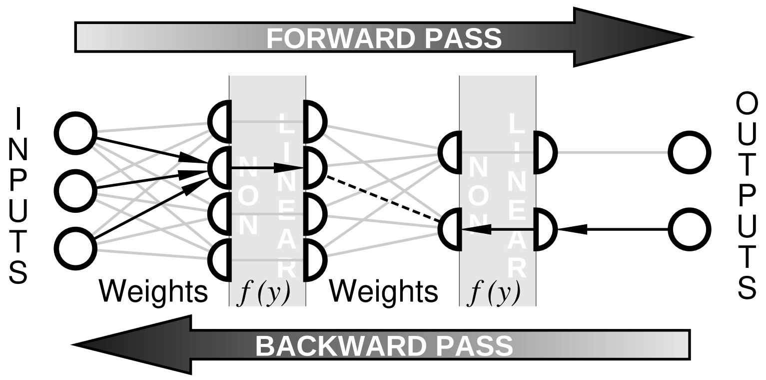


Figure 2.5: A typical small network highlighting the influence of a single weight, shown as a dashed line, on the outputs. To calculate the differential response of the output to the weight, we multiply the weight's activity (calculated by following the arrows pointing to the right) by the sensitivity of the output to the weight (calculated by following the arrows pointing to the left).

2.7 Calculating the curvature

Having calculated the gradient of our objective function we must now consider its curvature. For large problems, the storage requirements preclude the direct representation and manipulation of the curvature matrix; we will consider instead schemes which implicitly apply the Hessian to an arbitrary vector.

Sometimes we will use an approximate form of the Hessian even when we could calculate the exact form. This is because the exact Hessian will not in general be positive-definite. However, most of the algorithms that use the Hessian assume that it is the curvature of multi-variate Gaussian probability distribution, and is consequently positive definite. I feel that it is better to use an approximate form of the Hessian which breaks down gracefully when the Gaussian approximation is poor than to use the exact result which would entail countless kludges to work in this regime.

When we find a maximum in weight space we know that the Hessian must have strictly positive eigenvalues (otherwise there would not be a maximum!) and so we could in principle change our algorithm from the positive-definite approximation to the exact form when we are `sufficiently close' to the solution. However, in practical problems the approximations are often good enough for us to ignore this refinement.

2.7.1 Numerical differencing

If we take the gradient, with respect to the weight vector, of the Taylor expansion of the objective function, we find:

$$\nabla Q|_{w+\Delta w} = \nabla Q|_w + \nabla\nabla Q|_w \Delta w. \tag{2.42}$$

By setting Δw to be a small fraction of each of the N unit vectors in turn we can compute an approximate form of the Hessian. The resulting matrix will not in general be symmetric, and so a better approximation will be the symmetrized version, $(A + A^T)/2$. Although this approach commends itself with its simplicity and ease of implementation it is inefficient, opaque and is subject to rounding error. Moreover it imparts no insight which might lead us to a positive-definite cousin.

2.7.2 An analytic approach

Instead of numerical attack, we can proceed analytically. Splitting the objective function into Q_W and Q_D as before, we note that the curvature of the weights is diagonal:

$$\frac{\partial^2 Q_W}{\partial w_i \partial w_j} = \frac{1}{\sigma_i^2}\delta_{ij} = A_W. \tag{2.43}$$

For finite σ this is clearly positive-definite, and so we only require the eigenvalues of $\nabla\nabla Q_D$ to be non-negative.

To calculate the contribution from the likelihood, we begin with

$$Q_D = \sum_{i,k} q_i^k, \tag{2.44}$$

and thus

$$\nabla\nabla Q_D = \sum_{i,k} \nabla\nabla q_i^k, \tag{2.45}$$

$$= \sum_{i,k} \nabla\left(\dot{q}_i^k \nabla z_i^k\right), \tag{2.46}$$

$$= \sum_{i,k} \ddot{q}_i^k \nabla z_i^k \nabla z_i^k + \dot{q}_i^k \nabla\nabla z_i. \tag{2.47}$$

Ignoring the second of these terms leaves an approximation due to Denker and Le Cun (1991),

$$\nabla\nabla Q_D = \sum_{i,k} \ddot{q}_i^k g_i^k g_i^{k\,T}, \tag{2.48}$$

where

$$g_i^k = \nabla z_i^k. \tag{2.49}$$

Here q_i^k is strictly positive for most output functions, so that equation 2.47 is a weighted sum of positive-semidefinite matrices. Thus this approximation leads to a positive-definite total curvature.

The sensitivities g_i^k may be calculated by the same backpropagation code used to calculate the gradient. Equation 2.27 gives us

$$\nabla Q_D = \sum_{i,k} \dot{q}_i^k \nabla z_i^k, \tag{2.50}$$

and so to calculate g_i^k we seed the back-propagation by setting all the error terms to zero save the one associated with q_i^k, which we set to unity. Ignoring the outer product required to form the Hessian given the sensitivities, this algorithm will deliver a positive-definite approximation to the Hessian at a cost of N gradient evaluations.

The normal justification for neglecting the second term in equation 2.47 is that the signs of $\dot{q}_i^k$ are equally likely to be positive and negative. Accordingly, the total sum is likely to be small. This approximation is often made in similar fields; examples include the MemSys package (see Gull and Skilling, 1991) and the Leavenberg-Marquardt algorithm for least-squares fitting (see Marquardt, 1963).

2.7.3 R-Backprop

For large problems direct storage of any matrix is impractical and we turn to algorithms that will compute the product $\nabla\nabla Q x$, for an arbitrary vector x. Pearlmutter (1994) derives an elegant algorithm which he calls `R-Backprop' , the R referring to a differential operator that lies at the heart of his algorithm. Our notation is slightly different to Pearlmutter's so we reproduce the algorithm here.

The product $\nabla\nabla Q b$ tells us how much the gradient changes if we move a little way along b in weight space. To see this, consider the Taylor expansion of Q,

$$Q(w+v) = Q(w) + v^T \nabla Q\big|_w + \frac{1}{2} v^T \, \nabla\nabla Q|_w \, v + O(v^3). \tag{2.51}$$

Taking the gradient of this with respect to v gives

$$\nabla Q|_{w+v} \approx \nabla Q|_w + \nabla\nabla Q|_w \, v. \tag{2.52}$$

This form suggests that we consider the differential operator $\mathcal{R}_v$ defined by:

$$\begin{aligned}
\mathcal{R}(x) &= \frac{\partial}{\partial \varepsilon} x(w+\varepsilon v)\bigg|_{\varepsilon=0} && (2.53)\\
&= \lim_{\varepsilon \to 0} \left(\frac{x(w+\varepsilon v) - x(w)}{\varepsilon} \right) && (2.54)\\
&= (v.\nabla)x && (2.55)
\end{aligned}$$

We can now rewrite equation 2.52 in terms of $\mathcal{R}$,

$$\mathcal{R}(\nabla Q) = \nabla\nabla Q \, v. \tag{2.56}$$

Applying $\mathcal{R}$ to the expression for the gradient, equation 2.41, gives

$$\mathcal{R}\left(\frac{\partial Q_D}{\partial w_{ij}^k}\right) = \mathcal{R}(x_j^k)\eta_i^k + x_j^k\mathcal{R}(\eta_i^k). \tag{2.57}$$

It is also instructive to apply $\mathcal{R}$ to the propagation equations 2.12 and 2.40. The resulting equations are a new set of propagators which allow us to calculate the $\mathcal{R}(x)$ and $\mathcal{R}(\eta)$ in equation 2.57. The forward equations become

$$\left.\begin{aligned} \mathcal{R}(y_i^k) &= \sum_j v_{ij}^k x_j^k + w_{ij}^k \mathcal{R}(x_j^k), \\ \mathcal{R}(z_i^k) &= \dot{f}^k \mathcal{R}(y_i^k), \\ \mathcal{R}(x_i^{k+1}) &= \mathcal{R}(z_i^k), \end{aligned}\right\} \tag{2.58}$$

and the backwards ones

$$\left.\begin{aligned} \mathcal{R}(e_i^k) &= \mathcal{R}(\xi_i^{k+1}) + \ddot{q}_i^k \mathcal{R}(z_i^k), \\ \mathcal{R}(\eta_i^k) &= \ddot{f}_i^k \mathcal{R}(y_i^k) e_i^k + \dot{f}_i^k \mathcal{R}(e_i^k), \\ \mathcal{R}(\xi_i^k) &= \sum_j \left(v_{ji}^k \eta_j^k + w_{ji}^k \mathcal{R}(\eta_j^k)\right). \end{aligned}\right\} \tag{2.59}$$

2.7.4 A positive-definite approximation

Although R-backprop solves the problem of large matrices it does not provide a positive-definite Hessian. To do this I have extended the method by applying Pearlmutter's operator to equation 2.27. This gives

$$\mathcal{R}(\nabla Q_D) = \sum_{i,k} \mathcal{R}(\dot{q}_i^k)\nabla z_i^k + (\dot{q}_i^k)\mathcal{R}\nabla z_i^k. \tag{2.60}$$

The first term is just $v^T(\nabla z_i^k)(\nabla z_i^k)^T$, which is positive-definite, and I accordingly drop the second term. This is the same approximation that Denker makes.

To evaluate equation 2.60, compare it with equation 2.27 and note that both are weighted sums of ∇z_i^k. Accordingly we evaluate both by a backpropagation algorithm, but seed the backpropagation step with $\ddot{q}_i^k\mathcal{R}(z_i^k)$ instead of $\dot{q}_i^k$. Happily we get the same result by simply removing some terms in the R-backprop framework. The forward propagation terms remain unchanged but equation 2.59 becomes

$$\left.\begin{aligned} \mathcal{R}(e_i^k) &= \mathcal{R}(\xi_i^{k-1}) + \ddot{q}_i^k \mathcal{R}(z_i^k), \\ \mathcal{R}(\eta_i^k) &= \dot{f}_i^k \mathcal{R}(e_i^k), \\ \mathcal{R}(\xi_i^k) &= \sum_j w_{ji} \mathcal{R}(\eta_j^k). \end{aligned}\right\} \tag{2.61}$$

and equation 2.57

$$\mathcal{R}(\frac{\partial Q_D}{\partial w_{ij}^k}) = x_j^k\mathcal{R}(\eta_i^k). \tag{2.62}$$

2.7.5 A summary of Hessian methods

I have presented four different ways to calculate the Hessian for a network, three of them established, the fourth novel. To decide which of these algorithms is appropriate, two distinct choices must be made.

The first and most important choice is whether we should compute the exact Hessian, or an approximate form which is guaranteed to be positive definite. For the reasons outlined above I prefer the latter option.

Secondly a choice must be made between R-Backprop based algorithms and the others. On this point I recommend the former, as it will happily calculate Ax or the full matrix. In general there is little to choose between the algorithms on accuracy grounds; in test problems both exact methods agree as do both positive-definite approximations.

2.8 A network catalogue

So much for general theory. We turn now to some of the standard choices for practical networks.

2.8.1 Regression networks

This is perhaps the simplest kind of network and amounts to a default choice; if we are trying to learn an arbitrary mapping between inputs and outputs it is our first choice. The network has two layers of active nodes: that is, one `hidden' layer of nodes which is connected neither to the inputs nor the outputs. The transfer function for these nodes is a $\tanh(y)$ function, the resulting outputs being linearly combined into the network outputs. This is summarized in table 2.1.

Function	Hidden layer ($k=1$)	Final layer ($k=2$)
f^k	$\tanh(y)$	y
$\dot{f}^k$	$\mathrm{sech}^2(y) = (1+f)(1-f)$	1
$\ddot{f}^k$	$-2\tanh(y)\,\mathrm{sech}^2(y) = -2f\dot{f}$	0

Table 2.1: The linear and non-linear functions used in a regression network.

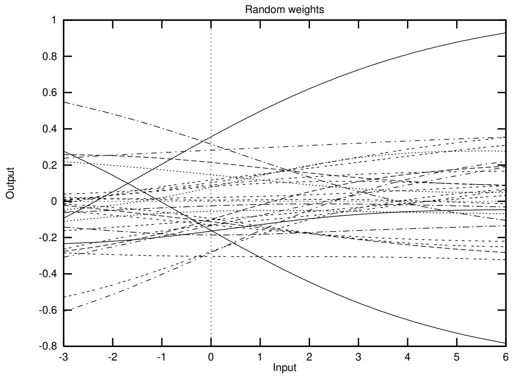
We choose a quadratic error measure, which is equivalent to assuming that the residuals (differences between the target and real outputs of the network) are expected to have a Gaussian distribution. Besides the target value from the training set, t, we also require a noise-level σ_ν. Accordingly

$$q^k = \frac{1}{2\sigma_\nu^2}(z-t)^2. \tag{2.63}$$

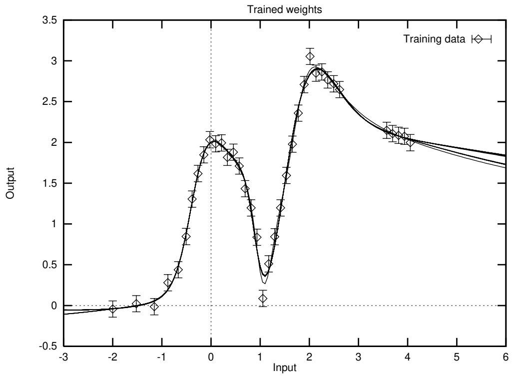
This choice of error measure leads to a non-zero Ψ_D in equation 2.17 of

$$\Psi_D = \frac{M}{2}\log(2\pi\sigma_\nu^2), \tag{2.64}$$

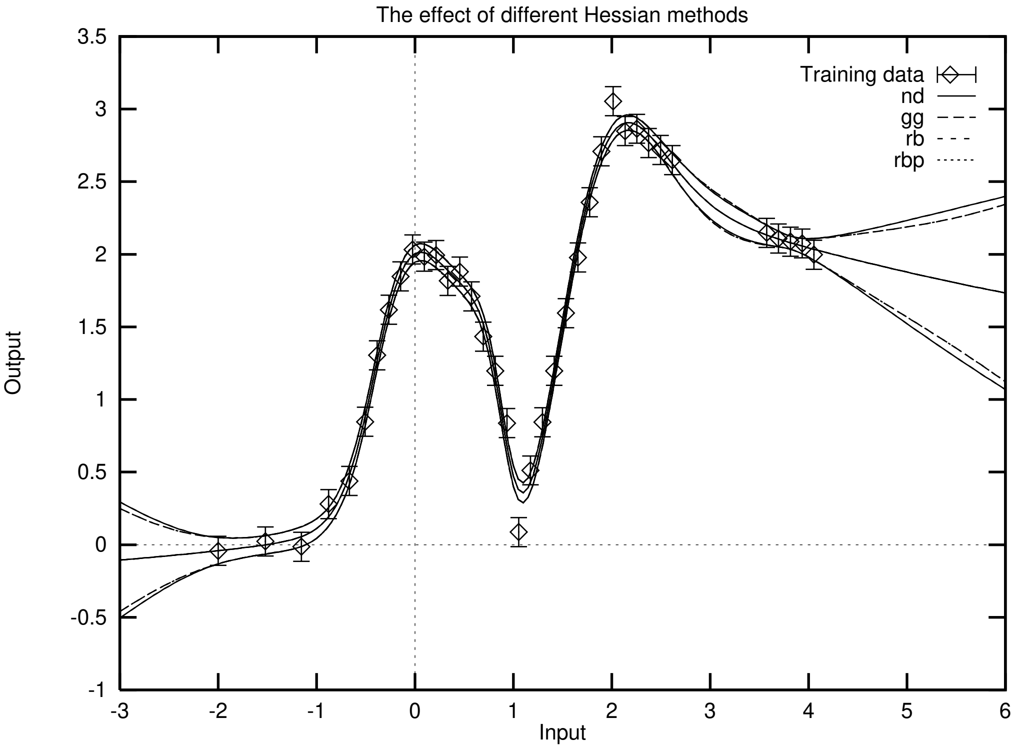
where M is the number of output nodes.



(a) Sample outputs for weights drawn from a Gaussian distribution with zero mean and a standard deviation of 0.3



(b) Sample outputs for weights drawn from a random distribution and then trained. The training data have also been plotted.



(c) The uncertainties in the sample outputs for a given set of trained weights. The uncertainties require the evaluation of the Hessian matrix which can be accomplished in several different ways. Each curve represents a different algorithm.

Figure 2.6: Illustrations of a typical regression network. This network has a single input, eight hidden nodes and a single output. Notice how the uncertainties grow in the areas devoid of training data.

2.8.2 Classification networks

If the target of our net is a scalar probability the Gaussian error measure is no longer appropriate. For example, suppose the training inputs describe an object *a priori* known to belong to one of two classes as specified in the target output. As before an entropic measure is the appropriate log-likelihood function and accordingly we choose the error function shown in table 2.2.

For general t, the error function diverges as z approaches either 0 or 1. If our training data are binary, i.e., we know with certainty the true class of each example, then our calculations will diverge only if our prediction is certain yet wrong. We can ameliorate things by choosing carefully the transfer function of the output layer's nodes. If we choose this function to approach 0 and 1 only asymptotically we remove the singular behaviour. The functions traditionally used are shown in table 2.3.

Function	General case	$t=0$	$t=1$
q^k	$-t\log(z)-(1-t)\log(1-z)$	$-\log(1-z)$	$-\log z$
$\dot{q}^k$	$(z-t)/z(1-z)$	$1/(1-z)$	$-1/z$
$\ddot{q}^k$	$t/z^2+(1-t)/(1-z)^2$	$1/(1-z)^2$	$1/z^2$

Table 2.2: The error functions for the binary classification network.

Function	Hidden layer ($k=1$)	Final layer ($k=2$)
f^k	$\tanh(y)$	$1/(1+e^{-y})$
$\dot{f}^k$	$\operatorname{sech}^2(y)=(1+f)(1-f)$	$f(1-f)$
$\ddot{f}^k$	$-2\tanh(y)\operatorname{sech}^2(y)=-2f\dot{f}$	$\dot{f}(1-2f)$

Table 2.3: The propagation functions used in the binary classifier.

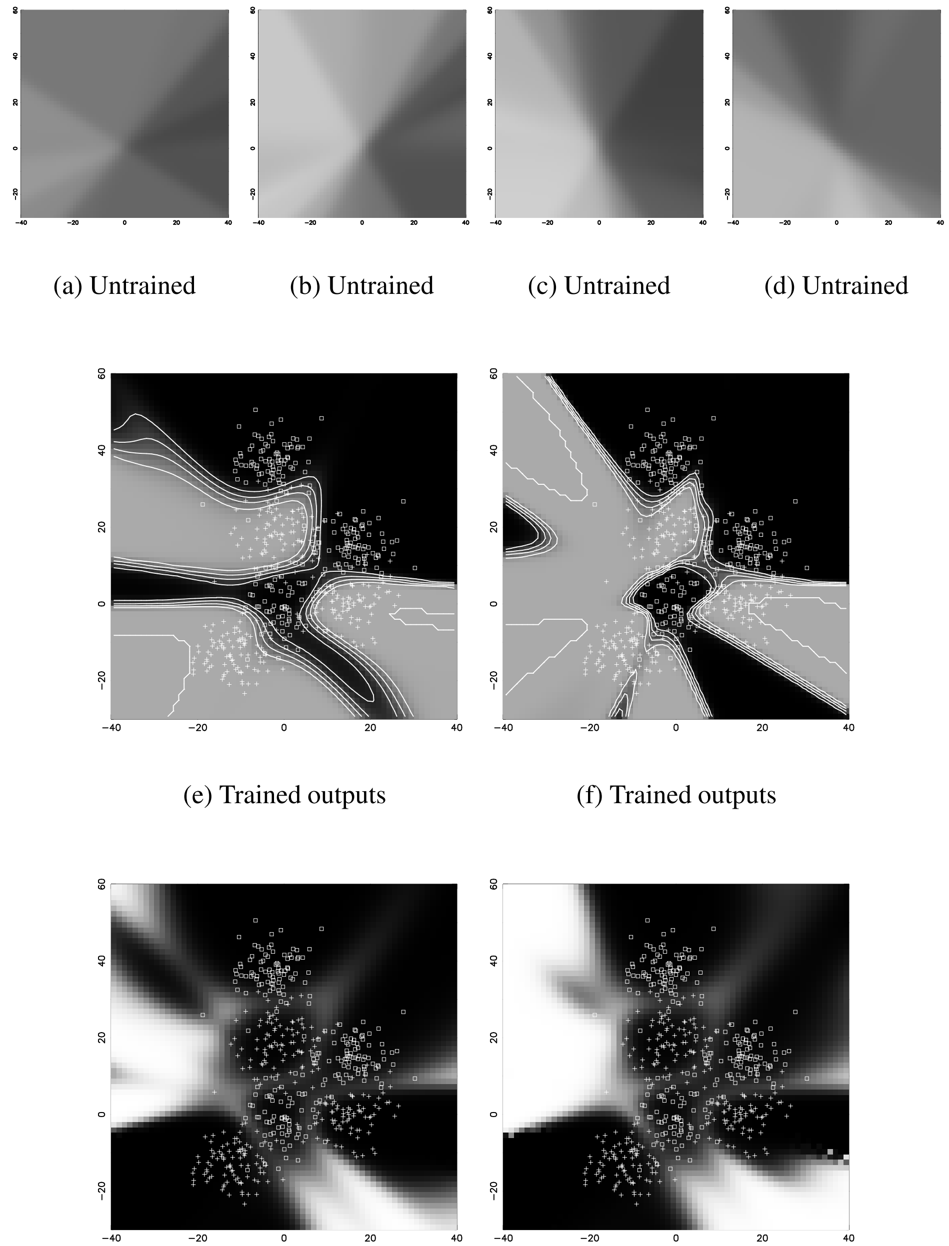
If we now examine the backward propagation equations 2.40 we see that the factor of $z(1-z)$ in the denominator of $\dot{q}$ cancels with the factor of $z(1-z)$ in $\dot{f}$. Accordingly I choose to split the q into p and r (such that $q=p+r$) and modify the equations so as to `inject' errors to both the ξ and η stages directly:

$$\left.\begin{aligned} e_i^k &= \xi_i^{k+1}+\dot{r}_i^k, \\ \eta_i^k &= \dot{f}^k e_i^k + \left(\dot{f}^k \dot{p}_i^k\right), \\ \xi_i^k &= \sum_j w_{ji}^k \eta_j^k. \end{aligned}\right\} \tag{2.65}$$

Of course this changes the value of ξ, but this is only an intermediate value. We have not changed the value of η in which all important results of the backward pass are contained.

Substituting the expression for $\dot{f}$ into that for η leads to the nice result that

$$\left.\begin{aligned} e_i^k &= \xi_i^{k+1}, \\ \eta_i^k &= (z_i^k - t_i^k) + \dot{f}^k e_i^k. \end{aligned}\right\} \tag{2.66}$$



(a) Untrained (b) Untrained (c) Untrained (d) Untrained

(e) Trained outputs (f) Trained outputs

(g) Interquartile range of the outputs calculated using the full Hessian

(h) Interquartile range of the outputs calculated using a positive-definite approximation to the Hessian

Figure 2.7: Illustrations of a typical classification network. This network had two inputs, eight hidden nodes and a single output. The two axes of each plot correspond to the two inputs, both the grey-scale and contours indicating the output of the network. Plots (a) – (d) show the outputs for random samples from the weight distributions. Plots (e) and (f) show the posterior distributions after the weights used to produce (a) and (b) have been trained. The training data in question have been overlaid on these plots; crosses correspond to $t = 1$, boxes to $t = 0$. Notice that (e) and (f) differ significantly where the outputs are uncertain; these regions are shaded brightly on plots (g) and (h).

We play a similar trick to implement R-backprop. Ignoring the propagated terms from further `up' the network we have, from equation 2.59,

$$\mathcal{R}(e_i^k) = \ddot{q}_i^k \mathcal{R}(z_i^k), \tag{2.67}$$

$$\mathcal{R}(\eta_i^k) = \dot{f}_i^k \mathcal{R}(e_i^k). \tag{2.68}$$

Substituting for f and q give the alternative formulation

$$\mathcal{R}(\eta_i^k) = \mathcal{R}(z_i^k)\left(t_i^k(1-z_i^k)/z_i^k + (1-t_i^k)z_i^k/(1-z_i^k)\right). \tag{2.69}$$

or substituting for $\mathcal{R}(z_i^k)$ from equation 2.58 as well,

$$\mathcal{R}(\eta_i^k) = \mathcal{R}(y_i^k)\left(t_i^k(1-z_i^k)^2 + (1-t_i^k)(z_i^k)^2\right). \tag{2.70}$$

2.8.3 Softmax networks

Our final standard network is an extension of the binary classifier to accommodate more than two-classes of data. This, the so-called `softmax' classifier, was first proposed by Bridle (1989).

We follow the previous section by first specifying an error function. We adopt the obvious generalization shown in table 2.4. The total error function for the network, Q, is the sum of the q_i over every output node. This gives the same result as the binary classifier if there are only two classes. To see this substitute $t_1 = t$, $t_2 = 1-t$, $z_1 = z$, and $z_2 = 1-z$. This makes the total error $Q = q_1 + q_2$ identical to that of the binary classifier shown in table 2.2.

Error Function	Softmax network
q^k	$-t_i \log(z_i)$
$\dot{q}^k$	$-t_i/z_i$
$\ddot{q}^k$	t_i/z_i^2

Table 2.4: The error measure appropriate to a softmax network.

Our choice of output function is more complicated; it must be conjugate to the error measure (lest the objective function diverge) and normalized so that $\sum_i z_i = 1$. In a softmax network we choose

$$z_i^k = \frac{\exp y_i^k}{\sum_j \exp y_j^k}. \tag{2.71}$$

This satisfies the two criteria, but we must extend our algebra slightly because z_i now depends on all the y and not just y_i. In particular,

$$\frac{\partial z_i}{\partial y_k} = z_i(\delta_{ik} - z_k), \tag{2.72}$$

and equation 2.40 becomes

$$\eta_i^k = \sum_j \frac{\partial z_j^k}{\partial y_i^k} e_j^k \tag{2.73}$$

$$= z_i^k \left(e_i^k - \sum_j z_j^k e_j^k \right). \tag{2.74}$$

If we use the same trick to inject the error term into η directly, as we did above, we find that

$$\eta_i^k = z_i^k \left(\sum_j z_j^k e_j^k - e_i^k \right) + z_i^k \left(\sum_j z_j^k \dot{p}_j^k - \dot{p}_i^k \right), \tag{2.75}$$

$$= z_i^k \left(\sum_j z_j^k e_j^k - e_i^k \right) + z_i^k \left(\sum_j z_j^k \frac{t_j^k}{z_j^k} - \frac{t_i^k}{z_i^k} \right), \tag{2.76}$$

$$= z_i^k \left(\sum_j z_j^k e_j^k - e_i^k \right) + (z_i^k - t_i^k). \tag{2.77}$$

The corresponding result for R-Backprop is a little messy but, if we neglect terms not involved directly in the softmax outputs, it is readily shown that equations 2.58 and 2.59 must be changed to

$$\mathcal{R}(z_i^k) = \sum_j \frac{\partial z_i^k}{\partial y_j^k} \mathcal{R}(y_i^k), \tag{2.78}$$

$$\mathcal{R}(\eta_i^k) = \sum_j \frac{\partial z_j^k}{\partial y_i^k} \ddot{q}_i^k \mathcal{R}(z_i^k), \tag{2.79}$$

or, expressing $\mathcal{R}(\eta)$ in terms of $\mathcal{R}(y)$

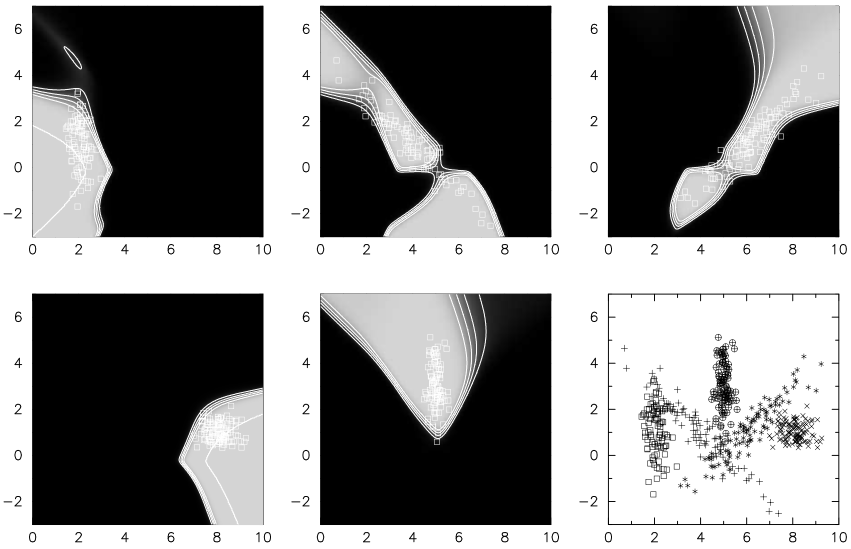
$$\mathcal{R}(\eta_i^k) = t_i^k \left(\mathcal{R}(y_i^k) - \sum_j z_j \mathcal{R}(y_j^k) \right) + z_i^k \left(\sum_j z_j \mathcal{R}(y_j^k) - \sum_j t_j \mathcal{R}(y_j^k) \right). \tag{2.80}$$

2.9 Quantifying Predictions

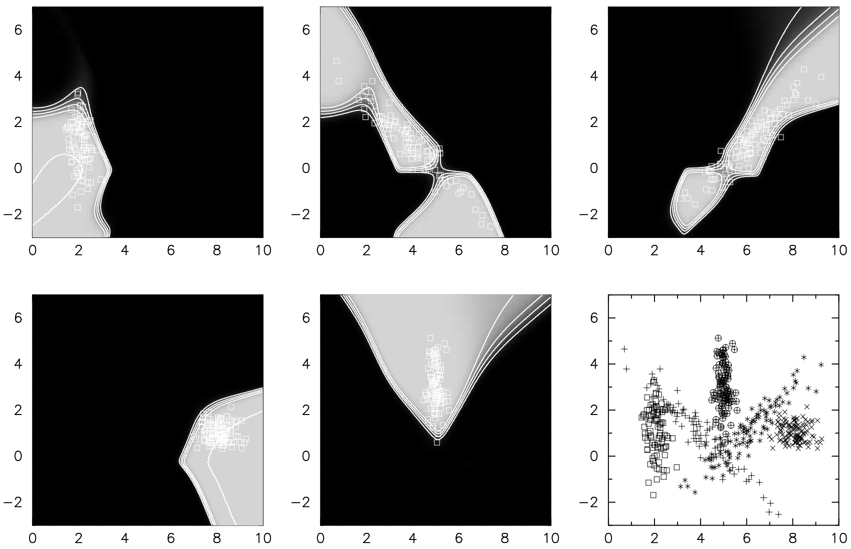
So far we have discussed only generating a single output from networks, namely the output corresponding to the modal weights. Just as the training process leads to a probability distribution on the weights, our predictions should generate a probability distribution on the network's outputs. Marginalizing over the weight vector gives

$$\mathrm{p}(z|D) = \int dw\, \mathrm{p}(z|w)\mathrm{p}(w|D). \tag{2.81}$$

The non-linear nature of the network makes it difficult to proceed exactly, so we make a locally linear approximation for $\mathrm{p}(z|w)$ and a Gaussian form for the posterior weight



(a) Trained outputs



(b) Trained outputs

Figure 2.8: Illustrations of a typical softmax network. The data are two dimensional and belong to one of five classes. In each group of six graphs, each panel corresponds to a different class, each axis of the plot corresponding to an input of the network. The output of the network is shown as both a grey scale and evenly spaced contours. Additionally those training data belonging to the class are overlaid as crosses. The sixth panel shows the union of all training data, each class being displayed with a different marker.(a) and (b) correspond to different samples from the distribution of starting weights; all networks had eight hidden units.

probability $p(w|D)$. Then z is normally distributed about the modal-value z with a variance given by

$$\sigma^2 = (\nabla z)^T A^{-1} \nabla z, \tag{2.82}$$

where A is the Hessian matrix for the weight distribution. This represents the uncertainty in the prediction: for Gaussian samples we should add an extra term, σ_ν^2, to get the expected deviation of a subsequent data point from the modal prediction.

If there are multiple outputs, correlations are possible amongst them. The same linear approximation as above leads to a covariance matrix with elements

$$C_{ij} = (\nabla z_i)^T A^{-1} \nabla z_j. \tag{2.83}$$

This is proved from the marginalization equation,

$$p(\mathbf{z}) \propto \int dw\, \delta(\mathbf{z} - (\mathbf{z} + (w)^T \nabla \mathbf{z})) \exp - \left(\frac{1}{2} (w - w_0)^T A (w - w_0) \right), \tag{2.84}$$

by writing the delta function as the limit of a narrow Gaussian; the integral is now easy and the limit trivial to take after integration.

If the output stage is non-linear then it is better to make a linear approximation for y rather then z (MacKay, 1991). Gaussian distributions are both symmetric and translationally invariant whereas z is strictly limited to lie within zero and one; progress towards these limits is much harder than towards $1/2$. Accordingly a Gaussian approximation is unlikely to be particularly faithful. The asymmetry in z has a qualitative effect on the marginalized predictions: they will be less dogmatic than their most probable cousins and MacKay accordingly calls the process `moderation'.

