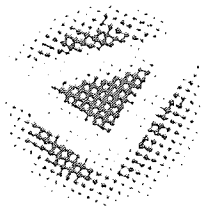
The n-tuple network

Coping with sub-optimality

MICHAŁ MORCINIEC

Doctor Of Philosophy



THE UNIVERSITY OF ASTON IN BIRMINGHAM

October 1996

THE UNIVERSITY OF ASTON IN BIRMINGHAM

The n-tuple network

Coping with sub-optimality

MICHAŁ MORCINIEC

Doctor Of Philosophy, 1996

Thesis Summary

The subject of this thesis is the n-tuple network (RAMnet). The major advantage of RAMnets is their speed and the simplicity with which they can be implemented in parallel hardware. On the other hand, this method is not a universal approximator and the training procedure does not involve the minimisation of a cost function. Hence RAMnets are potentially sub-optimal. It is important to understand the source of this sub-optimality and to develop the analytical tools that allow us to quantify the generalisation cost of using this model for any given data. We view RAMnets as classifiers and function approximators and try to determine how critical their lack of universality and optimality is.

In order to understand better the inherent restrictions of the model, we review RAMnets showing their relationship to a number of well established general models such as: Associative Memories, Kanerva's Sparse Distributed Memory, Radial Basis Functions, General Regression Networks and Bayesian Classifiers.

We then benchmark binary RAMnet model against 23 other algorithms using real-world data from the StatLog Project. This large scale experimental study indicates that RAMnets are often capable of delivering results which are competitive with those obtained by more sophisticated, computationally expensive models.

The Frequency Weighted version is also benchmarked and shown to perform worse than the binary RAMnet for large values of the tuple size n. We demonstrate that the main issue in the Frequency Weighted RAMnets is adequate probability estimation and propose Good–Turing estimates in place of the more commonly used Maximum Likelihood estimates.

Having established the viability of the method numerically, we focus on providing an analytical framework that allows us to quantify the generalisation cost of RAMnets for a given dataset. For the classification network we provide a semi-quantitative argument which is based on the notion of Tuple distance. It gives a good indication of whether the network will fail for the given data. A rigorous Bayesian framework with Gaussian process prior assumptions is given for the regression n-tuple net. We show how to calculate the generalisation cost of this net and verify the results numerically for one dimensional noisy interpolation problems.

We conclude that the n-tuple method of classification based on memorisation of random features can be a powerful alternative to slower cost driven models. The speed of the method is at the expense of its optimality. RAMnets will fail for certain datasets but the cases when they do so are relatively easy to determine with the analytical tools we provide.

Keywords: n-tuple classifier, RAMnets, generalisation cost, Bayesian inference, Gaussian process

Acknowledgements

I would like to thank my supervisor, Dr. Richard Rohwer, for his encouragement, support and advice that were always there when needed most. I am grateful to Richard for teaching me how to do research and spending so much time with me discussing RAMnets.

My thanks extend to all the people in the Neural Computing Research Group at Aston, who were always interested in my work and problems, be it technical or theoretical.

I am grateful to John van der Rest, Mike Tipping, Cazhaw Qazaz, Marcus Svensén, Ansgar West and David Barber for help, creative discussions and exchanging ideas. Chris Williams and Huaiyu Zhu were always eager to discuss numerous theoretical details that my research work brought about. Phil Barrett and Richard Lister did a great job as systems administrators maintaining our ever expanding network and software resources.

Thanks to Marcus, Anna, Ansgar and David for being a merry company on the numerous trips to the Peak District and Welsh mountains.

I thank Louis Wehenkel of Universite de Liege for useful correspondence and permission to report results on the BelgianI and BelgianII data sets, Trevor Booth of the Australian CSIRO Division of Forestry for permission to report results on the Tsetse data set, and Reza Nakhaeizadeh of Daimler-Benz, Ulm, Germany for permission to report on the Technical, Cut20 and Cut50 data sets.

I thank Amstrad Plc for providing the funding in my first year and to the British Council for administering this grant within the Excalibur Scholarship. My research could not continue for the next two years without the support of the Neural Computing Research Group at Aston for which I am grateful.

Chapter 6 of this thesis is a result of my close collaboration with Richard Rohwer who carried out the calculation of the formula for the expected generalisation cost of RAMnets. I am responsible for the calculation of the cost variance and all the numerical simulations.

Contents

List of Figures

List of Tables

Chapter 1

General Introduction

1.1 Pattern Recognition and Classification

Pattern recognition and classification are tasks that humans perform routinely and apparently without effort. The invention of a digital computer initiated numerous attempts to automate and accelerate solutions of practical problems in these domains. Representative examples of applications involving pattern classification include, for example, signature verification, speech recognition, medical images analysis or customer credit authorisation.

One of the most fruitful approaches to pattern recognition has been by statistical methods. Statistical pattern recognition (Bishop, 1995; Ripley, 1996) provides a rigorous unifying framework, which allows us to describe formally the process of classification.

The raw data x is given in terms of measurements (features) $\alpha_i(x)$, which may be written as a T-dimensional vector $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_T\}$. The feature vector $\boldsymbol{\alpha}$ is a function $\boldsymbol{\alpha}(x)$ of the pattern x belonging to class c which is an element of the finite set of classes $\{1, 2, \ldots, C\}$. A classifier can be viewed as a function $g(\boldsymbol{\alpha})$ that assigns an unlabeled pattern x to a class, on the basis of the measurements $\boldsymbol{\alpha}$. A correct classification corresponds to $g(\boldsymbol{\alpha}) = c$ while a missclassification corresponds to any other classification, i.e., $g(\boldsymbol{\alpha}) \neq c$. The structure of the classifier is learned from the data, which is regarded as a sample from a population of patterns. The function g can be constructed by estimating the class probability distributions of features and applying a statistical decision theory to arrive at an optimal decision.

Bayesian statistical inference (Berger, 1993; Gelman *et al.*, 1995) enables us to make decisions about c in terms of probability statements so as to minimise the probability of error. Before any data is observed our prior belief about c is specified by the probability distribution $p(c)$. When the measurement $\boldsymbol{\alpha}$ becomes available we condition c on the known value of $\boldsymbol{\alpha}$ using the Bayes' rule

$p(c|\boldsymbol{\alpha}) = \frac{p(\boldsymbol{\alpha}|c)p(c)}{p(\boldsymbol{\alpha})}$. Thus, in the Bayesian approach, the pattern x is assigned to the class c which yields the maximum posterior class density $p(c|\boldsymbol{\alpha})$. We use Bayesian statistical methods extensively in chapter 6.

Pattern recognition and classification is very much an interdisciplinary field. There exists a plethora of algorithms contributed from such diverse areas as physics and psychology. However, there is a number of desirable features that every classification method should exhibit. Some of them are listed below:

- Accuracy. The method should be accurate regardless of the data it is applied to.
- Speed of operation. The algorithm should process the data as fast as possible.
- Comprehensibility. It is important to understand the principles under which the method operates. Otherwise it may be not implemented correctly or the results may be misinterpreted.
- Learning time. In the real-world environment, it is essential that the classification rule be learned quickly.

The ideal classification method would score highly on all the above features. In practice, however, one has to trade-off speed and learning time for accuracy. It is difficult to answer the question "Is a method A better than method B?" because the answer will depend on the dataset used for evaluation and the importance that we attach to each of the factors. It may happen, for certain data, that a fast, but potentially sub-optimal model will outperform a slower but more accurate method.

In this thesis we discuss a classification method which belongs to the wider class of models called artificial neural networks (Haykin, 1994; Zurada, 1992). These models were inspired by psychological and biological studies of humans performing pattern recognition tasks but quickly evolved into rich mathematical structures having little or no resemblance to biological neural nets.

An artificial neural network consists of a large number of processing elements (neurons) connected by weighted links (synapses). Each processing element has the same simple non-linear functionality and the power of the network comes from its massively parallel structure.

Neural networks have a number of attractive features. They have the ability to adapt to changes in the surrounding environment by adjusting their weights or the activations of the processing elements. It is often claimed, that they are inherently fault tolerant because the processing is distributed among a large number of elements and the failure of one unit little affects the performance of the entire ensemble. Furthermore, neural nets can compute non-linear mapping functions, which is important due to the inherent non-linearity of many physical processes. All these features make neural networks well suited for the purpose of pattern recognition.

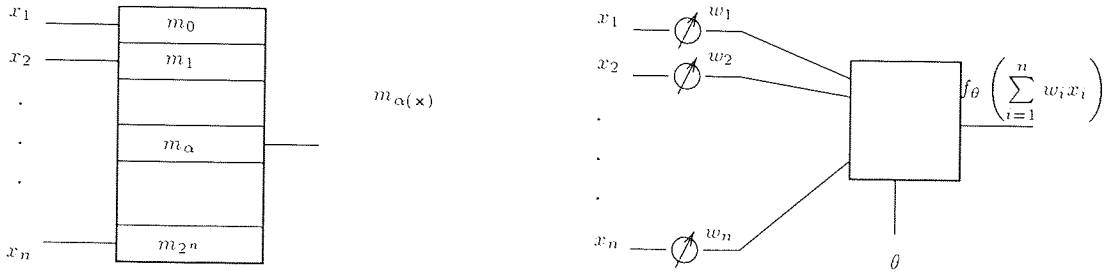


Figure 1.1: Structure of digital and analogue neurons. The digital neuron (RAM node) samples at random n components of the binary input vector $\mathbf{x}$. This n-tuple of bits $x_1, x_2, \ldots, x_n$ can be viewed as an address $\alpha(\mathbf{x})$. The output of the RAM node is equal to the contents of the addressed memory cell. The analog neuron with linear threshold θ calculates the weighted sum of the inputs. If the sum is greater than the threshold θ the output is 1 and 0 otherwise.

1.2 Motivation for Researching RAMnets

RAMnets (Random Access Memory networks) are networks of simple processing elements operating on binary data. Unlike other neural networks, the processing element is a RAM node performing a simple decoding and look-up operations. Each RAM node has n inputs sampled at random from the input vector $\mathbf{x}$. The n-tuple of bits $x_1, x_2, \ldots, x_n$ can be viewed as an address $\alpha(\mathbf{x})$. The output of the RAM node is equal to the contents of the addressed memory cell.

The analog neuron with a threshold θ calculates the weighted sum of the inputs. If the sum is greater than the threshold θ the output is 1 and 0 otherwise (The thresholding function $f_\theta(z) = 1$ if $z > \theta$ and 0 otherwise.) The structure of the digital and analog node is shown in figure 1.1.

The RAM nodes are usually arranged into a one-layer structure and their outputs are added in order to provide an overall output (score). Because of the summation involved, one can argue that the structure is essentially two layer with the second layer performing the addition. However, in the n-tuple community the former view prevails, i.e., only the layers of RAM nodes are counted.

It is interesting, that due to the nonlinear processing carried out by each RAM node, one layered RAMnets can cope with problems that cannot be solved by a single layer of linear threshold units (LTU). It has been demonstrated (Al-Alawi & Stonham, 1992) that the functional capacity (defined as the number of Boolean functions that can be computed by a node) of RAM nodes far exceeds that of LTUs.

One layer RAMnets, unlike analog neural networks, possess a number of features which jointly contribute to their short training times and operational speed:

- The cost function is not used for training.
- Nonlinear processing is accomplished using one layer of nodes.

- There are no weights to be adjusted[1].
- There is no pattern credit assignment problem.

Lack of the cost function. The training algorithm for RAMnets does not involve a cost function. It is non-iterative and each pattern is presented to the network just once. The n-tuple sampling process involves basic operations such as decoding, addressing and storing in the memory which are low level, fast operations well suited for implementation in binary hardware.

RAMnets can be compared with the Multilayer Perceptrons and Radial Basis Function (Hertz *et al.*, 1991; Bishop, 1995), models in which the training algorithm is cost driven. MLPs are notorious for their slow training. The generalised delta rule and its derivatives (scaled conjugate gradient, quasi Newton methods) minimise the cost function iteratively. Calculation of the first or second order derivatives is required to facilitate this task. The training data has to be presented many times to the network so that the algorithm can carry out a search for the minimum of the error function using the gradient information.

RBF models in which the basis functions ϕ_j are combined linearly using weights w_j to provide the network output $\mathbf{y} = \mathbf{W}\mathbf{\Phi}$ can be trained much faster than MLPs. For the quadratic cost function, the weights $\mathbf{W}$ can be found by calculating the pseudo-inverse $\mathbf{\Phi}^\dagger$ and multiplying it by the matrix $\mathbf{T}$ of the target outputs, i.e., $\mathbf{W}^T = \mathbf{\Phi}^\dagger\mathbf{T}$.

It is obvious, that the matrix inversion or the gradient descent are computationally more expensive than the memorisation of simple input data features.

RAMnets are weightless one layer networks One layer RAMnets can be viewed as networks with all weights set to one (which amounts to no weights at all, hence RAMnets are also known as weightless neural networks). Each node performs a simple decoding operation and the output of the network is just a sum of the contents of memory locations addressed by the test pattern.

MLPs as well as RBFs contain the nodes which implement complicated non-linear functions (e.g., Gaussian, sigmoid, hyperbolic tangent) of the inputs. In RBFs there is one and in MLPs two or more layers of adjustable weights on the connections between the nodes. Consequently, propagating the input pattern through the network is more time consuming than in weightless systems with simple units.

Pattern credit assignment Pattern credit assignment problem is concerned with the ability to determine appropriate parameters of the network for the given training data. In MLPs information

[1] In the principle, the memory contents m_a could be viewed as weights but because m_a are binary they have traditionally been viewed as RAM node's activation values. Hence the term "weightless neural networks".

acquired from the data is distributed across the real-valued weights as the result of back-propagating the derivatives of the cost function with respect to the network parameters. It is not possible to determine, for a given training pattern, the contribution it has made to the weight matrix. Therefore, if some patterns are removed from the training set, we have to retrain the system in order to determine the new weights.

This type of problem does not occur in RAMnets. For any training pattern we know exactly what memory locations have been updated. Consequently, given the memory contents of a RAMnet trained with dataset of size N we can infer the state of the system trained on $N - k$ patterns (but we need to keep a $k + 1$ integer tally). One of the benefits of this property is, for example, the ability to perform cross-validation (a method for calculating the error of the network) at recognition time.