Having taken an approximate form for y, the consequences for z must be calculated. Since the moderated output, $\bar{z}$, must also lie on $[0,1]$, it can be written as

$$\bar{z} = \frac{1}{1 + \exp(-y')} \quad \text{for some } y'. \tag{2.85}$$

MacKay's approximation is to take $y' = \kappa \hat{y}$. By definition

$$\bar{z} = \int dy \frac{1}{1 + \exp(-y)} \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp - \frac{(y - \hat{y})^2}{2\sigma_y^2}, \tag{2.86}$$

where $\hat{y}$ and σ_y define the Gaussian approximation for y.

Now to solve for κ we match equations 2.85 and 2.86. The requirement that $\bar{z} = z$ for $\sigma_y = 0$, and $\bar{z} \to 1/2$ as $\sigma_y \to \infty$, is satisfied for any κ tending to 1 and 0 in these limits. To find an explicit form MacKay matches the gradients in the large σ_y limit for $y = 0$. Thus he assigns

$$\kappa = \frac{1}{\sqrt{1 + \pi\sigma^2/8}}. \tag{2.87}$$

Figure 2.10 shows the large scale effects of moderation. A more quantitative assessment of the efficacy of the method can be found by calculating the test errors of the moderated output and comparing them to the test error of the most-probable output. MacKay does this and notes that `The improvement is most substantial for underdetermined networks with relatively poor performance. In a small fraction of the solutions however, especially among the best solutions, the moderated outputs are found to have a slightly but significantly inferior performance.' I support this statement, but my own simulations suggest that the first class of networks are in the minority and accordingly moderation makes the average test-error worse.

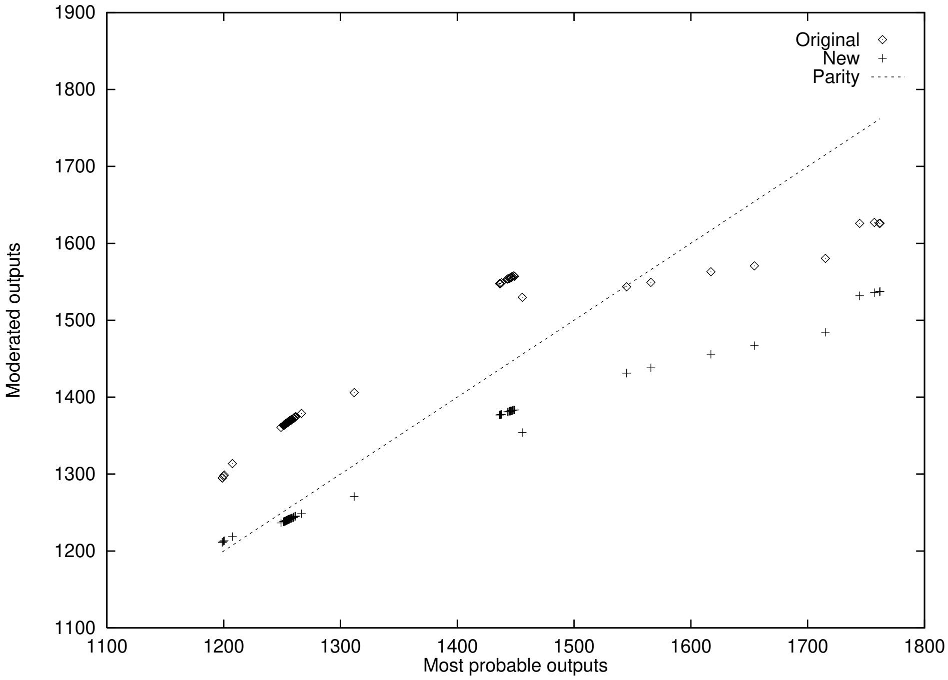


Figure 2.9: The effect of moderation on the test error of a binary classifier. Two schemes are shown: MacKay's original formulation and a new approximation derived in the text. Predictions below the parity line have a smaller test error when moderated; those above should be left unmoderated.

To improve matters, we seek a better approximation in the low σ limit. As the existing process is harmful to well trained networks it is plausible that matching the approximation more closely at low σ_y than high will correct the problem. If σ_y is small we can evaluate equation 2.86 by expanding the logistic function about $\hat{y}$; the resulting series is a power series in σ_y,

$$\bar{z} \approx \hat{z} + \frac{1}{2}\hat{z}(1-\hat{z})(1-2\hat{z})\sigma^2 + O(\sigma^4). \tag{2.88}$$

Retaining the same functional form for κ and expanding that in powers of σ gives

$$\kappa = 1/\sqrt{1+\alpha\sigma^2}, \tag{2.89}$$

$$\bar{z} \approx \hat{z} - \frac{1}{2}\hat{z}^2\frac{(1-\hat{z})}{\hat{z}}\hat{y}\alpha\sigma^2 + O(\sigma^4). \tag{2.90}$$

Matching these series leads to a new approximation

$$\kappa = 1/\sqrt{1+(2\hat{z}-1)\sigma_y^2/\hat{y}}. \tag{2.91}$$

Calculating the test error using this approximation gives better results. Figure 2.9 plots the moderated test error against the most probable error for one hundred runs; the new prescription is always better than MacKay's original formulation and gives smaller errors than the most-probable outputs in all but three cases.

Perhaps the correct conclusion to draw from this is that moderation is a good thing but, with apologies to Disraeli, sometimes there is excess in moderation.

We can also generate the inter-quartile range, a useful diagnostic in the classification process. To do this observe

$$z(\hat{y}+\alpha\sigma_y) = \frac{\hat{z}}{\hat{z}+\exp(\alpha\sigma_y)(1-\hat{z})}. \tag{2.92}$$

Now evaluate z for $\alpha = \pm 0.6745$; these values are the quartiles.

2.9.1 A practical comment

In general $\hat{y}$ and σ_y are `hidden variables' buried somewhere inside our program; we have access only to $\hat{z}$ and σ_z. However

$$\hat{z} = 1/(1+\exp(-\hat{y})) \Longrightarrow \hat{y} = \log(\hat{z}) - \log(1-\hat{z}), \tag{2.93}$$

$$\sigma_z = \frac{dz}{dy}\sigma_y \Longrightarrow \sigma_y = \sigma_z/z(1-z). \tag{2.94}$$

And so

$$z' = \frac{\hat{z}^\kappa}{\hat{z}^\kappa + (1-\hat{z})^\kappa}. \tag{2.95}$$

This prescription can be applied to any network without re-writing the code, although for predictions where $z \approx 0$ or 1, rounding may become a problem. In practice this is rarely troublesome if the inputs are not too far from the domain covered by the training data.

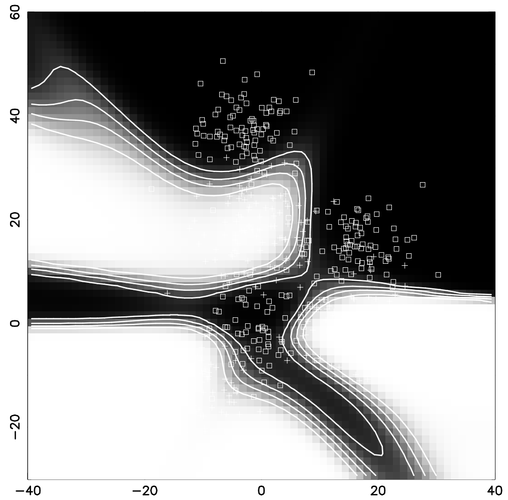
2.9.2 Moderating softmax networks

Moderating softmax networks is a more challenging task, because the probability distribution over the z_i or the y_i is multi-dimensional and correlated; accordingly it is difficult to proceed as above by finding a global solution with the correct limiting behaviour, and then matching it to the exact solution in the low uncertainty limit.

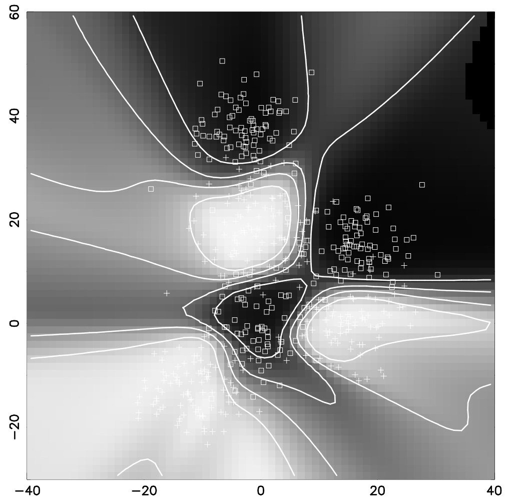
The most obvious global form to choose is

$$z'_i = \frac{z_i^{\kappa_i}}{\sum_j z_j^{\kappa_j}}, \tag{2.96}$$

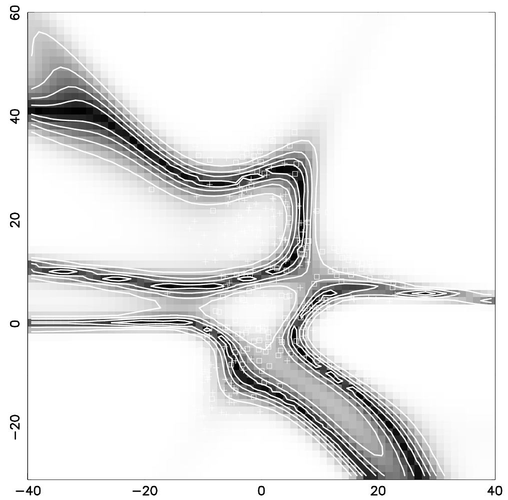
but we have been unable to find a suitable set of κ_i which moderate yet don't destroy correct predictions; accordingly the test errors for networks moderated in this fashion are often appallingly poor. To proceed we must seek a different way of sharing the probabilities between the classes.



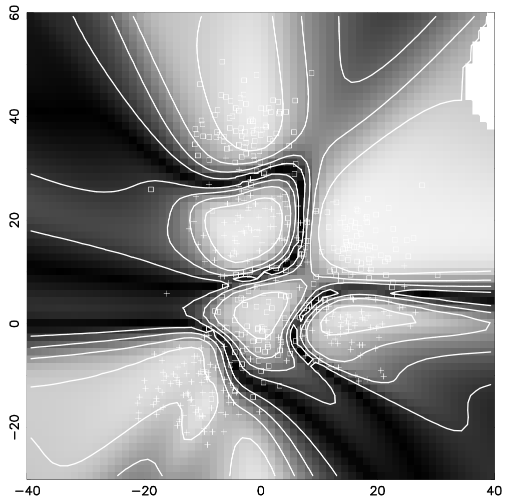
(a) The unmoderated outputs.



(b) The moderated outputs.



(c) Highly dogmatic regions (unmoderated)



(d) Highly dogmatic regions (moderated)

Figure 2.10: The effect of moderation on a network's predictions. The moderated predictions (right) show a pleasingly modest attitude far from the training data when the unmoderated outputs are overly dogmatic. To be explicit, the lower plots show $2\,|z-1/2|$.

In my experience a similar problem is often found in a restaurant after a meal with more than about three people present; when the bill arrives, a debate often ensues. Typically people `claim' some fraction of the total with varying degrees of certainty, the remaining shortfall being divided amongst the uncertain diners. We can apply the same algorithm to moderate the softmax network.

Given a series of unmoderated probabilities z_i subject to uncertainties σ_i we assert that some fraction of the probability is `certain' and the remainder should be pooled. The moderated probabilities are calculated by adding the `certain' part to an equitable share of the `pool'. Explicitly,

$$z_i' = \exp(-\kappa\sigma_i^2)z_i + (1-\exp(-\kappa\sigma_i^2))Z, \tag{2.97}$$

$$Z = (1-\sum_i \exp(-\kappa\sigma_i^2)z_i)/(N-\sum_i \exp -\kappa\sigma_i^2) \tag{2.98}$$

This has the desirable feature of leaving z unchanged if $\sigma = 0$ and sharing the probability equally for outputs with $\sigma = \infty$. There are many choices for κ but setting it to $1/2$ gives good results. Figures 2.11 and 2.12 illustrate the different predictions of moderated and unmoderated networks.

2.10 A summary of probabilistic modelling

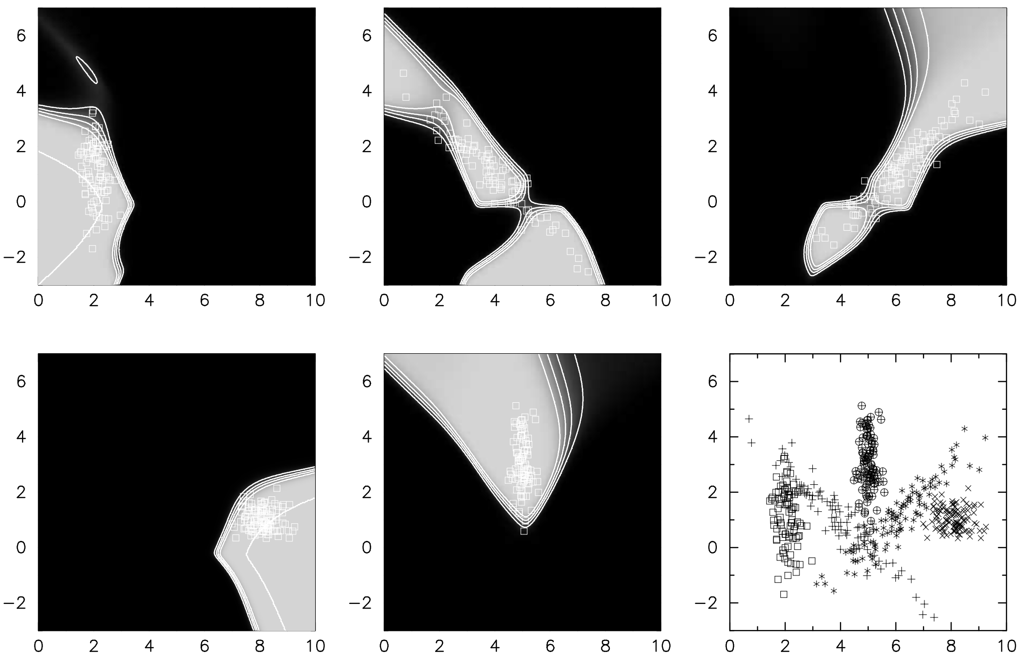
If the central thesis of the preceding chapter was that probability theory is the language of reasoning and inference, this chapter has added some vocabulary to the grammar.

I have presented a new algorithm for calculating a positive definite approximation to the network's Hessian. This draws heavily on previous work but facilitates the construction of robust models and training procedures. Such robustness is a *sine qua non* for the acceptance of probabilistic networks by a non-specialist audience.

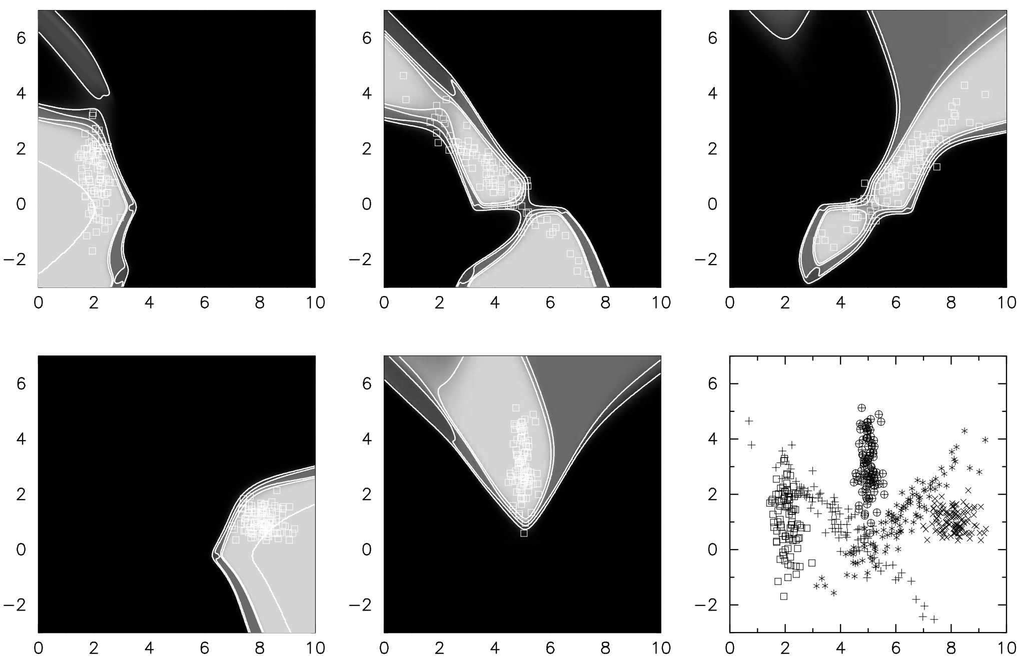
I have also improved the performance of binary classifiers with a new approximation to the exact marginalization result — this too is more robust than present techniques which damage good solutions.

Finally I have introduced a simple marginalization scheme for softmax classifiers. This scheme both improves the test error of the network and leads to less dogmatic predictions.

Much work remains to be done. The models considered here often rely on Gaussian approximations, which may do harm. Monte-Carlo techniques can provide an antidote, for example Neal (1994), and we look forward to future progress in the direction.

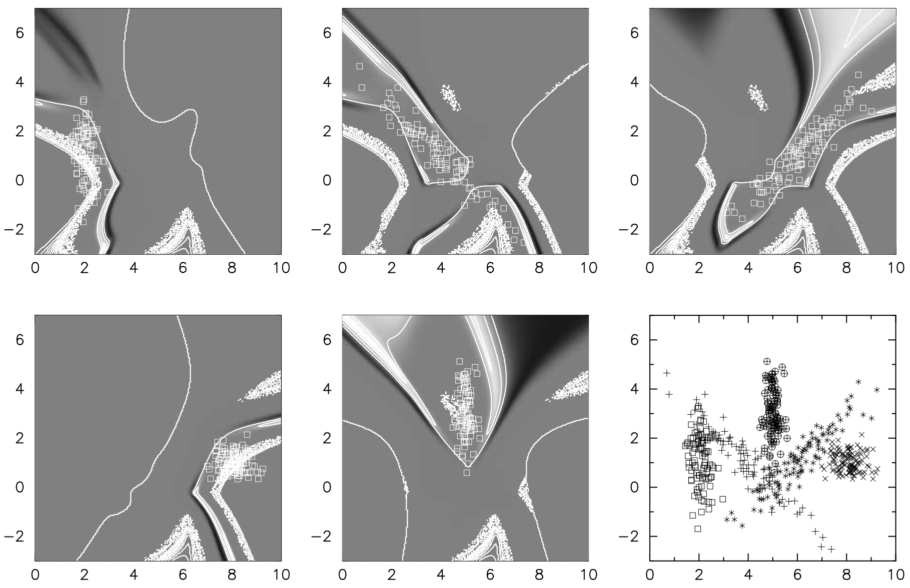


(a) The unmoderated outputs.

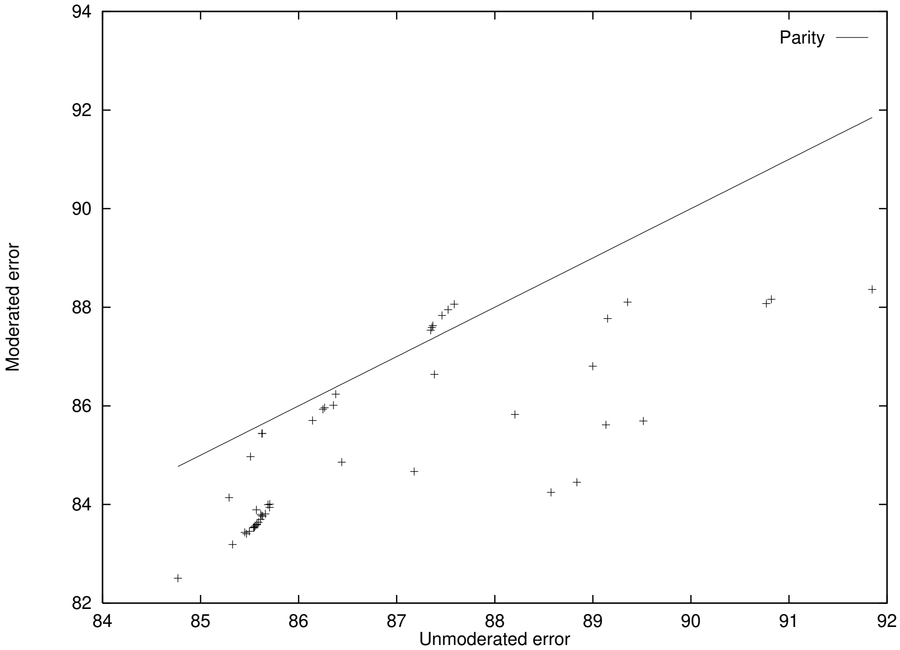


(b) The moderated outputs.

Figure 2.11: The effect of moderation on a softmax network. The upper group of plots shows the unmoderated outputs which should be compared with their moderated cousins below; the differences here are less obvious than in similar plots for the binary classifier.



(a) The difference made by moderation.



(b) The effect on test-error.

Figure 2.12: The top plot shows the differences between the probabilities of each class for the softmax network. Most of the plots are grey; these regions have not changed much by moderation. The lower plot investigates the test error scores for fifty different weight samples before and after moderation. In most cases moderation can be seen to improve the test error of the network.

Chapter 3

The learning experience

> Here let us breathe and haply institute a course of learning and ingenious studies.
>
> *Act I Scene I, The Taming of the Shrew, William Shakespeare.*

In the previous chapter I introduced neural networks as flexible, general-purpose, probabilistic models. However I did not suggest how these models might be used in practice and in particular did not discuss how to find an optimal set of weights $\hat{w}$, nor how `hyper-parameters', such as σ_W which appear in the prior on the weights, could be handled. I now rectify that omission.

3.1 The weights

Finding an optimal set of weights is a standard optimization problem and accordingly we can apply any of the standard techniques available. For small problems where the size of the weight vector does not prohibit the storage of an explicit Hessian matrix, quasi-Newtonian `variable-metric' methods are efficient; they are described in Press *et al.* (1992).

For larger problems we are limited to vector quantities taking $O(N)$ storage where N is the number of weights. In particular we will use algorithms generically called `conjugate-gradients'.

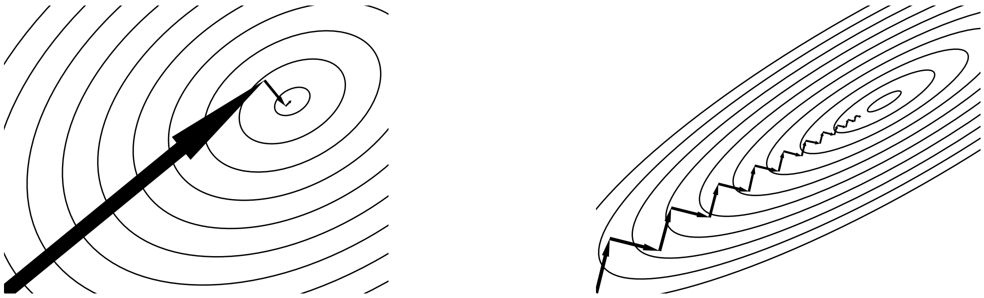
3.1.1 Conjugate-gradients.

Two vectors, a and b, are said to be A-conjugate to each other if

$$a^T A b = 0. \tag{3.1}$$

To understand why this idea is important for weight optimization we first consider a model algorithm (which is of little practical use).

The simplest algorithm to optimize a function is perhaps the `steepest-descent' method. This prescription has us choose a point, find the direction of steepest descent from the point and then travel straight along this line until we reach a minimum. We then repeat the process along the new downward direction until we finally reach a local minimum. At any stage the new direction must be orthogonal to its predecessor, and so our path will be a series of straight paths and right-angled turns. At first sight this seems sensible, and indeed would work well in the gently undulating Cambridgeshire countryside (if we ignore ditches and drains) but suppose we find ourselves in a narrow Alpine valley. In such geography we are likely to tack slowly along the valley floor, only reaching the minimum after a large number ($\gg N$) steps. This is shown in figure 3.1.



(a) Circular valley (b) Elliptical valley

Figure 3.1: A comparison of the steepest descent algorithm in circular and elliptical valleys. In the latter case the algorithm tacks inefficiently along the valley floor.

The key difference between these scenarios is the differing scale sizes in different directions or, more formally, the range of the eigenvalues of the curvature matrix. Steepest-descents is effective only if this range is small; otherwise myopia prevents the algorithm from matching the step to the overall scale of the function. However, we can transform the general problem to one with a circular curvature by stretching the space along the principal axes of its Hessian. If we then ask which vectors in the original space correspond to orthogonal vectors in its isotropic cousin, we find that these are A-conjugate to each other. This is why conjugate-gradient ideas are important in multi-dimensional optimization theory: in a general space conjugate vectors are minimally interfering.

Although successive minimizations along gradients are an inefficient method, the sequence of subspaces spanned by successive gradients is a useful framework within which to conduct minimization. This was first noted by Hestenes and Stiefel (1952); a good description can be found in Golub and Van Loan (1990), whose notation we follow. However our path through the theory is the opposite of the usual one. Most derivations start from a set of conjugate vectors and consider the space spanned by these; we start with the space (though spanned by the gradient vectors) and show that successive minima in this space are related by vectors obeying equation 3.1. As well as generating the necessary results, this derivation is clearer than the conventional approach.

Definition 3.1 $\phi(x)$ *is the function whose minimum we seek:*

$$\phi(x) = \phi(\hat{x}) + \frac{1}{2}(x-\hat{x})^T A(x-\hat{x}). \tag{3.2}$$

Definition 3.2 *x_0 is defined to be* 0. *For $j = 1, 2, \ldots$, x_j is the point in $\mathcal{S}_j$ that minimizes the value of ϕ.*

Definition 3.3 *r_j for $j = 0, 1, \ldots$ is the negative gradient, evaluated at the j^{th} local minimum x_j.*

$$r_j = -\nabla \phi|_{x=x_j} = A(\hat{x} - x_j). \tag{3.3}$$

Definition 3.4 *$\mathcal{S}_j$ is the j-dimensional subspace* $\text{span}\{r_0, r_1, \ldots, r_{j-1}\}$.

Definitions 3.2 to 3.5 are recursive, not circular. We could program a computer to evaluate the sequence x_0, r_0, $\mathcal{S}_1$, x_1, $\ldots$ without referring to any undefined quantities.

Theorem 3.1 *r_j lies wholly outside the subspace $\mathcal{S}_j$ i.e. r_j is orthogonal to every member of the set $\{r_0, \ldots, r_{j-1}\}$. Hence this statement is equivalent to the statement that the set $\{r_0, \ldots, r_j\}$ are orthogonal:*

$$r_i^T r_j = 0, \quad i \neq j. \tag{3.4}$$

We prove this by showing that its converse leads to a contradiction. If r_i has a non-zero projection onto $\mathcal{S}_i$ we could reduce the value of ϕ by moving along it; this contradicts the definition of x_i as the $x \in \mathcal{S}_i$ which minimizes ϕ. Consequently $\mathcal{S}_j$ grows along the directions `important' to ϕ; successive r_i suck out more of variation from ϕ, leaving progressively insipid structure in the space orthogonal to $\mathcal{S}_j$.

Definition 3.5 *s_i is the step between successive minima:*

$$s_i = x_i - x_{i-1}. \tag{3.5}$$

Theorem 3.2 *The set $\{s_i\}$ are conjugate to each other and consequently satisfy*

$$s_k^T A s_j = 0, if \quad j \neq k. \tag{3.6}$$

To prove this, suppose (without loss of generality) that $j \geq k$, and apply A to definition 3.5. Now use definition 3.3 to eliminate Ax_i in favour of r_i:

$$As_k = r_{k-1} - r_k. \tag{3.7}$$

Now, from the definition of s_k,

$$s_k^T A s_j = x_k^T r_{j-1} + x_{k-1}^T r_j - x_{k-1}^T r_{k-1} - x_k^T r_k. \tag{3.8}$$

Theorem 3.1 tells us that $r_j \notin \mathcal{S}_j$, while $x_j \in \mathcal{S}_j$ by construction. Hence, all but the first term on the right hand side vanish. Furthermore, the remaining term is non-zero only when $j = k$ proving the theorem. As a bonus we have also shown that

$$s_k^T A s_k = x_k^T r_{k-1}. \tag{3.9}$$

Theorem 3.3 $\mathcal{S}_j = \operatorname{span}\{r_0, Ar_0, \ldots, A^{j-1}r_0\}$.

We prove this inductively. Suppose that the theorem is true for j and then consider $\mathcal{S}_{j+1}$. By definition

$$\mathcal{S}_{j+1} = \operatorname{span}\{r_0, r_1, \ldots, r_{j-1}, r_j\}. \tag{3.10}$$

Now we use the orthogonality of $\{r_i\}$ and the supposition above to replace the first j of these vectors. This gives

$$\mathcal{S}_{j+1} = \operatorname{span}\{r_0, Ar_0, \ldots, A^{j-1}r_0, r_j\}. \tag{3.11}$$

But

$$r_j = A(\hat{x} - x_j) = r_0 - Ax_j \tag{3.12}$$

and since $x_j \in \mathcal{S}_j$ we can use the supposition again

$$r_j \in \operatorname{span}\{Ar_0, \ldots, A^{j-1}r_0, A^j r_0\}. \tag{3.13}$$

Now by definition $\mathcal{S}_{j+1} = \operatorname{span}\{\mathcal{S}_j, r_j\}$ and so if the supposition is true for j it is also true for $j+1$. The supposition is manifestly true for $j = 1$ and hence, by induction, for all positive j.

Subspaces of this type, generated by the repeated action of a matrix on a vector, are called Krylov subspaces after the Russian mathematician.

Theorem 3.4 $\mathcal{S}_k = \operatorname{span}\{s_1, s_2, \ldots, s_{k-1}, r_{k-1}\}$.

To prove this, note first that $\{s_j : j = 1, \ldots, (k-1)\}$ are linear multiples of $\{r_j : j = 0, \ldots, (k-2)\}$, both being independent sets of k vectors. So $\operatorname{span}\{s_1, s_2, \ldots, s_{k-1}\}$ and $\operatorname{span}\{r_0, r_1, \ldots, r_{k-2}\}$ are equal, yet by the definition of $\mathcal{S}_k$,

$$\mathcal{S}_k = \operatorname{span}\{\mathcal{S}_{k-1}, r_{k-1}\}, \tag{3.14}$$

thus proving the result.

We have described a seemingly sensible approach to function optimization but our algorithm currently requires us to solve a number of j-dimensional ($j \in \{0, 1, \ldots, k\}$) minimizations. Although we hope that our minimization will be effective even when $k \ll N$, this is still less than ideal. The real power of the conjugate-gradient approach appears because it is possible to generate the set of vectors $\{s_i\}$ without first calculating the orthogonal set $\{r_i\}$. Thus we can proceed to each new x_i without carrying out an explicit minimization.

Theorem 3.5

$$s_j = \frac{s_j^T A s_j}{r_{j-1}^T r_{j-1}} r_{j-1} - \frac{s_j^T A s_j}{s_{j-1}^T A s_{j-1}} s_{j-1}. \tag{3.15}$$

s_j must lie in $\mathcal{S}_j$ and so from theorem 3.4 we can write, without loss of generality,

$$s_j = \sum_{i=1}^{j-1} \eta_i s_i + \xi r_{j-1}. \tag{3.16}$$

To prove the theorem we now pre-multiply equation 3.16 by $s_i^T A$ and use theorems 3.1 and 3.2. Hence

$$\xi = \frac{s_j^T A s_j}{r_{j-1}^T r_{j-1}}, \tag{3.17}$$

$$\eta_i = \begin{cases} -s_j^T A s_j / s_{j-1}^T A s_{j-1} & \text{for } i = j-1, \\ 0 & \text{for } i = \{1, 2, \ldots, j-2\}. \end{cases} \tag{3.18}$$

Substituting these results into equation 3.16 proves the result.

Traditionally a scaled set of conjugate vectors, p_i, are used:

$$p_i = \frac{p_i^T A p_i}{r_{j-1}^T r_{j-1}} s_i. \tag{3.19}$$

Substituting this into equations 3.15 and 3.19 gives

$$p_j = r_{j-1} - \frac{r_{j-1}^T r_{j-1}}{r_{j-2}^T r_{j-2}} p_{j-1}, \tag{3.20}$$

$$x_j = x_{j-1} + \frac{r_{j-1}^T r_{j-1}}{p_j^T A p_j} p_j. \tag{3.21}$$

If we now substitute this formula for x_j into equation 3.2 we derive a recursive formula for the minimum,

$$\phi_j = \phi_{j-1} - \frac{1}{2} \left(r_{j-1}^T r_{j-1} \right)^2 / p_j^T A p_j. \tag{3.22}$$

Finally, applying A to equation 3.21 leads to a recursive formula for r_j.

$$r_j = r_{j-1} - \frac{r_{j-1}^T r_{j-1}}{p_j^T A p_j} A p_j. \tag{3.23}$$

These equations, applied recursively, will find the minimum of ϕ in no more than N steps if our arithmetic is exact. In practice we regard the algorithm as an iterative technique, terminating when the gradient falls below some fraction of the initial gradient or when we exceed some iteration limit. Thus our basic conjugate-algorithm for the minimization of a quadratic function is given below.

Algorithm 3.1 (Conjugate-gradient minimization of $\phi(x)$)

Set tolerance, ε, and iteration limit, j_{max}.

$$
\begin{array}{l}
x_0 = 0 \\
\phi_0 = \phi(x_0) \\
r_0 = -\nabla\phi|_{x_0} \\
\beta_0 = r_0^T r_0 \\
j = 1 \\
\text{do} \\
\quad \{ \\
\qquad \begin{array}{ll}
p_j & = \begin{cases} r_0 & \text{if } j = 1 \\ r_{j-1} - (\beta_{j-1}/\beta_{j-2}) p_{j-1} & \text{otherwise} \end{cases} \\
\alpha & = \beta_{j-1}/p_j^T A p_j \\
x_j & = x_{j-1} + \alpha p_{j-1} \\
\phi_j & = \phi_{j-1} - \left(r_{j-1}^T r_{j-1}\right)^2 / \left(2 p_j^T A p_j\right) \\
r_j & = r_{j-1} - \alpha A p_{j-1} \\
\beta_j & = r_j^T r_j \\
j & = j + 1
\end{array} \\
\quad \} \text{ while } (\,(j < j_{max})\ \&\&\ (\beta_j < \varepsilon\beta_0))
\end{array}
$$

On exit, x_j contains the approximate position of the minimum.

3.1.2 Conjugate-gradients without the Hessian

Sometimes it will not be easy to calculate Ax. If our function is sufficiently quadratic, we can still apply the ideas of the previous section at the expense of some gradient evaluations and one-dimensional line-searches.

The matrix A appears explicitly in the updating equations for both r_j and x_j. Accordingly we must find alternative ways to calculate these quantities. The first is easy: we return to the definition of r_j (equation 3.3) and calculate the gradient of ϕ directly. To calculate x_j, recall that x_j is defined to minimize ϕ on $\mathcal{S}_j$. Thus we can find x_j by minimizing

$$\phi_j(\lambda_j) = \phi(x_{j-1} + \lambda_j p_j) \tag{3.24}$$

over λ.

This algorithm, often called the Fletcher-Reeves algorithm, was modified by Polak and Ribiere (see Polak, 1971) who modified the update rule for p_j to

$$p_j = r_{j-1} + \frac{r_{j-1}^T (r_{j-1} - r_{j-2})}{r_{j-2}^T r_{j-2}} p_{j-1}. \tag{3.25}$$

Although these expressions are identical when ϕ is perfectly quadratic, the latter variant is found to be more efficient when faced with a quasi-quadratic function.

3.1.3 Pre-conditioning

It is natural to ask for which problems the conjugate-gradient technique works effectively. There is a substantial body of formal work (see, for example, Luenberger, 1973) but the most important conclusion is that the algorithm works well if the condition number (the ratio of the largest and smallest eigenvalues) of the matrix is close to unity, or, in other words, the matrix is close to some multiple of the identity.

In general, matrices from real problems do not have such a limited spread of eigenvalues. Accordingly we can improve the performance of a conjugate-gradient routine by transforming the problem into a space that compresses the eigenspectrum. As an example, consider the minimization of $\phi(x)$:

$$\phi(x) = \phi(\hat{x}) + \frac{1}{2}(x-\hat{x})^T A(x-\hat{x}), \tag{3.26}$$

$$= \phi(\hat{x}) + \frac{1}{2}(x-\hat{x})^T C^{-1}CACC^{-1}(x-\hat{x}), \tag{3.27}$$

$$= \phi(\hat{x}) + \frac{1}{2}(y-\hat{y})^T (CAC)(y-\hat{y}). \tag{3.28}$$

where

$$y = C^{-1}x, \qquad \hat{y} = C^{-1}\hat{x}. \tag{3.29}$$

If we now minimize with respect to y, having chosen C to make CAC closer to the identity, the minimization will proceed more quickly. We then solve equation 3.29 for $\hat{x}$. This step limits our choice of C; in particular we require C to be easily invertible. Consequently diagonal or pre-factored matrices are often used.

Since network training often requires several optimizations, one direction for further research is to use the results of one optimization to precondition the next.

3.1.4 Summary of weight optimization

Many different algorithms have been used to optimize the weights of a neural network; a recent summary is given by Jervis and Fitzgerald (1993). For the demonstrations contained in this dissertation I have used a mixture of line-based conjugate-gradient methods and variable-metric methods. Figure 3.2 compares the efficiency of three different optimization algorithms on a typical test problem.

The efficiency of both methods relies crucially on the line-search method chosen. In particular, variable-metric methods can benefit greatly from a crude line-search algorithm, but some caveats are required. It is relatively easy for crude algorithms to become stuck in non-quadratic parts of weight space or to stop before finding the true minimum. In this work I have erred on the side of caution and used robust line searches.

Future work might develop robust, efficient optimizers based around variable-metric methods and explore hybrid schemes which store low-rank approximations to the Hessian; see, for example, Gill *et al.* (1981). Another plausible approach involves working directly in weight space and foregoing line-searches entirely. The positive-definite R-backprob algorithm is expected to be useful in this context.

3.2 The hyper-parameters

Both the prior and likelihood functions contain unknown parameters which the Bayesian prescription has us marginalize over. Since we regard these parameters as being a higher

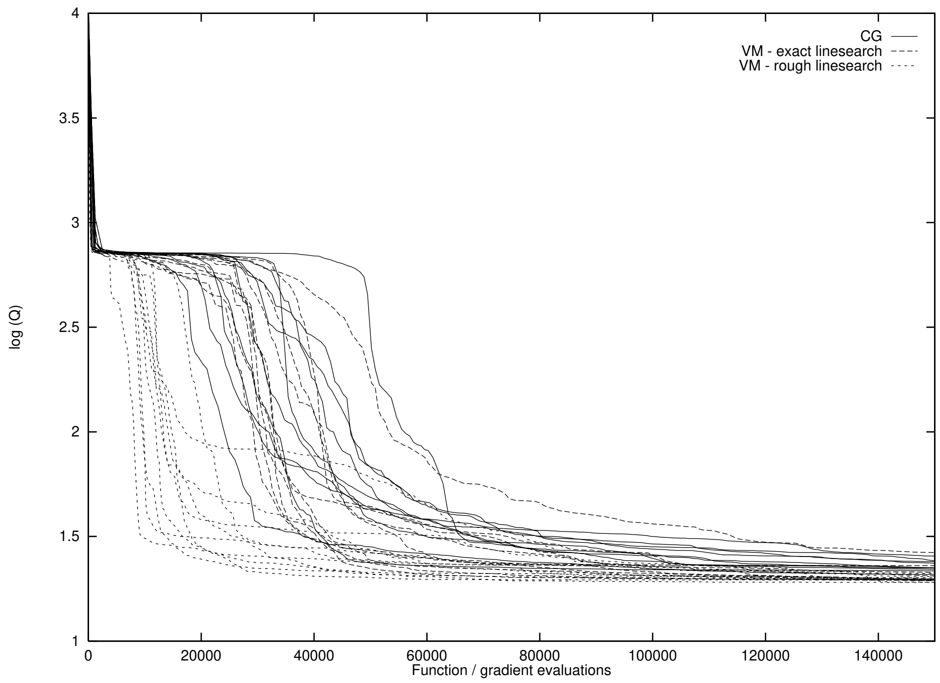


Figure 3.2: An illustration of the relative efficiency of different multi-dimensional optimization schemes. Three different algorithms are shown: Conjugate-gradient minimization without explicit use of a Hessian (denoted CG), a variable-metric method using exact line-searches; and the same method coupled to a cruder one-dimensional minimizer. Of these three, the last is the most efficient though all methods show a considerable variation in performance. Each line charts the progress of the minimization of Q against the number of function evaluations required by the minimization algorithm. In this case, Q is the objective function of a multi-layer regression network, the ten different traces for each minimizer corresponding to different starting positions.

level of an inference hierarchy, it is natural to call them hyper-parameters. However if we adopt this approach the resulting distribution is very skewed, and so difficult to compute with; instead we make the `Evidence' approximation due to Gull (1988) which we used for the simple Gaussian problem in subsection 1.5.3. Instead of integrating the function over the hyper-parameters we approximate this integral by setting the hyper-parameters to the values which maximize the evidence (equation 2.25). This idea has been discussed at length in Wolpert (1993) and MacKay (1993).

Before we find the optimal values of the hyper-parameters we present a mathematical preliminary

Definition 3.6 *The set $\{\mathcal{I}_c\}$ form a partitioning of the identity matrix. That is, they are diagonal matrices with either zero or unit diagonal entries and they sum to the identity. Thus*

$$\sum_c \mathcal{I}_c = I. \tag{3.30}$$

Theorem 3.6

If

$$A \equiv \sum_C \alpha_c \mathcal{I}_c + B \tag{3.31}$$

then

$$\frac{\partial}{\partial \alpha_i} \log \det A = \operatorname{Tr}\left(A^{-1} \mathcal{I}_i\right) \tag{3.32}$$

Proof: As is so often with symmetric real matrices, we proceed by stepping through the looking glass, into the frame in which the matrix is diagonal. We introduce the diagonal matrix $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_N)$ and the orthogonal transformation matrix O. Together they satisfy

$$A = O^T \Lambda O. \tag{3.33}$$

Both the determinant and trace of a matrix are unaffected by orthogonal transformations, so that

$$\begin{aligned}
\frac{\partial}{\partial \alpha_i} \log \det A &= \frac{\partial}{\partial \alpha_i} \log \det \Lambda, && (3.34)\\
&= \sum_i^N \frac{\partial}{\partial \alpha_i} \log \lambda_i, && (3.35)\\
&= \sum_i^N \frac{1}{\lambda_i} \frac{\partial \lambda_i}{\partial \alpha_i}, && (3.36)\\
&= \operatorname{Tr}\left(\Lambda^{-1} \frac{\partial \Lambda}{\partial \alpha_i}\right), && (3.37)\\
&= \operatorname{Tr}\left(O A^{-1} O^T O \mathcal{I}_i O^T\right), && (3.38)\\
&= \operatorname{Tr}\left(A^{-1} \mathcal{I}_i\right). && (3.39)
\end{aligned}$$

It is now simple to differentiate equation 2.25 with respect to σ_i (assuming $\hat{w}$ remains fixed). This gives

$$-\frac{\partial}{\partial\sigma_c}\log \mathrm{p}(D) = -\frac{1}{\sigma_c^3}E_{W,c} + \frac{2}{\sigma_c^3}\mathrm{Tr}A^{-1}\mathcal{I}_i + \frac{N_c}{\sigma_c}. \tag{3.40}$$

Accordingly the best set of σ_cs must satisfy

$$\frac{2}{\sigma_c^2}E_{W,c} = N_c - \frac{1}{\sigma_c^2}\mathrm{Tr}(A^{-1}\mathcal{I}_c) \equiv \mathcal{G}_c. \tag{3.41}$$

Gull (1989) christened the term on the right-hand side of the equation `Good' as it tells us how many of the parameters in the class we have determined from the data and, conversely, how many from the prior. To see this consider the special case where A is diagonal and there is only a single class of weights (so $\mathcal{I}_c = I$). If we now introduce the eigenvalues of the likelihood's curvature, β_i, we can rewrite good, $\mathcal{G}$, as

$$\mathcal{G} = N - \sum_i \frac{1/\sigma^2}{1/\sigma^2 + \beta_i} \tag{3.42}$$

$$= \sum_i \frac{\beta_i}{1/\sigma^2 + \beta_i} \tag{3.43}$$

If $\beta_i \ll 1/\sigma^2$ the uncertainty from the data dominates that from the prior and the term will be small; conversely if the data dominate then the term will tend to one. Thus the full sum tells us how many terms are determined the data, or the number of good measurements.

We follow MacKay (1991) in using equation 3.41 to estimate the optimum values of the hyper-parameters once we have found an optimal set of weights, $\hat{w}$. We then iterate this procedure until a self-consistent solution has been found. There is no formal proof of this method's convergence but it does seem to work in practice.

Algorithm 3.2 (The training algorithm)

Randomize the weight vector
Choose starting values for $\{\sigma_W\}$
do
  {
    Minimize $Q(w)$ to find a set of weights, $\hat{w}$.
    Calculate good, $\mathcal{G}$, by making a Gaussian approximation at $\hat{w}$.
    $\sigma_W = 2E_W/\mathcal{G}$
    Reduce tolerance for weight optimization
  } while (!converged)

Typically the loop is traversed 6 – 10 times during a training run.

3.3 Large problems

Both the optimization of the hyper-parameters and the evaluation of the evidence require us to calculate functions of the Hessian matrix, A. Many algorithms exist to calculate the determinant and trace of a matrix but this matrix will become unwieldy when the number of weights, N, exceeds a few hundred.

Skilling (1989) has shown that the trace of a matrix A is the expectation of the A-norm of a random vector v if $< v_i v_j >= \delta_{ij}$. Explicitly,

$$\mathcal{T} = v^T A v, \tag{3.44}$$

$$\mathrm{Tr} A =< \mathcal{T} >, \tag{3.45}$$

$$\approx \frac{1}{M} \sum_i^M \mathcal{T}_i. \tag{3.46}$$

Proof:

$$< \mathcal{T} > = \left\langle \sum_{i,j} v_i v_j A_{ij} \right\rangle, \tag{3.47}$$

$$= \sum_{i,j} A_{ij} \left\langle v_i v_j \right\rangle, \tag{3.48}$$

$$= \sum_i A_{ii}. \tag{3.49}$$

This statistical method inevitably introduces noise into the evaluation of the trace; to quantify the effect we calculate the variance of $\mathcal{T}$

$$\mathrm{var}\, \mathcal{T} =< \mathcal{T}^2 > - < \mathcal{T} >^2, \tag{3.50}$$

$$= \sum_{i,j,k,l} < v_i v_j v_k v_l > A_{ij} A_{kl} - (\mathrm{Tr} A)^2. \tag{3.51}$$

If we characterize the uncertainty in $\mathcal{T}$ by a Gaussian distribution then its standard deviation is $\mathrm{var}\, \mathcal{T} / \sqrt{(N-1)}$.

If the vector v is drawn from a Gaussian distribution with zero mean and unit variance then $< v_i v_j >= \delta_{ij}$ as required and

$$< v_i v_j v_k v_l >= \delta_{ij}\delta_{kl} + \delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}. \tag{3.52}$$

Thus

$$\mathrm{var}\, \mathcal{T}_{Gauss} = \left(\sum_i A_{ii}^2 \right) + 2 \sum_{i,j} \left(A_{ij} \right)^2 - (\mathrm{Tr} A)^2, \tag{3.53}$$

$$= 2\, \mathrm{Tr} \left(A^2 \right). \tag{3.54}$$

For the MaxEnt program MemSys, Skilling advocates the use of vectors with elements chosen randomly from $\{1, -1\}$. Accordingly

$$< v_i v_j v_k v_l >= \begin{cases} 1 & \text{if } i = j = k = l, \\ \delta_{ij}\delta_{kl} + \delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk} & \text{otherwise.} \end{cases} \tag{3.55}$$

Although this is not an isotropic choice of v (given v it would be easy to spot the coordinate axes), it reduces the variance significantly to

$$\mathrm{var}\, \mathcal{T}_{unit} = \sum_{i \neq j} A_{ij}^2. \tag{3.56}$$

This is smaller than the variance in the Gaussian case and, if we are lucky enough to encounter a diagonal A, we will get the trace exactly right with just a single random sample.

Another plausible choice for v is a normalized Gaussian, which can also be thought of as a random point on the surface of an N dimensional hypersphere. This too is a much better choice than the un-normalized Gaussian. To see this, consider moving off the hypersphere's surface; this will scale $\mathcal{T}$ and correspondingly increase its variance. The remaining variations occur because, broadly, don't quite get the `direction right', and mix the eigenvalues. Thus we expect terms in the variance to contain differences between eigenvalues rather than the eigenvalues themselves. Skilling considers this too (*pers. comm.*) thus

$$< v_i v_j >= \int_{\mathbb{R}^N} (2\pi)^{-N/2} \exp\left(-V^2/2\right) \frac{x_i x_j}{V^2} d^N x \tag{3.57}$$

For $i \neq j$ the integrand is odd and the expression is zero. For $i = j$ symmetry dictates that the expression is independent of i and so

$$\begin{aligned}
< v_i^2 > &= \frac{1}{N} \sum_j < v_j^2 >, && (3.58)\\
&= \frac{1}{N} \int_{\mathbb{R}^N} (2\pi)^{-N/2} \exp\left(-V^2/2\right) \frac{\sum_j x_j^2}{V^2} d^N x, && (3.59)\\
&= 1. && (3.60)
\end{aligned}$$

Similarly,

$$< v_i v_j v_k v_l >= \int_{\mathbb{R}^N} (2\pi)^{-N/2} \exp\left(-V^2/2\right) \frac{x_i x_j x_k x_l}{R^4} d^N x \tag{3.61}$$

vanishes if any of i,j,k or l don't match. This leaves only the two cases $< v_i^4 >$ and $< v_i^2 v_j^2 >$

$$\begin{aligned}
< v_i^4 > &= \int_{\mathbb{R}^N} (2\pi)^{-N/2} \exp\left(-R^2/2\right) \frac{x_i^4}{V^4} d^N x, && (3.62)\\
&= \int_{\mathbb{R}^+} (2\pi)^{-N/2} \exp\left(-V^2/2\right) dV \int \frac{x_i^4}{V^4} dS_N, && (3.63)\\
&= \int \frac{x_i^4}{V^4} dS_N / \int dS_N, && (3.64)\\
&= \frac{\int_{\mathbb{R}^+} (2\pi)^{-N/2} f(V) \exp\left(-V^2/2\right) dV \int x_i^4/V^4 dS_N}{\int_{\mathbb{R}^+} (2\pi)^{-N/2} f(V) \exp(-V^2/2) dV \int dS_N}, && (3.65)
\end{aligned}$$

for any $f(V)$. Now choose $f(V) = V^4$ and reassemble the integrals

$$\begin{aligned}
&= \frac{\int_{\mathbb{R}^N} (2\pi)^{-N/2} \exp\left(-V^2/2\right) x_i^4 d^N x}{\int_{\mathbb{R}^N} (2\pi)^{-N/2} \exp(-V^2/2) V^4 d^N x}, && (3.66)\\
&=< x_i^4 > / < V^4 >, && (3.67)\\
&= 3/\left(3N + (N^2 - N)\right), && (3.68)\\
&= \frac{3}{N(N+2)}. && (3.69)
\end{aligned}$$

Furthermore

$$< v_i^2 v_j^2 > = < x_i^2 x_j^2 > / < V^4 >, \tag{3.70}$$

$$= \frac{1}{N(N+2)}. \tag{3.71}$$

Finally substituting these into equation 3.51 reveals

$$\operatorname{var} \mathcal{T}_{norm} = \frac{2N}{N+2} \operatorname{Tr}(A)^2 - \frac{2}{N+2} (\operatorname{Tr} A)^2. \tag{3.72}$$

Or in terms of the eigenvalues, λ_i, and the mean eigenvalue, $\overline{\lambda}$,

$$\operatorname{var} \mathcal{T}_{norm} = \frac{2N}{N+2} \sum_i \left(\lambda_i - \overline{\lambda}\right)^2. \tag{3.73}$$

This confirms our intuition that the range of the eigenvalues is all important.

The choice between the normalized Gaussian and the unitary distributions is a non-trivial one. The two-by-two case is easy to analyse completely and we find that there is no best choice; in particular if

$$A = \begin{pmatrix} a & c \\ c & b \end{pmatrix}, \tag{3.74}$$

equations 3.56 and 3.72 give

$$\operatorname{var} \mathcal{T}_{unit} = 4c^2, \tag{3.75}$$

$$\operatorname{var} \mathcal{T}_{norm} = \frac{1}{2}(a-b)^2 + 2c^2, \tag{3.76}$$

However the 2×2 case is not representative of large matrixes for which the $N/(N+2)$ term becomes essentially unity. This factor controls the `c^2' term in equation 3.76, so for large matrices we should choose unitary vectors.

We have evaluated the variance of twenty samples of $\mathcal{T}$ at thirty points during the training of a regression network containing twenty-four hidden units. Table 3.1 and figure 3.3 show the results of this experiment. As expected the normalized Gaussian method is about better by about $1 - 24/(24+2) \approx 10\%$ but the unitary method better still.

3.3.1 The trace of an inverse

The re-estimation formula for the hyperparameters, equation 3.41, contains $\operatorname{Tr} A^{-1} \mathcal{I}_c$. To calculate this we evaluate

$$\mathcal{T}_c = r^T A^{-1} \mathcal{I}_c r \tag{3.77}$$

$$= r^T \mathcal{I}_c A^{-1} \mathcal{I}_c r, \tag{3.78}$$

$$= r_c^T A^{-1} r_c. \tag{3.79}$$

where

$$r_c = \mathcal{I}_c r. \tag{3.80}$$

Distribution	Mean of $(\sqrt{\operatorname{var}\mathcal{T}}/\mathcal{T})$	$(\sqrt{\operatorname{var}\mathcal{T}}/\mathcal{T})_i$
Gaussian	0.45	0.8240 0.7722 0.6701 1.1854 0.7145 0.6243 0.4543 0.3707 0.4296 0.3689 0.3227 0.3734 0.3553 0.3246 0.3603 0.3560 0.3244 0.3646 0.3544 0.3264 0.3573 0.3547 0.3267 0.3566 0.3545 0.3266 0.3569 0.3546 0.3266 0.3568
Normalized	0.41	0.8096 0.7230 0.6587 1.1405 0.6657 0.6090 0.4135 0.3335 0.3729 0.3296 0.2869 0.3305 0.3161 0.2866 0.3234 0.3151 0.2876 0.3242 0.3129 0.2896 0.3183 0.3132 0.2898 0.3177 0.3131 0.2897 0.3180 0.3131 0.2898 0.3178
Unitary	0.30	0.7819 0.6457 0.7218 1.1057 0.5825 0.6371 0.2054 0.2314 0.2779 0.1176 0.1949 0.2252 0.1324 0.1984 0.2168 0.1475 0.2079 0.2107 0.1508 0.2179 0.2098 0.1508 0.2181 0.2091 0.1508 0.2180 0.2094 0.1508 0.2180 0.2093

Table 3.1: The table shows twenty samples of the trace of the partitioned Hessian matrix taken from a regression network. Three different sampling strategies are compared; for a full discussion see the text.

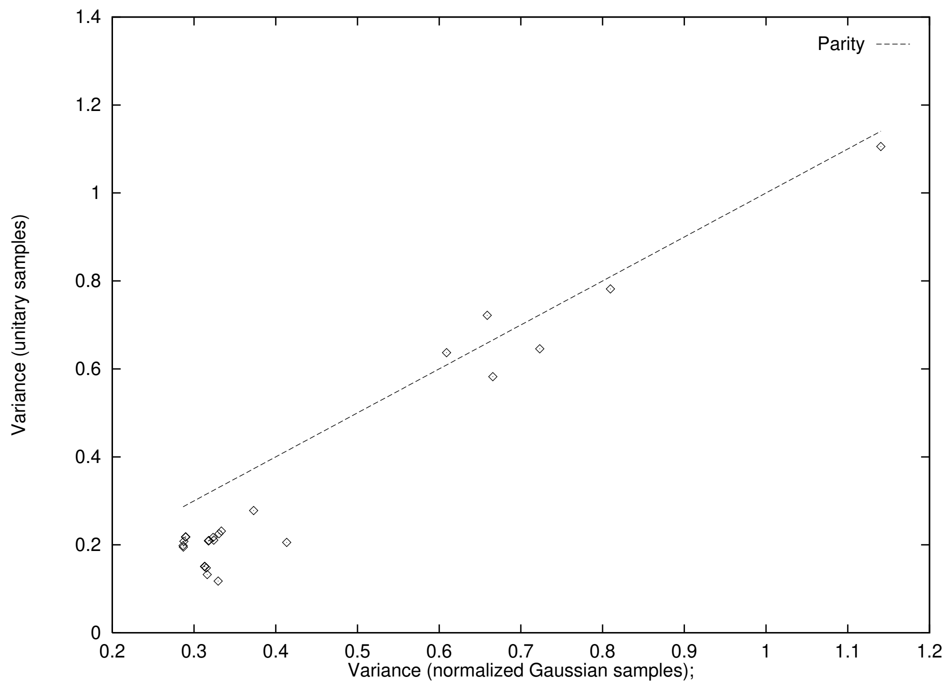


Figure 3.3: A comparison of the variance of different sampling schemes for evaluating Good. The data points are taken from a typical training run of a regression network containing twenty-four hidden units; the line indicates equal performance from both schemes. In practice the unitary sampling scheme performs better.

Thus we require the trace of the *inverse* of a matrix when that matrix is too large to invert directly. This is not a problem; we solve the inverse problem $x = A^{-1}r$ and then take the dot product $(r^T x)$. There are many ways to accomplish this but a particularly useful approach is via the ideas of section 3.1.1. We note that

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b \tag{3.81}$$

is minimized when

$$x = A^{-1}b, \tag{3.82}$$

with a minimum value

$$\phi_{min} = -\frac{1}{2}b^T A^{-1}b. \tag{3.83}$$

Accordingly we can cast the matrix inverse problem as a multidimensional minimization.

Solving $x = A^{-1}r_c$ with a conjugate gradient algorithm requires an effective preconditioner. A plausible choice is the curvature of the prior, since this is diagonal and thus easy to manipulate. Introducing

$$A_W = \sum_c \frac{1}{\sigma_c^2}\mathcal{I}_c, \tag{3.84}$$

we apply the preconditioning thus

$$\mathcal{T}_c = r_c^T A_W^{-1/2} A_W^{1/2} A^{-1} A_W^{1/2} A_W^{-1/2} r_c. \tag{3.85}$$

But

$$A_W^{-1/2} r_c = \sum_{c'} \sigma_{c'} \mathcal{I}_{c'} r_c \tag{3.86}$$

$$= \sigma_c r_c. \tag{3.87}$$

Thus

$$\mathcal{T}_c = \sigma_c^2 r_c^T \left(A_W^{1/2} A^{-1} A_W^{1/2} \right) r_c, \tag{3.88}$$

$$= r_c^T A^{-1} r_c, \tag{3.89}$$

where A is the preconditioned matrix. If we ignore the statistical error inherent in the sampling process, successive steps of the conjugate gradient algorithm lead to monotonically increasing approximations to the trace, bounded above by the exact result. Accordingly, the values of good calculated in this way form a progressively tighter upper bound on the true value.

However, equations 3.41 and 3.79 imply

$$\mathcal{G}_c = N_c - < r_c^T A^{-1} r_c >, \tag{3.90}$$

$$= < r_c^T (I - A^{-1}) r_c >, \tag{3.91}$$

$$= < r_c^T A^{-1} (A - I) r_c > . \tag{3.92}$$

We can evaluate this expression also by conjugate gradients, leading to a progressively better lower bound on the true value. By running the two algorithms in parallel we progressively refine the bounds on $\mathcal{G}$. The difference between these two expressions provides us with a natural termination criterion; typically we will optimize until the difference is less than 10% of the average value.

Robinson (1992) suggests running both conjugate gradient algorithms in the same subspace. Accordingly we can write a more efficient algorithm by solving equation 3.92 in the subspace generated by the solution of equation 3.79. Rather than derive a special case algorithm, we have recourse to Lanczos theory (see Lanczos (1950) or Golub and Van Loan (1990)).

3.3.2 Lanczos methods

Lanczos methods generate a low-dimensional tri-diagonal representation of a large matrix. It is not surprising that conjugate-gradient methods, which handle high-dimensional matrices by working in a low-dimensional subspace, can be seen in a Lanczos context. To be precise, the central result that makes the connection is

$$QAQ^T = T, \tag{3.93}$$

where Q is the matrix of normalized r vectors generated by the conjugate gradient algorithm:

$$Q = \begin{pmatrix} r_0^T / \sqrt{r_0^T r_0} \\ r_1^T / \sqrt{r_1^T r_1} \\ \vdots \\ r_{j-1}^T / \sqrt{r_{j-1}^T r_{j-1}} \end{pmatrix} \tag{3.94}$$

and T is tri-diagonal matrix. The elements of T are easy to compute; introduce

$$\sigma_i = p_i^T A p_i, \tag{3.95}$$

$$\rho_i = r_i^T r_i. \tag{3.96}$$

and express r_j in terms of p_j and p_{j+1} using equation 3.20. Hence

$$r_i^T A r_j = \left(\sigma_{j+1} + \frac{\rho_j \rho_k}{\rho_{j-1}\rho_{k-1}}\sigma_j\right)\delta_{jk} - \frac{\rho_j}{\rho_{j-1}}\sigma_j \delta_{j(k+1)} - \frac{\rho_k}{\rho_{k-1}}\sigma_k \delta_{j(k-1)}. \tag{3.97}$$

Clearly this is non-zero only when $j = (k-1)$, k or $(k+1)$, proving that the matrix is tri-diagonal. To find the diagonal elements set $j = k$, whereupon

$$q_j^T A q_j = \frac{1}{\rho_j} r_j^T A r_j = \frac{\sigma_{j+1}\rho_{j-1}^2 + \sigma_j \rho_j^2}{\rho_j \rho_{j-1}^2}; \tag{3.98}$$

while for the sub-diagonal set $j = k+1$,

$$q_j^T A q_{j+1} = \frac{1}{\sqrt{\rho_j \rho_{j+1}}} r_j^T A r_{j+1} = -\sqrt{\frac{\rho_j}{\rho_{j-1}^3}}\sigma_j. \tag{3.99}$$

If Q were square it would be orthogonal and we could invert equation 3.93 to express A in terms of Q and T. Even though Q isn't square a consequence of the Lanczos method is that we can often treat it as if it were. For example, to within the accuracy provided by the Lanczos method,

$$A \approx Q^T T Q, \quad A^n \approx Q^T T^n Q. \tag{3.100}$$

More precisely,

$$AQ = QT + \frac{\sigma_j}{\rho_{j-1}^{3/2}} r_j e_j^T. \tag{3.101}$$

For a formal proof see the earlier references; an intuitive understanding can be gained by considering $y = (AQ)x = A(Qx)$. Our definition of $\mathcal{S}_j$ is such that y must lie in the subspace unless $r_{j-1}^T Qx \neq 0$ or, equivalently $x^T e_j \neq 0$. In this case we need a bigger subspace $\mathcal{S}_{j+1}$ to contain AQx, but, as the difference between these subspaces is r_j we must have

$$AQ = QT + \alpha r_j e_j^T. \tag{3.102}$$

To find α, sandwich the equation between r_j^T and e_j and then use equation 3.99. Perhaps the most important idea from Lanczos theory is that A's behaviour on $\mathbb{R}^N$ is mimicked by T's on $\mathbb{R}^j$, and that the rectangular matrix Q maps between these two spaces.

Having derived the method we now apply it. As an example consider our expression for the trace, equation 3.46. Lanczos theory allows us to rewrite this as

$$< r^T A^{-1} r > \approx < r^T Q^T T^{-1} Q r >, \tag{3.103}$$

but r is orthogonal to all but the first column of Q and so

$$< r^T A^{-1} r > \approx < r^T r >< e_0^T T^{-1} e_0 > . \tag{3.104}$$

More formally,

$$AQ = QT + \alpha r_j e_j^T; \tag{3.105}$$

sandwiching this between A^{-1} and T^{-1} we have

$$QT^{-1} = A^{-1}Q + \alpha A^{-1} r_j e_j^T T^{-1}, \tag{3.106}$$

and so

$$T^{-1} = Q^T A^{-1} Q + \alpha Q^T A^{-1} r_j e_j^T T^{-1}. \tag{3.107}$$

$$r_0^T A^{-1} r_0 = e_0^T Q^T A^{-1} Q e_0. \tag{3.108}$$

Now substitute for $Q^T A^{-1} Q$ from above

$$= r_0^T r_0 \left(e_0^T T^{-1} e_0 - \alpha e_0^T Q^T A^{-1} r_j e_j^T T^{-1} e_0 \right), \tag{3.109}$$

$$= r_0^T r_0 \left(e_0^T T^{-1} e_0 - \alpha (r_{j-1} A^{-1} r_j)(e_j^T T^{-1} e_0) \right). \tag{3.110}$$

As the algorithm converges $r_j \to 0$ and accordingly $e_0^T T^{-1} e_0$ converges to $\mathcal{T}$.

We will face some fairly complicated manipulation of T, and pause at this point to consider its factorization.

Theorem 3.7 *T can be factorized as*

$$T = \Delta M^T \Sigma M \Delta \tag{3.111}$$

where Δ and Σ are diagonal and M is upper bi-diagonal. Explicitly,

$$\Delta = \begin{pmatrix} 1/\sqrt{\rho_0} & 0 & \dots & 0 \\ 0 & 1/\sqrt{\rho_1} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1/\sqrt{\rho_{n-1}} \end{pmatrix} \tag{3.112}$$

$$M = \begin{pmatrix} 1 & -\rho_1/\rho_0 & \dots & \dots & 0 \\ 0 & 1 & -\rho_2/\rho_1 & & \vdots \\ 0 & 0 & \ddots & \ddots & 0 \\ \vdots & & \ddots & 1 & -\rho_{n-1}/\rho_{n-2} \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix} \tag{3.113}$$

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n \end{pmatrix} \tag{3.114}$$

This factorization is more than simply blind algebra; Σ is a representation of A in the space spanned by p_i, M a mapping between this space and that spanned by r_i and Δ the scaling needed to normalize the r_i.