There are two main one layer RAMnet architectures: WISARD (Wilkie, Stoneham and Aleksander's Recognition Device) and ADAM (Advanced Distributed Associative Memory). The former was originally proposed by Bledsoe and Browning (Bledsoe & Browning, 1959) and further researched by Aleksander and Stonham (Aleksander & Stonham, 1979) in the context of pattern recognition. It has been successfully applied to inspection and quality assurance, identification and sorting, movement and change detection and robot vision (see chapter 3 for full review). Recently, modified one layer RAMnets have been shown (Allinson & Kolcz, 1995b) to perform well in solving regression problems.

The Associative Neural Architecture ADAM (Austin, 1989a) can be viewed as an extension of the binary RAMnet. It found its application in infra-red image processing (Austin *et al.*, 1993), analysis of document fax images (O'Keefe & Austin, 1994), uncertain reasoning (Austin, 1992; Lees *et al.*, 1995) and in knowledge based systems (Austin, 1991). It can be, as most binary RAMnets, easily implemented in parallel hardware (Austin *et al.*, 1991). Unlike WISARD, ADAM features two stage processing which increases the capacity of the system at a cost of a slight increase in the amount of computation. The first stage associates an n-tuple processed input with a randomly generated class vector (in WISARD this class vector is a deterministic 1-out-of-C pattern class code) and the second stage associates this class vector with the desired output.

It should be kept in mind that the appealing features of one layer RAMnets tend to disappear if the functionality of the RAM node is modified or the units are arranged in multiple layers. There is, in fact, a number of weightless neural network architectures which modify the basic RAM node to extend the functionality of the system. Probabilistic Logic Node (PLN) (Aleksander & Morton, 1995) can store three states: 0, 1 and u (uncertain). If the cell in an uncertain state is addressed, the node outputs 1 with probability 0.5. Other derivatives include probabilistic RAM (pRAM) (Gorse & Taylor, 1993) and the Goal Seeking Neuron (GSN) (Filho *et al.*, 1991). They all can be arranged in

multiple layers and trained using reinforcement algorithms. However, the additional functionality of these systems comes at the cost of the time to learn.

Let us consider as an example a multi-layer PLN network (Al-Alawi & Stonham, 1992) in which the outputs of one layer are tuple sampled by the next one. We note that this architecture (RAM pyramid) cannot be trained with the simple one pass algorithm used in WISARD because the output of the net depends on the activations in the hidden layers. The training algorithm is iterative and requires a repeated presentation of the training data. Initially, all nodes are in the uncertain (probabilistic) state. The training vector is presented to the net and the activations propagated towards the output. If the output of a node in the final layer agrees with the teaching activation, the RAM cells addressed by the teaching input retain their randomly generated values, i.e., they assume deterministic states 0 or 1. If the activation of the output node is different from the teaching output the procedure is more complicated. Essentially, the algorithm tries to match the actual response to the teaching output. This may be achieved by considering all possible output combinations of the uncertain RAM nodes in the previous layers. If this fails, the whole network is punished by putting all the RAM nodes into an uncertain state, the training data is randomly permuted and the algorithm restarted.

Consequently, the RAMnets with an extended architecture are much slower than ADAM or WISARD which implement a one pass training algorithm. In this thesis we will focus on the fast, most basic one-layer WISARD architecture. Although very good results have been reported in a number of applications (Rohwer & Tarling, 1993; Rohwer & Lamb, 1993; Ullmann & Kidd, 1969; Ullmann, 1969) very few studies exist that compare the performance of WISARD with other methods (Rohwer & Cressy, 1989). In fact, Austin ends his review of RAMnets (Austin, 1995) with the following statement:

> "The most useful addition to the future research in RAM based systems would be a comparison of the methods against a set of standard benchmark problems."

In order to fill this gap, we conduct a large scale experimental study of the binary RAMnet using the StatLog data (Michie *et al.*, 1994). The StatLog project was launched to compare 23 classification algorithms, testing them on large-scale and commercially important problems. Building on this research work allows us to compare the performance of the n-tuple classifier with well established models such as Radial Basis Functions, Multilayer Perceptrons or k-Nearest Neighbours.

It is well known that one-layer RAMnets are not universal approximators. This was demonstrated for the intra ex-or function (Austin, 1995). Inability of the net to represent the best solution can be viewed as a model error. Another potential source of the sub-optimality of WISARD architectures is the training algorithm which does not minimise an expected generalisation cost. This can be viewed as an estimation error. Because these two sources of error exist in RAMnets, it is difficult to predict,

for a given dataset, if the method will fail or succeed.

We reexamine the issue of the accuracy/speed trade-off in the n-tuple classifier both experimentally and analytically. We point out the factors determining the speed and sub-optimality of the method.

A fast but sub-optimal method can be competitive for a certain class of problems. It is highly desirable to identify quickly whether a given dataset belongs to this class or not. We develop formal tools that explain why RAMnets perform better on certain datasets than on others and propose a Bayesian framework that allows one to calculate the generalisation cost of regression RAMnets.

1.3 Overview of the Thesis

Chapter 2 contains the definition of the n-tuple method and an overview of the fundamental properties of this model. Alternative RAMnet models as well as algorithms for mapping real valued vectors into bit strings are presented. The issue of saturation in binary RAMnets is also discussed.

Chapter 3 reviews RAMnets and the theoretical tools that can be used to analyse them. RAMnets are compared with well-established models such as Associative Memories, Bayesian Classifiers, and General Regression Networks. A connection to the Walsh expansion and discrete probability estimators is also outlined.

Chapter 4 presents the experimental evidence of the feasibility of RAMnets. The method is tested on 11 real-world datasets used in the StatLog project and the results are compared with other classification methods. A semi-quantitative method is developed which gives a good indication of whether a RAMnet will perform well on the given dataset or not.

Chapter 5 discusses the Frequency Weighted version of the n-tuple classifier and compares its performance with that of the binary RAMnet. The underlying "zero tally problem" is identified and an alternative probability estimation method proposed.

Chapter 6 introduces a Bayesian framework for estimation of the expected cost of using regression RAMnets for noisy interpolation problems. The approach taken can cater for any method and is especially useful for reasoning about models that do not use an explicit cost function during training.

Chapter 7 concludes the thesis with a summary of the results and suggests directions for future research.

1.4 Publications

The content of this thesis represents original research. The work within has not previously appeared elsewhere, with the exception of those research papers produced during the normal course of its preparation.

Material from chapters 3,4,5 was presented at the Weightless Neural Network Workshop 1995. The research work described in chapter 6 is to be presented at the NIPS 96 Conference in December 1996. The following research papers have been published or have been accepted for the publication:

- Rohwer, R., & Morciniec, M. 1995a. Benchmarking the N-Tuple Classifier with StatLog Datasets. *Pages 29 - 34 of:* Bisset, D. (ed), *Proceedings of the Weightless Neural Network Workshop 1995, Computing with Logical Neurons.*

- Rohwer, R., & Morciniec, M. 1995b. Good–Turing Estimation for the Frequentist N-Tuple Classifier. *Pages 93 - 98 of:* Bisset, D. (ed), *Proceedings of the Weightless Neural Network Workshop 1995, Computing with Logical Neurons.*

- Rohwer, R., & Morciniec, M. 1996a. A Theoretical and Experimental Account of the N-Tuple Classifier Performance. *Neural Computation*, **8**(3), 657–670.

- Rohwer, R., & Morciniec, M. 1996b. The Theoretical and Experimental Status of the N-Tuple Classifier. *Neural Networks.* To appear.

- Rohwer, R., & Morciniec, M. 1997. The Generalisation Cost of RAMnets. *In:* Jordan, M.C. Mozer M.I., & Petsche, T. (eds), *Advances in Neural Information Processing Systems*, vol. 9. Morgan Kaufman. To appear.

1.5 Notation

I have tried to develop a consistent notation following generally approved conventions. I use bold face to denote vectors (lower case) and matrices (upper case). All vectors are considered to be column vectors so that, for example, $\mathbf{x}^{\mathrm{T}} = (x_1, x_2, \ldots, x_d)$. In general, I use a subscript to denote components of a more complex object and a superscript to list a number of objects of the same type. The element of a matrix $\mathbf{A}$ will thus be denoted as A_{ij} (with the first subscript referring to a row) whereas a set of training vectors will be $\{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\}$. The Euclidean length of a vector $\mathbf{x}$ will be referred to as $\|\mathbf{x}\|$, the Hamming length of a binary vector $\mathbf{x}$ will be symbolised with $|\mathbf{x}|_H$ (the Hamming distance between two vectors as $H(\mathbf{u}, \mathbf{v})$) in order to distinguish it from the Manhattan distance $|\mathbf{x}|_M$ (also

denoted as $M(\mathbf{x}, \mathbf{y})$). I use $\det[\mathbf{A}]$ to stand for a determinant of a matrix $\mathbf{A}$ and $\mathrm{tr}[\mathbf{A}]$ to denote the trace of a matrix $\mathbf{A}$. The Kronecker delta symbol appears quite often and has the usual meaning $\delta_{i,j} = 1$ if $i = j$, $\delta_{i,j} = 0$ otherwise. The letter P denotes discrete probability while p is reserved for continuous probability density. The expectation of a random variable y is denoted as $\mathcal{E}[y]$ and the variance as $\mathrm{var}[y]$.

The most important objects and the corresponding notation are gathered in the table below.

Symbol	Meaning
n	The size of a tuple
T	The number tuples
d	The dimension of the original input space
R	The dimension of the discretised input space
$\alpha_k(\mathbf{u})$	The address generated by the k-th tuple for a pattern $\mathbf{u}$
$\eta(k, j)$	The input vector's component corressponding to the j-th bit in the k-th tuple
$m_{ci\alpha}$	The memory contents at discriminator c, i-th RAM node, address α in that node
D^{trn}	The number of training patterns
D_c^{trn}	The number of training patterns of class c
$\mathcal{D}^{trn}$	The training dataset
D^{tst}	The number of test patterns
$\mathcal{D}^{tst}$	The testing dataset
$\boldsymbol{x}^i$	The i-th input vector
x_j^i	The j-th components of the i-th input vector
y^i	The i-th teaching output
y_c	The output of the c-th discriminator
ρ	The Tuple distance
$\rho^{(K)}$	The Kanerva distance
a	Bit resolution of the CMAC code substrings
K	CMAC code threshold of Hamming-Manhattan distance proportionality

Table 1.1: Notation

Chapter 2

N-Tuple Classification Method

2.1 Introduction

This short chapter introduces the n-tuple method of classification. Firstly, an informal description of how it operates is given, then a suitable notation is developed in order to allow us to reason about this model in a more formal way. In section 2.4 we present two tuple mapping schemes: random and exclusive and discuss their suitability for the classification RAMnets.

Various extensions to the basic binary RAMnet can be easily obtained by manipulating the tally information in different ways. A number of WISARD based architectures is briefly presented in section 2.5.

RAMnets process binary valued data. If the inputs are real-valued vectors they must be transformed into the binary domain. Suitable mapping algorithms are presented in section 2.4.

An important issue in the binary RAMnets is the memory saturation which occurs for small values of the parameter n. In section 2.7, we give upper and lower bounds on the expected saturation level for an arbitrary n, given the saturation of the system comprising one-tuples.

2.2 Informal Description of RAMnets

N-tuple models have been originally developed for image classification (Bledsoe & Browning, 1959; Ullmann & Kidd, 1969). Let us consider, as an example, a simple image recognition task of distinguishing uppercase letters. Suppose that the images of letters have somehow been captured, rescaled to fit into a rectangular area of size R and binarised. The data is then divided into two datasets: one used for training and the other for testing the model's performance. Images in the training set are hand labeled with the correct class, whereas the classifier is expected to provide a correct label for

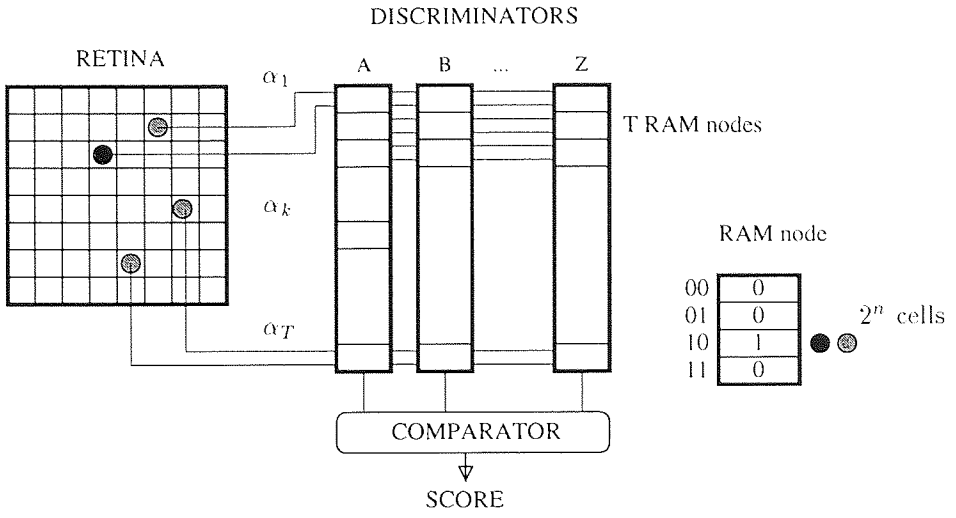


Figure 2.1: The Architecture of the Binary RAMnet.

the patterns in the test set.

In order to accomplish this task, RAMnet system makes a number, say T, of measurements $\boldsymbol{\alpha}(\mathbf{v}) = \{\alpha_1(\mathbf{v}), \ldots, \alpha_i(\mathbf{v}), \ldots, \alpha_T(\mathbf{v})\}$ of the unknown pattern $\mathbf{v}$. The measurements are randomly chosen subsets of n pixels. The value of the i-th measurement (feature), $\alpha_i(\mathbf{v})$, is a positive integer in range $[0, 2^n - 1]$. The original binary pattern $\mathbf{v}$ of length R is therefore represented by a T dimensional feature vector $\boldsymbol{\alpha}(\mathbf{v})$.

The RAMnet stores information about which *components* of the feature vector occurred in the training sample (which requires only $2^n \times T$ bits per class). Consequently, the storage requirements for the system are substantially reduced (naive lookup table of patters would call for as much as 2^R bits per class). On the other hand, because the capacity of this "feature space" is smaller than that of the original binary space, a number of different input patterns will have exactly the same representation in the feature space and it will not be possible to reconstruct the original patterns from the information available. This can, however, be an advantage if similar patterns are mapped into the same (or a similar) feature vector. (This procedure imposes a smoothness in the tuple space.) When trained with a set $\mathcal{D}^{trn}$ of training vectors $\{\mathbf{u}^1, \ldots, \mathbf{u}^{D^{trn}}\}$ the system should be able to generalise and classify correctly a set $\mathcal{D}^{tst}$ of test vectors $\{\mathbf{v}^1, \ldots, \mathbf{v}^{D^{tst}}\}$ larger than the training set.