The theorem facilitates the manipulation of T; in particular it allows us to solve $x = T^{-1}e_0$ and then, from equation 3.104 to evaluate $\mathcal{G}_c$. Unsurprisingly we get the same answer as solving for $A^{-1}r$ and then explicitly doing the dot-product *viz.*,

$$\mathcal{G}_u = \rho_0 - \sum_i \frac{\rho_{i-1}^2}{\sigma_i}. \tag{3.115}$$

We can now evaluate the lower bound on good given in equation 3.92

$$\mathcal{G}_l = < x^T A^{-1}(A-I)x > . \tag{3.116}$$

To evaluate this we introduce the scalar function

$$\phi(z) = \frac{1}{2} z^T (A-I)Az - x^T (A-I)z \tag{3.117}$$

which is maximized at $z = A^{-1}x$. In the interests of brevity we have suppressed the tilde from A. If z is chosen from the subspace $\mathcal{S}_j$, $z = Qz$ for some $z \in \mathbb{R}^j$ and so

$$\phi(z) = \frac{1}{2} z^T Q^T (A-I)AQz - x^T (A-I)Qz. \tag{3.118}$$

Maximizing over z requires

$$(Q^T AAQ - Q^T AQ)z = (Q^T A - Q^T)x. \tag{3.119}$$

But

$$Q^T x = \sqrt{\rho_0}\, e_0, \tag{3.120}$$

and

$$AQ = QT + \frac{\sigma_j}{\rho_{j-1}^{3/2}} r_j e_j^T. \tag{3.121}$$

so

$$(T^2 - T + \frac{\rho_j \sigma_j^2}{\rho_{j-1}^3} e_j e_j^T)z = \sqrt{\rho_0}(T - I)e_0 \tag{3.122}$$

This is close to the desired result, except that the matrix on the left-hand-side of this equation is penta-diagonal. However the factorization of T expressed in equation 3.111 facilitates reducing the problem to a tri-diagonal inverse. Explicitly,

$$P = (T^2 - T + \rho_j e_j e_j^T), \tag{3.123}$$

$$= \Delta M^T \left(\Sigma M \Delta^2 M^T \Sigma - \Sigma + \frac{\sigma_j^2 \rho_j}{\rho_{j-1}^3} \left(M^T\right)^{-1} \Delta^{-1} e_j e_j^T \Delta^{-1} M^{-1} \right) M\Delta, \tag{3.124}$$

$$= \Delta M^T T M \Delta. \tag{3.125}$$

and inverting both sides gives

$$P^{-1} = \Delta^{-1} M^{-1} T^{-1} 1 \left(M^T\right)^{-1} \Delta^{-1}. \tag{3.126}$$

Substituting this into equation 3.92 gives

$$\mathcal{G}_l = < x^T A^{-1}(A - I)x >, \tag{3.127}$$

$$= < z^T Q^T (A - I)x >, \tag{3.128}$$

Now substitute for z from equation 3.122 and $Q^T A$ from equation 3.121 to give

$$= < y^T T^{-1} y > . \tag{3.129}$$

where

$$y = \sqrt{\rho_0} \left(M^T\right)^{-1} \Delta^{-1} (T - I) e_0. \tag{3.130}$$

Though these equations look fearsome, the components of both T and y are in fact easy to calculate. Explicitly,

$$T_{ii} = \frac{\sigma_i^2}{\rho_{i-1}} (1 + \rho_i/\rho_{i-1}) - \sigma_i, \tag{3.131}$$

$$T_{i(i+1)} = -\frac{\sigma_i \sigma_{i+1}}{\rho_{i-1}}, \tag{3.132}$$

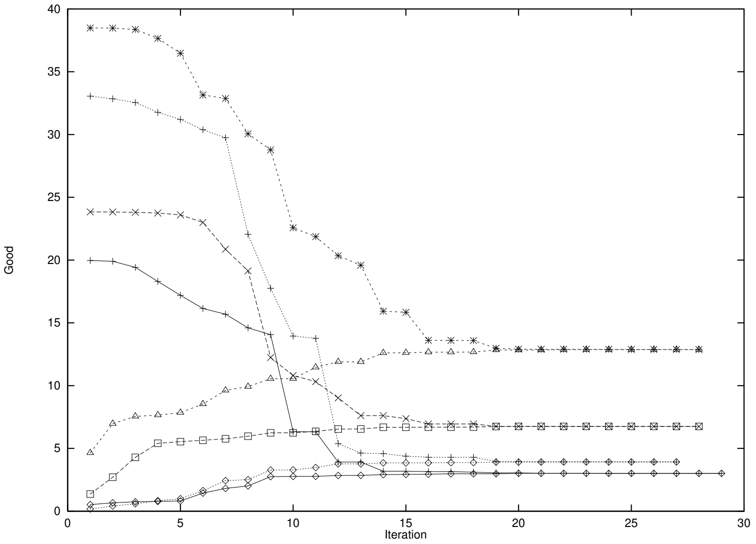


Figure 3.4: Successive estimates of Good generated by the Lanczos method. Each iterate of the algorithm adds a new vector to the search space and finds tighter upper and lower bounds on Good. Typically the algorithm terminates when these bounds agree to within about 10%, but here we have set the convergence tolerance to a much smaller number to illustrate that both limits do indeed converge to a common value. The four pairs of lines correspond to different random starting seeds.

and

$$z^T = \left(\sigma_1 - \rho_0, -\rho_1, -\rho_2, \ldots, -\rho_{j-1}\right). \tag{3.133}$$

It can be seen that every iteration need only extend the matrix and vector by adding new components; accordingly we need only store the values of ρ and σ from the previous iterations which takes negligible space.

As previously described, equations 3.92 and 3.115 provide strict bounds on $\mathcal{T}$. Their difference also provides a natural termination criterion; we continue to iterate until this falls below some fraction of the average. Since the training process is relatively insensitive to the value of the hyper-parameters, and hence also to Good, this fraction can be quite large, e.g. 1-10%.

3.3.3 A summary of hyper-parameter estimation

The results of the previous subsections allow us to re-estimate the regularizing constants for large networks without explicit use of the Hessian matrix A. This method is complementary to existing algorithms which directly manipulate matrices; the fractional sampling errors inherent in the random vector approach scale as $1/\sqrt{N_W}$ and a correspondingly large numbers of samples must be averaged to achieve the desired accuracy for small problems. As a rule of thumb, direct methods should be used for networks having fewer than 100–200 weights.

Algorithm 3.3 (Re-estimating the regularizing constants for large networks)

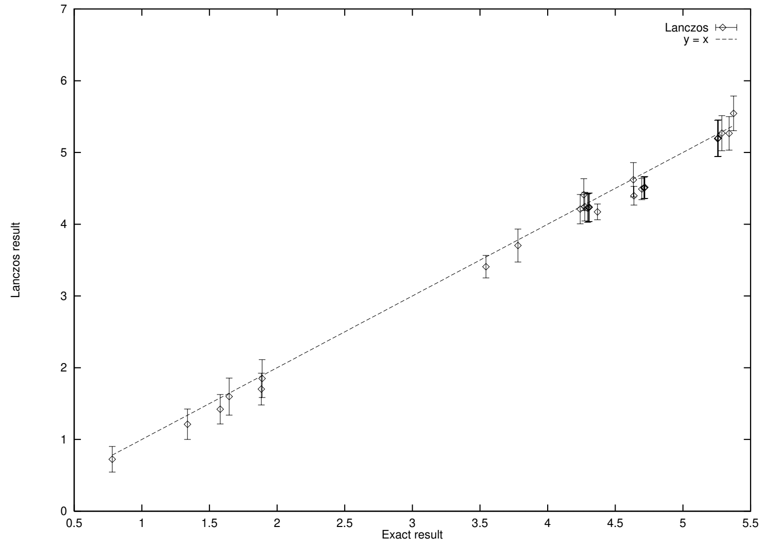


Figure 3.5: A comparison of the Lanczos and exact methods for calculating Good. The graph shows thirty evaluations of Good which took place during a network training run. The tolerance parameter in the conjugate-gradient code, ε, has been set to a negligible level to highlight the statistical fluctuations from the true traces. Each point shows the mean of 20 samples from the unitary distribution, with estimated error calculated as $\text{var}\,\mathcal{T}/\sqrt{n-1}$. The regularization classes each contained 24-25 weights.

```
for(regularization classes, i)
    {
        for(j = 0; j < M_i; j++)
            {
                r_j = random_vector();
                do
                    {
                        Calculate new G_l and G_u using Lanczos method.
                        G_j = (G_l + G_u)/2;
                    } while((G_u − G_l) > εG_j);
                }
            G = mean(G_j);
        }
```

Of course, it would be more efficient to include new random vectors until the variance of the samples drops below some fraction of $\mathcal{G}$. However we would then, in principle, average over some subset of random vectors on each pass, leading to unnecessary fluctuations in good. Accordingly we set M_i by hand at the start of the calculation and try to accept the inefficiency with good grace.

3.3.4 Calculating the Evidence

We can also use Lanczos methods to calculate the Evidence. Since

$$\log \det A = \operatorname{Tr} \log A \approx \left\langle e_0^T \log T e_0 \right\rangle, \tag{3.134}$$

it follows that to evaluate the evidence we need the logarithm of a matrix. This may be found by factorizing T with a QL-algorithm (see Wilkinson and Reinsch (1971)) into $H^T \Lambda H$ where H is an orthogonal matrix of eigenvectors and Λ is the matrix of eigenvalues. Then

$$e_0^T \log T e_0 = (He_0)^T \log \Lambda (He_0), \tag{3.135}$$

$$= \sum_i H_{i0}^2 \log \lambda_i. \tag{3.136}$$

Accordingly, to find the determinant, we need to calculate both the eigenvalues and first elements of the eigenvectors. The extremal eigenvalues of T tend very rapidly to those of A, so that calculating the determinant becomes a matter of weighting the interior eigenvalues appropriately. This weighting is the rôle of H; were Q square its orthogonality would ensure that $\sum_i H_{i0}^2 = 1$ and we could interpret them as weights on the eigenvalues. There is a substantial body of work discussing the convergence of the eigenvalues of T, the seminal reference being Kaniel (1966).

3.4 Summary

The ideas presented in the chapter enable us to apply the Bayesian machinery to large probabilistic networks. In particular they allow the use of networks where the number of weights, N, precludes storing the Hessian matrix directly. Although Lanczos theory is not new, it has not previously been applied to probabilistic networks.

A program implementing the ideas of this and previous chapters is available from the author; for more details see Appendix A. This program has been used for all the demonstrations contained in this dissertation.

Chapter 4

Probabilistic networks at work

This chapter has two aims. The first is to illustrate the ideas of the previous two chapters by applying probabilistic networks to a simple test problem. The second is to reconcile some apparently contradictory ideas about the performance of networks as the number of hidden units increases indefinitely.

4.1 The robot arm problem

The example chosen, a two joint robot arm, has been analysed before by MacKay (1991), but the second half of our analysis is novel and we explore a larger range of networks than previously. The input data consist of pairs of angles, (θ_1, θ_2), and the corresponding positions of the arm's end, (z_1, z_2). These are related by

$$z_1 = r_1 \cos\theta_1 + r_2 \cos(\theta_1 + \theta_2), \tag{4.1}$$

$$z_2 = r_1 \sin\theta_1 + r_2 \sin(\theta_1 + \theta_2). \tag{4.2}$$

Here r_1 and r_2 have been set to 2.0 and 1.3 respectively, θ_1 and θ_2 are sampled randomly from a restricted range of angles, and Gaussian noise with a standard deviation of 0.05 is added to the outputs. Six hundred samples were produced and split into equal thirds; this provides a training set and two independent test sets.

We have trained numerous networks on this problem, using the Bayesian framework to control the network hyper-parameters. The networks are not too large and so direct manipulation of the Hessian matrix is possible; we do not use the Lanczos ideas of the last chapter. We have investigated a number of networks, with hidden layers containing between six and twenty-eight hidden units, trying ten different starting weights for each network topology.

4.1.1 Training error

Figure 4.1 shows how the training error Q_D varies with the number of hidden units. Unsurprisingly this error falls as the number of units increases. To understand why, observe that the number of hidden units sets an upper bound on the complexity of the network; as the number of hidden units increases, the network is able to implement increasingly complicated maps between inputs and outputs. The robot-arm map is of different functional form from that provided by the network; accordingly the network will find an approximation to the true map. As the number of hidden units increases, the possible maps fill a larger volume of `map-space' and the distance between the true map and the best network map is likely to fall, reducing the training error.

4.2 Good parameters

One of the useful insights afforded by the Bayesian viewpoint is that the number of weights in a network is often less important than Good, the number of weights that are well determined by the data. This is illustrated by figure 4.2 which shows how Good changes as the size of the network increases. If the network is small then an increase in the number of available weights is often accompanied by a similar increase in the number of well determined weights. As the network grows, however, Good increases at a much slower rate. In a loose sense many of the new weights duplicate behaviour already present in the network; if we group these weights together the data may be informative only about the group as a whole and not individual weights in the group. Additionally there is also only a finite amount of information contained in the training set; at some point the data must run out of things to tell us.

In equation 3.41 Good is related to the eigenvalues of the Hessian matrix of the weights and the hyper-parameters, σ_W. If there is only a single regularization class we have, from equation 3.43, the expression for Good

$$\mathcal{G} = \sum_i \frac{\beta_i}{1/\sigma^2 + \beta_i} \tag{4.3}$$

where β_i are the eigenvalues of the likelihood's curvature. When there are few hidden units the weights are often well constrained: $1/\beta_i \ll \sigma^2$. Accordingly each term in the sum is close to unity and $\mathcal{G} = N_W$. Conversely, if the network has become large, it is likely that the information in the data used to constrain the weights has already been used up, and accordingly $\beta_i \to 0$. Thus the sum tends to a constant limit.

4.3 The Evidence

In previous chapters we have placed great importance on the utility of the Evidence as an objective measure of model fitness. Since we can generate almost arbitrarily large numbers of different models within the probabilistic network framework, such a measure is obviously useful. We have evaluated the Evidence using a Gaussian approximation for each of the training runs and plotted the results in figure 4.3.

Two features are evident. If we consider only those networks with fewer than about fifteen hidden units, there is a definite peak in the evidence at around eight hidden units. Accordingly we conclude that if the network has fewer than eight hidden units then it lacks the complexity to model our network, whilst networks with more than eight hidden units are penalized for their complexity by an Occam factor.

The other obvious feature on the graph is a steady rise in the Evidence beyond fifteen hidden units. We do *not* consider these models to be more probable, but interpret the

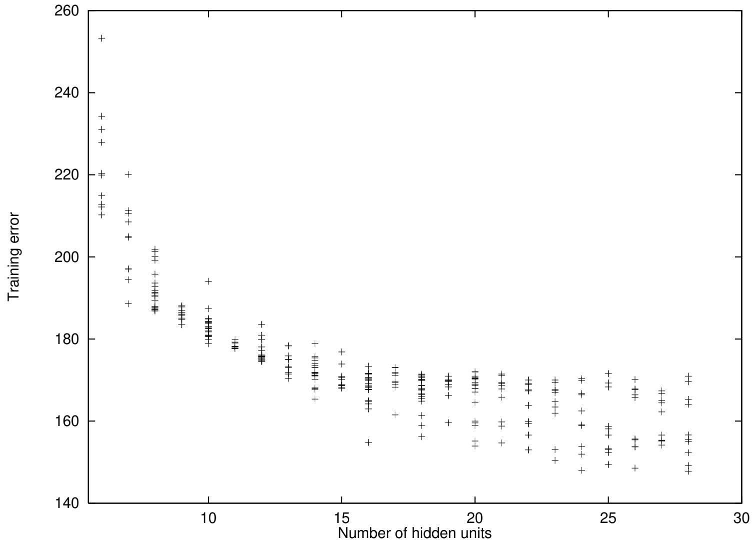


Figure 4.1: The graph shows how the training error of the network is related to the number of hidden units in the network. As this number increases, the network becomes more complex and accordingly can synthesise a better approximation to the true behaviour. We have trained each network ten times, starting from a different set of random weights.

result instead as a breakdown of the Gaussian approximation. As the number of weights increases beyond a certain limit much of the distribution will be unconstrained by the data; and our calculation of the Evidence, which makes a Gaussian approximation about the most probable value and then integrates over the entire weight space, is likely to be poor.

The Gaussian approximation is probably still serviceable for hyper-parameter estimation; certainly figure 4.2, which plots Good as a function of the number of hidden units, still seems well behaved when the number of hidden units exceeds fifteen. However this calculation relies upon the trace and not the determinant of the matrix, and is accordingly less sensitive to the quality of the approximation.

4.3.1 How well does the Evidence predict test error ?

To be useful as a measure of network quality the Evidence should be strongly related to some established measure of the fitness of a network. The test error has often been used for this purpose. This is simply the error between the predictions and targets for an independent set of data drawn from the same sampling distribution that produced the training set. Figure 4.5 shows the correlation between evidence and test error for the two independent test sets and each of the trained networks. We have split the results into two halves, the first containing only those networks with fourteen or more hidden units, the second containing the smaller networks. This division allows us to ignore networks in the former class for which we believe our evaluation of the Evidence is poor, and concentrate instead on the latter class. We expect that the second plot will show a strong correlation between Evidence and test error. This prediction is confirmed: there is indeed a good correlation when the network is small, which breaks down as the network grows and the

Gaussian approximation fails.

4.4 A paradox

So far we have presented a coherent picture of modelling with probabilistic networks; it is time now to consider an apparent inconsistency in the approach.

4.4.1 The penalty for over-complexity

It has long been held that models with too many hidden units have poor generalization properties. Whilst it is true that the training error continues to fall as the number of hidden units increases, the test error is seen to rise. The robot-arm example behaves no differently in this respect. Figure 4.6 shows the test errors for networks with between six and twenty-eight hidden units; there is a definite trend towards larger test errors as the number of hidden units increases. Consequently the training score is not a good predictor of a network's ability to generalize; figure 4.7 shows how the test and training errors are related.

Conventional wisdom accordingly holds that the number of hidden units is an important parameter of the model, which we should optimize to give the best Evidence or test error. Extrapolating the graph of test error against hidden units suggests that it would be foolish to contemplate an infinite number of hidden units; even if we could compute with such a network the test error appears to diverge. So much for convention!

Recently Neal (1994) has shown that if an appropriate prior is placed on the hyper-parameters, an infinite number of hidden units *is* a sensible limit to take. The idea is that, if the number of hidden units is large, it is the hyper-parameters, rather than the network topology, which control the complexity of the network; it may be helpful to recall that in the discussion above we spoke of the number of hidden units placing an upper bound on the complexity.

Formally, Neal shows that for networks whose weights are drawn from a Gaussian distribution, and if the hyper-parameter for the output layer σ_{out} scales as

$$\sigma_{out} = \frac{1}{\sqrt{N_H}}, \tag{4.4}$$

where N_H is the number of hidden units, then the function produced by the network will tend to a Gaussian process as the number of hidden units tends to infinity.

If the infinite limit is well defined it seems intuitively unlikely that some finite number of hidden units will be a better model. In a Bayesian sense the complexity of the model tends to a fixed limit as the hyper-parameters constrain the network's behaviour, but the likelihood increases as the model is better able to mimic the true map. Of course there are exceptions to the rule; if we use a standard regression network to model $z = \tanh x$ it does the job remarkably well with a single hidden unit, but such examples are unlikely to arise by chance!

Such is the problem. On one hand we have a body of conventional research which shuns the infinite limit, on the other mathematics which suggests that infinite networks are the way forward.

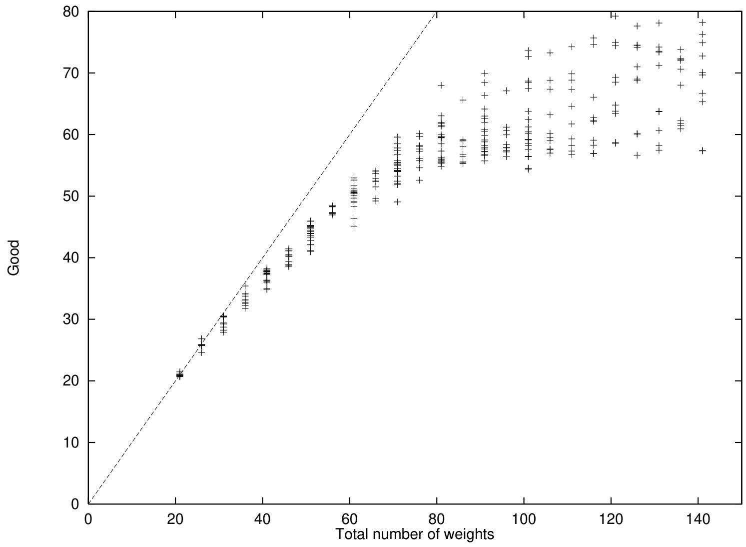


Figure 4.2: The graph shows the relation of Good, the number of well determined weights, to the total number of weights in the network. The dashed line has unit slope and shows the behaviour of a network in which all the weights are determined by the data. In real networks the data can not determine all the parameters, so that Good is always less than the number of weights.

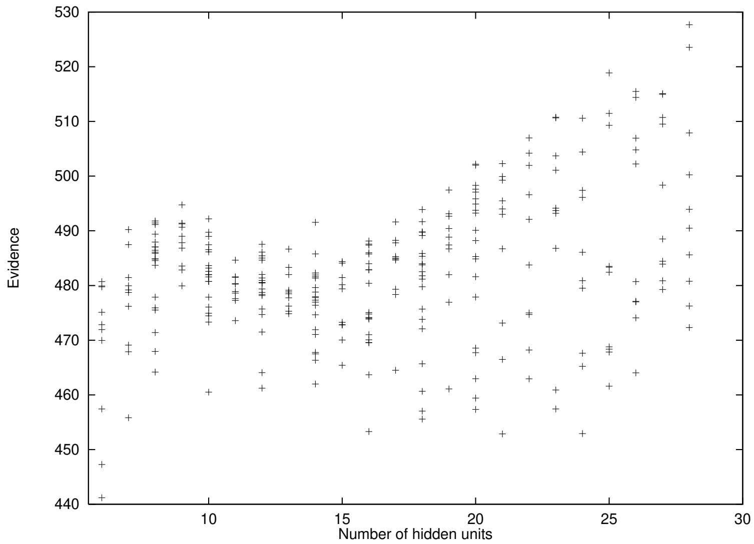


Figure 4.3: The graph shows the variation between Evidence and the number of hidden units in a network. If the number of hidden units is small the Evidence rises as the size of the network increases but, beyond a certain point (about eight units in this example), extra units incur an Occam penalty which outweighs the benefit gained from greater fidelity to the training set.

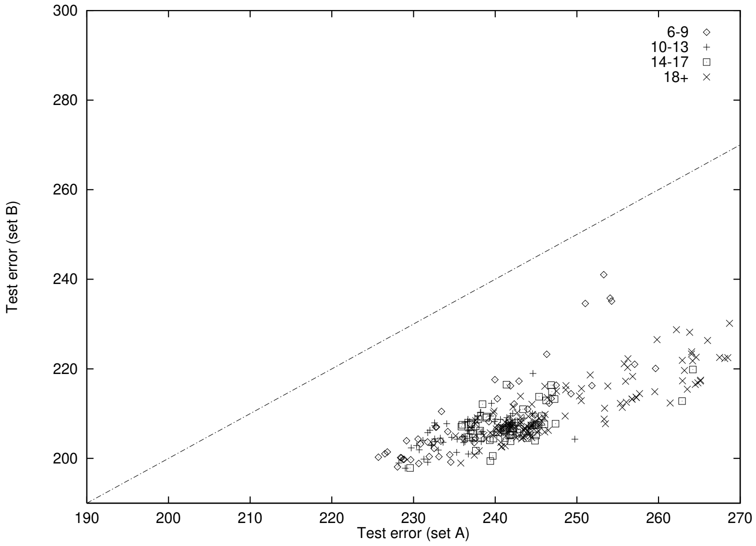
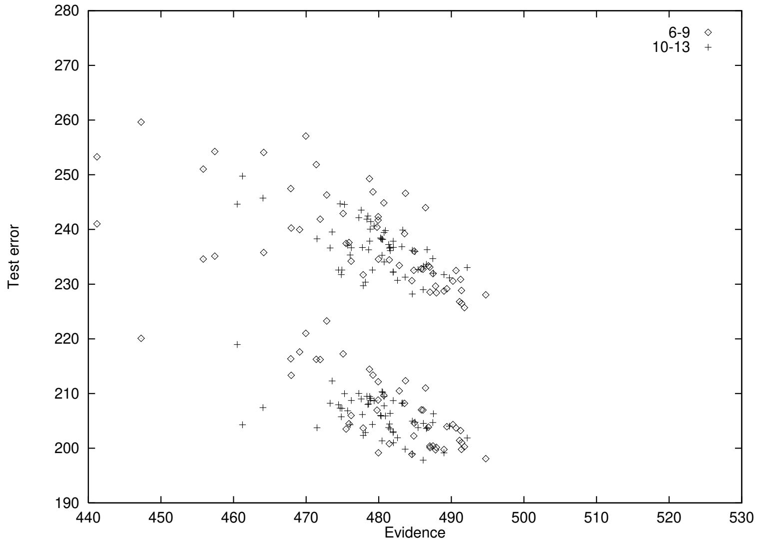
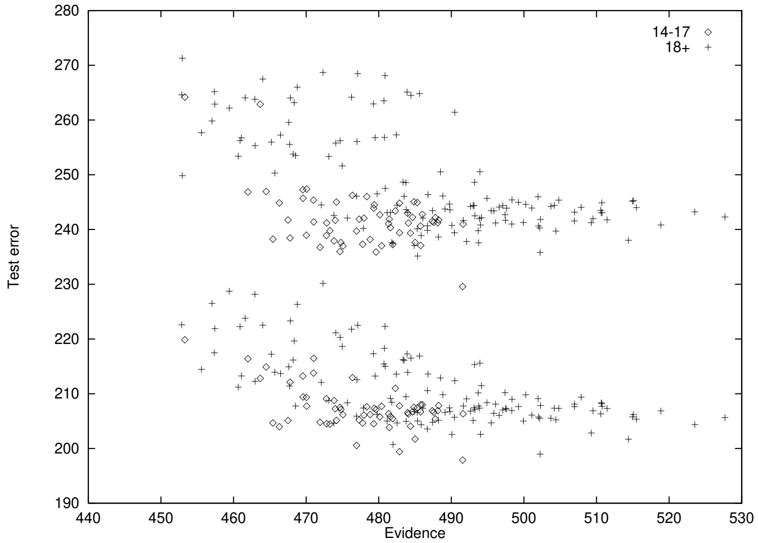


Figure 4.4: The graph compares the test error calculated on two independent test sets. Two features are obvious: the test errors are strongly correlated, and there is an apparently large difference between them. This difference is to be expected: samples drawn from this distribution have a mean of N and a standard deviation of $\sqrt{N}$, thus the difference between the test error of two different test sets has zero mean but a standard deviation of $\sqrt{2N}$. Here $N = 200$ and so the uncertainty in the test error is 20.



(a) Small networks with fewer than fourteen hidden units



(b) Larger networks with at least fourteen hidden units

Figure 4.5: The two graphs show how well the Evidence predicts the test error of a network. The top graph shows results taken from small networks, for which we believe that the Gaussian approximation is justified. The lower graph deals only with larger networks and the lack of correlation is due to the failure of the Gaussian approximation used to integrate over weight space. For a full discussion see the text.

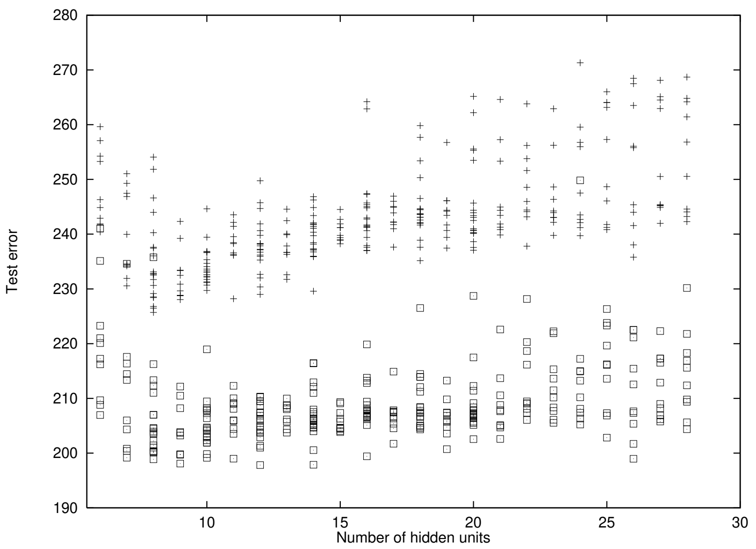


Figure 4.6: The graph shows how test error changes as the network grows. These results support the conventional wisdom that, as the number of hidden units increases, the test error first falls and then rises again.

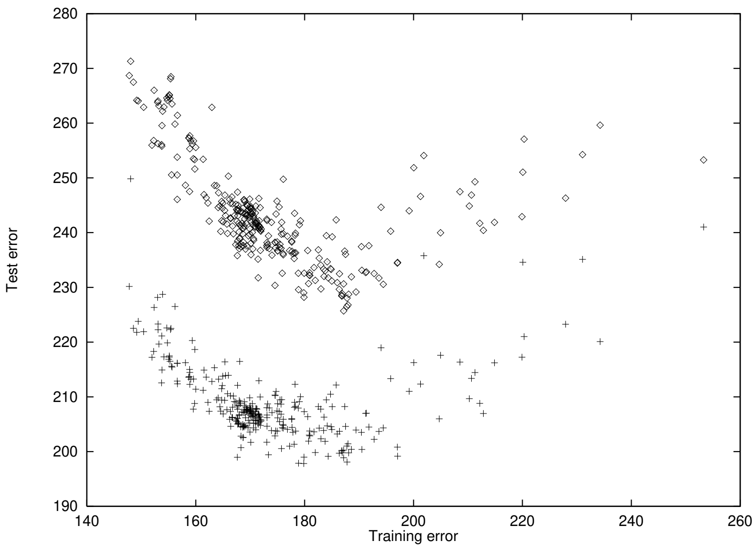


Figure 4.7: The graph illustrates the relationship between test and training errors and shows that training error is not a good predictor of test error. However networks which have a high training error have a high test error too; this is because the network lacks enough complexity to model the data accurately.

4.4.2 A different error measure

From the probabilistic standpoint, the test error can be seen as a Gaussian log-likelihood with a variance of σ^2. This is all very well, but a Gaussian approximation to the output's posterior distribution has already been calculated in equation 2.83 on page 46. Both distributions have the same form, but with different covariance matrices. Explicitly the conventional test error is

$$Q_T = \frac{1}{2\sigma^2}(z-t)^T(z-t), \tag{4.5}$$

while the log-predictive error is given by

$$-\log \mathrm{p}(z|D) = \frac{1}{2}(z-t)^T C(z-t) + \frac{1}{2}\log\det C + \frac{N_z}{2}\log(2\pi) \tag{4.6}$$

where

$$C = \left((\nabla z)^T A(\nabla z) + \sigma^2 I\right)^{-1}, \tag{4.7}$$

z and t are the vector-valued modal and target outputs, A is the Hessian matrix for the network weights, and ∇ is with respect to the weight vector. The factor of $\frac{1}{2}N_z\log(2\pi)$ is constant for a given problem and has been ignored in the following demonstrations.

The first question is, of course, how this measure compares with the conventional test error. Figure 4.8 illustrates this by plotting one against the other. Networks with only a few hidden units show a strong correlation between the two errors, but all is not lost because as the network size increases the correlation disappears. Indeed the graph suggests that the log-predictive may tend to a limit as the number of hidden units increases. To test this hypothesis, figure 4.9 plots log-predictive against the number of hidden units; it does indeed decrease monotonically to a fixed, finite limit.

4.4.3 The output distribution

These results give us some insight into the nature of the paradox; still more can be gained by looking at the outputs from different runs of the program. Figure 4.10 shows a single sample from networks with eight and twenty-eight hidden units and ten samples from a network with eighteen hidden units. Two conclusions can be drawn from these plots.

First, the uncertainty of the predictions becomes larger as the number of hidden units increases; if this were not the case then we could not explain the limiting behaviour of the log-predictive.

Second, as the number of hidden units increases, samples drawn from the distribution of outputs show greater variety in their modes. This is not surprising, since the increased complexity afforded by the hidden units must increase the number of maps between inputs and outputs available to the network. However, if we sample randomly from this distribution, there is an increasing probability that our sample will lie some distance from the mode of the overall distribution; each Gaussian sample is an imperfect approximation to the true predictive distribution. As a consequence the conventional test error for outputs drawn from the distribution of possible outputs is likely to rise as the number of

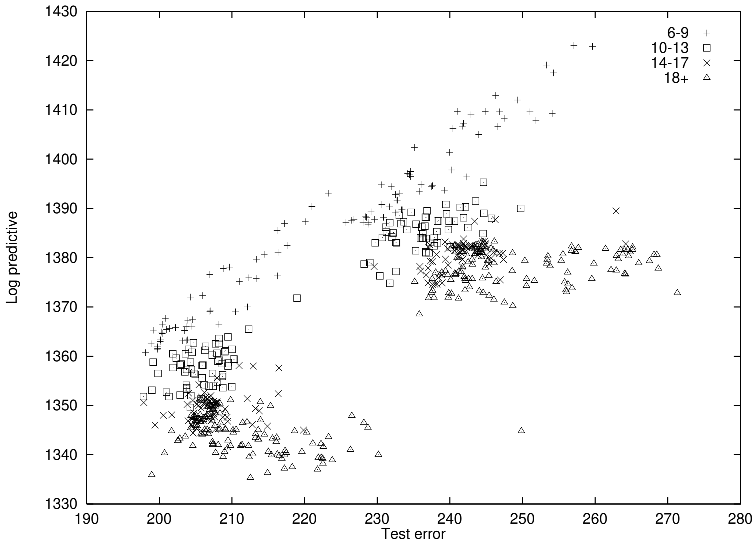


Figure 4.8: The graph illustrates the relationship between the conventional error and the log predictive density. Each marker type corresponds to a range of network sizes; if the number of hidden units is small the networks are strongly correlated, but as the number of hidden units increases the correlation disappears. Nets with a large number of hidden units appear to have a constant log predictive density; the importance of this result is discussed in the text.

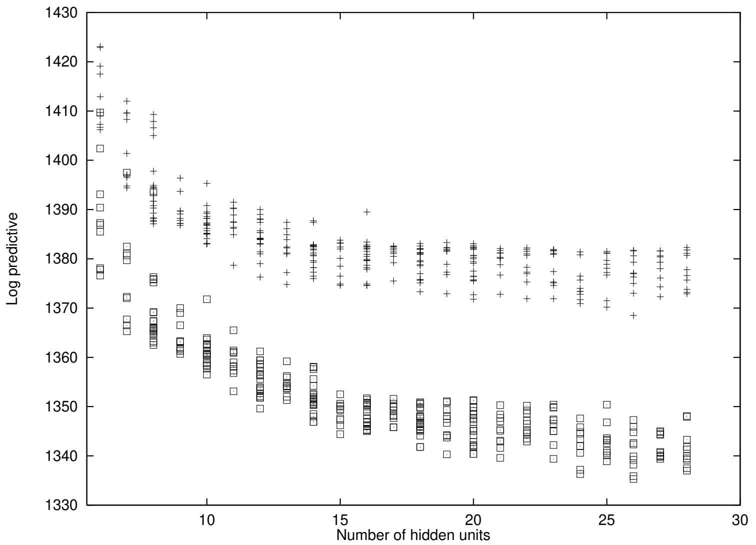


Figure 4.9: The graph illustrates the behaviour of the log-likelihood error measure as the number of hidden units increases. The monotonic decrease to a limit suggests strongly that this measure of model fitness remains well defined in the limit of an infinite number of hidden units.

hidden units increases. Mathematically, we employ equation 2.83 to write the posterior probability for the outputs as

$$p(z|D) = \sum_i p(z|D, \mathcal{H}_i) p(\mathcal{H}_i|D). \tag{4.8}$$

Thus the hypothesis outlined above asserts that random samples of $p(z|D, \mathcal{H}_i)$ are unlikely to be close to $p(z|D)$. This might be due to the inherent problem of sampling in a high-dimensional space, or it might be that *all* the $p(z|D, \mathcal{H}_i)$ are quite different from the overall distribution.

Another way to understand this is to consider all the possible maps from inputs to outputs allowed by the network's structure. Placing an appropriate prior on the hyper-parameters constricts the network behaviour to a subspace in the space of all maps; adding data makes the subspace shrink even more as maps incompatible with the data are rejected. As the number of hidden units increases it is quite reasonable that the number of maps still compatible with the data should increase (although at some level the range of maps may become limited by the prior).

4.4.4 The committee approach

If our hypothesis is correct then the test error of a committee of different samples from the distribution of weights should perform well; indeed I conjecture that this too should tend to a limit if the committee becomes infinitely large. Figure 4.11 shows the results of this experiment. These results are inconclusive, presumably because ten samples are not sufficient to accurately approximate the true posterior distribution. There is scope for further research here.

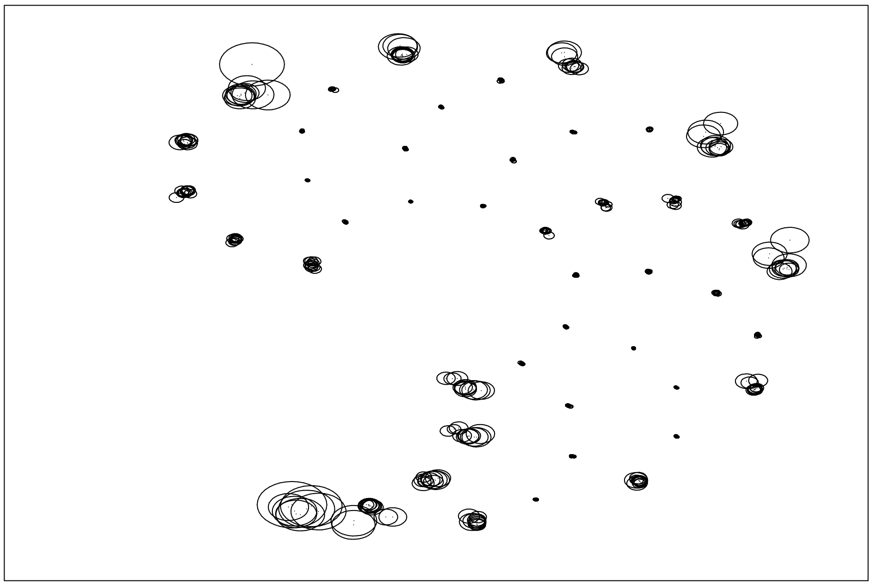
4.5 Conclusions

In the second half of this chapter I have addressed the apparent paradox between the conventional view of networks with an infinite number of hidden units which dismisses them as useless, and more recent research which suggests that, with proper hyper-parameter control, they are a useful tool.

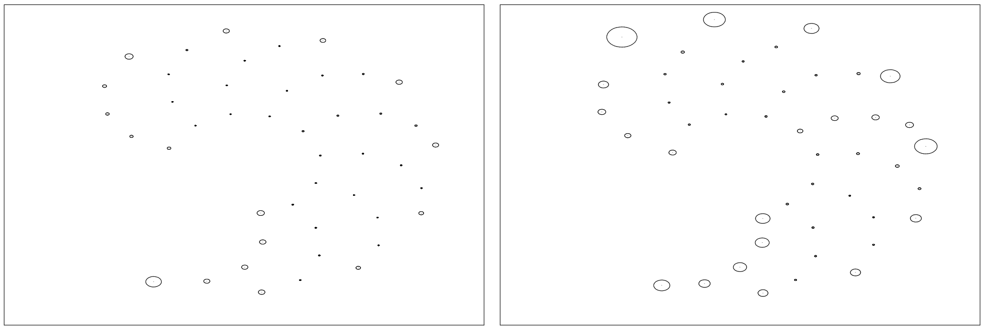
To understand this discrepancy we have scrutinized at the traditional arbiter of model quality, the test error, and concluded that it may not always be appropriate; I suggest that the log-predictive measure should be used. Finally I conjecture that the test error *will* behave sensibly as the number of hidden units grows, but only if applied to the predictions of a committee with a large number of members.

There is clearly further work to be done in this area. Many of the outstanding issues — including an investigation of the test error of large committees — will be addressed in a forthcoming publication (MacKay and Oldfield, 1995).

A final outstanding point concerns the evidence in the limit of few hidden units. In particular the peak in the evidence at eight hidden units warrants further study. It is conceivable that the Gaussian approximation has already broken down at this stage, or it might be that the robot arm map lies fortuitously close to a map expressible by an eight layer network. This question will best be answered by performing similar experiments on different problems.



(a) Ten samples from eighteen hidden units



(b) Eight hidden units

(c) Twenty-eight hidden units

Figure 4.10: Each plot shows the two-dimensional output space of the network, with ellipses indicating the 1σ region around each prediction. The top plot shows ten different samples of the posterior density over the outputs; although the modes vary they do so only by $\approx 1\sigma$. These plots are discussed at more length in the text.

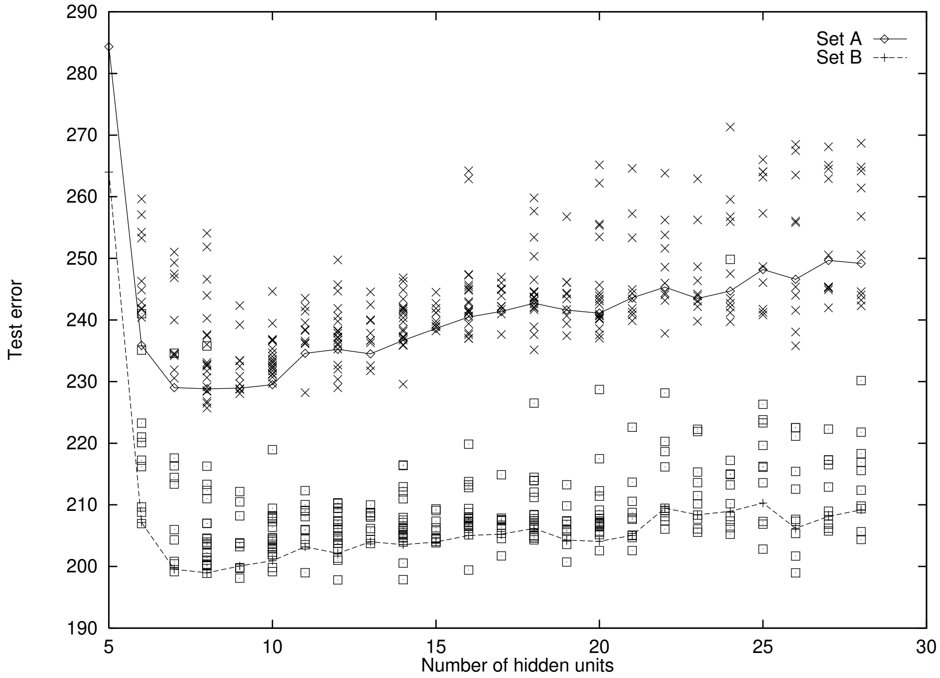


Figure 4.11: The graph illustrates the behaviour of the test error from a small committee of ten samples as the size of the network increases. Overlaid on this graph are the test errors from the individual samples which make up the committee.

Chapter 5

The deck of cards revisited

In previous chapters we have considered both traditional Bayesian analysis and more abstract approaches based around probabilistic networks. For simple problems, a full Bayesian analysis is often easy to perform, but it is easier still to throw the problem at a general purpose program. Setting aside the essential ugliness of this approach, the crucial question becomes `How well does the general-purpose program perform ?'

To answer this question we return to a slightly modified version of the playing card example from chapter 1. Rather than inferring one of thirty-six cards from multiple observations with bit-wise noise, we will use cards drawn from a six-card deck and seen only once through a fog of Gaussian noise. Figure 5.1 shows these cards; figure 5.2 shows typical images after noise has been added. If the noise level is small this is clearly an easy problem, since it is possible to discriminate the cards by eye. At the highest noise level the problem is more challenging and visual classification difficult.

In the image each pixel is stored as an eight-bit integer and, if the noise-level becomes large, there is a strong possibility that some pixels will `wrap-around' ; consequently pixels which we might have expected to be dark are very bright, and *vice-versa*.

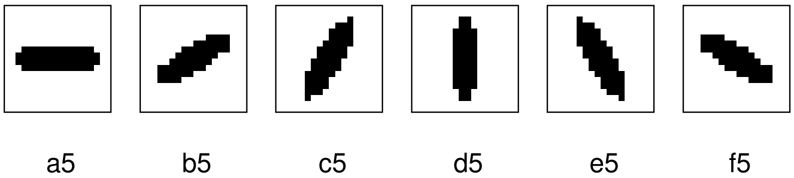
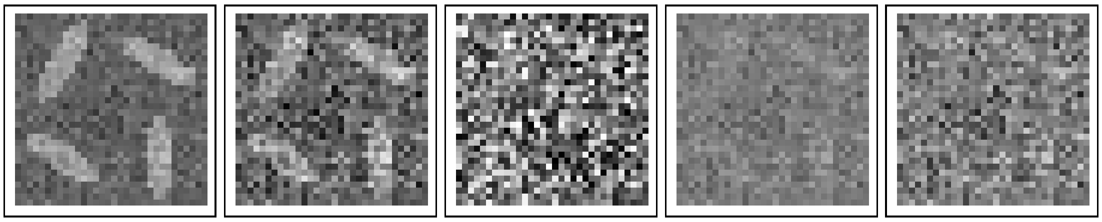


Figure 5.1: The set of six cards used to compare the performance of probabilistic networks with the exact Bayesian solution.

5.1 The exact Bayesian solution

We denote the M pixel intensities in the image by the vector D and the known, unblemished images of the j^{th} card by $\hat{D}_j$. If each pixel is subject to uncorrelated Gaussian noise with a standard deviation of σ then the likelihood is

$$\mathrm{p}(D|j) = \left(2\pi\sigma^2\right)^{-M/2} \exp -\frac{1}{2\sigma^2}(D-\hat{D}_j)^T(D-\hat{D}_j). \tag{5.1}$$



$\sigma = 16$ $\sigma = 32$ $\sigma = 64$ $\sigma = 16$ $\sigma = 32$

Figure 5.2: Typical samples of the data we will try to analyse. The samples are labelled with the noise level, σ; the leftmost three samples have an initial contrast Δ of 64 the others 16. The middle sample shows some evidence of pixel wrap-around; some dark, background, pixels have become very bright. Each panel shows a group of four different cards.

A flat prior on the cards therefore implies

$$\log \mathrm{p}(j|D) \propto -(D-\hat{D}_j)^T(D-\hat{D}_j), \tag{5.2}$$

so that, if we were forced to pick a most probable card we would pick that one closest, in a least-squares sense, to the image. We have made one approximation in deriving this result: assuming that the pixel values D are real numbers rather than range-limited integers, this assumption has two consequences.

First, our likelihood will be slightly wrong because we have ignored the discrete nature of the measurements. This is difficult to quantify accurately, but as both Δ and σ are always larger than sixteen, enough states will be used to make the effect a small one. If the data were uniformly distributed, rounding to the the nearest integer would introduce noise with a variance of $1/12$, which is the variance of a random variable distributed uniformly on $[0,1]$.

The second consequence, the `wrap-around' effect alluded to above, is potentially far more serious. Least-squares error measures are often dominated by a small number of outlying data and unless we modify the likelihood to include the possibility of wrap-around, such pixels will dominate the classification. For example, if a bright pixel is brightened too far by the noise it will suddenly become very dark and the Bayesian classifier will almost certainly match it to template in which the corresponding pixel was dark. Such a problem is not outside the Bayesian approach; we could build a more complicated model which considers all possible wrap-arounds and then marginalize the possibilities out. However this would obscure the comparison.

Applying the algorithm to 4096 sample images gives the results shown in table 5.1. Several points are worthy of note. As expected the algorithm works very well in the low noise case and manages to classify almost perfectly for $\sigma \le \Delta$. However if the noise model is not accurate, as is the case for $\sigma = 64$, the prediction fidelity falls dramatically. The important conclusion is that Bayesian models can only be as good as the assumptions put into the model!

As the problem gets harder it is interesting to see where the classification goes wrong. Table 5.2 shows the results of the Bayesian classifier in the high-noise case ($\Delta = 16$,

Noise level	Template contrast	Number correct	Fraction correct
16	64	4096	1.00
32	64	4096	1.00
64	64	3704	0.90
16	16	4094	0.9995
32	16	3607	0.88

Table 5.1: The performance of the exact Bayesian solution in a variety of different regimes; the different results between $\sigma = \Delta = 64$ and $\sigma = \Delta = 16$ can be attributed to pixel wrap-around, which has not been modelled in the likelihood function.

$\sigma = 32$). In principle there is nothing stopping us calculating the entries in this table given only the card templates and the noise level, but such a calculation is messy. Intuitively we expect that a5 is more likely to be mistaken for b5 than c5 by virtue of having more pixels in common with the former than the latter. This structure may be seen in the table; the most common errors are between adjacent cards (remembering that a5 and f5 are neighbours too).

More formally we note that this classification scheme assigns the highest posterior probability to the template nearest to the data point, in a Euclidean sense. If we introduce hyper-planes in the M dimensional data space which bisect the template points and then truncate the planes so that only the regions which form bounding boxes around each template remain, then the M-dimensional polyhedra surrounding each template correspond to images which will be assigned to each class. To calculate the expected number of wrong classifications, note that sample images are drawn from hyper-spherical Gaussian distributions centred on the templates with a radius appropriate to the noise-level. To find the expected outcome it is necessary to integrate the distribution over the each of the polyhedra. This is messy. More usefully we can calculate the distance between the templates in this space; we expect these values to be monotonically related to the results of the integrations. This has been done in table 5.3, which is a negative picture of table 5.2; points which are a small distance apart are often confused and *vice versa*.

Target card	Most probable card					
	a5	b5	c5	d5	e5	f5
a5	582	28	8	6	6	25
b5	23	590	41	8	9	11
c5	2	31	601	26	7	3
d5	5	7	26	617	30	9
e5	10	7	3	30	618	23
f5	35	13	8	12	37	599

Table 5.2: An error analysis in the high noise limit. The table shows the Bayesian classification of the test set when contaminated with Gaussian noise having a standard deviation of twice the contrast of the motif on the cards. The strongly diagonal nature of the table indicates that the classifier still works well.

Card	Card B					
A	a5	b5	c5	d5	e5	f5
a5	0	42	66	68	66	42
b5	42	0	40	58	64	60
c5	66	40	0	42	68	64
d5	68	58	42	0	42	58
e5	66	64	68	42	0	40
f5	42	60	64	58	40	0

Table 5.3: The table shows the square of the distance between cards A and B in data-space assuming a Euclidean metric i.e. $(D_A - D_B)^T(D_A - D_B)$. The distances have been scaled so that the difference between a bright and dark pixel is one unit.

5.2 The softmax solution

This is a classification problem and consequently a softmax network is appropriate. Even though the problem is a simple one, the number of weights is still quite large and we will use the Lanczos ideas introduced in chapter 3 to control the network.

The data were divided into halves; one half was used as a test set to quantify the success of the method, the other used as a training set. The data were scaled to lie on the range $[0,1]$ before being fed to the network. Now for the crucial test: how do these results compare with the exact results above ?

Noise	Template	Size of training set							
level	contrast	16	32	64	128	256	512	1024	2048
16	64	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0
32	64	0.65	0.999	1.0	1.0	1.0	1.0	1.0	1.0
64	64	0.25	0.32	0.50	0.64	0.69	0.78	0.81	0.85
16	16	0.34	0.69	0.95	0.98	0.99	0.998	0.996	0.998
32	16	0.25	0.30	0.47	0.59	0.65	0.73	0.79	0.82

Table 5.4: The performance of the softmax network solution. These should be compared with table 5.1; for a full discussion see the text.

Table 5.4 shows the performance of a probabilistic softmax network on the different classes of data as a function of the number of training examples. There are no great surprises; the accuracies all increase monotonically with more training data and are bounded above by the Bayesian result. The network performs perfectly for the two cases in which the noise-level is lower than the template contrast, but these are hardly difficult problems. As the task becomes more difficult the network performs less well; even with a reasonably large training set the network performance lies a few per-cent behind the performance of the exact solution.

To investigate how well the network performs we first return to the Bayesian solution. The crucial quantities in the Bayesian analysis are the misfit statistics

$$\chi_i^2 = (D - \hat{D}_i)^T(D - \hat{D}_i)/\sigma^2, \tag{5.3}$$

which can be expanded to give

$$\chi_i^2 = (D^T D - 2\hat{D}_i^T D + \hat{D}_i^T \hat{D}_i)/\sigma^2. \tag{5.4}$$

The first term does not depend upon the template and so cancels in the posterior probabilities. Thus

$$\mathrm{p}(j|D) = \exp(\hat{D}_j^T D + \hat{D}_j^T \hat{D}_j/2) / \sum_i \exp(\hat{D}_i^T D + \hat{D}_i^T \hat{D}_i/2). \tag{5.5}$$

The key point here is that this is the mapping used by the softmax network; the first term represents weights coming from the image, the second represents any bias terms. Accordingly we might expect the weights to look like the templates and the biases to be roughly constant. Figure 5.3 shows the weights for a network trained on data in the low noise regime. While it is true that those pixels set in the template correspond to large positive weights, there is more structure which demands to be explained.

The preceding argument considered only the likelihood; to go further we must also consider the prior on the weights. The prior tries to limit the size of the weights, so that they tend towards zero unless the likelihood pulls them up. Consequently we expect weights associated with pixels which do not contribute to the classification process to be small; equivalently we note that if we expand the dot-products in the likelihood distribution, these pixels correspond to common terms and may be cancelled.

The final piece of insight we require can be found in the noise-free limit. In the absence of noise we could classify the images by setting all but six of the weights to zero. This is a degenerate choice; we need to pick one pixel from each template which is set in that template but none of the others. Then set all the weights to zero save those belonging to the six pixel/template pairs which should be set to some positive value.

Of course as soon as we introduce noise the situation becomes more complicated. By considering more pixels we gain some degree of noise-immunity at the expense of a less favourable prior factor. This is enough to understand, broadly, the weights produced by the training process. Pixels that are uniquely in the relevant template give rise to large positive weights, those in pixels shared with other shapes are positive but less so, and those pixels which are set not in our template but in other templates close to our own have a small negative weight.

5.3 A different formulation

Rather than inferring a particular card, we could reformulate the problem to infer the inclination of the ellipse on the card, or some parameterisation of the inclination. In other words we will try to infer a single scalar parameter, and not the posterior probabilities of each card.

Since the mapping between input and outputs is now rather more complicated, we introduce a layer of hidden units between inputs and outputs. The inputs are as before, the training target set are integers between 0 and 5 corresponding to the cards a5 to f5. A regression net is suitable for this task, except that we do not know the noise level. However we need only introduce another hyper-parameter and control this with the Evidence framework. A full treatment is given by MacKay (1991). There is one further practical problem: a network with multiple hidden units and over two hundred inputs rapidly

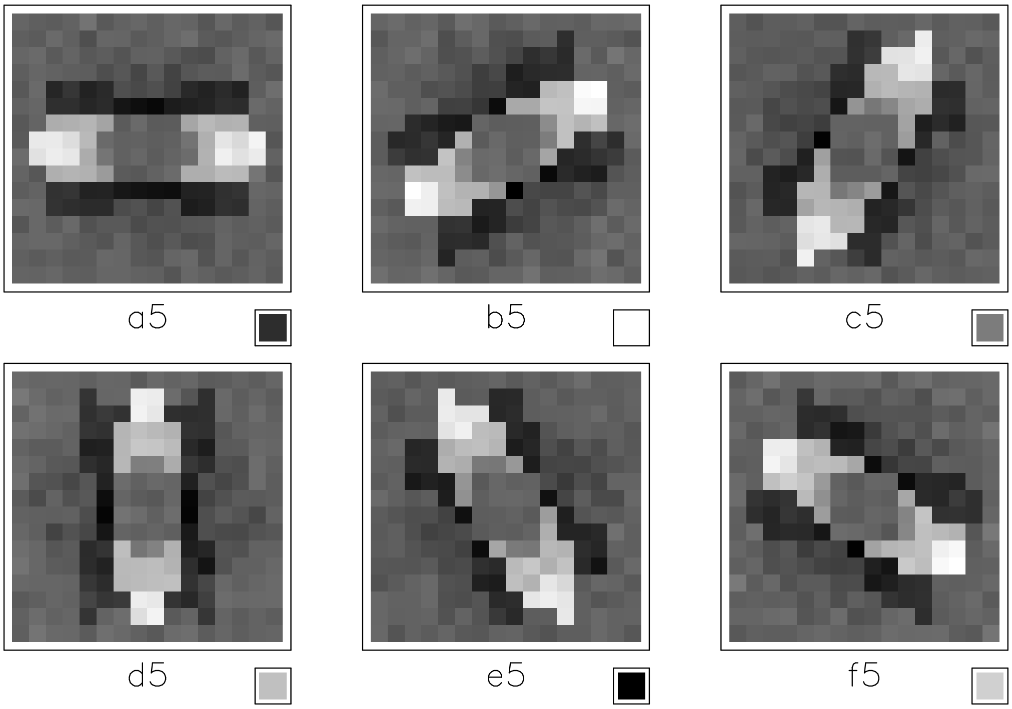


Figure 5.3: A typical set of weights from the trained network. The weights are strongly correlated with the templates, but are smaller for pixels in the centre region which are common to every template, and strongest for those pixels uniquely set in each template. Inputs weights and biases are shown on different scales; the inputs weights are of order unity, the biases of order 10^{-5}.

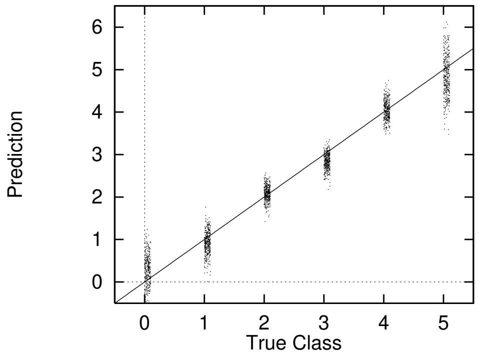
acquires a large number of weights. Accordingly we have reduced the size of the input image to a 4×4 array. This inevitably makes the problem harder but in the low-noise regime that we have investigated here ($\sigma = 16, \Delta = 64$) both the exact solution and the softmax network were able to classify perfectly.

Clearly this approach is unlikely to predict only those precise inclinations printed on the cards; rather we will probably see a range of inclinations around the true value. If the method is to be useful this spread must be less than the spacing between inclinations. To assess our performance will we take the network's prediction to be the inclination nearest to the scalar output.

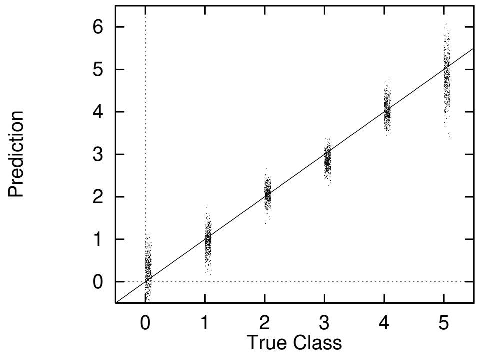
Figure 5.4 shows the results of training a variety of different networks with different amounts of training data. The effect of network committees, i.e. averaging the predictions of multiple trained networks, is also shown.

Hidden units	Training examples	Reading accuracy (different starting weights)	Committee accuracy	Test error
4	256	0.89 0.89 0.88 0.89 0.88 0.88 0.88 0.90 0.89 0.88	0.886	133.110
6	256	0.89 0.90 0.89 0.89 0.89 0.89 0.90 0.89 0.88 0.88	0.889	124.141
8	256	0.89 0.89 0.89 0.90 0.89 0.89 0.89 0.89 0.89 0.89	0.893	120.510
4	512	0.95 0.92 0.90 0.92 0.92 0.92 0.92 0.92 0.92 0.92	0.943	78.247
6	512	0.92 0.92 0.92 0.92 0.92 0.92 0.92 0.92 0.92 0.96	0.932	89.593
8	512	0.95 0.96 0.91 0.92 0.95 0.92 0.95 0.92 0.92 0.95	0.957	61.985
4	1024	0.98 0.96 0.96 0.98 0.97 0.98 0.96 0.96 0.97 0.97	0.979	46.220
6	1024	0.97 0.97 0.96 0.97 0.96 0.98 0.96 0.97 0.97 0.97	0.975	49.260
8	1024	0.98 0.98 0.94 0.97 0.97 0.97 0.96 0.96 0.96 0.97	0.978	46.332
4	2048	0.98 0.98 0.99 0.99 0.98 0.98 0.98 0.98 0.99 0.98	0.988	25.465
6	2048	0.98 0.98 0.98 0.98 0.99 0.98 0.99 0.99 0.98 0.99	0.988	25.795
8	2048	0.99 0.98 0.99 0.99 0.99 0.98 0.98 0.98 0.99 0.99	0.989	23.900

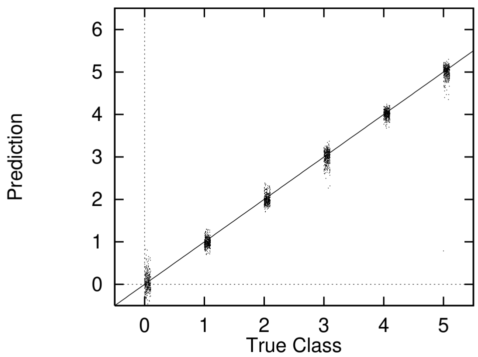
Table 5.5: The table shows the accuracy of classifications made by the regression network. Each training set and hidden layer size combination was trained ten times starting from a different random starting position. The fractional number of correct predictions made by each of these solutions is shown, as is the accuracy of the committee's prediction. See the text for a full discussion.



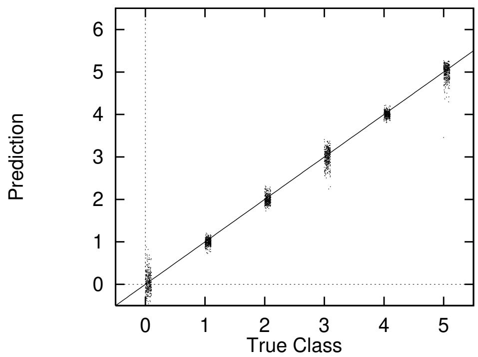
(a) Four hidden units, 256 examples



(b) Eight hidden units, 256 examples



(c) Eight hidden units, 2048 examples



(d) Committee

Figure 5.4: This figure shows the predictions of various regression networks applied to the card classification problem. Each graph is a scatter plot of the networks prediction (plotted vertically) against the true value (plotted horizontally). Horizontal scatter has been introduced to indicate the density of points in each clump. The committee results in (d) come from a committee of ten networks each with eight hidden units and trained on a set of 2048 examples.
.

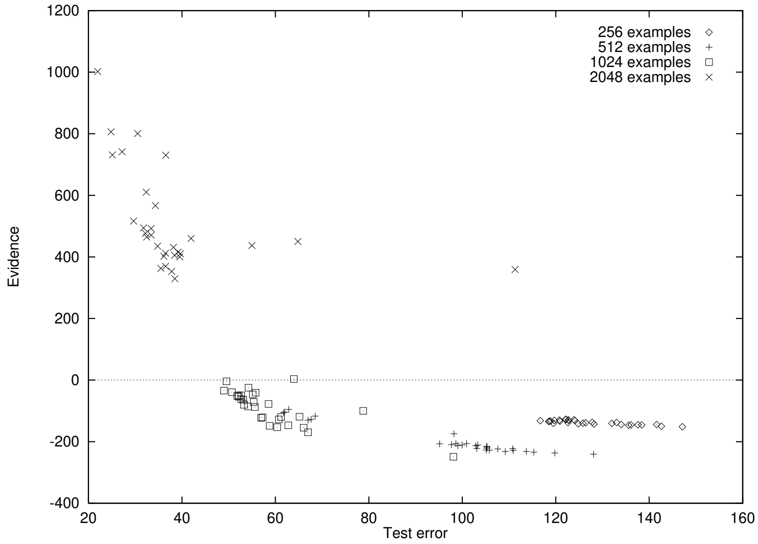


Figure 5.5: The graph shows the test error and Evidence statistics for a number of different regression networks. The variations are due to both different training sets and different starting seeds. With the exception of a few outlying data points when the training set is large, the Evidence and test errors are strongly correlated; in other words the Evidence is a good predictor of test error.

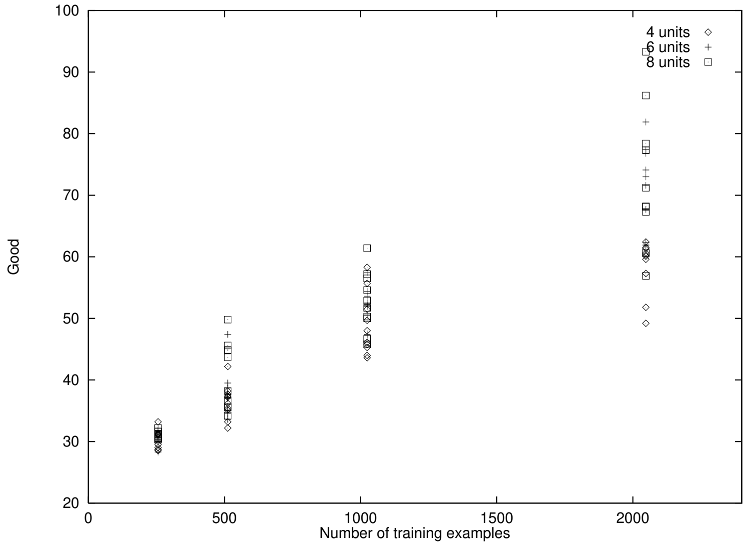


Figure 5.6: The graph shows the variation of Good, the number of well determined weights, as the number of training examples increases. For small training sets, the extra capacity of the larger nets is not used; instead the prior determines most of the weights.

5.4 Conclusions

In this chapter we have considered a simple toy problem and compared the performance of the exact Bayesian solution to that of probabilistic networks trained on the problem. Despite a common foundation in probability theory, the two approaches differ; in terms of the discussion at the beginning of chapter two, the former is a traditional model-based approach whilst the latter is a non-specific solution. Accordingly there will be situations in which we have little choice but take the latter route—if that is the case then we hope that this chapter will help us intuit the likely performance.

We have been rather unfair on the networks when making this comparison. Our soft-max network had no hidden layers, which could have been used to infer something about wrap-around. Similarly the regression network is not ideally suited to classifying images. However if we consider only complicated models we gain no insight into their behaviour; similarly we will sometimes be forced to use a network which we know to be less than optimal.