RAMnets store the information about the components of the feature vector $\boldsymbol{\alpha}(\mathbf{u})$ separately for each pattern class c in the memory blocks called discriminators. The i-th component $\alpha_i(\mathbf{u})$ is stored in the i-th RAM node within the given discriminator. Broadly speaking, trained discriminators contain templates of the patterns representative for a given class. The classification is accomplished by selecting the best fitting template for a given test pattern $\mathbf{v}$.

The architecture of the RAMnet is extremely simple. As shown on figure 2.1 it consists of the retina (binary array of the size of the images), tuples, memory and a comparator. Before the training commences T tuples of size n are attached to the retina. The number of tuples is usually selected so as to ensure a good coverage of the retina. A tuple is simply a subset of n pixels selected at random from the retina.

The ordered set of pixels can be viewed as a feature of the pattern or an address ranging from 0 to $2^n - 1$ (there are 2^n possible arrangements of black and white pixels). The address, with a value say α, can then be used to access a memory node (comprising 2^n cells) in order to store some information about the observed feature. Two 2-tuples are shown in figure 2.1. One of them samples a black and a white pixel and has the corresponding tuple address $\alpha_1 = 2_{(10)} = 10_{(2)}$. The other tuple samples two white pixels and generates the address $\alpha_T = 0_{(10)} = 00_{(2)}$.

For each i-th n-tuple there are as many memory nodes, say c, as there are classes of patterns to be distinguished (in our example $c = 26$). These RAM nodes are supplied with the same address α_i by the i-th tuple. A collection of RAM nodes associated with the same class is often referred to as a discriminator because after training it contains information that allows us to discriminate between pattern classes. In total $2^n \times c \times T$ memory cells are allocated for a RAMnet with T n-tuples. The memory is reset (all cells contain a 0) just before the training commences. Note, that once the tuple mapping to the retina is established it is not changed during the operation of the classifier (a new mapping would require retraining of the system).

RAMnets are adaptive systems and have the ability to learn, given the teaching input (e.g., image of a letter) and output (a class label, for example, "A"). Training requires only one pass through the data set. Patterns are projected (copied) onto the retina and processed by n-tuples in parallel. The set of addresses $\{\alpha_1, \ldots, \alpha_i, \ldots, \alpha_T\}$ is generated and supplied *only* to the discriminator corresponding to the teaching output. All addressed memory cells are set to 1.

Suppose that the RAMnet pictured in figure 2.1 is in the training mode. The first tuple sampled the training pattern $\mathbf{u}$ representing a letter "A" and generated an address $\alpha_1 = 10_{(2)}$. This address is used to set a 1 in the first RAM node within the discriminator A. The memory contents in other discriminators is not affected.

The process of tuple address generation and memory updating is carried out for all the training patterns. If a memory cell is selected more than once it does not change its contents[1] and retains a 1.

Intuition tells us that patterns from the same class will tend to "overlap on tuples", i.e., they will share a significant proportion of the addresses generated for them. Therefore, we expect that a pattern "A" will access more cells containing a 1 in the discriminator associated with the correct class

[1] Other RAMnet architectures store a full tally of cell's accesses made during training allowing one to estimate the probabilities of features. We discuss such models in section 2.5.

than with any other.

Similarity of the patterns to the class templates stored in discriminators is determined during the test phase. The patterns with unknown class assignment are processed by n-tuples. This time, however, tuple addresses are supplied to *all* discriminators. For each one of them the number of non-zero cells addressed is calculated. The resulting numbers ranging from 0 to T, the class scores, can then be compared and the pattern is assigned to the class with the highest score.

2.3 Formal Description of RAMnets

The informal description given above will now be formalised. Introducing a suitable notation will allow us to reason about the model in a more organised and principled manner. We will also generalise the original binary n-tuple method to other RAMnet architectures such as the Frequency Weighted version.

The n-tuple recognition method is used to classify patterns, which are bit strings of a given length R. If the inputs are real-valued vectors, say $\mathbf{x} \in \mathbb{R}^d$ they must be mapped onto a binary hypercube $\mathbb{H}^R$.

Efficient methods exists (see section 2.6) for converting real scalars into bit strings in a way that preserves the metric properties likely to be relevant to generalisation for pattern vectors over the reals. Therefore, no essential limitation is introduced by the requirement that patterns must be bit strings.

Several (let us say T) sets of n distinct[2] bit locations are selected randomly. These are the n-tuples. The restriction of a pattern to an n-tuple can be regarded as an n-bit number α which, together with the identity of the n-tuple, constitutes a feature of the pattern. The standard n-tuple recogniser operates simply as follows:

> *A pattern is classified as belonging to the class for which it has the most features in common with a union of features generated by the training patterns of that class.*

Precisely, the class assigned to unclassified pattern $\mathbf{v}$ is

$$\underset{c}{\operatorname{argmax}} \left(\sum_{i=1}^{T} \Theta_\theta \left(\sum_{\mathbf{u} \in \mathcal{D}_c^{trn}} \delta_{\alpha_i(\mathbf{u}),\alpha_i(\mathbf{v})} \right) \right) \tag{2.1}$$

where $\mathcal{D}_c^{trn}$ is the set of training patterns in class c,

$$\Theta_\theta(x) = \begin{cases} x & \text{if} \quad 0 \leq x < \theta \\ \theta & \text{if} \quad x \geq \theta, \end{cases}$$

[2]Relaxing the requirement that a given n-tuple has n *different* bit locations amounts to introducing a mixture of differently sized n-tuples. Note the restriction does not disallow a single pattern component from being shared by more than one n-tuple.

$\delta_{i,j}$ is the Kronecker delta ($\delta_{i,j} = 1$ if $i = j$ and 0 otherwise) and $\alpha_i(\mathbf{u})$ is the i-th feature of pattern $\mathbf{u}$:

$$\alpha_i(\mathbf{u}) = \sum_{j=0}^{n-1} u_{\eta(i,j)} 2^j. \tag{2.2}$$

Here u_k is the k-th bit of $\mathbf{u}$ and $\eta(i,j)$ is the bit position in $\mathbf{u}$ corresponding to j-th bit location of the i-th n-tuple. The system discussed in section 2.2 corresponds to the version with $\theta = 1$.

With c classes to distinguish, the system can be implemented as a network of Tc RAM nodes. As a result of the training procedure, the memory content $m_{ci\alpha}$ at address α of the i-th node allocated to class c is set to

$$m_{ci\alpha} = \Theta_\theta \left(\sum_{\mathbf{u} \in \mathcal{D}_c^{trn}} \delta_{\alpha, \alpha_i(\mathbf{u})} \right). \tag{2.3}$$

Thus, $m_{ci\alpha}$ is set to a non-zero value if any pattern of $\mathcal{D}_c^{trn}$ has feature α and set to zero otherwise.

Recognition is accomplished by tallying the set bits in the nodes of each discriminator at the addresses given by the features of the pattern to be classified. The output $\mathbf{y}$ of the classifier is a c dimensional integer-valued vector with components

$$y_c = \sum_{i=1}^{T} m_{ci\alpha_i(\mathbf{v})}. \tag{2.4}$$

Finally, the pattern $\mathbf{v}$ is assigned to class with the highest output y_c

$$\underset{c}{\text{argmax}} \; y_c. \tag{2.5}$$

2.4 Tuple Mapping

During the construction of the n-tuple network, the subsets of n bits are randomly selected. For the i-th tuple we have a positive integer-valued mapping function $\eta(i,j)$ which returns the number (between 1 and R) of a bit in an input bit string $\mathbf{u}$ which corresponds to the j-th bit of the i-th tuple, $\mathbf{u}_{\eta(i,j)} = \alpha_j^i(\mathbf{u}), \forall i, j$. Here α^i is the binary vector representation of the i-th positive-integer-valued component α_i of the address vector $\boldsymbol{\alpha}$.

Let $\boldsymbol{\eta}$ denote a vector with T components, so that the i-th component is equal to the i-th tuple mapping function $\eta(i,\cdot)$. The function's vector $\boldsymbol{\eta}$ will be referred to as a tuple mapping. Once the mapping has been selected, it becomes permanent for the entire life of the n-tuple classifier.

There are two main types of random mappings used in RAMnets: exclusive and random. In the former mapping type, each bit of the retina is sampled at most once. This constrains the number of tuples to at most $\lfloor R/n \rfloor$. Up to $n-1$ bits may not be sampled at all, depending on the retina size R and tuple size n. A possible theoretical advantage of an exclusive mapping is that the tuples can be treated as independent assuming that the pixels in the retina are uncorrelated.

Random mapping allows a retina bit to be sampled by more than one tuple. The number of tuples T is practically unlimited, but there are at most $\binom{R}{n}$ different tuples. It is possible that an n-tuple samples just one bit n times (becoming 1-tuple), or that two n-tuples sample the same subset of n pixels (making one of them redundant). Clearly, this sampling procedure introduces correlations between tuples. Therefore, even assuming that the pixels are uncorrelated (which is clearly implausible in reality) one could not trivially calculate the distribution of the tuple score.

The advantage of a random mapping is that a large number of tuples can be taken even for a moderate retina size R delivering a better statistic. Furthermore, because all the pixels are sampled uniformly for each tuple, the analytical form of the statistic is simpler than for the exclusive mapping.

Therefore, the use of an exclusive mapping in RAMnets applied to pattern recognition appears to be guided more by a tradition than the theoretical considerations. The reason for this can be traced back to the paper by Bledsoe and Browning (Bledsoe & Browning, 1959) in which they state the following:

> "Some experiments were made in which non-exclusive n-tupling was used for the photocells. [...] non-exclusive pairing resulted in some improvement in the percent of characters recognized. But this improvement was at the expense of more storage space and longer computing time. We feel that a larger gain in percent recognized can be realized, for the same amount of storage and same length of computing time, by increasing the number of photocells (N) and continuing to use exclusive n-tuples. In other words, we see no real advantages in non-exclusive grouping."

This statement is inaccurate as the exclusive mapping captures much less pixel correlation information than random sampling and the benefits of the simplified formal analysis for the latter mapping are significant as will be demonstrated in section 3.3.2.

2.5 Modifications of the Original RAMnet Architecture

A number of extensions to the original method have been proposed. All of them allow frequencies of tallies to be stored. They differ in the way that the tally information is manipulated. One of the earliest modifications introduced was the so called Frequency Weighted n-tuple system where instead of binary information, frequencies of features are stored. This procedure amounts to setting the tally truncation variable θ to infinity in equations 2.1 and 2.3. We review this version in sections 3.2 and 3.4.2.

It is tempting to use normalised tallies so that the memory contents can be viewed as an estimate of the probability. For example, the ratio $m_{ci\alpha} / \sum_{\alpha} m_{ci\alpha}$ is a Maximum Likelihood estimate of the probability $P(\alpha_i|c)$ of the i-th feature α_i conditioned on class c.

Assuming that features are independent, the output of the classifier can be related to the condi-

tional class probability $P(c|\boldsymbol{\alpha})$:

$$P(c|\boldsymbol{\alpha}) \approx P(c) \prod_i \frac{P(\alpha_i|c)}{P(\alpha_i)}. \tag{2.6}$$

This RAMnet version is interesting because of its relationship to the well understood Bayes Classifier (see section 3.4.2). Normalised Frequency Weighted RAMnets can employ other methods of estimation than Maximum Likelihood. We discuss this issue in chapter 5.

Yet another interesting modification makes it possible to apply RAMnets to regression problems or as a function interpolator. For such problems we require two sets of tallies: weights $m_{k\alpha_k(\mathbf{u}^i)}$ that accumulate the teaching outputs y^i at the addresses generated by the training inputs $\mathbf{u}^i$ and tallies $r_{k\alpha_k(\mathbf{u}^i)}$ which are incremented if training input $\mathbf{u}^i$ generates address α for the k-th tuple.

The network's output $y(\mathbf{v})$ for an input test vector $\mathbf{v}$ is calculated as a ratio

$$y(\mathbf{v}) \approx \mathcal{E}[y|\mathbf{v}] = \frac{\sum_{k=1}^{T} m_{k\alpha_k(\mathbf{v})}}{\sum_{k=1}^{T} r_{k\alpha_k(\mathbf{v})}} \tag{2.7}$$

which can be shown to approximate the conditional regression $\mathcal{E}[y|\mathbf{v}]$ of the output given an unknown input $\mathbf{v}$. This RAMnet is known as the n-tuple regression network and is reviewed in detail in section 3.5. We calculate the expected generalisation cost of this method in chapter 6.

2.6 Input Encoding

In many practical applications the input data is real-valued rather than binary. Because a RAMnet can process only binary vectors $\mathbf{u} \in \mathbb{H}^R$, the original vectors $\mathbf{x} \in \mathbb{R}^d$ must somehow be encoded using a suitable mapping function $M : \mathbb{R}^d \rightarrow \mathbb{H}^R$, $M(\mathbf{x}) = \mathbf{u}$. The mapping should ideally preserve the metric properties of the original space (Allinson & Kołcz, 1994b), because the distances between the inputs are likely to affect the generalisation properties of the classifier. (We use Manhattan distance as the the original metric because it can be computed efficiently.) On the other hand, it is desirable to keep the dimensionality R of the binary input vector $\mathbf{u}$ low, so that it can be sampled with relatively few tuples. Amongst the encoding schemes discussed, only the thermometer code (see section 2.6.1) ensures that the Hamming distance is proportional to the Manhattan distance for all the inputs. Other mapping schemes such as CMAC (Allinson & Kołcz, 1994a) provide a limited proportionality of distances in the $\mathbf{x}$ and $\mathbf{u}$ space, but they lead to shorter codes.