It is quite clear that we could improve upon both approaches. For example, the Bayesian model, which currently relies upon a knowledge of both the templates and the noise level, could be recast to infer both from the data. Inferring the latter is easy; even if we did not know the templates it would soon become clear that the distribution of intensities of certain pixels was uni-modal. These pixels can then be unambiguously classified as foreground or background; we can then infer both the foreground and background intensities, and the noise level. A Bayesian approach would do this implicitly but the conclusions are still valid. The exact calculation also assumes a particular noise model; if the noise model is valid then this is a bonus, if not then it counts against us. I speculate that the non-specific model could perform better than the exact solution if the data were contaminated by certain kinds of noise.

To return to our original question: `How do the exact and network solutions perform ?' then there are two important conclusions. Firstly we see that the best method for this problem is the exact calculation. Calvin would no doubt be happy to find that the extra effort above the network method that is required to set up the problem, solve the equations and write the computer programs brings with it an attendant improvement in overall performance and efficiency of computation.

The second conclusion is less clear cut. Although a `hand-crafted' solution gives the best performance, very good results can be obtained by using a non-specific model; in this case a simple probabilistic network. Our second conclusion is that no model is truly general purpose. Probabilistic networks are not panaceas; simply throwing data at a network chosen at random will not, in general, give good results. Rather it is still important to model the problem properly and choose an appropriate likelihood function of the network and also sensible representations for both training and target data; a non-specific model is one which is applicable to a *range* of problems and not *all* problems.

Chapter 6

Reading number plates

> The Stranger: I will tell, since you on your part are prepared to listen. We know that children, when they are just getting some knowledge of the letters—
>
> Young Socrates: Well ?
>
> The Stranger: Recognize the several letters well enough in the short and easy syllables, and can make correct statements against them.
>
> *The Statesman,* Plato.

As a practical application of the theory expounded in the previous chapters, we treat the automatic reading of number plates. As the quote above shows, reading has been subject to academic study for thousands of years but Plato is not known to have pondered this particular application.

Reading number plates is clearly a real-world problem and a successful solution could be applied to such tasks as traffic-flow monitoring and vehicle entry control. There is some commercial interest in the former, particularly for monitoring the flow of traffic in urban areas where pointwise velocity measurements are uninformative about large-scale congestion.

A system called `VIA` has been developed in Cambridge by Non-Cooperative Target Recognition Limited (NCTR) to read vehicle registrations. This chapter documents research independently undertaken to apply probabilistic networks to the reading problem. As a prelude to the discussion we outline the experimental arrangement of `VIA`. This both sets the scene and gives some insight into the problem at hand.

A typical `VIA` installation consists of a personal computer equipped with an image digitizer and a video camera. Such equipment is readily available commercially although care must be taken over the choice of camera. In particular three features must be considered: the exposure time, which must be short enough to prevent motion blur, the sensitivity of the camera, and the aperture of the lens. Most CCD cameras fix the exposure time and then regulate the intensity of the image by controlling the iris and the gain of the amplifiers inside the camera, although a few now dynamically adjust the exposure time too. Many of the algorithms for doing this are useless in this application because they consider the intensity of the scene as a whole, and this is only weakly correlated with the intensity of a highly retro-reflective number plate. This makes the optical properties of number plates differ from most other things in the scene. Spot-metering, which considers only a portion of the scene, also fails because the spot is static; to be useful it would have to move with the car. In practice, reliable exposure control is possible only by delegating the task to the computer running `VIA`.

Current computing hardware is capable of analysing every frame from the camera, which samples the scene every 40ms. In that time the vehicle, if it is travelling at 60 m.p.h., will have travelled about 1m. Typically the distance between the camera and vehicle is about 30m, which allows a reasonably inexpensive lens to survey an entire carriageway, and so the uncertainty in position leads to a variation in character size of roughly 3%. In fact this is an underestimate; a combination of missed triggers and timing problems in the computer program conspire against us and a variation of 10% is a better estimate.

6.1 Statement of the problem

6.1.1 The characters

In the United Kingdom the optical and mechanical properties of number plates must conform to British Standard BSAU145A (1980) and the lettering on the plates to HMGA (1971). The registration mark written on the plate is structured and typically consists of a year affix, index mark and index number. For example, given the registration `A123BCD` the `A` indicates that the vehicle was registered between August 1983 and July 1984, and the `CD` that the vehicle's first registered keeper had an address in Sussex. The remaining characters give more precise information concerning the time of registration but it is impossible to interpret this without reference to Glass's guide (1995). This document is published annually and lists the serial letters issued by the registration offices.

Registration marks are unlikely to contain the letters `Q` or `Z`. They are not entirely unknown; `Q` has been used since 1983 as a year prefix when the age of the vehicle is unknown and `Z` has been used since 1903 for registration marks issued in Ireland. However such characters remain sufficiently scarce that, to a first approximation, we can forget them. Furthermore, we are unable to distinguish between `I` and `1`, and `O` and `0`, so we amalgamate these characters in the data set and rely on the syntactical post-processing to distinguish them. This leaves thirty-two character classes; a formal statement of the problem is to calculate the probability that a character belongs to each of these classes, given an image of the character.

No regulation can tell us the size of the characters expressed as a number of pixels as this depends upon the camera geometry, the choice of lens and the frame-grabber design. In a typical installation each character will be about fourteen pixels square and located within a twenty square array of pixels. This size represents a compromise between resolution and coverage. Whatever the size, each pixel is digitized to an eight-bit integer value, although the least significant bits of this value are often contaminated with noise or simply set to zero by the image grabbing hardware. An example of a character taken from an image is shown in figure 6.1.

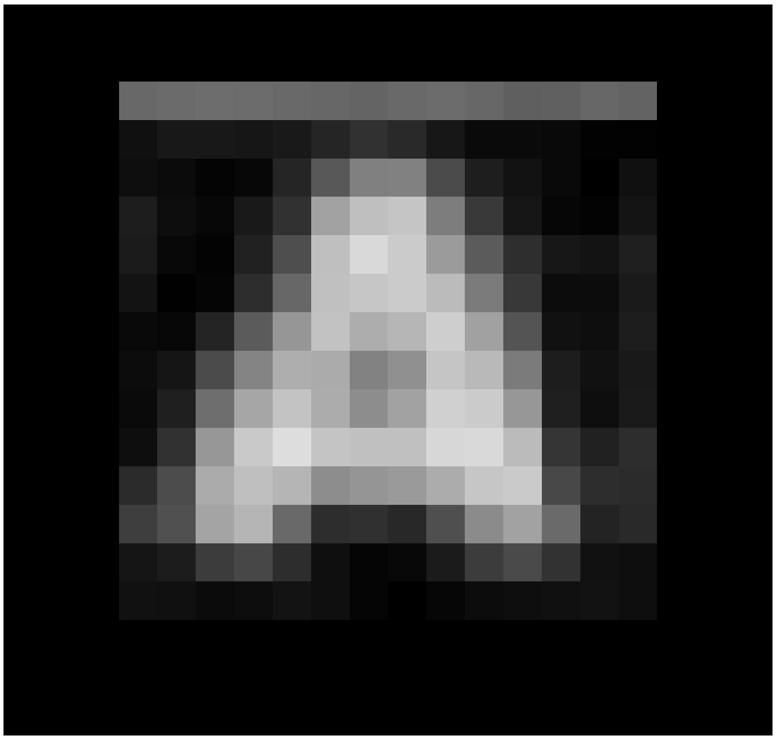


Figure 6.1: A typical character from `VIA`. Both electronic and optical effects have conspired to make this `A` a less than perfect image of the character; as is often the case it looks rather better held at arm's length.

6.1.2 Getting the characters

In essence, `VIA` can be divided into a number of functions: capturing an image of a vehicle, locating the number plate on an image, segmenting the plate into individual characters, reading those characters and finally reassembling the characters into a valid registration mark.

At its simplest level this final stage is nothing more than syntax checking; if the most probable answer isn't a valid registration mark we reject it and consider the next most probable one. However this is simple Boolean calculus and can be seen as a simple case of a better algorithm built upon probability theory.

If we are observing a road in Cambridge we are far more likely to see a vehicle from Peterborough than Preston, or from Ipswich than Inverness. Consequently registrations should not simply be classed as valid or invalid but should instead be assigned a prior. If the mark would have been rejected by a syntax-check, this approach should assign zero prior probability and reject it; in other cases this prior information will improve our reading accuracy. For example, the index mark `EC` indicates that the vehicle was registered in Preston whilst the similar mark `EG` indicates Peterborough. If the character reading part of `VIA` considered both choices equally likely and we were forced to choose just one mark, we should clearly choose the latter if we were observing in Cambridgeshire and the former if in Cumbria. We will not develop these ideas here, but concentrate instead on reading individual characters.

The data analysed in this chapter consist of nearly five thousand characters extracted from images of number plates observed by the `VIA` system on the A14 in Cambridgeshire during the summer of 1994. A full *ab initio* Bayesian calculation would start with a detailed model of the image collection process and calculate the likelihood of the data but for this problem that task seems intractable. Accordingly we content ourselves with the vague notion that the data have been `damaged' during the measuring process and let the network ponder the precise nature of the vandalism.

6.2 Simple data preparation

The characters provided by the first stages of `VIA` suffer from defects which must be addressed before we introduce them to the network. This is against the hard-line Bayesian attitude which abhors modifying data in any way and favours improving the model to fit the data better, but that viewpoint is usually as an antidote to *ad hoc* processing which destroys our model of uncertainties in the data.

We can identify two obvious differences between examples of a given letter: the intensity of the letter varies, and the different examples are not perfectly aligned with each other. Tackling the former issue is relatively simple: we will normalize the values in each character so as to stretch the range of intensities to some fixed range, in this case $[0,1]$.

The alignment problem is slightly more complicated. A plausible solution is to calculate the centroid of each character and then to move the character so that the centroid always falls in the same place; since the character size is somewhat smaller than the array selected by the preceding segmentation stage, we can take this point to be the middle of the image without losing any of the character. None of the characters in this particular set is larger than 20×19 pixels, and this size was also chosen for the aligned characters. Figure 6.2 shows how this algorithm performs on a selection of letters taken from our sample.

We don't have a theoretical noise model for our data and our pre-processing is mild; accordingly none but the most zealous of Bayesians will object too strongly to this method.

The canon of simplicity suggests that we start with a single-layer softmax network devoid of any hidden units. The network has 380 inputs corresponding to the 20×19 normalized pixel values and 32 outputs corresponding to the valid characters. The network thus has 12,192 weights. To establish a performance yardstick we will not be concerned with alternative regularization schemes or hyper-parameter adjustment, and we accordingly set $\sigma_W = 1$.

This suffices to try reading some characters. We have halved the initial observations, and thus have about two and half thousand training examples, and a similar number of independent test examples, with which to assess our performance. In the interests of efficiency, we have trained the net with different proportions of the set described above: to be precise we have divided the training set into fifths and trained on one-fifth, two-fifths and so on.

Finally we would like to introduce some creativity into the discussion of read rates. To do this we introduce a distinction between getting the answer wrong when the network is sure of the result, and merely picking the wrong member of a set of relatively implausible possibilities. Essentially we would like to separate the test data set into those examples

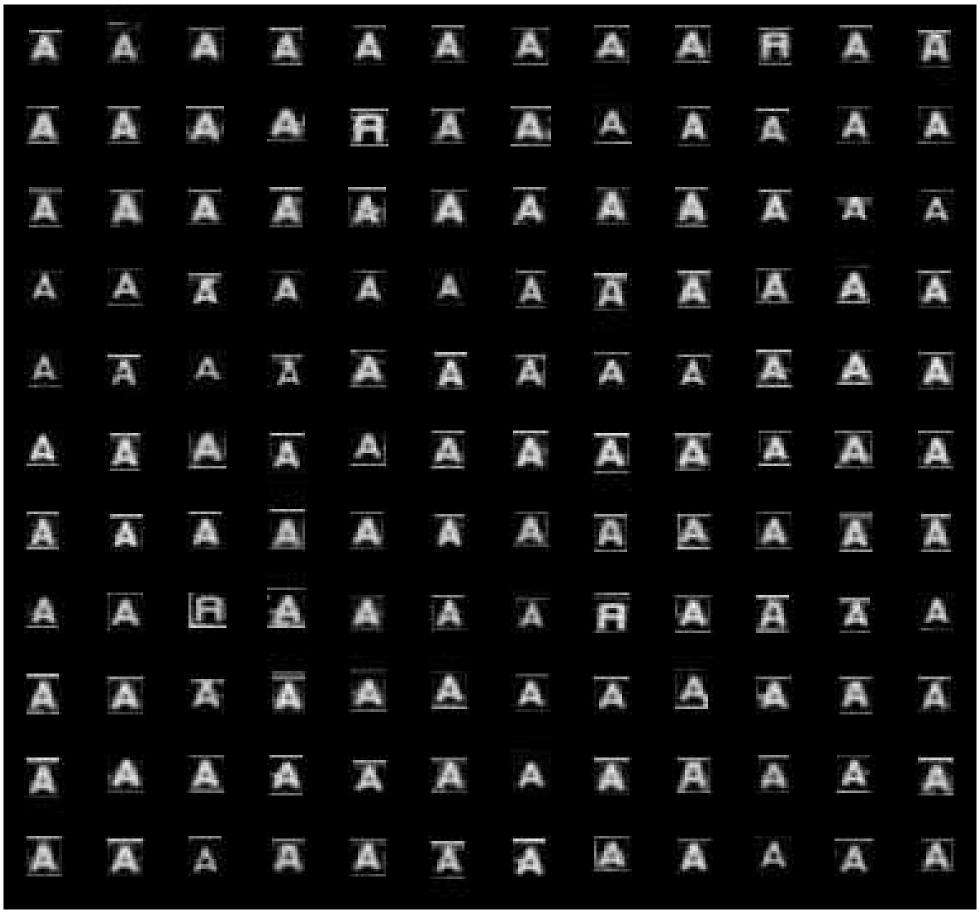


Figure 6.2: Although each of these characters is an A, they differ quite significantly. Each letter has been aligned so that its centroid lies on a rectangular lattice point; when fed to a network this scheme is also used, so that the misalignment of characters in this figure shows the variation that the network is required to handle.

where the posterior distribution is well represented by a Kronecker delta, and those where the probability is more widely spread. These correspond to type 1 and type 2 errors in traditional statistical theory.

Examples in the first class are likely to dominate any prior information and so these errors will lead to an error in the overall mark. Conversely posteriors in the second class may easily be over-ruled by the prior; given the choice between the index marks EG (Peterborough) and EJ (Haverfordwest) when observing in Cambridge, the prior prefers the former by a factor of order one thousand.

To distinguish between these classes we use a simple ad-hoc method; choosen pragmatically because the Hessian is too large to store directly and the *ab initio* calculation of $\nabla\nabla Qx$ takes too long for a real-time system. We look at the probability of the most probable output; if this is above some threshold (taken here as 50%) then we assume that the net is confident in its prediction and include the test example in the rightmost column of the table; it seems appropriate to call this `creative accounting'. Unsurprisingly, as we increase the size of the training set the network becomes more confident in its predictions and the number of examples in this class increases.

Table 6.1 shows the results of training such a network with these data. The results contain little of note; as the number of training examples increases, the test error falls and the proportion of correct reads increases to around 90%. The results show that there is no evidence for multiple maxima of the objective function; despite training the network many times, all the solutions converged on the same point.

Training performance		Test performance	
$Q(\hat{w})$	Number correct	Number correct	Creative accuracy
322.4	453 / 455	1828 / 2271 (80.5%)	1428 / 1520 (93.9%)
⋮	⋮	⋮	⋮
322.4	453 / 455	1828 / 2271 (80.5%)	1428 / 1520 (93.9%)
505.4	903 / 909	1961 / 2271 (86.3%)	1729 / 1825 (94.7%)
505.4	903 / 909	1961 / 2271 (86.3%)	1729 / 1825 (94.7%)
662.1	1348 / 1363	2026 / 2271 (89.2%)	1841 / 1924 (95.7%)
662.1	1348 / 1363	2026 / 2271 (89.2%)	1841 / 1924 (95.7%)
804.2	1796 / 1817	2048 / 2271 (90.2%)	1931 / 2012 (96.0%)
804.2	1796 / 1817	2048 / 2271 (90.2%)	1931 / 2012 (96.0%)
930.1	2244 / 2271	2074 / 2271 (91.3%)	1964 / 2033 (96.6%)
930.1	2244 / 2271	2074 / 2271 (91.3%)	1964 / 2033 (96.6%)

Table 6.1: Training and test performance for a simple network as the size of the training set is increased. The repeated lines in the table are not a mistake, but correspond to different random starting positions. The first group of results (with 455 training examples) contains twelve identical lines, of which only two are shown. For an explanation of the final column, see the text.

Figure 6.3: The weights of the simple network after 455 training examples have been assimilated. Each image array shows the input weights for the respective character classes; the small box to the lower right shows the bias for that class. The weights and biases have been separately scaled to make the smallest black and the largest white. Accordingly pixels in the grey region surrounding each character have weights close to zero, while the centre region resembles a matched filter: a positive region shaped like the relevant symbol surrounded by a negative sea.

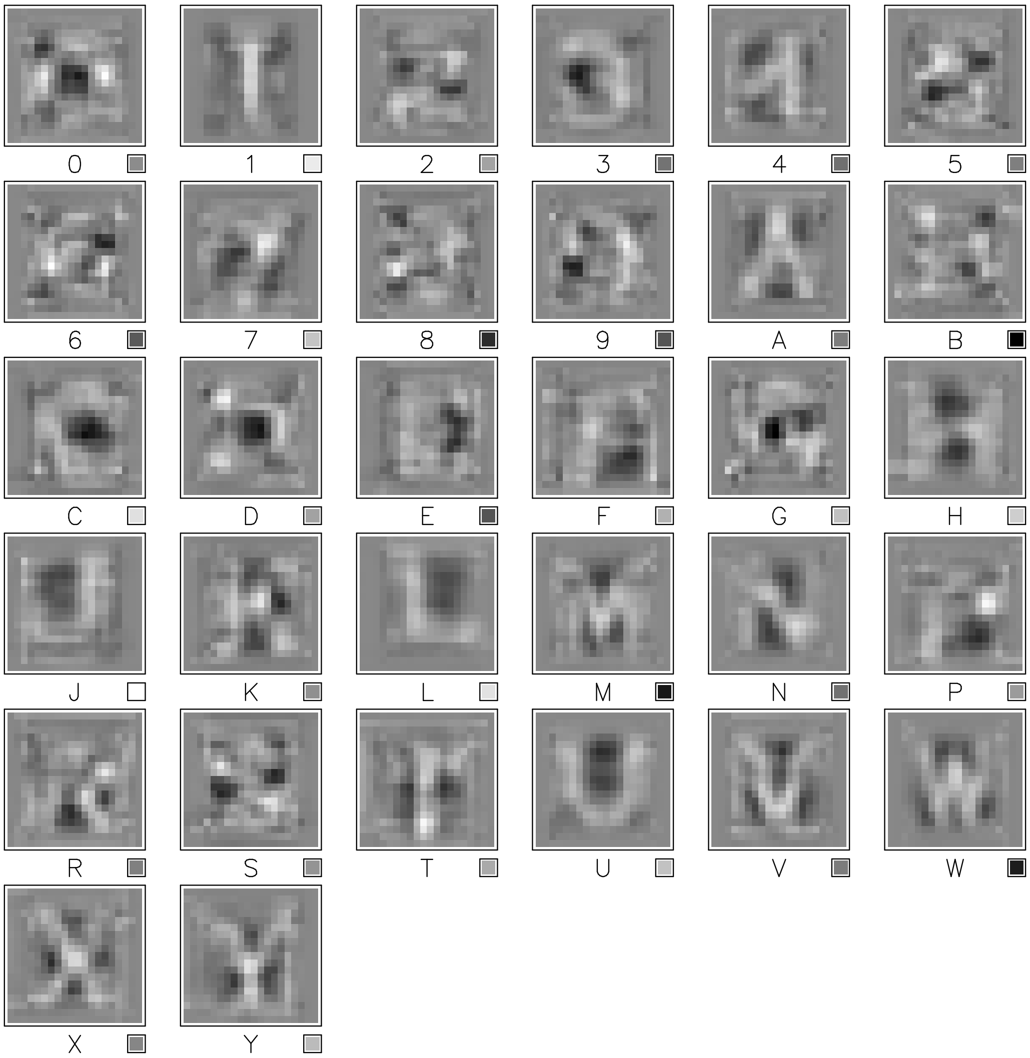


Figure 6.4: The weights of the simple network after 2244 training examples have been assimilated. In comparison with its predecessor this figure shows the weights for a network trained with more examples. These weights are considerably more fuzzy and the characters are correspondingly less distinct.

6.3 Junk categories

Despite this disconcerting title, adding junk categories to our network can help us read registration marks (see Bromley and Denker (1993)). These categories provide an escape for unrepresentative data in the training set; if we train the network on a dozen examples of the letter `A` and one of them has, for example, a dirty crossbar, it may well be better to classify the latter as junk lest it damage classification of the other `A`s. Table 6.2 shows the effect of adding junk categories to our network. The most obvious change is an improvement in read rates, particularly when a small number of training examples is used; this supports the hypothesis that junk categories ameliorate unrepresentative or sparse training data. The second consequence of the change is that despite making more correct predictions, fewer of them are `good'. On a cosmetic level this makes the creative read percentages higher; perhaps more usefully it serves to moderate the outputs of the network, though only in an approximate fashion.

There is another reason for introducing junk categories. Sometimes algorithms in `VIA` incorrectly partition the image and present the network not with a picture of a character but rather with a section of road or a bit of the vehicle. In such circumstances it is obviously good that the network recognizes the image as something other than a character, rather than picking a character essentially at random. Accordingly a reasonable choice of training data for the junk categories would be images that the splitting and location algorithms have wrongly selected as characters. In the table below we have used three junk categories broadly corresponding to the road, bits of vehicle and badly separated characters. This is a somewhat arbitrary division and accordingly the network performs better at classifying letters than classifying junk. The rightmost column in the table shows the performance on a test set that includes some typical junk. The performance falls by approximately 4%; this corresponds to a marginal read rate for the junk of less than 50%. This percentage does not improve significantly as we increase the size of the training set; not only is it junk, it is unclassifiable as well!

Training performance		Test performance		
		Characters only		Characters and junk
$Q(\hat{w})$	Number correct	Number correct	Creative accuracy	Number correct
407.2	493 / 497	1908 (84.0%)	1440 / 1503	1998 (80.5%)
676.1	969 / 994	1993 (87.8%)	1741 / 1825	2089 (84.1%)
926.2	1442 / 1491	2038 (89.7%)	1825 / 1895	2137 (86.1%)
1168	1908 / 1988	2063 (90.8%)	1901 / 1960	2162 (87.1%)
1375	2371 / 2484	2081 (91.6%)	1929 / 1989	2181 (87.8%)

Table 6.2: The performance of a network with junk categories. By adding these junk categories to the network we can improve the reading accuracy even when the data are all known to be valid images of letters.

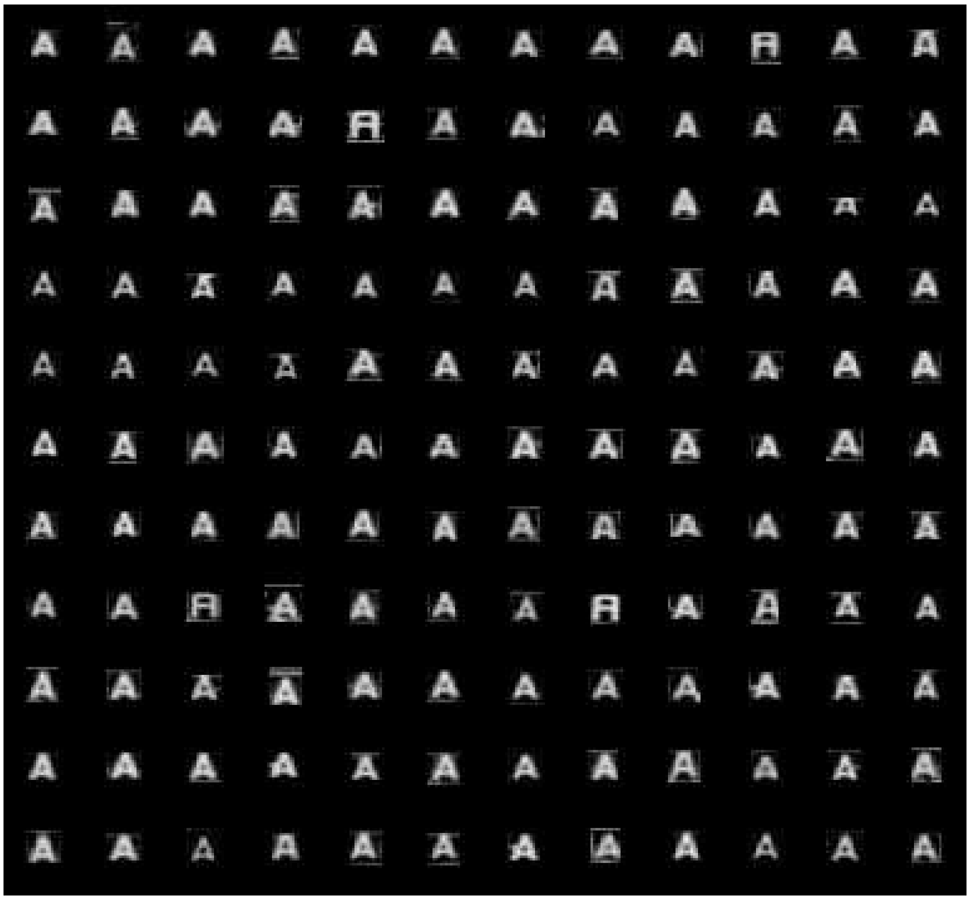


Figure 6.5: These characters have been cleaned prior to alignment; comparing them with figure 6.2 shows better alignment and fewer extraneous artefacts surrounding the letter.

6.4 Dejunking the characters

In the last section we found that adding junk categories to a classification network improves its performance. In this section we show how removing junk, this time from the data, can also improve matters.

A glance at figure 6.2 shows a lot of `junk' surrounding the letters in the images used to train the network, often taking the form of a bright border surrounding the letter. This is typically an artefact of the splitting algorithm which, in its zeal to capture any data of even marginal relevance to the reading problem, also includes a portion of the surrounding car in the image of the number plate. The remedy for this problem is simple; since such lines are fully the width or height of the image, we will remove any lines whose intensity is above some threshold from the image prior to training. The median is a more robust average than the mean and is thus a suitable choice for this threshold. Figure 6.5 illustrates a set of characters after they have been processed in this way. Just as bicycle wheels wobble when extraneous objects are attached to their perimeters, the junk that surrounds characters adversely affects their alignment. We therefore expect `dejunked' characters to be rather better aligned, with an attendant improvement in read rates. This is indeed found: table 6.3 shows the effects of dejunking.

Training performance		Test performance	
$Q(\hat{w})$	Number correct	Number correct	Creative accuracy
386.5	488 / 497	1989 / 2271 (87.6%)	1618 / 1660 (97.5%)
647	960 / 994	2083 / 2271 (91.7%)	1866 / 1914 (97.5%)
858.7	1432 / 1491	2117 / 2271 (93.2%)	1947 / 1991 (97.8%)
1054	1910 / 1988	2137 / 2271 (94.1%)	2004 / 2040 (98.2%)
1226	2375 / 2484	2142 / 2271 (94.3%)	2033 / 2068 (98.3%)

Table 6.3: The performance of a network when the characters have been shorn of extraneous junk. Processing the characters in this way makes them more alike and improves their alignment. This is reflected in the higher read rates.

6.5 A more Bayesian approach

So far we have not deployed the Bayesian machinery developed in chapters two and three. In particular we have considered networks with a single hyper-parameter set to an ad-hoc value, rather than networks with multiple hyper-parameters estimated according to the Evidence prescription. We will continue to use a network without any hidden units, but divide the weights into those fed from the image itself, and bias units.

Training performance						Test performance	
Number Correct	Bias		Image		Evidence	Number Correct	Creative Accuracy
	σ	Good	σ	Good			
496 / 497	5.1	4.4	4.2	86	-153	2019 (88.9%)	1943 / 2091
993 / 994	6.0	6.5	4.3	145	-282	2078 (91.5%)	2042 / 2165
1487 / 1491	6.6	8.6	4.2	204	-416	2118 (93.3%)	2081 / 2188

Table 6.4: The performance of a network when the characters have been shorn of extraneous junk. Processing the characters in this way makes them more alike and improves their alignment. This is reflected in the higher read rates.

Perhaps surprisingly, the Bayesian approach has not significantly improved the test scores. Certainly there are changes; the Bayesian choice for σ is significantly larger than unity and accordingly these nets have larger weights. This implies that such nets place more importance on the data, and we accordingly expect them to make stronger predictions. This is confirmed by the table; the number of confident predictions has risen by approximately $10-25\%$. If we have only a limited amount of training data the predictions are also improved a little, but when the number of training examples exceeds about a thousand the Bayesian choice of σ does not herald any significant improvement in reading accuracy.

6.6 The size of the network

A cursory examination of the character images suggests that the size of our templates is too large. This is reflected in the trained weights, which typically have a border of very small weights; these weights contribute little to the classification process and are accordingly squashed by the prior.

It is obviously inefficient to have such a large number of inputs; not only does it make the process slow, it might adversely affect the classification process. Essentially we can divide the image weights into those likely to be fed by a character and those likely to be fed by the border. One approach would be to establish another regularization class for the border weights but that would not improve the program's efficiency. Instead we will investigate pruning the input image.

We have investigated the performance of the network for a number of input images of different sizes, ranging from 6×6 to 18×18, and have typically used a variety of training sets for each size. Table 6.5 shows the reading rates for the experiments, which are also shown graphically in figures 6.6 and 6.7. An alternative measure of the networks' performance is the test error; these are shown in figures 6.8 and 6.9.

Several conclusions can be drawn from these results, the most obvious being that the size of the image doesn't matter very much beyond a certain point. There is some evidence that the predictions improve slightly for images larger than 12×12, but the data are not conclusive. There is much stronger evidence for the obvious result that more training data improve the performance of the classifier — though this regime is computationally very expensive and thus difficult to explore.

More interesting conclusions can be drawn from the test error results, in particular the differences between figures 6.8 and 6.9; the former test set containing samples drawn from the junk-categories, the latter not. Although the first test set contains fewer than 10% extra examples, the test error has risen by over 80%. Thus a large proportion of the test error comes from the junk-categories, and indeed if we analyse those characters wrongly classified by the 10×10 network when trained on 497 examples, we find that 67 of the 235 mistakes correspond to characters read as junk. Using a larger training set does not solve the problem; when 2484 examples were used the fraction rose slightly to 45 out of 145.

Image size	Number of training examples			
	497	1491	1988	2484
6×6	1628 (71.7%)	-	-	-
8×8	1922 (84.6%)	-	-	-
10×10	2036 (89.7%)	2085 (91.8%)	2113 (93.0%)	2126 (93.6%)
12×12	2059 (90.7%)	2102 (92.6%)	2109 (92.9%)	2118 (93.3%)
14×14	2052 (90.4%)	2101 (92.5%)	2111 (93.0%)	2111 (93.0%)
16×16	2049 (90.2%)	2101 (92.5%)	2116 (93.2%)	2121 (93.4%)
18×18	2053 (90.4%)	2100 (92.5%)	2116 (93.2%)	2128 (93.7%)
20×19	2019 (88.9%)	2078 (91.5%)	2118 (93.3%)	-

(a) True reading accuracy

Image size	Fraction correct				Fraction strong			
	Number of training examples							
	497	1491	1988	2484	497	1491	1988	2484
6×6	64.7%	-	-	-	79.4%	-	-	-
8×8	80.8%	-	-	-	90.6%	-	-	-
10×10	87.1%	89.6%	90.7%	91.5%	93.7%	95.2%	95.2%	95.5%
12×12	87.7%	90.8%	91.8%	91.6%	93.6%	96.2%	96.7%	96.5%
14×14	87.0%	90.5%	91.5%	91.4%	92.6%	96.1%	96.3%	96.3%
16×16	86.9%	90.4%	91.7%	92.3%	92.6%	95.7%	96.2%	97.0%
18×18	86.6%	90.6%	91.8%	92.4%	92.1%	95.9%	96.4%	96.7%
20×19	85.6%	89.9%	91.6%	-	92.1%	95.3%	96.3%	-

(b) Creative reading accuracy

Table 6.5: The upper table shows the reading accuracy of networks of different sizes as the number of training examples is increased. Each entry in the table shows the number of test examples read correctly, followed, in parentheses, by the percentage which that figure constitutes of the whole set. The lower table shows only those examples for which the network made a strong prediction, and the proportion of predictions which were strong. `Strong' in this sense means that the network assigned more than 50% of the probability to a single class.

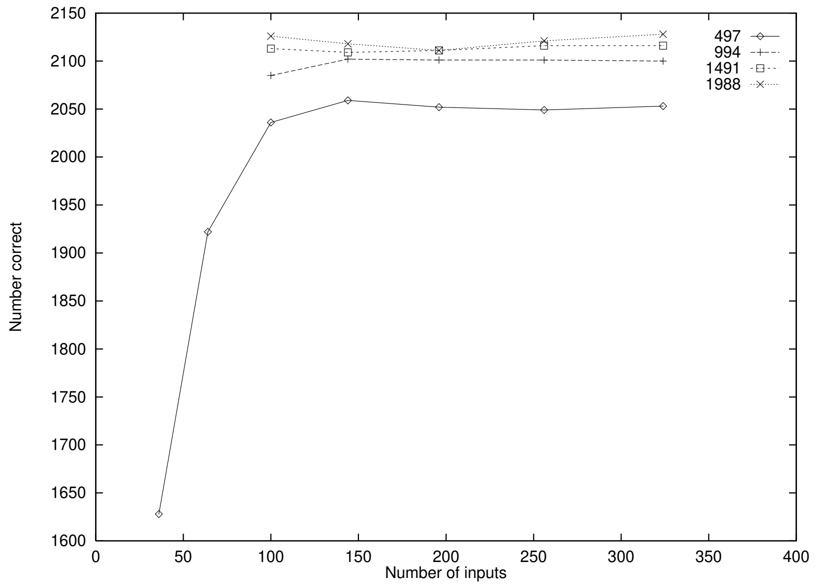


Figure 6.6: The graph charts the rise of prediction accuracy with increasing amounts of training data. Little variation is found with image size, assuming that the network is larger than some critical value.

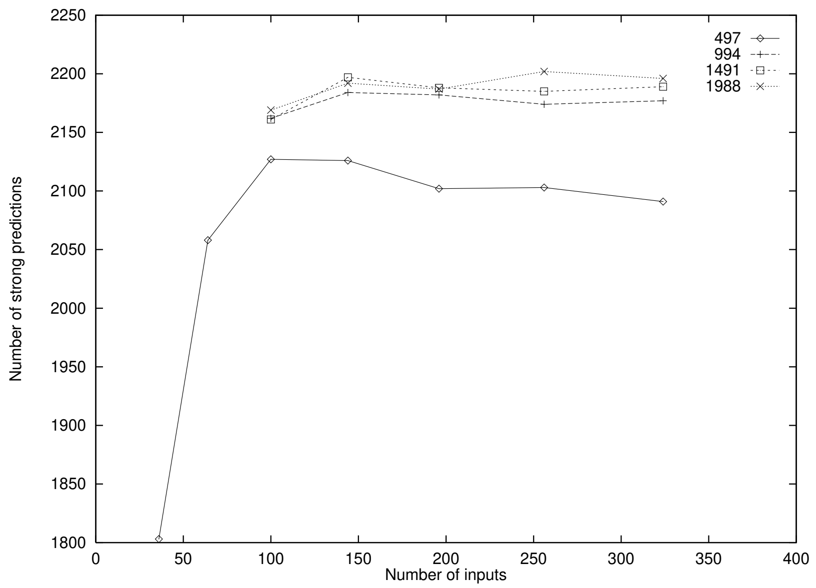


Figure 6.7: The graph charts the variation of prediction strength with increasing amounts of training data. There is some evidence that the predictions become less strong as the image size is increased.

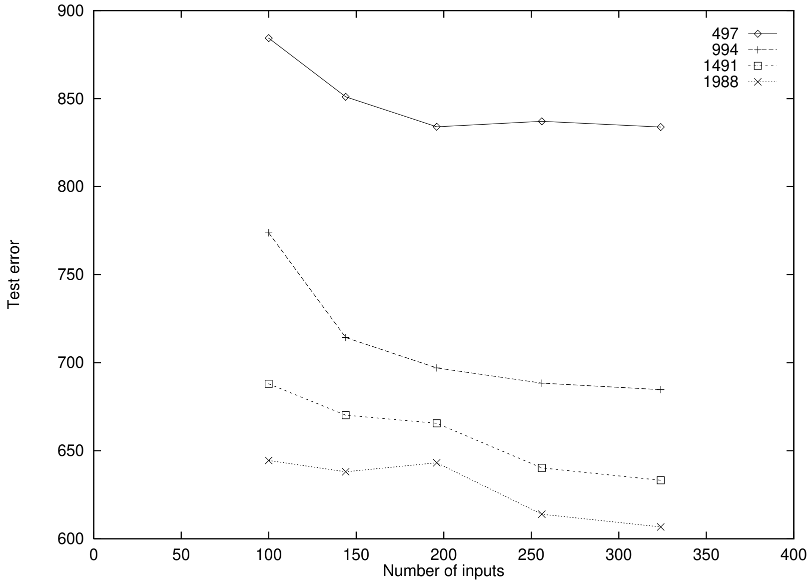


Figure 6.8: The graph charts the fall of test error accuracy with increasing amounts of training data. As image size increases the test error falls. This is in contrast to the number of correct predictions which changes little once the network has grown above a critical size.

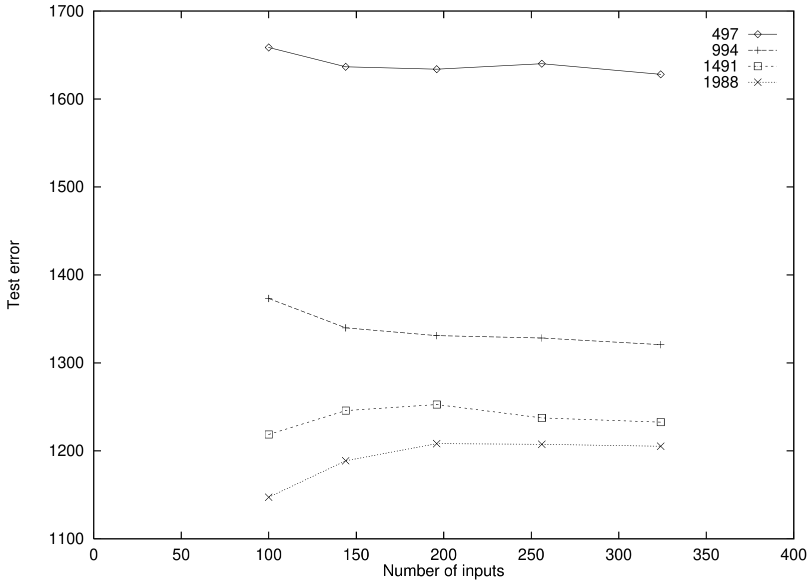


Figure 6.9: The graph charts the fall of test error accuracy with increasing amounts of training data. Little variation is found with image size. The test set differs from the graph above because it includes an extra 212 examples drawn from junk categories. Although the set has grown by only 10% the test error has nearly doubled.

6.7 Does Bayes get it right ?

In the previous section we examined the performance of the network as a function of the input size, and concluded that the size made very little difference once it had exceeded some threshold value. This conclusion implicitly assumes that the hyper-parameters have been set to a sensible value by the Bayesian re-estimation formulae,and we should investigate whether this claim is valid.

As a test of this, we have trained networks with fixed hyper-parameters on 497 examples from the 10×10 set described above. The read accuracies and test errors for these experiments are shown below. In these experiments we are trying to test the Bayesian method alone and accordingly use a test set which includes some junk-characters.

		Image σ		
		1	5	10
Bias σ	1	2003	1876	2090
	5	1892	1679	1986
	10	1883	1986	1927

(a) Test error

		Image σ		
		1	5	10
Bias σ	1	2104	2106	2078
	5	2102	2117	2120
	10	2102	2120	2119

(b) Read accuracy

Table 6.6: The table shows how the test error and read accuracy change as the hyper-parameters are changed. The Bayesian Evidence formalism sets σs for image and bias weights to 4.66 and 4.76 respectively; this achieves an error of 1659 and reads 2118 of the 2271 examples in the same test set correctly.

In terms of the test error the Bayesian method performs well; its test error is below any on the grid. The read accuracies are more difficult to interpret since any variation on the grid is masked by the noise. Accordingly we are content to say that the Bayesian method reads neither significantly more nor significantly fewer characters correctly than networks having different hyper-parameters.

6.8 Semi-supervised classification

As noted above our junk categories have essentially been allocated arbitrarily. Accordingly we are interested in schemes which automatically choose suitable classifications. One approach to this problem is to classify the characters manually into different classes and then to give this training set to the network. An alternative plan is to introduce a more complicated network designed to exploit this structure. We note that such a network is also at liberty to subdivide the character classes; accordingly the network may subdivide characters into individual fonts.

Suppose we believe that each character has two variants, called `Aa` and `Ab`. Then the probability of the character being an `A` is just a sum,

$$p(\mathtt{A}) = p(\mathtt{Aa}) + p(\mathtt{Ab}). \tag{6.1}$$

We could implement this by adding a linear post-processing layer to the softmax network, but handling the calculation explicitly is preferable as it avoids potentially singular behaviour. The algebra required is simple, if a little tedious, and we reproduce the key results below.

It is convenient to introduce ζ_i as the probability of an image being in a subclass, while z_i remains the probability of the image being a particular character. With this notation, equation 6.1 becomes

$$z_i = \zeta_{2i} + \zeta_{2i+1}. \tag{6.2}$$

The forward propagation equation 2.58 becomes

$$\zeta_i = \frac{\exp(y_i)}{\sum_j \exp(y_j)}, \tag{6.3}$$

$$z_i = \zeta_{2i} + \zeta_{2i+1}. \tag{6.4}$$

To propagate backwards we inject the errors into η and so, ignoring contributions from any other output units,

$$\eta_{2i} = \frac{\partial Q_D}{\partial y_{2i}} \quad = \theta_{2i}(z_i - t_i), \tag{6.5}$$

$$\eta_{2i+1} = \frac{\partial Q_D}{\partial y_{2i+1}} = \theta_{2i+1}(z_i - t_i), \tag{6.6}$$

where $\theta_i = \zeta_i / z_{\mathrm{int}(i/2)}$. This should be compared with equation 2.77; the essential difference is the fractional multiplier which diverts an equitable proportion of error to each subclass. Inevitably we also require the R-backprop analogue which I have derived.

$$\begin{aligned}\mathcal{R}(\eta_{2i}) =& t_i\theta_{2i}\left(\theta_{2i}\mathcal{R}(y_{2i}) + \theta_{2i+1}\mathcal{R}(y_{2i+1}) - \sum_j \zeta_j \mathcal{R}(y_j)\right) \\ &- \zeta_{2i}\left(\sum_j t_j(\theta_{2j}\mathcal{R}(y_{2j}) + \theta_{2j+1}\mathcal{R}(y_{2j+1})) - \sum_j \zeta_j \mathcal{R}(y_j)\right).\end{aligned} \tag{6.7}$$

This scheme has been used with some success to segment and classify handwritten characters (Keeler and Rumelhart, 1992).

So much for messy algebra! Does this network actually predict more accurately ? Table 6.7 compares the new network to a standard softmax network when fed with a training set of 497 10×10 images. The performance has improved both in the overall number of correct classifications, and more dramatically in the number of test characters read as junk; this has fallen by nearly a quarter.

The weights of this classifier are shown in figure 6.10. As predicted the network has split one of the junk classes into two subclasses; this accounts for the fewer junk / character mis-classifications. None of the other sub-classes has been used; instead the prior has pulled the weights towards zero. I speculate that this is because characters of significantly different fonts are comparatively rare; the extra cost of turning on weights in a subclass is only infrequently offset by an extra correct prediction.

	Softmax net	Bi-softmax net
Test error	1659	1564
Correct classifications	2036	2048
Characters read as junk	67	52
Evidence	-197	-187

Table 6.7: The table shows some preliminary results from a classifier with two subclasses assigned to each character class. This performs better than the traditional approach.

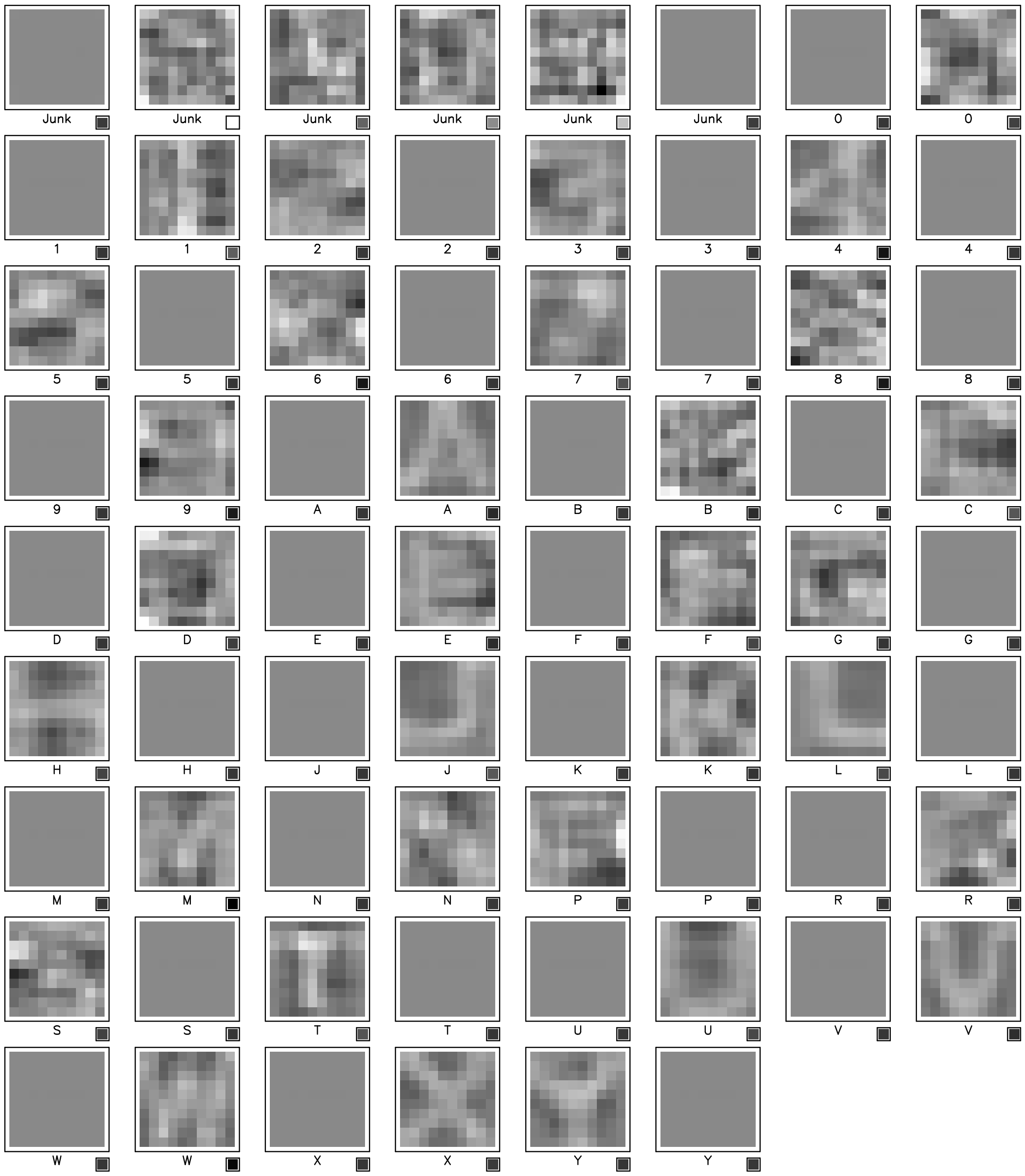


Figure 6.10: The plot shows the weights of the two sub-class classifier when trained on 497 examples. Each character has two sets of weights associated with it, corresponding to the two subclasses. In most cases, one of the subclasses has been `turned off', but one of the junk categories has been sub-divided.

6.9 Blurring the data

It has been suggested (Guyon *et al.*, 1991) that the performance of hand-writing classifiers can be improved by blurring the data before they are fed to the network. We have investigated this suggestion in the context of reading registration marks.

Blurring the data in this way can be justified in a Bayesian context by noting that it is equivalent to putting a correlated prior on the weights. To see this, observe that each node in the first stage of the network calculates

$$y = \sum_i w_i D_i, \tag{6.8}$$

where D_i is the value of the i^{th} pixel and w_i is the appropriate weight. The blurring process is a convolution and accordingly the blurred data, D_b, can be expressed as a matrix multiplication:

$$D_b = BD. \tag{6.9}$$

Hence the ys in the network fed with blurred data y_b are given by

$$y_b = \sum_i w_i \sum_j B_{ij} D_j, \tag{6.10}$$

which can also be written as

$$y_b = \sum_j \left(\sum_i B_{ij} w_i \right) D_j. \tag{6.11}$$

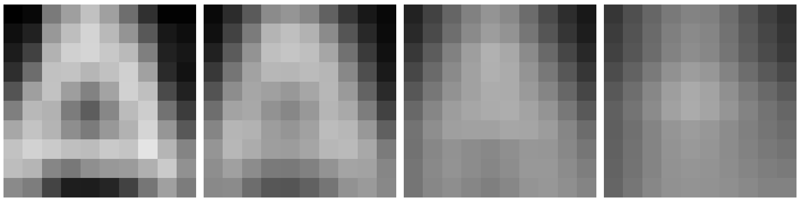
This is the Intrinsic Correlation Function (ICF) model for priors used by Gull (1989) to infer smooth distributions from an entropic prior; ironically this model is sometimes referred to colloquially as `pre-blur'.

A top-hat function seems a reasonable function with which to convolve the image. Although we will blur the image, the high-frequency structure will be preserved; in other words we will introduce extra correlations into the image without destroying too much of the information content. Figure 6.11 shows a typical character after various degrees of blurring have taken place. The data have been convolved with a square top-hat functions before being aligned and finally cropped to the 10×10 pixel array.

Sadly the results of this experiment are extremely poor. Table 6.8 shows how the character read-rate falls monotonically with increasing blur strength.

The ICF model is designed to model smooth structure in an image. Sample weights drawn from an ICF prior have little structure on scales smaller than the scale of the ICF. Accordingly this prior will only work well if the data are similarly devoid of structure on small scales. This is not true for our data set.

Why then does it work for Guyon ? I speculate that although his data individually have data at small scales, there is also significant variation on larger scales. Weights trained on the data must accordingly have little high frequency structure; such weights are precisely those produced by an ICF prior.



(a) 1 pixel (b) 3 pixels (c) 5 pixels (d) 7 pixels

Figure 6.11: The images show a character from the training set after it has been blurred with a top-hat function. The captions on each figure indicate the linear size of the top-hat in question.

Blur size	Number correct	Evidence
1	1636	-246
3	1706	-369
5	2051	-471
7	2366	-625

Table 6.8: The table shows how the reading accuracy of the network falls as the data become progressively blurred.

6.10 Ideas for future research

We conclude by outlining plans for further research on this topic.

6.10.1 Template manufacture

One of the avenues we intend to explore in the future is the manufacture of artificial character templates. Besides providing cleaner data, there are three reasons why such an undertaking is attractive:

- Foreign plates. It would clearly be useful to install a system in, for example, Germany, without first having to collect thousands of images, classify them by hand and train the network on the resulting set.

- Rare letters. As discussed above, the letters `Q` and `Z` occur very infrequently in the spectrum of registration marks. Indeed it is doubtful if we even now, after some years of observation, have enough examples to include them in the network.

- Different camera geometries. As noted above, the characters depend upon the geometry of the installation. Accordingly it is difficult to install the system in locations where the geometry is significantly different from the geometry of the site used to collect the training data. Currently the data are bi-linearly interpolated onto a new grid but this must damage the data.

For all of these reasons, any scheme which provides a ready supply of training data under our control is an attractive one. This is likely to be a two stage scheme: firstly generating a set of `perfect' letters and then abusing them by adding noise, shear and skew to generate a more reasonable training set. We are presently constructing a METAFONT description for suitable characters and intend to use PostScript to degrade these images at high resolution before sub-sampling them and adding noise.

6.10.2 AutoClass

All the classification schemes considered so far have been supervised; in other words a human has hand-classified the training set before feeding it to the network. Autoclass (Cheeseman *et al.* (1988)) seeks to solve the more difficult problem of assigning the classes automatically. It would be very interesting to see how Autoclass classifies our data; in particular it might provide a more objective classification of the junk categories.

Chapter 7

On British traffic

> The motor-car went Poop-poop-poop,
> As it raced along the road.
> Who was it steered it into a pond ?
> Ingenious Mr Toad!
>
> *The Wind in the Willows,* Kenneth Grahame.

In the previous chapter we discussed how the Bayesian network concepts which form the body of this thesis can be harnessed to read vehicle registration marks. Having tested the system on the highways and byways of the land, we have accumulated a body of data concerning the distribution of vehicles around the country.

Whilst this is far from a complete survey of the nation's driving habits, in this chapter we present some preliminary results and show how Bayesian modelling can be successfully applied to real-data.

7.1 The geographical distribution of vehicles

As explained previously, one of the most cogent pieces of information in a registration mark is the index mark, a two-letter code which can be related to the vehicle registration office (VRO) at which the vehicle was originally registered. At the time of writing there are some fifty-three offices spread over the country; their locations are plotted on figure 7.1 on the next page.

From observations of index marks we can infer where the vehicle was registered and, assuming that the vehicle's home has remained in the vicinity of the registration office, we can also infer something about the distribution of journeys to the observation point.

This is clearly a difficult problem; qualitatively we expect to see fewer cars from far away than nearby, but a sophisticated quantitative model is beyond the scope of this work. Instead we present two simple models, both as examples of Bayesian analysis applied to real data and as a pointer to further research. Our model assumes that there are two terms relevant to the probability of seeing a car from a given VRO: the total number of cars registered there, and the distance from the VRO to the observation point. The first model assumes that the probability has an exponential decay with distance, so that if N_i cars have been registered at an office i a distance x_i away then the fraction of the i that we would

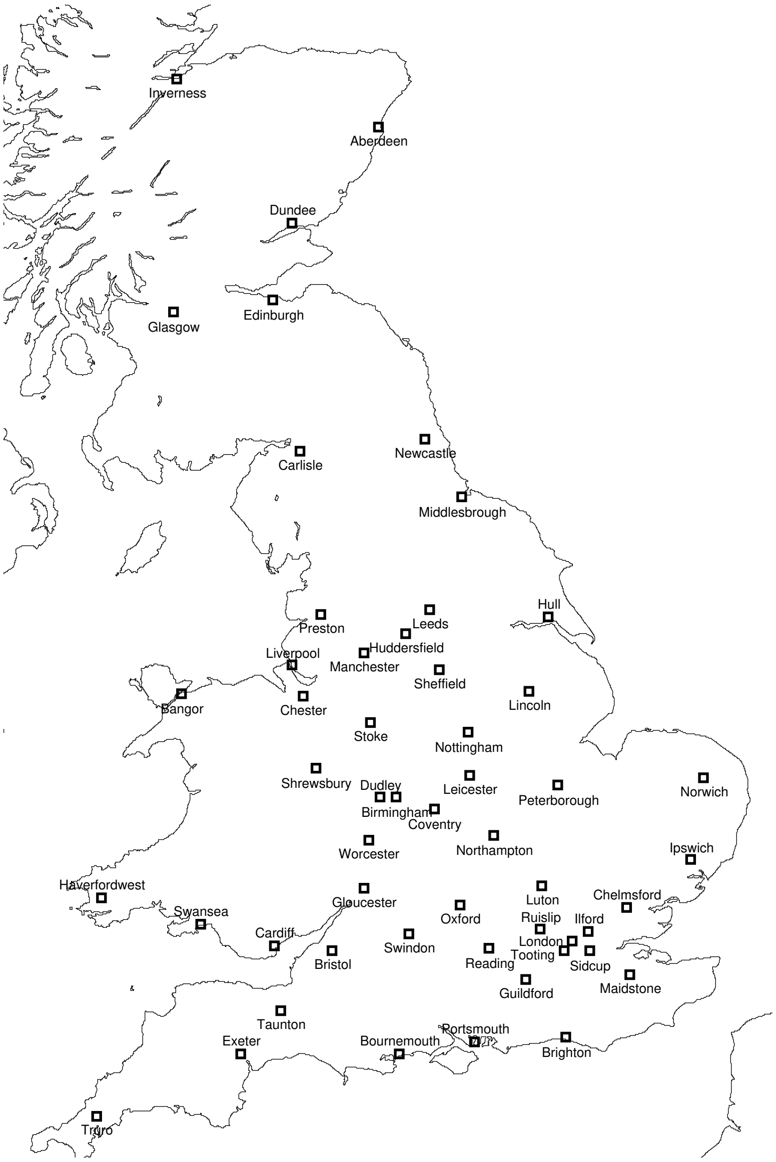


Figure 7.1: Vehicle Registration offices in Great Britain.

expect to see at the observation point from that office is

$$\phi_i = N_i \exp(-x_i/\lambda) / \sum_j N_j \exp(-x_j/\lambda), \tag{7.1}$$

at the observation point.

The second model is a traditional power-law, for which

$$\phi_i = N_i x_i^{-\alpha} / \sum_j N_j x_j^{-\alpha}, \tag{7.2}$$

Both models have one free parameter, which we will infer in the course of our analysis. To compare the two models, we must also assign prior densities over λ and α; the former is a scale parameter, for which a prior uniform in $\log\lambda$ is appropriate, whilst the latter should have a flat prior. We will truncate the valid ranges for these parameters at the ad-hoc limits shown in table 7.1.

	α		λ	
	Lower	Upper	Lower	Upper
Time	1	5	1/2 hour	6 hours
Distance	1	5	20 miles	500 miles

Table 7.1: The limits for the parameters in our models for national traffic flow

Similarly there are two sets of `magic constants', the x_i and N_i, which we must also calculate and which we now consider.

We have assumed that N_i number of vehicles licenced at the i^{th} VRO and still capable of travelling to the observation point is proportional to the number of vehicles licenced at the VRO over the last ten years. This is turn is assumed proportional to the number of serial letters issued in that period. The first assumption is quite difficult to quantify; it certainly assumes that vehicles age at the same rate in different parts of the country and so could be criticized by expecting an unduly high fraction of cars from areas with rough weather. However we can see no easy way of improving upon this for the present.

We have investigated three different distance measures between the VRO and the observation point: the straight line distance, the shortest distance along the trunk-road network and the shortest time assuming average driving habits. Rather than spend thousands of pounds and hours on an environmentally unfriendly study, we used the commercial program `Autoroute` (Nextbase, 1994) to calculate the data. The data are unwieldy and have accordingly been collated in table B.1.

Given these models we require a suitable likelihood to quantify their fidelity to the observations; given two sets of probabilities an entropic measure is appropriate and so

$$\log \mathrm{p}(\{\theta_i\}, M | \{\phi_i\}) = -M \sum_j \theta_j \log \frac{\theta_j}{\phi_j}. \tag{7.3}$$

where the $\{\theta_j\}$ are the proportion of vehicles seen from each VRO and M is the total number of observations.

Having described the model, we now turn to the data. We present four sets of data, three sets taken on different roads in and around Cambridge, the fourth on the M4 just north of Swansea. The observations are plotted pictorially in figure 7.2; on each plot the *area* of each square is proportional to the relevant θ_i. Even though three of these sites are very close geographically, they are all quite different in character.

M4 A motorway linking South Wales to London. Swansea lies close to the western end of the motorway.

Madingley Road A minor trunk road (A1303) carrying vehicles due west radially away from the centre of Cambridge.

A14 (formerly **A45)** Partly a northern bypass for Cambridge and partly a trunk route linking the east coast ports of Felixstowe and Harwich to the M1 motorway.

M11 Partly a western bypass for Cambridge and partly a trunk route linking Cambridge to London.

All the observations suffer from certain deficiencies which might make them unrepresentative. No systematic surveying plan was adopted; instead the data were collected at essentially random times, usually on weekdays and often around lunchtime. In particular practical considerations have limited observations to daylight hours. Furthermore biases will have been introduced by observing only a single carriageway; the inner lane of the M11 and the overtaking lane of the A14.

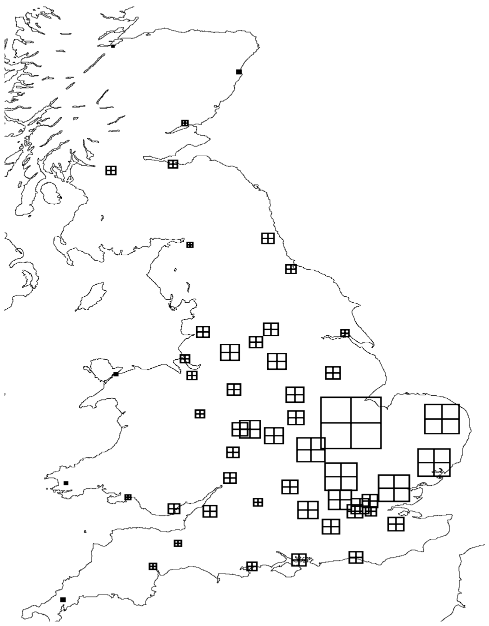
It is natural to ask how well these data are explained by the selection of models that we have described. A simple though non-Bayesian approach is to find some function of the mock-data which is linearly related to the `magic-numbers' in the model, and then to plot the same function of the real-data. For the exponential model we should plot $\log(\phi_i/N_i)$ against x_i; for the power-law we change the x-axis to $\log(x_i)$. Figure 7.3 shows the six different plots; a cursory visual inspection suggests that a power-law based upon journey time is likely to be the best model although both models seem to break down in the nearby/short-journey regime. There are two reasons why this might occur:

- We should really calculate the time not to the VRO but an average time over places served by the VRO. For distant VROs the distinction is unlikely to be important, but this is not true for vehicles licenced in, for example, Peterborough. Many of these vehicles will belong to Cambridge residents and will have travelled only a few miles rather than the forty or so from Peterborough.

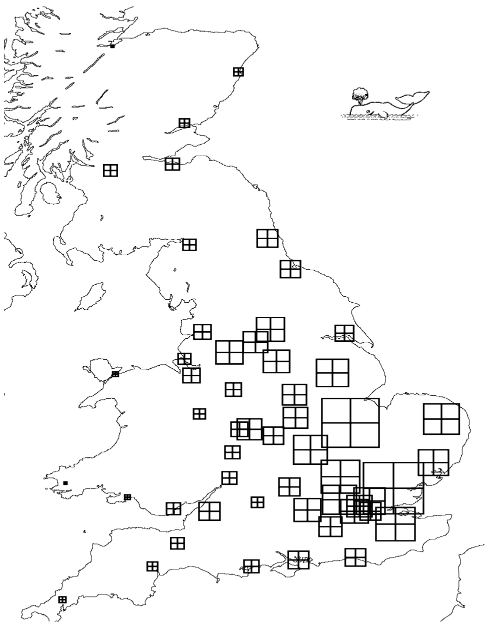
- For some roads, notably trunk routes, the population of vehicles conceivably admits a separation into long-distance and local traffic. It seems plausible that a more sophisticated model, which includes this distinction, would expect more local cars and accordingly fit the data better.

7.1.1 Bayesian model comparison

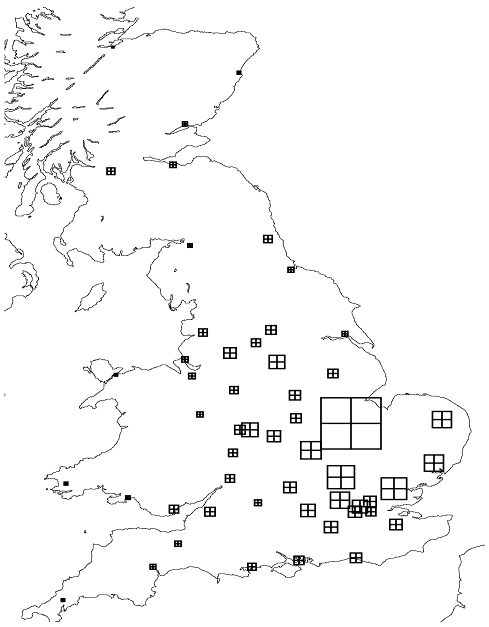
Model comparison is easy in the Bayesian framework; given the likelihood and priors we find the joint distribution and then marginalize over parameter space to find the Evidence.



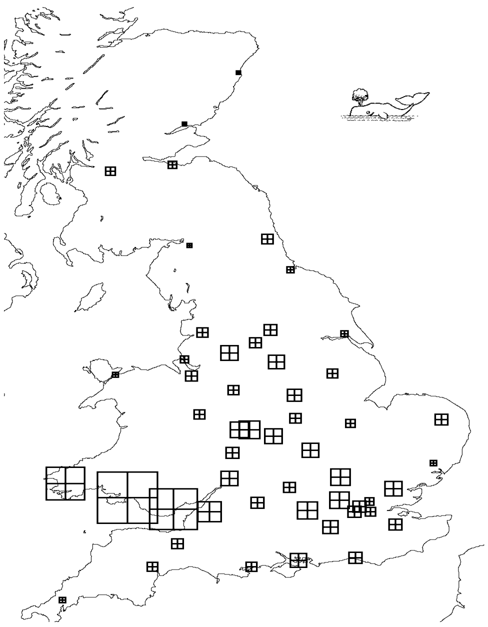
(a) Eastbound A14 near Cambridge



(b) Southbound M11 near Cambridge

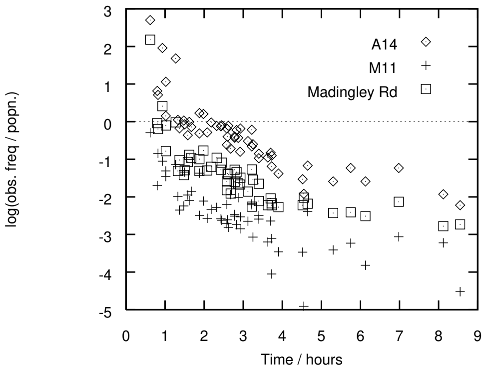


(c) Westbound Madingley Road in Cambridge

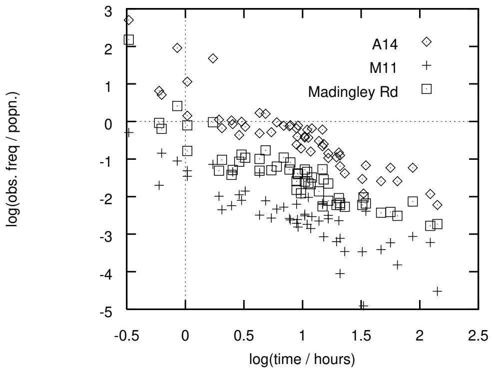


(d) Westbound M4 near Swansea

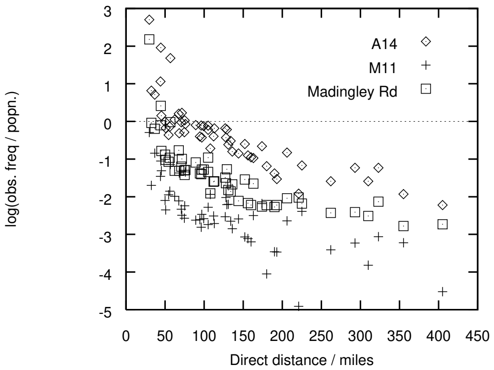
Figure 7.2: The registration offices of vehicles observed at four different sites on the UK road network. In each picture, the area of each plot is proportional to the number of vehicles observed from that VRO. The squares are scaled such the largest square is the same size on all maps. The important conclusion to draw from these maps is that the dynamic range of the data is greatest for the Madingley Road and smallest for the M11.