2.6.1 Thermometer Code

One of the simplest mapping schemes is the thermometer code. The idea is to quantise uniformly the dynamic range of the attribute domain into a number, say R/d, of bins and assign to each component

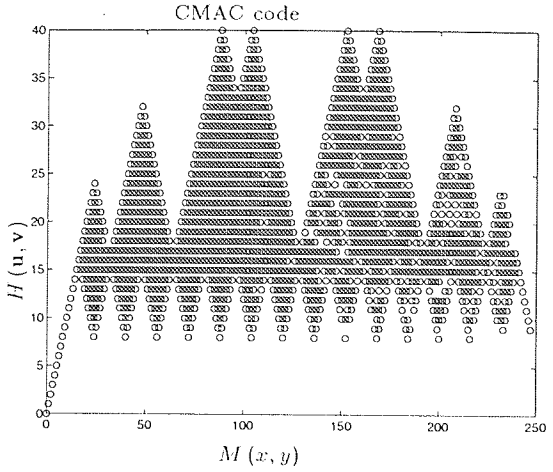


Figure 2.2: Hamming distance $H(\mathbf{u}, \mathbf{v})$ between two CMAC/Gray-transformed integers vs. their Manhattan distance $M(x, y)$, for 3×10^4 randomly chosen pairs of integers. $K = 8$ and $a = 5$. The relationship is linear for Hamming distances up to K, and the transformed distance is bounded below by K for greater Hamming distances.

x_j^i of the i-th input vector $\mathbf{x}^i$ a bit string $\mathbf{u}^j$ with the first $\frac{Rx_j'}{d \max_k x_j^k}$ bits set to one and those remaining set to zero. The R dimensional binary vector $\mathbf{u}$ is obtained by concatenating together the d bit strings $\mathbf{u}^j$. The advantage of this coding scheme is that Hamming distances between the binary vectors $\mathbf{u}$ are proportional to the Manhattan distances between the inputs $\mathbf{x}$ within a given quantisation resolution R/d. However, if the dynamic range or dimensionality of the inputs is large, the code length may become impractical.

2.6.2 CMAC Code

A memory-efficient method tailored to the Hamming distance underlying RAMnet generalisation (see section 3.3.1) exists (Allinson & Kołcz, 1993; Allinson & Kołcz, 1994a), that constitutes a mixture of CMAC (Albus, 1975) and Gray coding techniques.

This encoding scheme operates on positive integers so the components x_j of the input vector $\mathbf{x}$ must first be rescaled appropriately. Each component is encoded separately and the resulting binary strings $\mathbf{u}_j$ are concatenated together to yield the pattern vector $\mathbf{u}$. The prescription for encoding one component with integer value x is to concatenate K bit strings, the k-th of which (counting from 1) is $\frac{x+k-1}{K}$, rounded down and expressed as a Gray code. The Gray code of an integer i can be obtained as the bitwise exclusive-or of i (expressed as an ordinary base 2 number) with $i/2$ (rounded down). This provides a representation in aK bits of the integers between 0 and $(2^a - 1)K$ inclusive, such that if integers x and y differ arithmetically by K or less, their codes $\mathbf{u}(x)$ and $\mathbf{v}(y)$ differ by Hamming distance $H(\mathbf{u}(x) - \mathbf{v}(y))$, and if their Manhattan distance is K or more, their corresponding Hamming distance is at least K.

(a=3, K=4)	$k=1$	$k=2$	$k=3$	$k=4$
x=26	$\frac{26+1-1}{4}=6$	$\frac{26+2-1}{4}=6$	$\frac{26+3-1}{4}=7$	$\frac{26+4-1}{4}=7$
x=27	$\frac{27+1-1}{4}=6$	$\frac{27+2-1}{4}=7$	$\frac{27+3-1}{4}=7$	$\frac{27+4-1}{4}=7$
x=28	$\frac{28+1-1}{4}=7$	$\frac{28+2-1}{4}=7$	$\frac{28+3-1}{4}=7$	$\frac{28+4-1}{4}=7$
x=29	$\frac{29+1-1}{4}=6$	$\frac{29+2-1}{4}=6$	$\frac{29+3-1}{4}=7$	$\frac{29+4-1}{4}=8$

Table 2.1: Illustration of the CMAC encoding for the parameter values $a=3$ and $K=4$. The valid integer range is $(2^a-1)K=28$ and the code word for the integer x comprises 4 substrings of length 3. The total code length is $aK=12$ bits. The entries of the table are integer values of $\frac{x+k-1}{K}$ rounded down. These values are then Gray encoded and concatenated. For the out of range value $x=29$ the code word is 101 101 100 100. This is because the Gray code of an integer $x=8$ is 1100 and after truncation to 3 bits we have 100. The consequence of the Gray coding and truncation is that the increasing out of range integer values get mapped into decreasing in range values, i.e., $x=8$ has the same truncated Gray string as $x=7$, $x=9$ has exactly the same code as $x=6$ and so on.

The fact that the code preserves the Manhattan distance values up to K is illustrated in figure 2.2. An example of the CMAC code calculation appears in table 2.1. More comprehensive illustrations are given by (Allinson & Kołcz, 1993).

The size of the subspace where the Hamming distance is proportional to the Manhattan distance can be adjusted by changing the parameter K. For $a=1$ the CMAC code is equivalent to a thermometer code with the resolution of K bits. Increasing the value of the parameter a for the constant dynamic range $(2^a-1)K$ results in fewer code bits per attribute (if a increases, K must be reduced dramatically, hence the smaller overall code length $R=aKd$) at the expense of reducing the region of distance proportionality.

2.6.3 Minchinton Cells Code

Minchinton cells (Bishop *et al.*, 1990) were originally invented for processing gray level images with n-tuple networks. The Minchinton encoding scheme is suitable for high dimensional input data. The idea is to use R digital comparators (cells) which sample randomly a number (in the simplest case two) of components of the input pattern $\mathbf{x}$ and provide an output u_j. An arbitrary Boolean relation is then introduced, for example $x_j > x_k$, to describe the activity of the cell. The binary activity pattern $\mathbf{u}$ of all R Minchinton cells is a code for the input $\mathbf{x}$.

Research into the Minchinton encoding has been limited to the case of very high dimensional inputs (gray level images) and uniform Boolean relations of order two in the form of inequalities. Clearly, low dimensional data requires a number of different boolean relations because the number of Minchinton cells with different samplings is limited ($\binom{d}{2}$ for cells sampling two components).

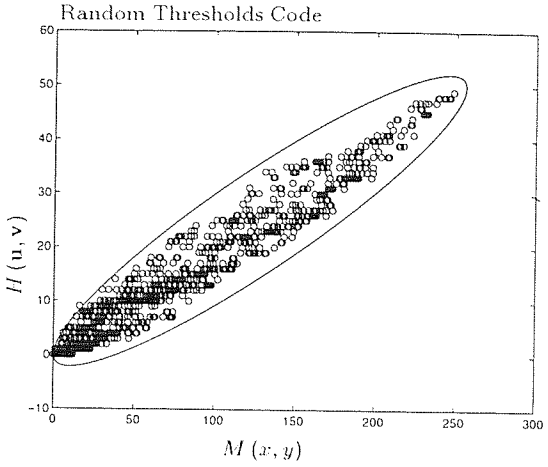


Figure 2.3: Hamming distance $H(\mathbf{u}, \mathbf{v})$ between two random threshold encoded integers vs. their Manhattan distance $M(x, y)$, for 3×10^3 randomly chosen pairs of integers. $d = 1$ and $a = 50$. The relationship is quasi-linear. Given that the Manhattan distance is M the expected Hamming distance is $\mathcal{E}[H] = aM$ with the standard deviation $\delta H = \pm\sqrt{aM(1-M)}$. The ellipse denotes the region within 3 standard deviations.

2.6.4 Gray N-Tuple Processing

A different scheme for processing gray level images was proposed by Austin. Instead of mapping integer pixel intensities into binary strings and using a binary n-tuple process defined in section 2.3 he proposed (Austin, 1988) an entirely new n-tuple technique (gray tuples). As it is very different from the binary RAMnet which we study and because it cannot be applied to lower dimensional data (most of the datasets we use have dimensionality of order 9–57) we will not elaborate on it and direct the reader to the above paper.

2.6.5 Random Thresholds Code

This new encoding scheme is a version of Minchinton cells code for low dimensional data. It introduces a set of thresholds $\{t_1, \ldots, t_k, \ldots, t_a\}$, for each component x_j of the input vector $\mathbf{x}$. The thresholds are chosen at random and cover the dynamic range of the component x_j. The actual value of the component is then compared with a thresholds yielding a bit string $\mathbf{u}_j$ of length a with bits denoting the relation $t_k > x_j$. The bit strings $\mathbf{u}_j$ are concatenated together yielding the pattern code vector $\mathbf{u}$ of length $R = da$. This encoding scheme is equivalent to a thermometer code if the thresholds are placed at monotonically increasing intervals. With random thresholds the relationship between the Manhattan distance and Hamming distance is quasi-linear throughout the dynamic range of the input. Given that the Manhattan distance of the j-th components of $\mathbf{x}$ and $\mathbf{y}$ is $M_j = |\mathbf{x}_j - \mathbf{y}_j|_M$, the expected Hamming distance between the resulting bit strings $\mathbf{u}(\mathbf{x})$ and $\mathbf{v}(\mathbf{y})$ is $\mathcal{E}[H(\mathbf{u}, \mathbf{v})] = a\sum_j M_j$ with standard deviation $\delta H(\mathbf{u}, \mathbf{v}) = \pm\sum_j \sqrt{aM_j(1-M_j)}$. (Consider just one attribute rescaled into the

real line $[0, 1]$ and two attribute values x and y. The Manhattan distance $M = |x - y|_M$ corresponds to the length of the line segment, which can also be interpreted geometrically as probability. If a threshold t_k falls onto the line segment, the k-th bit in the Hamming vector $\mathbf{u}(x) \oplus \mathbf{v}(y)$ will be one, otherwise it will be zero. If the thresholds are chosen at random this procedure is a binomial trial with the probability M, sample size a and number of "successes" H. If more than one attribute is processed with a independently selected thresholds, this procedure generalises as indicated above). This relationship is illustrated for one attribute case on figure 2.3.

2.7 Memory Saturation in Binary RAMnets

A well known phenomenon (Bledsoe & Browning, 1959; Rohwer & Tarling, 1993) occurring in the binary RAMnet systems is memory saturation. It occurs because the bits are never reset and for small enough n and a large number of training vectors all memory cells will eventually be filled out. This results in the loss of the discrimination power and consequently worsened performance of the recogniser.

The saturation level can be changed in two ways: by varying the amount of training data or the tuple size n. By increasing the latter we are able to decrease the saturation level for a given training set. The issue of saturation is interesting because it affects the performance of the classifier. High saturation levels decrease the ability of the system to discriminate between patterns, whereas very low saturation impairs the generalisation properties of the recogniser.

We will now give a bound on the average saturation as a function of the tuple size n. The saturation $s_c(n)$ of c-th discriminator is defined as the ratio of non-zero RAM locations to the total number of cells

$$s_c(n) = \frac{1}{T2^n} \sum_{i=1}^{T} \sum_{\alpha=0}^{2^n-1} m_{ci\alpha}. \tag{2.8}$$

The average saturation $s(n)$ for the whole RAMnet, as a function of n, is defined as

$$s(n) = \sum_{c} P(c)\, s_c(n). \tag{2.9}$$

Let us assume that the saturation level for $n = 1$ is known. We would like to extrapolate $s(n)$ for any n. Note, that the RAM nodes associated with 1-tuples can have either all the cells set or one zero and one non-zero cell. If all the cells of the node are set it means that the pixel sampled by this tuple changed its value during training. A node with just one zero location corresponds to a pixel that did not change its value at all. The latter pixels will be called stationary and the former non-stationary pixels.

Stationary pixels cause 50% saturation in the tuple's RAM node as opposed to 100% saturation induced by a non-stationary pixel. Given the saturation $s_c(1)$ caused by patterns from class c, we can calculate the number of stationary pixels sampled by T one-tuples as $2T(1 - s_c(1))$. The maximum likelihood estimate q_c of the probability that a randomly selected pixel in patterns of class c will be a stationary one is

$$q_c = 2(1 - s_c(1)) \tag{2.10}$$

and the average probability for all the discriminators is defined as

$$q = \sum_c P(c)\, q_c. \tag{2.11}$$

The probability that an n-tuple samples k stationary points is determined by the binomial distribution with the parameters k, n, q (If we do not let the tuple to sample the same pixel more than once the distribution is hypergeometrical.)

$$P(n,k) = \binom{n}{k} q^k (1-q)^{n-k}. \tag{2.12}$$

Parameter k induces a partition of the set of all tuples into $n+1$ subsets. Each n-tuple in the k-th partition samples k stationary pixels. For the non-exclusive tuple mapping (binomial sampling) the expected number $T(n,k)$ of tuples in the k-th partition is equal to

$$T(n,k) = \sum_{i=0}^{T} i \binom{T}{i} P(n,k)^i \left(1 - P(n,k)\right)^{T-i} = TP(n,k) \tag{2.13}$$

Given the average saturation $s(n,k)$ of a tuple from the k-th partition the total average saturation can be expressed as

$$s(n) = \frac{1}{T} \sum_{k=0}^{n} T(n,k) s(n,k) = \sum_{k=0}^{n} P(n,k)\, s(n,k). \tag{2.14}$$

The major problem is thus the estimation of $s(n,k)$. As we increase n, it is clearly impossible to determine the number of stationary pixels that a given n-tuple will sample, without knowing pixel correlations. However, upper and lower bounds for this quantity can be given. Let us first consider the upper bound on the saturation in a tuple sampling k stationary pixels. We observe that if k pixels do not change their value during training then k bits in the tuple address α are fixed, reducing the number of cells that can be addressed to at most 2^{n-k} which gives the upper saturation bound on $s(n,k)$ equal to $1/2^k$. The lower bound can also be easily found. If all n pixels happen to be stationary then only one RAM cell will have a non-zero value giving the lower bound of $1/2^n$. On the other hand, if k pixels are stationary and the remaining fully correlated than at most two cells can be addressed by the training data and this bound is increased to $1/2^{n-1}$. Consequently, the bounds on the total saturation $s^{lo}(n) \leq s(n) \leq s^{hi}(n)$ can be found to be

$$s^{lo}(n) = \frac{2 - P(n,n)}{2^n} \qquad s^{hi}(n) = \sum_{k=0}^{n} \frac{1}{2^k} P(n,k)$$

Saturation is equal to the lower bound if the pixels are fully correlated, i.e., non-stationary pixel changes are synchronised or if all of them are stationary. High saturation occurs for uncorrelated pixels because they give rise to different tuple addresses.

Figure 2.4 shows upper and lower bounds as well as the actual saturation level as a function of the tuple size n for a number of datasets (these will be discussed in chapter 4). In most cases the actual saturation value for a moderate n lies in the middle of the bounding curves indicating moderate pixel correlation. However, the individual bits for the given tuple addresses of patterns in DNA dataset appear to be uncorrelated. The Technical set is even more interesting because the upper and lower bounds almost overlap. This is because almost all the pixels sampled by the n-tuples are stationary (the measured 1-tuple saturation is close to 50%). In this case $P(n,n)$ is close to 1 and both bounds approach the $1/2^n$ curve. We also observe that as n increases the actual saturation curve approaches the lower bound curve. This effect is due to the finite amount of the training data.

We note that the bounds are tight for fully correlated pixels and diverge with increasing pixel decorrelation indicating that the correlation information is required to infer the saturation level. In other words, knowing the saturation level is as difficult as predicting the system's performance.

Figure 2.5 demonstrates how the performance of the classifier depends on saturation (and the tuple size n). The prevailing tendency is that the performance improves as the saturation decreases (which corresponds to the increase in the tuple size n). Moreover, the error-bars on all the curves decrease with saturation, i.e., systems performance becomes consistent with n increasing.

2.8 Summary

We have introduced the main ideas behind the n-tuple method of classification. The architecture of the system was explained and a suitable notation allowing its formal description was introduced.

RAMnets can implement two main types of mapping: random and exclusive. The first type of mapping is preferred because, as will become apparent in the next chapter, the underlying pixel sampling process is binomial and the number of tuples is not limited. This simplifies considerably the analysis of the statistical properties of the system.

RAMnets can be applied to classification problems involving real-valued inputs. The mapping of reals into the binary space should ideally ensure the proportionality of the Manhattan distances of inputs to the Hamming distances of their binary codes. On the other hand, the code strings should be as short as possible. These two properties can usually be traded-off using a parameter (K for CMAC, a for random thresholds code).

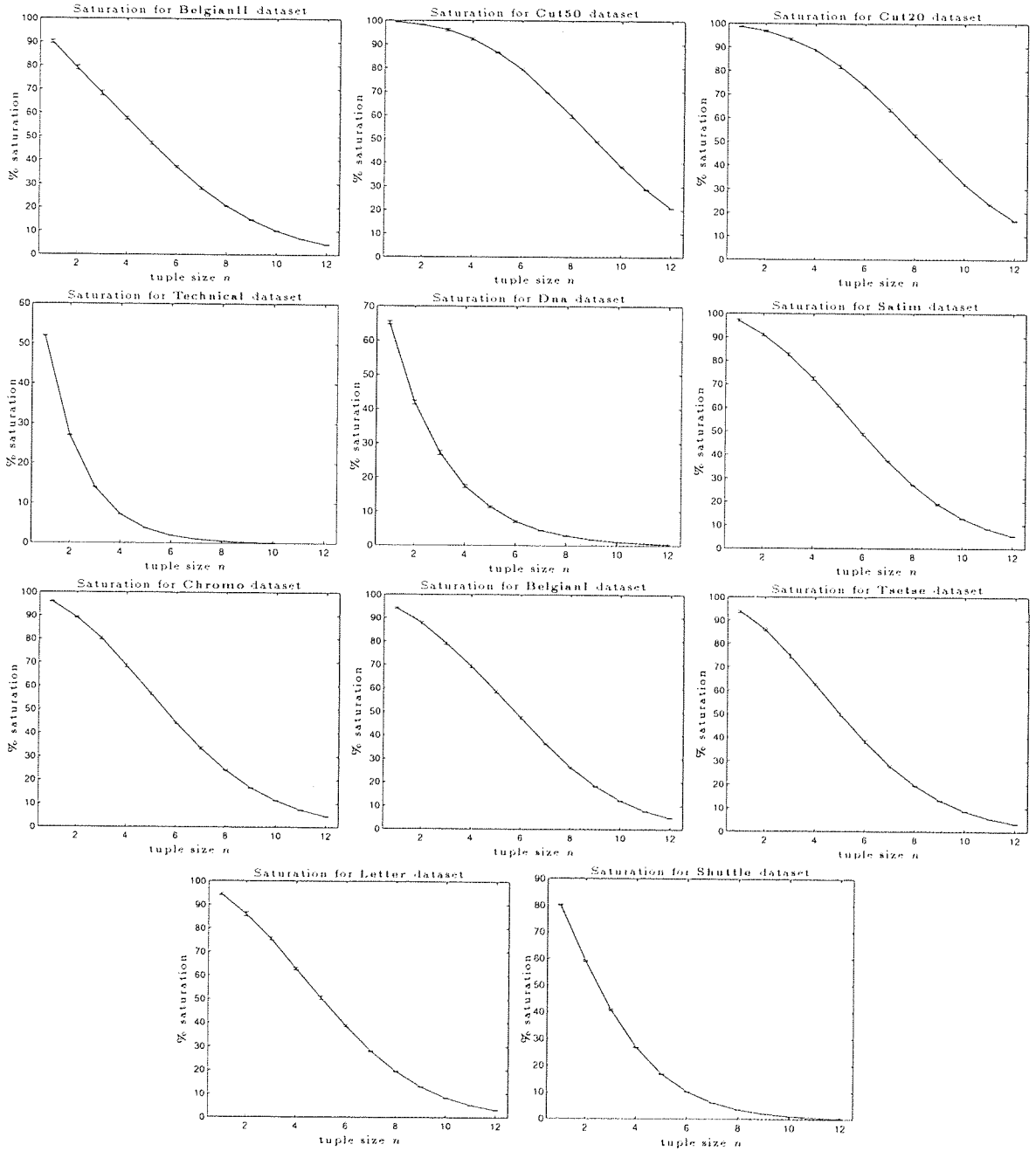


Figure 2.4: Upper and lower saturation bounds (dotted lines) with actual mean saturation level (solid line) as a function of n computed for a number of datasets. The one standard deviation error-bars were computed for 10 runs using a binary RAMnet with 1000 tuples.

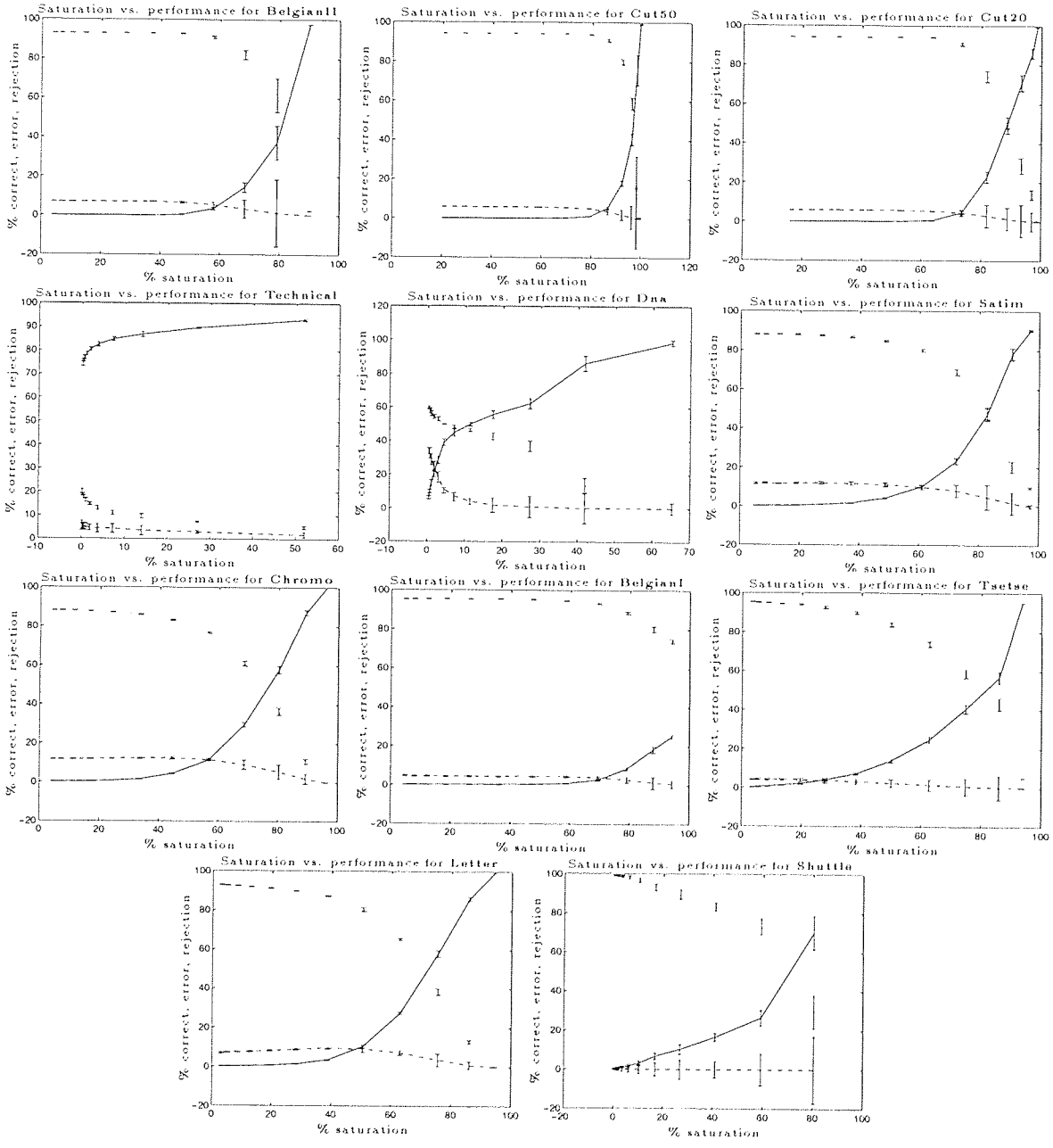


Figure 2.5: Binary RAMnet's performance as a function of the memory saturation (computed for n=1,2, ... ,12). The dotted line indicates the % of correct classifications, solid line denotes % errors and the broken line indicates the % of rejected patterns due to ties in the score. Large values of saturation correspond to small tuple sizes. The one standard deviation error-bars were computed for 10 runs using a RAMnet with 1000 tuples. Ignore the error-bars below zero.

Saturation occurs in binary RAMnets for small values of the parameter n. It affects the performance of the system in a negative way. For a given dataset it can be reduced by increasing the tuple size. Although it is difficult to predict the exact saturation level for a given n, upper and lower bounds on this quantity can be given. The saturation level is related to the amount of correlation in the bits that constitute an n-tuple. Low saturation levels indicate correlated pixels.

Chapter 3

Review of the N-Tuple Networks

3.1 Introduction

In this chapter we review the research on the n-tuple method and show how RAMnets relate to well established methods and models. We focus here on the one-layer (WISARD) architecture.

Early research work (section 3.2) tended to be experimental. Its main subject was the determination of classification performance as a function of tuple size n, tuple number T and type (random or exclusive) of tuple mapping η. In most cases the datasets used were small and not available to other scientists.

Contemporary research is analytical (section 3.3) and concentrates on the theoretical aspects of n-tuple classification. We discuss in detail two theoretical developments which shed some light on how RAMnets operate: Hamming distance and Tuple distance analysis.

N-tuple systems have been poorly understood (Highleyman & Kamentsky, 1960; Bledsoe, 1961) from their very beginning. One of the possible reasons why this method fell into a relative obscurity is that there is no theory that would compare with the sophisticated statistical techniques available (MacKay, 1992a; MacKay, 1992b) with optimisation-based methods.

By relating RAMnets to well established methods and models such as probability estimation and the Bayes Classifier (section 3.4), regression estimation and Regression Networks (section 3.5), Radial Basis Functions (section 3.6) and finally Kanerva's Distributed Memory and Associative Memory (section 3.7), we try to bridge the gap in our understanding of n-tuple systems or at least provide a way of viewing them form different perspectives.

3.2 Early Exploratory Research

Bledsoe et. al The n-tuple method was invented by Browning and Bledsoe (Bledsoe & Browning, 1959) in the context of character recognition. They analysed the influence of the tuple size n on classification error rate and pointed out that saturation of the memory can occur for $n = 1$ and is less likely as n increases. Because at that time storage was scarce and expensive, attention was paid to the calculation of the memory requirements ($T \times 2^n \times c$) for the recogniser.

The classification rate was measured for a number of different exclusive tuple mappings which were found to have no significant effect on the number of errors made by the recogniser. Furthermore, exclusive mappings were compared with random mappings. The RAMnet performed better for random mappings but Browning and Bledsoe concluded that the same improvement could be achieved by using an exclusive mapping and larger values of n (which is not necessarily true, as discussed in section 2.4). Consequently, in all the experiments exclusive mapping was used.

Two scoring methods were investigated: "maximum score" and "distribution processing". In the first scheme an unknown pattern $\mathbf{u}$ was assigned to the class c with the maximum discriminator output y_c. The latter utilised the entire score vector $\mathbf{y}$. The trained RAMnet was tested on a validation subset $\mathcal{D}^{val}$ of the data. For each validation vector $\mathbf{u}$ generated by the class c, the score distribution $\mathbf{y}(\mathbf{u}; c)$ was normalised by the number of tuples T so that $0 \leq y_{c'}(\mathbf{u}; c) \leq 1 \quad \forall c, c'$. The validation score distributions $\mathbf{y}(\mathbf{u}^i, c)$ were then averaged over the number of test patterns in class c yielding a typical score vector $\mathbf{y}^{(avg)}(c)$ with components $y_{c'}^{(avg)}(c) = \frac{1}{D_c^{val}} \sum_{i=1}^{D_c^{val}} y_{c'}(\mathbf{u}^i, c)$. For an unknown test pattern $\mathbf{v}$ the score vector $\mathbf{y}(\mathbf{v})$ was calculated and subtracted off from c class score distributions $\mathbf{y}^{(avg)}(c)$. Pattern $\mathbf{v}$ was then assigned to the class c yielding the minimum distance from the typical score distribution over c classes: $\underset{c}{\operatorname{argmin}} \sum_{c'} \left(y_{c'}(\mathbf{v}) - y_{c'}^{(avg)}(c) \right)^2$.

Distribution score processing was reported to perform better than the maximum score scheme, especially for the handwritten data.

A simple modification of the original system was also proposed in the paper. Instead of storing a bit of information in the memory cell m_{cia}, it would hold a tally of features α_i found in the training vectors. This modification was termed a Maximum Likelihood version and was researched by Bledsoe and Bison (Bledsoe & Bisson, 1962). (Here we call it the Frequency Weighted version with Maximum Likelihood estimation as other estimation methods can be used for RAMnets.)

When the address frequencies are normalised by the number of training samples, they can be viewed as the estimates of the probability $P(\alpha_i|c)$ of address α_i conditioned on the pattern class c. Assuming that tuples are independent, the joint conditional address vector probability $P(\boldsymbol{\alpha}|c)$ is a product of marginal probabilities $P(\alpha_i|c)$. Minimum classification error will be obtained if we assign an unknown pattern $\mathbf{u}$ to the class with the maximum posterior $P(c|\boldsymbol{\alpha})$. For a flat prior $P(c)$,

choosing a class with the maximum likelihood $P(\boldsymbol{\alpha}|c)$ is exactly equivalent.