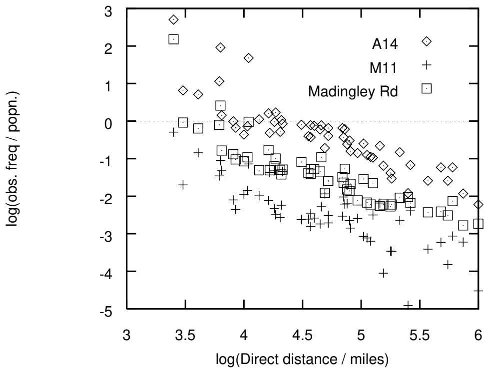
(a) Exponential model for time



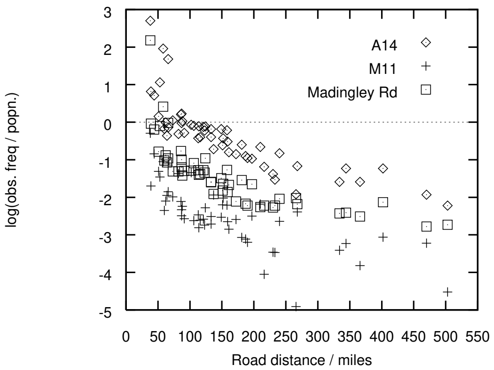
(b) Power-law model for time



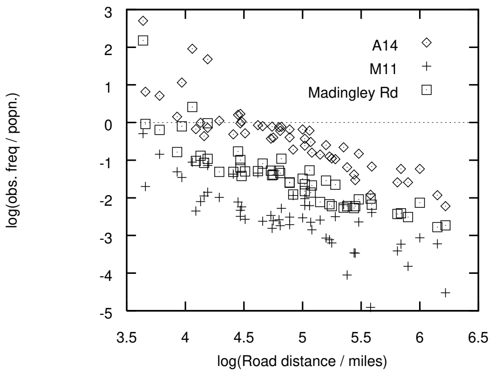
(c) Exponential model for direct distance



(d) Power-law model for direct distance



(e) Exponential model for road distance



(f) Power-law model for road distance

Figure 7.3: These plots would be straight lines if we plotted the mock-data; by instead plotting the real-data we get both a qualitative measure of model fidelity and insight into where the model fails to accurately predict the data. Each plot shows data from the three different observatories in the Cambridge area.

Exponential model			
	Time	Direct-dist.	Road-Dist.
A14	-0.971×10^4	-1.25×10^4	-1.23×10^4
M11	-430	-590	-604
Madingley Road	-5.7×10^3	-7.0×10^3	-7.2×10^3
Power-law model			
	Time	Direct-dist.	Road-Dist.
A14	-0.646×10^4	-0.97×10^4	-0.90×10^4
M11	-288	-427	-447
Madingley Road	-3.3×10^3	-4.6×10^3	-5.1×10^3

Table 7.2: The evidence, $-\mathrm{p}(D)$, for the different traffic models described in the text. Quantitative comparisons can be made between different models for a given data-set but not across different data sets.

All the models considered here have a single parameter and the marginalization required is just a one-dimensional integration performed numerically with the Romberg algorithm. The resulting Evidences are tabulated in table 7.2.

These results make quantitative what we had already observed in a qualitative fashion from the scatter plots, namely that the best model of the six is a power-law distribution based around an average transit time from the observation point to the VRO. At present we can only speculate why this model wins, but we note that the transit time to places connected by motorways is likely to be relatively shorter than their absolute distances and there is a suspicion that the motorways can be seen on the maps in figure 7.2. Of course there is a selection effect here; motorways link major towns and cities and it is precisely these places which tend to register more vehicles.

The better performance of the power-law over the exponential is also explicable. The key feature of an exponential model is that a certain fraction decays as the distance increases. For small distances this accords with our intuition but this is not the case for large distances. Although it is improbable that someone should travel from Haverfordwest to Cambridge, it is significantly more probable than an exponential model is likely to predict.

7.1.2 Parameter estimation

Having decided that the power-law model works moderately well in explaining the data, we now look at the posterior probabilities for the parameter α. The posterior densities for the three sets of observations are plotted in figure 7.4. The three observations lead to quite different values of α and it is instructive to consider why. In a power-law model, α tells us how fast things decay with distance; we are all familiar with conserved fluxes which behave as r^{-2} in three dimensions or r^{-1} in two. A very crude model for traffic flow might be the random ballistic transport of bullet-like cars from the VRO to the observation point which, if the principle of detailed balance is valid can also be seen as ballistic transport from the observation point to the VRO. Neglecting niceties, such as roads, this leads to an α of 1. If the isotropy assumption is valid, we expect $\alpha > 1$; the excess reflects the distribution of journey lengths.

In fact the observations all require $\alpha > 1$. We also find that, as the scale of the road increases, the distribution of cars becomes increasingly long-range; the motorway data have a most-probable α of 1.2, the trunk road 1.7 and the town-entry route 2.1. If we had more observatories, spread around the country, it would be interesting to see whether these numbers are relatively constant over a given class of road or whether there is a broad, featureless distribution over the range 1 to 2.5.

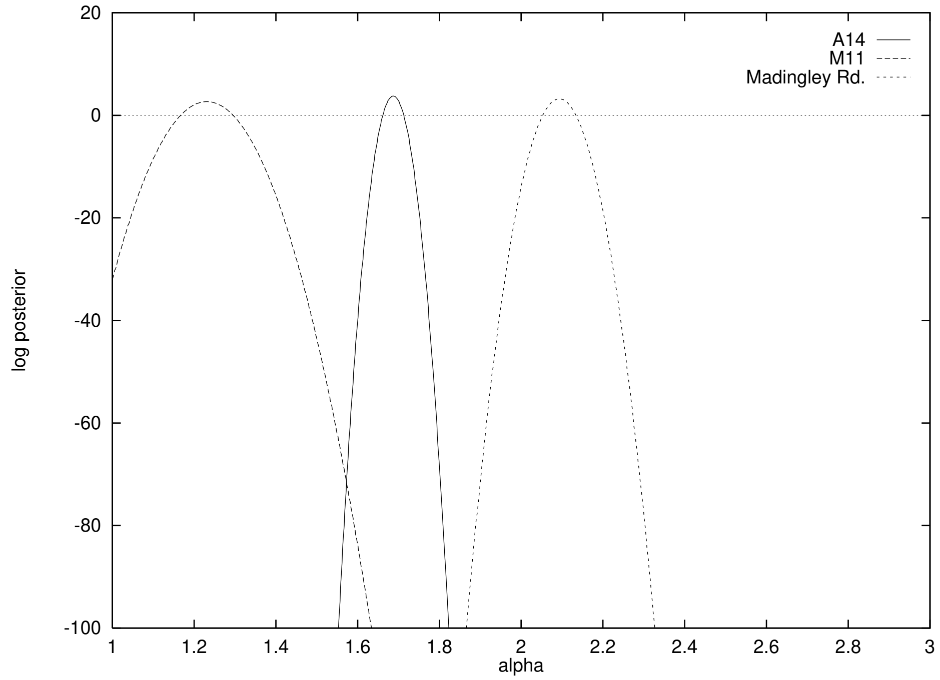


Figure 7.4: The posterior probability density for the power-law index, α, for the three different sets of observations. The three different roads have quite different distributions of traffic, corresponding to their different roles; as the road becomes larger a greater proportion of its traffic becomes long-distance, and the index falls.

7.2 The effect of age

In the previous section we discussed the distribution of registration offices for vehicles observed in the Cambridgeshire area. That analysis took no account of the age of the vehicles; we assumed, for example, that ten year old vehicles travel as widely as new ones. We can test this assumption by introducing a more sophisticated model to take account both of their registration office and index-letter.

To gain some insight into the form that this model should take, we begin by splitting the data-set by age when observed, grouping vehicles into years and then analysing each year's data separately. Having decided upon the transit-time power-law model, we are not interested in the evidence for different models, but in the posterior distribution for the index α. If these distributions are significantly and consistently different then we have some hope of finding a better model. Figure 7.5 shows the results of this experiment;

there is certainly a spread of αs between about 1.45 and 1.8 and the newer data tend to lie at the smaller end of this range. This is not a perfect correlation between α and age however, in particular the two-year old distribution is almost identical to the current one whilst quite different to the one-year old set.

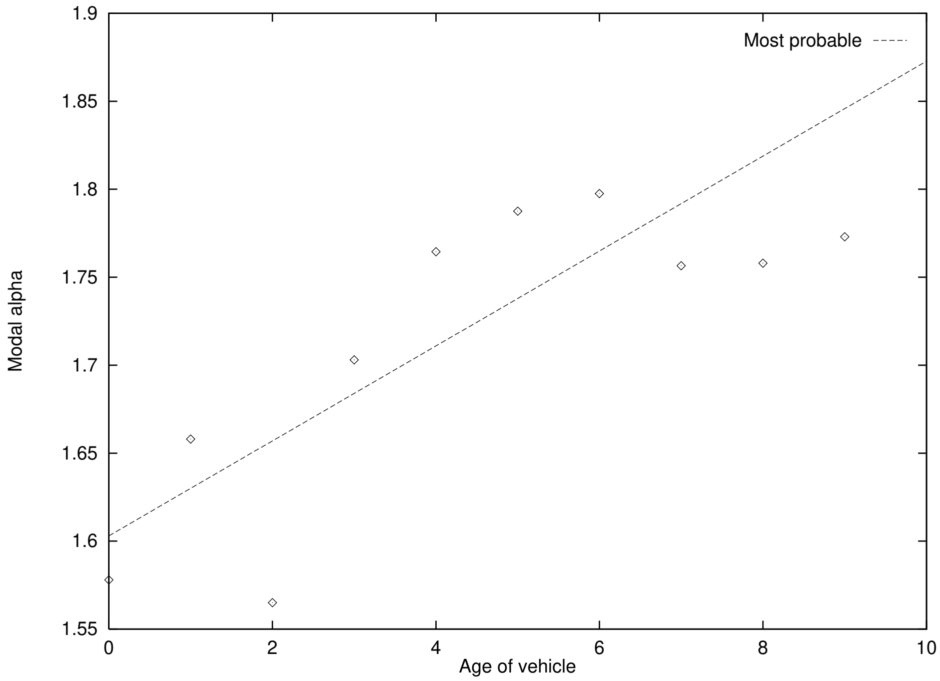


Figure 7.5: The graph shows ten different posterior densities for the power-law model of traffic based upon the transit-time from observation point to registration office. There is some evidence that newer vehicles undertake longer trips, i.e. that the best value of α tends to increase with vehicle age.

A very simple model for this effect is to suppose the α for the i^{th} year depend linearly upon the age of the vehicle:

$$\alpha_i = \alpha_0 + i\beta, \tag{7.4}$$

where α_0 is the index appropriate to new vehicles and β a parameter quantifies the effect of age. It is a legitimate criticism that the data in figure 7.5 do not lie on a perfect straight-line. This does not matter; in the Bayesian framework we can consider any model and probability theory will tell us if that model is better, or worse, that its predecessors. I do not rule out finding a better model in future, indeed such an endeavour might prove to be very interesting, but for the present I am content to improve upon the simple fixed spectral-index model.

With this assumption, the likelihood becomes

$$\log \mathrm{p}(\{\theta_i^j, M^j\}|\{x_i\}, \alpha_0, \beta) = -\sum_i M^i \sum_j \theta_j^i \log \frac{\theta_j^i}{\phi_j^i}, \tag{7.5}$$

$$\phi_j^i = x_j^{\alpha_0 + i\beta} / \sum_k x_k^{\alpha_0 + i\beta}. \tag{7.6}$$

We assign uniform priors on both α_0 and β, truncating them respectively to the ranges $[1.0, 2.0]$ and $[-0.1, 0.1]$. Over the ten years of data this allows the power-law index to change by unity, which is roughly the difference between the best-fit values for the M11 and Madingley Road.

As a reference model we use the power-law transit time model above, the likelihood now being a sum over successive years. This model has just one free parameter, α, and is accordingly a special case of equation 7.6 with $\alpha_0 = \alpha$ and $\beta = 0$. We assign the same prior on α as we did to α_0 above.

The likelihood for this model is shown in in figure 7.6. As expected the most probable value of the parameters has a small positive β; vehicles do become more parochial as they age. The best fit relationship has $\alpha_0 = 1.603$ and $\beta = 0.02698$. This line is overlaid on figure 7.5. It can be seen to fit the recent data rather better than the old; this is because fewer older cars are seen and so they affect the likelihood correspondingly less.

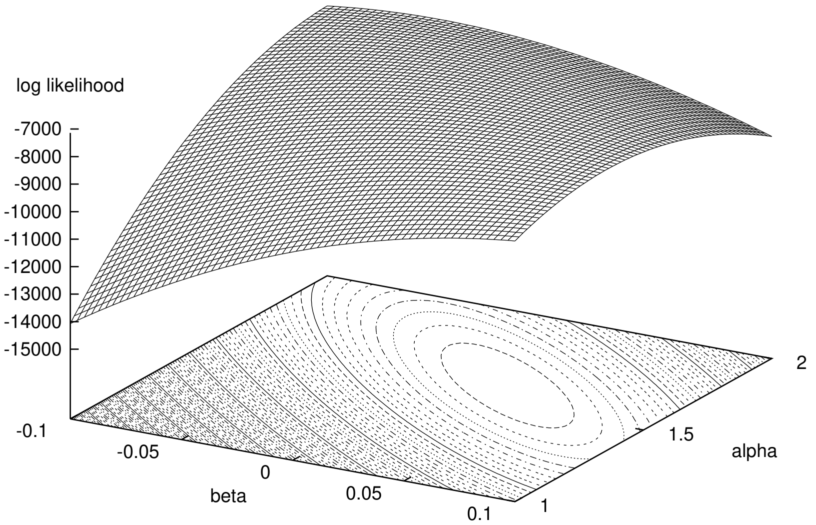


Figure 7.6: The graph shows the likelihood for the two-parameter model which crudely models the increased mobility of newer vehicles. The contours are evenly spaced.

To compare the two models, we must calculate their respective evidences. The priors are both constant and may be taken outside the integration. Denoting the one- and two-parameter models by $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively, we have

$$p(D|\mathcal{H}_1) = \frac{1}{\Delta\alpha} \int p(D|\alpha, \mathcal{H}_1) d\alpha, \tag{7.7}$$

$$p(D|\mathcal{H}_2) = \frac{1}{\Delta\alpha\Delta\beta} \iint p(D|\alpha, \mathcal{H}_2) d\alpha d\beta. \tag{7.8}$$

Accordingly we will prefer the more complicated model if

$$\int p(D|\alpha, \mathcal{H}_1) d\alpha > \frac{1}{\Delta\beta} \iint p(D|\alpha, \mathcal{H}_2) d\alpha d\beta. \tag{7.9}$$

We have calculated the evidences for the two models and found that the more complicated model is better; the evidences differ by 23 and accordingly the second model is consider 10^{10} times more probable.

7.3 Conclusions

In this chapter I have applied the techniques of Bayesian data-analysis to observations of traffic in Cambridgeshire. The models used have been extremely simple and serve mainly as a first step towards more useful models and as an illustration that these techniques can be applied to real data and not just simulations. I am not an expert in the field of traffic modelling and so it would be interesting to compare these results with any in the literature; sadly time has precluded any such study.

Modelling the distribution of vehicles is doubly useful. In the first instance it allows us to improve the read-rate of automatic number-plate reading systems which at present rely almost exclusively on the previous observations to build a prior, with the result that their performance suffers slightly when first installed in a new location. Given a good model for the distribution of vehicles at a random location we can ameliorate this problem.

Secondly the results are interesting and useful in their own right. Concluding that the journey time is a better predictor of journey frequency than the journey distance is an interesting conclusion which may be of interest to planners of large out-of-town shopping centres. Similarly the decay of journey time with vehicle age presumably reflects a correlation between long journeys and commercial travellers and another between commercial travellers and new vehicles. Given good data and more sophisticated models it might be possible to estimate the fraction of long-distance traffic in a given set of number-plates. To proceed further it is probably necessary to divide the vehicles into cars and lorries; currently this is not done, but it clearly is not an insurmountable problem.

Conclusions

There must be some way out of here, said the Joker to the Priest.
Bob Dylan

Probability theory might seem a queer subject to study in a physics laboratory; some people think it better suited to a statistics department. However this dissertation has demonstrated the essential utility of the subject; put simply it is the framework within which we should analyse data. Accordingly it is appropriate that it should be studied it be we physicists, engineers or indeed anyone who does experiments, collects data and makes inferences from them.

Despite this, probability theory still lies some distance from mainstream data analysis. The first chapter was written to address this problem and I hope that it achieves a wider readership than most PhD dissertations. I would like to emphasize that the title of the chapter is not pejorative. Rather it should be taken as synonymous with `A practical man's guide to probability theory'; the root of `engineer' is the Latin *ingenium* meaning skill. There is little new theory in this section, though I present two novel problems which I feel clearly illustrate the Bayesian methodology and are pedagogically useful.

The second chapter of this dissertation discussed the rôle of non-specific models in the form of probabilistic networks. I began with a handbook of traditional techniques in an attempt to make the work accessible to a lay audience, and extended existing methods of calculating the Hessian to produce a positive definite approximation; this is an important step towards robust network programs.

I then turned to the subject of marginalization and showed how to moderate the output of classifiers to take account of their non-linear nature. I have improved existing techniques to do this for binary classifiers and extended the idea to handle softmax networks. Here an essentially probabilistic idea lead to better predictions when judged by a traditional technique, the error on an independent test set.

In the third chapter I explored hyper-parameter control for large networks and showed how a combination of random sampling techniques and vector algebra obviate the need to compute and store the weights' Hessian matrix. This is a significant advance, as many practical applications of network based models require many more weights than traditional methods allow.

In this dissertation, the aim is always to provide things of practical worth, and so I proceeded to applications of our ideas. All the demonstrations in this thesis have been performed using the Backprob program, whose thirty thousand lines of code have taken a significant time to write and debug. By making this available to other researchers, I hope to facilitate more general acceptance of the Bayesian approach.

Chapter four began by illustrating the use of networks on a toy problem; we then explored networks with large numbers of hidden units. Such networks are conventionally held to be only of limited usefulness as it was believed they had a poor ability to generalize. I showed that this is not the case; probability theory suggests both a better error measure than the test error, and gives us insight into the large network limit.

Chapter five introduced a second synthetic example, which allowed us to compare the performance of model based and non-specific probabilistic analyses. The example is a simple one, yet it demonstrated that non-specific models are never completely general. This is both good and bad. On the positive side it means that we can interpret some of the model's parameters, while on the negative side it means that we must still be careful when we choose a model for a problem.

The last two chapters returned to the opening theme: probability as a tool for solving practical problems. We first addressed the problem of reading characters taken from vehicle number plates, and show that probabilistic networks are a suitable solution though as the number of weights is large we need to apply the theory developed in chapter three. A number of different approaches were considered, though their scope is limited by the size of the problem. Nevertheless I showed that by paying due attention to data preparation and analysis the reading accuracy is significantly improved.

Finally I analysed the data collected by the number plate reader and used Bayesian techniques to make inferences about the nations driving habits based upon observations of the number plates travelling along three roads near Cambridge. In particular I found that travelling time, rather than distance, better explains the observed distribution of traffic and furthermore that as vehicles age the journeys they undertake change significantly.

There is one unifying concept throughout this work; regardless of whether we analyse number plates, playing cards, traffic distributions or robot arms, probability theory is the right way to do it. The approach is simple to learn and only ignorance and inertia prevent its more widespread acceptance. This dissertation is my attempt to spread the gospel a little wider.

> Iamque opus exegi, quod nec Iovis ira, nec ignis, nec poterit ferrum, nec edax abolere vetustas.
>
> *Ovid*

Appendix A

Backprob - A Bayesian network program

All the demonstrations of probabilistic networks in the thesis have used the Backprob prgram which has been written during the course of our research. The program consists of about thirty thousand lines of ANSI-C and has been successfully ported to a variety of workstations and personal-computers running both Unix and MS-DOS.

I am keen that other researchers, both academic and industrial, should have access to the program and to this end have set-up a mailing list, `backprob@mrao.cam.ac.uk`, to which requests for licenses should be addressed.

At present Backprob

- implements regression, binary classification and softmax networks with arbitrary numbers of input, output and hidden units.
- optimizes the weight vector by using either conjugate-gradient or quasi-Newtonian optimizers. Several different line search algorithms are available.
- optimizes hyper-parameters for weight control and target noise level estimation using either matrix or Lanczos algorithms. Several different algorithms may be used to calculate the Hessian of the weights.
- predicts new targets using a Gaussian approximation.

It is easy to use Backprob for a new problem as the program's operation is entirely controlled by a set of text files. Explicitly these are:

The Netspec file describes the topology and class of the network.

The Datspec file describes the format of the data.

The Trainspec file describes broad features of the training process. For example, the desired accuracy and required diagnostics.

The Optspec file describes the fine details of the optimization process.

The Genspec file describes how predictions are made.

At present, Backprob has a number of deficiencies. The most important is a lack of documentation, in particular there is neither a user's guide, nor anything beyond a trivial `getting-started' example. On a technical level, the thirty thousand lines of code should probably only be about twenty-five thousand; like Pascal we have a written a long program because we lacked the time to shorten it.

Besides addressing the problems above, future developments are likely to include:

- A rewritten package to make new predictions.
- A multi-processor version for Solaris 2 and similar platforms.
- Revised low-level routines to take advantage of BLAS and similar libraries if available.
- An improved optimizer for problems too large for variable-metric methods.

Additionally some of the libraries written for the project may prove useful in a wider scope; these include an optimization package, a comprehensive package for handling user provided specification files and a selection of matrix and vector operations.

Appendix B

Magic numbers for traffic modelling

Registration Office	Index letters issued	Time / hour	Direct distance / mile	Road distance / mile
Aberdeen	275	8.1167	354.5975	470
Bangor	129	4.5167	193.3925	233
Birmingham	1072	2.4500	89.2960	106
Bournemouth	499	2.9167	136.1019	161
Brighton	638	2.7833	97.0125	113
Bristol	520	3.2167	128.8814	158
Cardiff	541	3.2500	151.7796	181
Carlisle	239	4.6500	225.1644	268
Chelmsford	1087	0.8167	36.8184	44
Chester	505	3.4000	144.3620	172
Coventry	638	1.8833	71.5334	87
Dudley	615	2.6167	96.2139	114
Dundee	277	6.9833	322.7021	402
Edinburgh	635	5.7500	293.0443	344
Exeter	444	3.9000	190.0574	230
Glasgow	1046	6.1333	310.1089	366
Gloucester	434	2.8333	104.8432	123
Guildford	892	2.0833	74.7629	91
Haverfordwest	136	4.5500	220.9435	266
Huddersfield	590	2.8667	128.0540	152
Hull	305	2.6833	108.4441	137
Inverness	92	8.5500	405.0531	503
Ipswich	334	0.9333	44.6428	58
Leeds	909	2.5833	131.1230	151
Leicester	565	1.3333	62.2212	73
Lincoln	509	1.5000	76.0998	102
Liverpool	471	3.7333	155.9338	187
London	862	1.3667	50.7189	60

continued on next page

continued from previous page				
Ruislip	1212	1.6167	49.9865	62
Ilford	464	1.0167	45.2704	51
Tooting	629	1.6667	56.0009	66
Sidcup	392	1.5833	54.5140	64
Luton	1050	0.8000	32.5328	39
Maidstone	783	1.8833	68.1210	82
Manchester	1338	3.1167	134.1193	159
Middlesbrough	461	3.2167	174.1091	210
Newcastle	774	3.7000	205.5466	240
Northampton	635	1.0167	44.0504	53
Norwich	513	1.2667	56.7468	66
Nottingham	701	1.4833	74.9795	88
Oxford	470	1.9833	67.5779	86
Peterborough	569	0.6167	29.8817	38
Portsmouth	674	2.7833	112.0866	133
Preston	929	3.6333	160.2943	190
Reading	954	2.1833	70.8053	87
Sheffield	877	2.3167	105.4265	124
Shrewsbury	227	2.9333	126.7527	149
Stoke	479	2.5833	112.5607	133
Swansea	230	3.7333	180.0790	216
Swindon	259	2.6000	94.4674	116
Taunton	280	3.4000	163.3013	197
Truro	181	5.3000	261.9809	334
Worcester	383	2.4333	99.4727	121

Table B.1: The magic numbers for our national traffic model.

Bibliography

Aczél, J. (1966). *Lectures on Functional Equations and their Applications*. Academic Press, New York.

Bridle, J. S. (1989). Probabilistic interpretation of feedforward classification network outputs, with relationship to statistical pattern recognition. In F. Fougelman-Soulie and E.J.Hérault, editors, *Neurocomputing: algorithms, architectures and applications*. Springer-Verlag.

Bromley, J. and Denker, J. S. (1993). Improving rejection performance on handwritten digits by training with "rubbish". *Neural Computation*, **5**(3), 367–370.

BSAU145A (1980). *BSAU 145A*. The British Standards Institute.

Cheeseman, P., Freeman, D., Kelly, J., Self, M., Stutz, J., and Taylor, W. (1988). Autoclass: a bayesian classification system. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 54–64.

Cox, R. T. (1946). Probability, Frequency, and Reasonable Expectation. *American Journal of Physics*, **14**, 1–13.

Denker, J. S. and Le Cun, Y. (1991). Transforming neural-net outputs to probability distributions. In R. P. Lippmann, editor, *Advances in neural information processing systems 3*, pages 853–859. Morgan Kaufmann.

Garrett, A. J. M. (1990). Ockham's razor. In W. T. G. Jnr and L. H. Schick, editors, *Maximum Entropy and Bayesian Methods, Laramie, Wyoming*, pages 357–364, Dordrecht. Kluwer Academic Publishers.

Gill, P., Murray, W., and Wright, M. H. (1981). *Practical Optimization*. Academic Press, London and New York.

Glass's guide (1995). *Glass's Index of Registration Marks*. Glass's Information Services, Weybridge, England. Some information in this guide is covered by Crown Copyright and as such is reproduced by permission of the Controller of Her Majesty's Stationery Office.

Golub, G. H. and Van Loan, C. (1990). *Matrix Computation*. Johns Hopkins University Press, Baltimore, 2 edition. Chapter 9 deals with the Lanczos method.

Gull, S. F. (1988). Bayesian inductive inference and maximum entropy. In G. Erickson and C. Smith, editors, *Maximum Entropy and Bayesian Methods in Science and Engineering*, pages 53–74, Dordrecht. Kluwer Academic Publishers.

Gull, S. F. (1989). Developments in maximum entropy data analysis. In J. Skilling, editor, *Maximum Entropy and Bayesian Methods*, pages 53–71, Dordrecht. Kluwer Academic Publishers.

Gull, S. F. and Skilling, J. (1991). *Quantified Maximum Entropy: MemSys 5 Users' Manual*. Maximum Entropy Data Consultants, Cambridge, England.

Guyon, I., Vapnik, V., Boser, B., Bottou, L., and Solla, S. A. (1991). Structural risk minimization for character recognition. In *Advances in neural information processing systems 3*, pages 853–859. Morgan Kaufmann.

Hebb, D. O. (1949). *The Organization of Behaviour*. John Wiley and Sons, New York.

Hestenes, M. R. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand.*, **49**, 409–436.

HMGA (1971). *Statutory Instrument 450*. Her Majesty's Government.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA*, **79**, 2554–2558.

Jaynes, E. T. (1994). Probability theory: The logic of science. Available by anonymous ftp from bayes.wustl.edu in Jaynes.book. Bernoulli's urn is discussed in the first few chapters of Jayne's Meisterwerk.

Jervis, T. and Fitzgerald, W. (1993). Optimization schemes for neural networks. Technical Report CUED/F-INFENG/TR 144, Cambridge University Engineering Department, Trumpington Street, Cambridge, England.

Kaniel, S. (1966). Estimates for some computational techniques in linear algebra. *Math. Comp*, **20**, 369–78.

Keeler, J. and Rumelhart, D. E. (1992). A self-organizing integrated segmentation and recognition neural net. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 496–503, San Mateo, California. Morgan Kaufmann.

Lanczos, C. (1950). An iteration method for the solution of the eigenvalue problem for linear differential and integral operators. *J. Res. Nat. Bur. Stand.*, **45**, 255–82.

Luenberger, D. G. (1973). *Introduction to Linear and Nonlinear Programming*. Addison Wesley, New York.

MacKay, D. J. C. (1991). *Bayesian Methods for Adaptive Models*. Ph.D. thesis, California Institute of Technology.

MacKay, D. J. C. (1993). Hyperparameters: Optimize or integrate out ? In G. Heidbreder, editor, *Maximum Entropy and Bayesian Methods*, Dordrecht. Kluwer Academic Publishers.

MacKay, D. J. C. and Neal, R. (1994). Automatic relevance determination for neural networks. Technical report, Cambridge University. In preparation.

MacKay, D. J. C. and Oldfield, M. J. (1995). Generalization error and the number of hidden units in a multilayer perceptron. In preparation.

Marquardt, D. (1963). *Journal of the Society for Industrial and Applied Mathematics*, **11**, 431–441.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas imminent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, 115–133.

Neal, R. M. (1994). *Bayesian Learning for Neural Networks*. Ph.D. thesis, University of Toronto.

Nextbase (1994). Autoroute plus. This is a commercial computer program available for MS-DOS machines.

Pearlmutter, B. A. (1994). Fast exact multiplication by the Hessian. *Neural Computation*, **6**(1), 147–160.

Polak, E. (1971). *Computational Methods in Optimization*. Academic Press, London and New York.

Press, W. H., Teukolsky, S. A., T.Vettering, W., and Flannery, B. P. (1992). *Numerical Recipes in C, The Art of Scientific Computing*. Cambridge University Press, Cambridge, second edition.

Robinson, D. R. T. (1992). *Developments in Maximum Entropy Data Analysis*. Ph.D. thesis, University of Cambridge.

Rumelhart, D. E., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, **323**, 533–536.

Shannon, C. (1948). A mathematical theory of communication. *Bell Systems Technical Journal*, **27**, 379. Reprinted in C.E. Shannon and Weaver, `The Mathematical Theory of Communication' , University of Illinois Press, Urbana (1949).

Skilling, J. (1989). The eigenvalues of mega–dimensional matrices. In J. Skilling, editor, *Maximum Entropy and Bayesian Methods, Cambridge 1988*, pages 455–466, Dordrecht. Kluwer.

Skilling, J. and Sibisi, S., editors (1994). *Maximum Entropy and Bayesian Methods*, Dordrecht. Kluwer Academic Publishers. These are the proceedings of the 14th annual workshop; proceedings from previous years have also been published by Klewer under the same title but with different authors.

Squires, G. L. (1985). *Practical Physics*, page 18. Cambridge University Press, Cambridge, third edition.

Wilkinson, J. and Reinsch, C. (1971). *Linear Algebra*, volume 2 of *Handbook for Automatic Computation*. Springer-Verlag, New York.

Wolpert, D. H. (1993). On the use of evidence in neural networks. In C. L. Giles, S. J. Hanson, and J. D. Cowan, editors, *Advances in Neural Information Processing Systems 5*, pages 539–546, San Mateo, California. Morgan Kaufmann.

Colophon

This dissertation was typeset by the author on Sun workstations, Linux computers and DOS boxes. The document was produced in a single LaTeX 2ε run and subsequently rendered as PostScript with dvips. All figures were produced as Encapsulated-PostScript and then included into the LaTeX source; about half were generated with gnuplot and pgplot, the remainder being written by hand. The bibliography was produced automatically by BibTeX and conforms to the generic natbib style. Both draft and final copies of the dissertation were previewed on screen using xdvi and printed at 600dpi on a Hewlett-Packard Laserjet 4M. The dissertation was bound by Stoakley Bookbinders, Cambridge.

The text of this dissertation was set in fonts designed by Adobe Systems. In the original thesis, Times Roman at 11 point was used for the body text, Helvetica for labelling figures and illustrations, and the Symbol and Zapf Chancery fonts for the mathematics. However for this printing slightly larger, 12 point, fonts were used. In both cases these fonts were augmented by symbols taken from the Computer Modern family of fonts designed by Donald Knuth.