Instead of computationally expensive products of marginal probabilities, Bledsoe and Bison calculated their logarithms and added them up, using $\log P(\boldsymbol{\alpha}|c) = \sum_{i=1}^{T} \log P(\alpha_i|c)$. If a memory content m_{cia} is zero, this poses a technical problem (the zero tally problem – we formalise it in section 5.3). In the experiments $\log 0$ was replaced with a small negative number ϵ chosen arbitrarily. The authors varied the tuple size n between 1 and 6 and reported better results for the Maximum Likelihood system than for the binary one. (This is not very surprising because for small n, binary RAMnets suffer from saturation.) They reported that the tuples' independence assumption, although theoretically implausible, worked well in practice.

They also proposed a way of finding an optimal mapping. The pixels of the retina were ordered according to their probability of being inked. Then $T < R/n$ tuples were formed using pixels with high probabilities. Again, increased system's performance was reported. The database used was small and comprised 500 images of handwritten numerals.

Ullmann et. al Experiments on a larger scale were conducted by Ullmann (Ullmann, 1969) who gathered 6500 numerals written by 650 different subjects. He was interested in the performance of a binary RAMnet and its Maximum Likelihood version as a function of the tuple size n. He carried out his study for a wider range of tuple size than Bledsoe and Bisson ($1 \leq n \leq 24$) and found that the binary system outperformed the Maximum Likelihood method.

He gathered the statistics of n-tuple states and noted that with a fixed amount of training data the probability of a particular tuple address α_i occuring decreases rapidly with increasing n. In most of the RAM cells $m_{cia} = 0$ and those which are non zero hold insufficient statistics to be used as probability estimates in the Maximum Likelihood system. Consequently, an increase of the tuple size beyond certain threshold will result in a worse performance of the system. Ullmann observed that the binary version, which does not rely on probability estimates, although suffering from saturation for small values of n, maintained better classification rates with n increasing than Maximum Likelihood system.

The problem of supplying enough of the training data when n is large was also mentioned by the author in the context of typed numeral recognition (Ullmann & Kidd, 1969). Memory was expensive in those days and a considerable research effort was put into finding storage efficient architecture modifications (Ullmann, 1971). Ullmann investigated the use of random superimposed coding (zato–coding) and hash arrays in n-tuple systems. It turns out that the former leads to an increase of classification errors and the latter, which involves storing the addresses of non zero cells is efficient only for large values of n. Hash arrays are used in n-tuple regression networks (see section 3.5).

Aleksander et al. A RAMnet has the ability to generalise from the training samples. This property has been analysed (Aleksander & Stonham, 1979) for the binary version of the classifier. Let us assume that the vectors to be classified have length $R = 9$. The universe set $\mathcal{U}$ containing all the vectors comprises $2^R = 512$ distinct patterns. Suppose the RAMnet consists of three tuples of size three ($T = n = 3$) arranged in an exclusive mapping. After training with a set of patterns $\mathcal{D}^{trn}$ each tuple i records occurrences of features α_i by setting bits at locations $m_{i\alpha_i}$. A generalisation set $\mathcal{G}_\theta$ is defined to contain vectors $\mathbf{v}$ which yield the tuple score y at least equal to the threshold θ, i.e., $y(\mathbf{v}) \geq \theta$. For $\theta = T$ any test pattern $\mathbf{v}$ composed of a combination of substrings α_i for which all addressed RAM cells $m_{i\alpha_i} = 1, \forall i$ belongs to $\mathcal{G}_T$. The total number of patterns in the generalisation set can be calculated as $G_T = \prod_i^T \sum_{\alpha_i=0}^{2^n-1} m_{i\alpha_i}$. This formula can be understood by imagining T bins, with $\sum_{\alpha_i=0}^{2^n-1} m_{i\alpha_i}$ objects in i-th bin. If one is allowed to select only one object from each bin in order to produce a sequence of T objects then G_T gives the total number of these sequences.

Aleksander noted, that generalisation[1] is affected by the diversity of the patterns in the training set, because the greater the diversity the lower the number of zero RAM cells. In general, $\mathcal{G}_\theta \subset \mathcal{U}$, however, if n is small enough the memory will saturate and $\mathcal{G}_\theta \equiv \mathcal{U}$. The generalisation set can be increased by lowering the threshold θ, allowing the vectors $\mathbf{v} \in \mathcal{G}_\theta$ to have up to $T - \theta$ subpatterns α_i corresponding to $m_{\alpha_i} = 0$. If a random mapping is used, the size of $\mathcal{G}_\theta$ can be reduced due to tuple overlap.

Trained RAMnets can be used as window operators for image processing (Aleksander & Wilson, 1985) much in the same way as Gaussian operators are applied in machine vision (Marr, 1982). If a RAMnet window is trained on just one bit string $\mathbf{u}$ of length R, its output from an input vector $\mathbf{v}$ can be easily calculated. A tuple will contribute to the score only if it samples the pixels that $\mathbf{u}$ and $\mathbf{v}$ have in common. The probability of this event X happening when the tuple mapping $\boldsymbol{\eta}$ is randomly chosen is

$$P(X) = \left(1 - \frac{H(\mathbf{u}, \mathbf{v})}{R}\right)^n, \tag{3.1}$$

where H is the Hamming distance. With T tuples the expected window output is

$$\mathcal{E}[y] = T\left(1 - \frac{H(\mathbf{u}, \mathbf{v})}{R}\right)^n. \tag{3.2}$$

For a window trained with more than one pattern, one has to consider the Hamming distances between training vectors $\mathbf{u}^i$ as well and the formula gets complicated (this issue is discussed in detail in section 3.3.1).

Aleksander considered the response of a RAMnet window trained to detect patches and noted the exponential decrease of the window's response as a function of its displacement from the origin. He was

[1] In the RAMnet literature the term generalisation refers to the generalisation set of patterns as defined earlier.

also aware that sharper response can be obtained by increasing the tuple size n. He investigated image processing with n-tuple windows trained to detect oriented edges and concluded that this scheme is comparable to that employing classical Laplace and Sobel transforms.

Igor Aleksander revived interest in RAMnets by publishing the details of a successful industrial application (Aleksander *et al.*, 1984) of the n-tuple method to image recognition. The WISARD vision system was implemented in purpose built hardware based on the Motorola 68000 processor. The retina of size 512×512 was sampled exclusively with n-tuples. The n-tuple address space was mapped directly onto the system's VME bus with 16Mb address space allowing fast processing. The tuple size n could be programmed in the range $4 \leq n \leq 8$. Images were captured into the 8 bit deep frame store of the size of the retina. Up to 16 discriminators could be supported. The system operates within 40ms which is sufficient to process images in real time.

3.3 Analytical Work on RAMnets

I review the main tools for understanding the standard n-tuple network: Hamming distance and Tuple distance. The former approach is mainly of theoretical value and is important because it reveals the combinatorial nature of RAMnets. The latter has a wide range of practical applications to the analysis of n-tuple systems and allows us to show later the relationship of the model to kernel regression and regression networks.

3.3.1 Hamming Distance Analysis

Hamming distance is a foundation of a principled method that allows us to calculate the expected output $\mathcal{E}[y]$ of the RAMnet. The original framework (Aleksander, 1970) which was based on the knowledge of the Hamming distance from the test vector $\mathbf{v}$ to all the training vectors $\mathbf{u}^i$ was later improved and extended (Stonham, 1977) whence it turned out that the network's output y also depends on the distances between training vectors $\mathbf{u}^i$ alone.

Let $\mathbf{v}$ denote a binary test vector of length R and $\mathcal{D}^{trn} = \{\mathbf{u}^i\}, \quad 1 \leq i \leq D^{trn}$ a training set of vectors. Let us assume initially that $i = 1$ and that the network consisting of T n-tuples was trained with just one vector $\mathbf{u}$. A binary Hamming vector $\mathbf{h}$ can be obtained by performing an XOR operation on the test and training vectors so that $\mathbf{h} = \mathbf{h}(\mathbf{u}, \mathbf{v}) = \mathbf{u} \oplus \mathbf{v}$.

The Hamming vector $\mathbf{h}$ denotes the similarity of the test vector $\mathbf{v}$ to the training vector $\mathbf{u}$. Note, that any n-tuple will output 1 provided that all components v_j that are sampled by it have a corresponding zero component in the Hamming vector, i.e., $\forall j : \mathbf{h}_{\eta(k,j)} = 0 \Rightarrow m_{ck\alpha_k(\mathbf{v})} = 1, \forall k$. Let $P(X_i)$ denote the probability of the event "an n-tuple will fire for a bit string $\mathbf{v}$ due to training with the i-th

bit string $\mathbf{u}^i$". Assuming that the tuple mapping is exclusive for each one tuple, this probability can be calculated as

$$P(X_i) = P(\boldsymbol{h}^i) = \binom{R-n}{H(\boldsymbol{h}^i)} \left\{ \binom{R}{H(\boldsymbol{h}^i)} \right\}^{-1}, \tag{3.3}$$

where the Hamming distance operator H is understood to return the number of non-zero components of the binary vector: in symbols, $H(\mathbf{h}^i) = \sum_{j=1}^{R} \mathbf{h}_j^i = H(\mathbf{v}, \mathbf{u}^i)$.

To understand this formula imagine that n different pixels are connected to an n-tuple. The components $\mathbf{h}_j$ corresponding to this n-tuple must be equal to zero so that the tuple will fire. The remaining bits of the Hamming vector $\mathbf{h}$ can be either 0 or 1. If the patterns $\mathbf{u}$ and $\mathbf{v}$ are H bits apart then there are $\binom{R-n}{H}$ ways of arranging H 1s on $R-n$ positions. The numerator in equation 3.3 gives the total number of Hamming vectors $\mathbf{h}^i$ with H out of R bits set to 1.

Note, that this formula assumes exclusive sampling within one tuple but still allows different tuples to overlap. It can be rewritten by expanding binomial coefficients and simplifying as

$$P(X_i) = \left(1 - \frac{H}{R-n+1}\right)\left(1 - \frac{H}{R-n+2}\right)\cdots\left(1 - \frac{H}{R}\right) \tag{3.4}$$

and differs from the formula 3.1 given earlier for a random mapping.

The expected scalar output y of the network with T n-tuples presented with a test pattern $\mathbf{v}$ could then be computed as $\mathcal{E}[y] = TP(X_i)$.

However, if a set of D^{trn} patterns $\mathcal{D}^{trn} = \{\mathbf{u}^i, \mathbf{u}^j, \mathbf{u}^k, \ldots \mathbf{u}^{D^{trn}}\}$ is used to train the network, the probability of an n-tuple firing depends on the overlap of zero components in the corresponding Hamming vectors $\mathbf{h}^i, \mathbf{h}^j, \mathbf{h}^k, \ldots, \mathbf{h}^{D^{trn}}$. In order to quantify the notion of overlap let us introduce a binary vector $\mathbf{e}^{i,j,k,\ldots}$ which represents this overlap due to training with patterns $\mathbf{u}^i, \mathbf{u}^j, \mathbf{u}^k, \ldots$.

The overlap vector can be easily obtained by performing an OR operation on the Hamming vectors $\mathbf{e}^{i,j,k,\ldots} = \mathbf{h}^i \vee \mathbf{h}^j \vee \mathbf{h}^k \vee \ldots$. A zero component in the overlap vector implies corresponding zero components in *all* Hamming vectors, i.e., $\mathbf{e}_l^{i,j,k,\ldots} = 0 \Rightarrow \forall i,j,k,\ldots : \mathbf{h}_l^i = \mathbf{h}_l^j = \mathbf{h}_l^k = \ldots = 0$. The probability of an n-tuple sampling zero components in the overlap vector can be calculated as

$$P(X_i, X_j, X_k, \ldots) = P(\mathbf{e}^{i,j,k,\ldots}) = \binom{R-n}{H(\mathbf{e}^{i,j,k,\ldots})} \left\{ \binom{R}{H(\mathbf{e}^{i,j,k,\ldots})} \right\}^{-1}. \tag{3.5}$$

The event X "any one tuple is firing after training" is a union of events X_i "any one tuple is firing after training with the i-th training pattern $\mathbf{u}^i$" so that $X = X_1 \cup X_2 \cup \ldots \cup X_{D^{trn}}$.

To determine the composite probability $P(X)$ we must consider all possible overlaps between events. In other words for every pair $[i,j]$, every triple $[i,j,k]$, etc., we must know the probability of composite events occuring simultaneously, i.e., $P(X_i)$, $P(X_i, X_j)$, $P(X_i, X_j, X_k)$.

Let us define S_r as the sum of all $\binom{D^{trn}}{r}$ probability terms involving r composite events:

$$S_r = \sum_{i,j,k,\ldots=1\ldots D^{trn}}^{\binom{D^{trn}}{r}} P(X_i, X_j, X_k, \ldots). \tag{3.6}$$

In the sum above each combination of subscripts appears just once; hence S_r has $\binom{D^{trn}}{r}$ probability terms. The probability $P(X)$ of the realisation of at least one among events $X_1, X_2, \ldots, X_{D^{trn}}$ is given by

$$\begin{aligned} P(X) &= S_1 - S_2 + S_3 - S_4 + - \ldots \pm S_{D^{trn}} \\ &= \sum_{i=1\ldots D^{trn}}^{\binom{D^{trn}}{1}} P(X_i) - \sum_{i,j=1\ldots D^{trn}}^{\binom{D^{trn}}{2}} P(X_i, X_j) + \sum_{i,j,k=1\ldots D^{trn}}^{\binom{D^{trn}}{3}} P(X_i, X_j, X_k) - \ldots \\ &\pm \sum_{i,j,k,\ldots=1\ldots D^{trn}}^{\binom{D^{trn}}{D^{trn}}} P(X_i, X_j, X_k, \ldots). \end{aligned} \tag{3.7}$$

Finally, the expected output y of a system employing T tuples can be computed

$$\mathcal{E}[y] = TP(X). \tag{3.8}$$

Consequently, to evaluate the probability $P(X)$ of an n-tuple firing after the system was trained with D^{trn} patterns, we face the task of calculating $\binom{D^{trn}}{1} + \binom{D^{trn}}{2} + \ldots + \binom{D^{trn}}{D^{trn}} = 2^{D^{trn}} - 1$ probability terms which is clearly not practical. Stonham could verify equation 3.8 using at most 15 training patterns before the computing requirements became excessive. In general, exact calculation of the expected network output based on the Hamming distance analysis is not possible due to combinatorial explosion demonstrated by equation 3.7.

3.3.2 Tuple Distance and Effective Kernel Function

The most productive theoretical concept for understanding the n-tuple method has been Tuple distance (Allinson & Kołcz, 1995a; Tattersall *et al.*, 1991) and its nonlinear statistical relationship with Hamming distance which is relevant to the network's generalisation properties. The Tuple distance analysis, although restricted to RAMnets trained with just one pattern, leads to the discovery of the effective kernel function which has numerous practical applications. It allows us to understand RAMnets as kernel smoothing methods In particular, regression RAMnet (see section 3.5) can be easily obtained by modifying the original n-tuple network architecture.

The Tuple distance $\rho(\mathbf{u}, \mathbf{v})$ between patterns $\mathbf{u}$ and $\mathbf{v}$ is the number of tuples (of a given input mapping) on which the patterns disagree:

$$\rho(\mathbf{u}, \mathbf{v}) = T - \sum_{k=1}^{T} \delta_{a_k(\mathbf{u}), a_k(\mathbf{v})}. \tag{3.9}$$

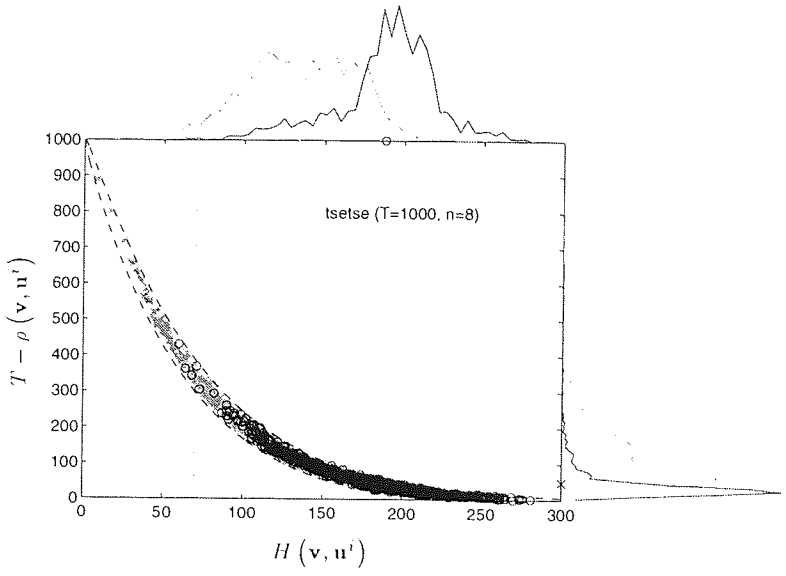


Figure 3.1: Tuple score vs. Hamming distance for a fixed test pattern $\mathbf{v}$ and training patterns $\mathcal{D}^{trn} = \{\mathbf{u}^i\}$ $1 \leq i \leq D^{trn}$ The score of $\mathbf{v}$ was computed for the system trained on just one pattern $\mathbf{u}^i$. Distances between patterns of the same class as $\mathbf{v}$ are marked $*$, and $\circ$ is used for different classes. Functions (3.11) and (3.13) are plotted as 3 standard deviation error curves. The total number of patterns of the correct class (dotted line) and incorrect classes (solid line) are plotted in the margins as functions of Hamming distance (top) and tuple score (right). The means of these distributions are indicated by $*$ and $\circ$ marks. The "generalisation distance" R/n is indicated by a vertical dotted line.

The number on which they agree, $y = T - \rho(\mathbf{u}, \mathbf{v})$, can[2] be called the tuple score or network output. The discussion below considers *expected* tuple distance (where the randomness is due to a tuple mapping) as a function of Hamming distance between a train vector $\mathbf{u}$ and test vector $\mathbf{v}$, i.e., $\rho(\mathbf{u}, \mathbf{v}) \approx \mathcal{E}[\rho(H(\mathbf{u}, \mathbf{v}))] = \mathcal{E}[\rho(H)]$. By writing $\rho(H)$ we mean here the expected[3] tuple distance given that the Hamming distance between two vectors is H. Note, that in contrast to the Hamming analysis, this approach is concerned with just one training pattern so that the tuple overlap of training patterns in the discriminator is not an issue.

An elementary argument based on the random selection of the n-tuple inputs from the R bits available shows that patterns $\mathbf{u}$ which lie a fixed Hamming distance $H = H(\mathbf{u}, \mathbf{v})$ from any one pattern $\mathbf{v}$ are distributed binomially in tuple distance:

$$P(\rho(H)) = \frac{T!}{\rho!(T-\rho)!}\left(1-\left(1-\frac{H}{R}\right)^n\right)^{\rho}\left(1-\frac{H}{R}\right)^{n(T-\rho)}. \tag{3.10}$$

More complicated expressions are available for more constrained n-tuple sampling procedures (Allinson & Kołcz, 1994c). Distribution (3.10) gives an expectation value for ρ of

$$\mathcal{E}[\rho(H)] = T\left(1-\left(1-\frac{H}{R}\right)^n\right), \tag{3.11}$$

[2]This is true only if the RAMnet was trained with one pattern u.

[3]knowing the tuple mapping η and the Hamming *distance* H is not sufficient to calculate the actual tuple distance $\rho(\mathbf{u}, \mathbf{v})$. Knowledge of η and the Hamming *vector* $\mathbf{h}$, however, is sufficient to obtain $\rho(\mathbf{u}, \mathbf{v})$.

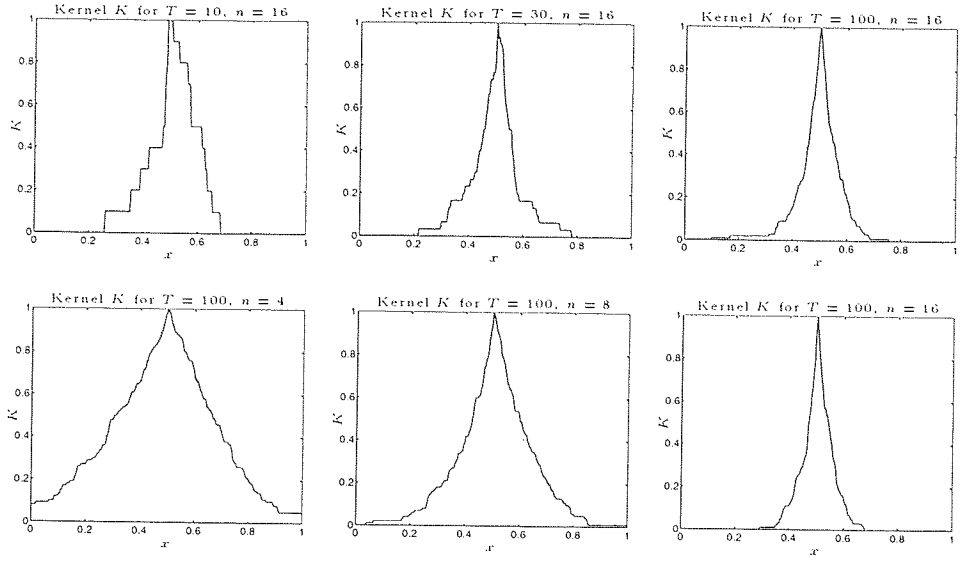


Figure 3.2: Dependence of the effective tuple kernel K on the tuple size and number. The panels show the kernel shapes (solid line) for $T = 10, 30, 100$ and $n = 4, 8, 16$. The dotted line indicates the analytical approximation according to equation 3.16. Real numbers $0 \leq x \leq 1$ were thermometer encoded with resolution 256 bits yielding a test set $\{\mathbf{v}^i\}$ $1 \leq i \leq 256$. The RAMnet was trained on a bit string $\mathbf{u}$ corresponding to $x = 0.5$.

and the expected score y of

$$\mathcal{E}[y] = T\left(1 - \frac{H}{R}\right)^n, \tag{3.12}$$

and indicates that ρ typically strays from this value by the standard deviation

$$\delta\rho(H) = \left[T\left(1 - \frac{H}{R}\right)^n \left(1 - \left(1 - \frac{H}{R}\right)^n\right)\right]^{\frac{1}{2}}. \tag{3.13}$$

This is illustrated in figure 3.1. If the patterns are nearby ($H \ll R$), then a convenient approximation is

$$\rho(H) \approx T\left(1 - e^{-n\frac{H}{R}}\right). \tag{3.14}$$

The n-tuple sampling variations then make $\rho(H)$ uncertain by about $\sqrt{\rho(H)}$.

It is clear from (3.14) that proximity in Hamming distance plays a role in the generalisation behaviour of n-tuple networks. Consider a network trained on just one example $\mathbf{u}^i$ of class c, and tested on a pattern $\mathbf{v}$ Hamming distance H from $\mathbf{u}^i$. Classifications are based on the network response $\sum_{k=1}^{T} m_{ck\alpha_k(\mathbf{v})}$ to pattern $\mathbf{u}^i$, which will be about $Te^{-n\frac{H}{R}}$. Alternatively, one could then say that the network generalises from a training pattern $\mathbf{u}$ to all test patterns $\mathbf{v}^i$ within a Hamming distance of about R/n of $\mathbf{u}$.

Hence, we can view the n-tuple network as a kernel smoother performing a mapping from the R-dimensional hypercube $\mathbb{H}^R$ populated by test patterns $\mathbf{v}^i$ to positive integer space $\mathbb{Z}^+$. The unnor-

malised effective kernel function $K^*(\mathbf{u}, \mathbf{v}^i) : \mathbb{H}^R \rightarrow \mathbb{Z}^+$ of two bit-strings (with the training pattern $\mathbf{u}$ viewed as a centre) is defined to be a function of their Hamming distance $H(\mathbf{u}, \mathbf{v}^i)$:

$$K^*(\mathbf{u}, \mathbf{v}^i) = K^*(H(\mathbf{u}, \mathbf{v}^i)) = K^*(H) = T - \rho(H) \approx Te^{-n\frac{H}{R}}. \tag{3.15}$$

It can be normalised by dividing by the number of tuples T,

$$K(\mathbf{u}, \mathbf{v}^i) = K(H) = \frac{K(H^*)}{T} = 1 - \frac{\rho(H)}{T} \approx e^{-n\frac{H}{R}}. \tag{3.16}$$

The kernel function $K(H)$ is called an "effective" kernel function because it involves the expectation of $\rho(H)$ and equation 3.16 holds on average. The instances of kernels are plotted in figure 3.2 for various numbers of tuples and tuple sizes. The input range $x \in [0, 1]$ was quantised into 256 bins and thermometer encoded. The RAMnet was trained with *one* vector $\mathbf{u}$ being the thermometer code of the value 0.5 with resolution $R = 256$ bits. Then, the kernel function $K(\mathbf{u}, \mathbf{v}^i) = 1 - \frac{\rho(H(\mathbf{u}, \mathbf{v}^i))}{T}$ was calculated for all 256 vectors $\mathbf{v}^i$ and plotted against its approximation $\exp\left(-n\frac{H(\mathbf{u}, \mathbf{v}^i)}{R}\right)$ for various values of parameters n and T. As expected, the approximation is very good around the maximum of the kernel (where the condition $H \ll R$ holds) and gets better as the number of tuples increases. The width of the kernel can be controlled by changing the tuple size n and decreases as n goes up.

3.4 RAMnet and Probability Estimation

The relationship of RAMnets to probability estimators is interesting because a principled statistical framework could be applied to analyse their properties. In particular, we would like to be able to demonstrate that the output of the n-tuple classifier is somehow related to the posterior probability $P(c|\boldsymbol{\alpha})$ of the class given a feature vector $\boldsymbol{\alpha}(\mathbf{u})$, because this would allow us to compare RAMnets with the optimal Bayesian classifier (Hughes, 1968).

We show the relationship of the binary n-tuple system to the Walsh Expansion, which is the classical method for approximating discrete probability functions. We also demonstrate that the unnormalised Frequency Weighted RAMnets (with parameter $\theta = \infty$) can be interpreted, under the feature independence assumption, as class likelihood estimators and upon introducing certain modifications, related to a Bayesian classifier.

3.4.1 Binary RAMnet and the Walsh Expansion

The relationship between the n-tuple method and the Rademacher–Walsh expansion (Beauchamp, 1975) is rather intricate but reveals several interesting points. Therefore, we present the Walsh expansion, discuss its motivation and finally show how it compares with binary RAMnets.

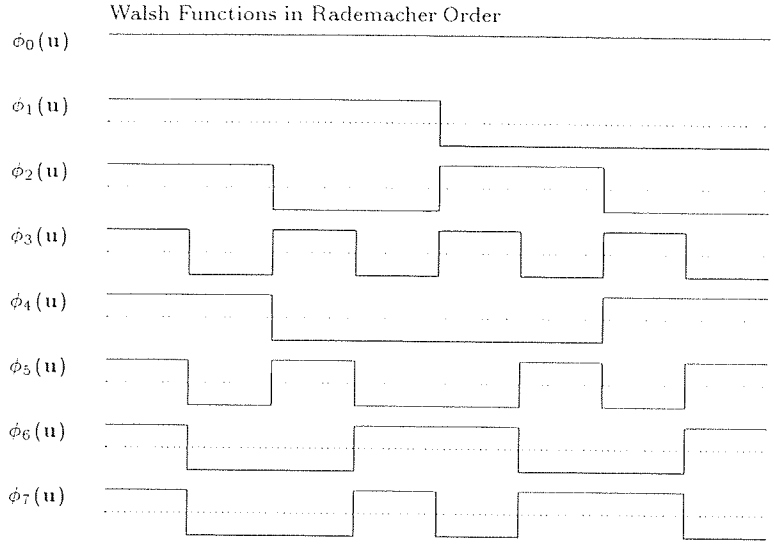


Figure 3.3: Set of Walsh functions $\{\phi_i\}$ in Rademacher order for $n = 3$. X-axis is $\mathbf{u}$, regarded as a single binary number.

Suppose that we are facing a problem of estimating a discrete probability function $P(\mathbf{u} = \mathbf{v}_k)$ of a binary vector $\mathbf{u}$ taking a value $\mathbf{v}_k$. In principle, we just have to observe, using a large sample, the frequencies of $\mathbf{v}_k$. However, if vector $\mathbf{u}$ has a large number R of components then we would have to estimate 2^R probabilities, which is not practical. The solution to this problem (Duda & Hart, 1973) is to expand the function $P(\mathbf{u})$ and approximate it as a partial sum. Walsh functions ϕ_i (the Rademacher ordering (Hurst *et al.*, 1985) of the expansion is used here) form a complete set of orthogonal polynomials, i.e., they have the property $\frac{1}{2^n} \sum_{\{\mathbf{u}\}} \phi_i(\mathbf{u})\phi_j(\mathbf{u}) = \delta_{i,j}$ which allows us to expand any function $P(\mathbf{u})$ as

$$P(\mathbf{u}) = \sum_{i=0}^{2^n-1} c_i \phi_i(\mathbf{u}) \tag{3.17}$$

where 2^n is the number of functions used. Values of coefficients c_i can be calculated as $c_i = \frac{1}{2^n} \sum_{\{\mathbf{u}\}} P(\mathbf{u})\,\phi_i(\mathbf{u})$. Walsh polynomials are plotted in figure 3.3 for $n = 3$. It is easy to verify that they form an orthonormal set of functions with respect to the weight $\frac{1}{2^n}$. We can sample this set discretely at 2^n points and arrange the result in the form of a transformation matrix $\mathbf{\Phi}$:

$$\mathbf{\Phi} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

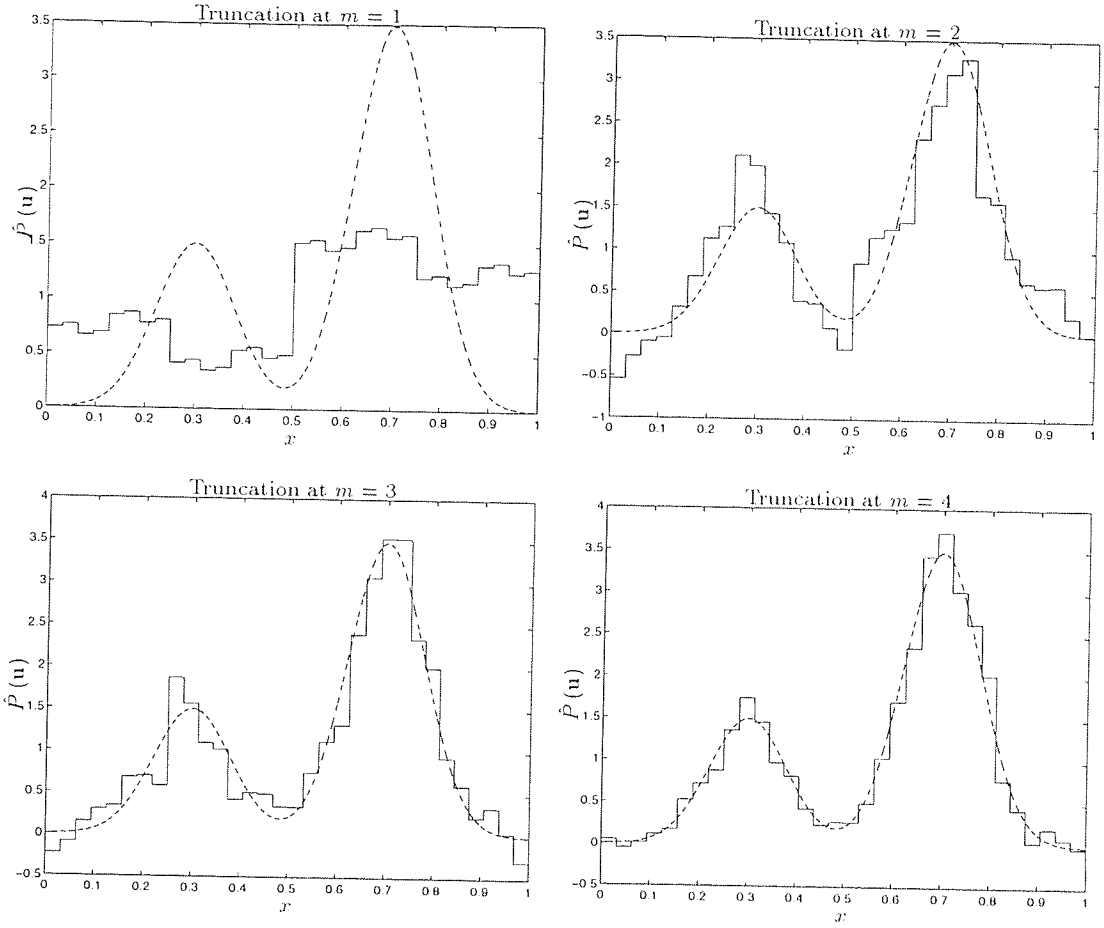


Figure 3.4: Truncated Rademacher–Walsh approximation to a Gaussian mixture probability density function. Approximation $\hat{P}(\mathbf{u})$ improves with increasing m. The coefficients were estimated from a sample of size 10^5 and the approximation calculated according to the equation 3.18. The real valued input x was thermometer encoded into a bit-string $\mathbf{u}$.

If we also represent discrete probability function $P(\mathbf{u})$ with a vector $\mathbf{P}$, the coefficient vector $\mathbf{c}$ can be calculated as $\mathbf{c} = \mathbf{\Phi P}$ and the probability function recovered from the relation $\mathbf{P} = \frac{1}{2^n}\mathbf{\Phi}^{\mathrm{T}}\mathbf{c}$.

If the function to be expanded is a probability function the coefficients can be written as $c_i = \frac{1}{2^n}\mathcal{E}[\phi_i(\mathbf{u})]$. Consequently, we can estimate them from the data $\hat{c}_i = \frac{1}{D^{trn}}\sum_{j=1}^{D^{trn}}\frac{1}{2^n}\phi_i(\mathbf{u}^j)$ where D^{trn} is the sample size. Of course we do not want to estimate all 2^n coefficients, but we would truncate the expansion after 2^m terms so that

$$P(\mathbf{u}) \approx \hat{P}(\mathbf{u}) = \sum_{i=0}^{2^m} \hat{c}_i \phi_i(\mathbf{u}) \tag{3.18}$$

Truncated approximations to a Gaussian mixture probability function are plotted on figure 3.4. The approximation is improving with increasing m. Writing the Walsh functions in the analytical form

$$\phi_i(\mathbf{u}) = \begin{cases} 1 & i = 0 \\ (1-2u_1) & i = 1 \\ \vdots & \vdots \\ (1-2u_n) & i = \binom{n}{1} \\ (1-2u_1)(1-2u_2) & i = \binom{n}{1} + 1 \\ \vdots & \vdots \\ (1-2u_{n-1})(1-2u_n) & i = \binom{n}{2} + \binom{n}{1} \\ (1-2u_1)(1-2u_2)(1-2u_3) & i = \binom{n}{2} + \binom{n}{1} + 1 \\ \vdots & \vdots \\ (1-2u_1)\ldots(1-2u_n) & i = 2^n - 1 \end{cases}$$

reveals the relationship of RAMnets to this expansion.

We note that the complete Walsh expansion includes all tuples of components of $\mathbf{u}$ of sizes from 1 to n. The components have been encoded using values $[-1, 1]$ rather than $[0, 1]$ used in the n-tuple system. Furthermore, the RAMnet contains only a limited number T of tuples of the chosen size. It is clear that we cannot hope to be able to recover the shape of the probability function $P(\mathbf{u})$ using RAMnet in its original form. However, we can view the binary n-tuple system as a discrete probability approximation system in which the probability function is projected onto a set of T non-orthogonal functions $\phi_i^{Ram}(\mathbf{u})$.

The tuple "eigenfunctions" ϕ^{Ram} can be seen in two ways: either as a set of $T2^n$ binary valued or a set of T positive integer valued functions. In the first interpretation

$$\{\phi_{k,\alpha}^{Ram}(\mathbf{u})\} = \{\delta_{\alpha,a_k(\mathbf{u})}\}_{k=1,\alpha=0}^{k=T,\alpha=2^n-1} \tag{3.19}$$

and the coefficients of the expansion are given by the memory contents $m_{k\alpha}$. In the second view it is not the coefficients of the expansion that are estimated during the training procedure (they are all 1) but the basis functions themselves. There are as many functions as tuples

$$\{\phi_i^{Ram}(\mathbf{u})\} = \{m_{i\alpha_i(\mathbf{u})}\}_{i=1}^{i=T} \tag{3.20}$$

and the values of the i-th function are tabulated in the i-th RAM node.

As a minor point, we note that given the contents $m_{i\alpha_i}$ of the i-th RAM node defining a RAMnet basis function $\phi_i^{Ram}(\mathbf{u}; n)$ in the n-tuple system we can reconstruct $\binom{n}{n'}$ functions $\phi_j^{Ram}(\mathbf{u}; n')$ corresponding to smaller tuple size n'. We obtain them by integrating out $n - n'$ bits from the addresses α_i and adding and thresholding corresponding locations $m_{ci\alpha}$. From one n-tuple we can recover $\binom{n}{n'}$

n'-tuple functions. Thus an n-tuple net contains a similar amount of information as a truncated Walsh expansion with terms involving up to n components of $\mathbf{u}$. As no integration is ever performed, only a part of this information is utilised in RAMnets, which allows them to reduce the memory requirements (a table of size $T \times 2^n$ is required as opposed to 2^R entries needed by the Walsh expansion) at a cost of the lack of universality.

The fact that RAMnets are not universal approximators has been noted (Roy & Sherman, 1967) in the aspect of Φ-learning machines. A Φ-machine (Nilsson, 1965) is any pattern classifying machine employing a Φ-function, i.e., a function $\Phi(\mathbf{u}; \mathbf{w})$ which depends linearly on the parameters $\mathbf{w}$ so that

$$\Phi(\mathbf{u}) = w_0 + \sum_{i=1}^{T} w_i \phi_i(\mathbf{u}) \tag{3.21}$$

where linearly independent, real, single valued functions ϕ_i are independent of the weights $\mathbf{w}$[4]. It can easily be noted that an n-tuple system is equivalent to a n-th order polynomial Φ-processor with all parameters $w_i = 1$ only if all pattern vector component n-tuples are taken. Because in practical applications one selects a fixed number, T, of tuples rather than all $\binom{R}{n}$ of them, certain terms in the expansion will be missing. Therefore, the original binary RAMnet is *not* a universal approximator and potentially introduces a model error. Furthermore, the estimation error can occur as a result of the training procedure.

3.4.2 Frequency Weighted RAMnet as a Crude Likelihood Estimator

With $\theta = \infty$, expression 2.3 for the memory content $m_{ci\alpha}$ can be interpreted as an estimate of the probability $P(\alpha_i|c)$ (up to a normalisation factor) that a data point from a given class c will have subpattern α in the i-th n-tuple. Assuming these distributions for different n-tuples to be independent of each other, the joint distribution over all the sub-patterns taken together is

$$P(\boldsymbol{\alpha}|c) = \prod_i P(\alpha_i|c)\,.$$

This can be used for Maximum Likelihood classification, or converted to posterior class probabilities using Bayes' rule with class priors $P(c)$, if they are available. The independence assumption lacks plausibility, because it would be remarkable for the correlations required to make the classes distinguishable not to be reflected in correlations between the n-tuples, but never the less, good results have been reported with this formulation (see section 3.2), which has been re-invented from time to time (Sixsmith *et al.*, 1990; Badr, 1993). Aside from its implausibility, its main practical problem is that factors of zero appear if a naive estimate of $P(\alpha_i|c)$ is used for subpatterns which never appear in

[4] specific examples of Φ function families include for example linear, quadric, n-th order polynomial functions or RBFs

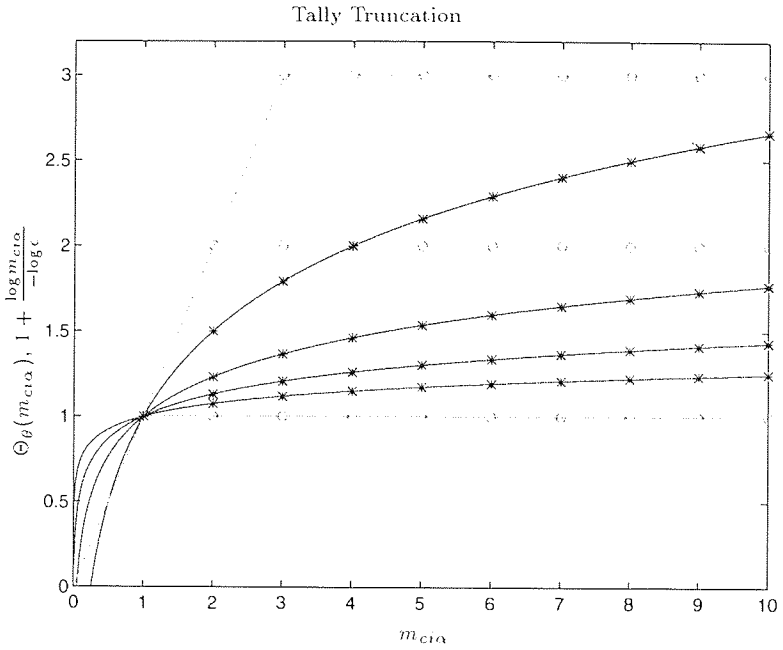


Figure 3.5: $1+\frac{\log m_{ci\alpha}}{-\log\epsilon}$ truncated to 0 for tally values less than ϵ (dark lines) for $\epsilon = 0.25, 0.05, 0.005$ and 0.0001 (counting from the top of the figure), and $\Theta_\theta(m_{ci\alpha})$ (light lines) for $\theta = 1, 2$ and 3 (counting from the bottom of the figure), as functions of the tally value. This shows that (3.22) can be a reasonable approximation at integral values of $m_{ci\alpha}$ in some circumstances, particularly for $\epsilon \to 0$ with $\theta = 1$.

the training data. In practice these are replaced with a small *ad hoc* positive constant, say ϵ, leaving scope for more principled approaches to estimating these probabilities (see chapter 5).

The n-tuple method with finite threshold can be seen as a scaled and translated approximation to the logarithm of the class conditional probability in equation 3.22. Suppose that pattern $\mathbf{u}$ has sub-patterns $\boldsymbol{\alpha}(\mathbf{u})$, and the D_c^{trn} training patterns from class c supply the tallies $m_{ci\alpha} = \sum_{\mathbf{v}\in\mathcal{D}_c} \delta_{\alpha,\alpha_i(\mathbf{v})}$ used to estimate $P(\alpha_i|c)$ by $m_{ci\alpha}/D_c^{trn}$. Furthermore, the number of tuples in the system is T. We can translate and scale outputs y_c by $T + \log({D_c^{trn}}^T)/-\log\epsilon$ and $\frac{1}{-\log\epsilon}$ respectively so that the network's output is equal to

$$y_c = T + \frac{\log({D_c^{trn}}^T)}{-\log\epsilon} + \frac{\log\left(P\left(\boldsymbol{\alpha}(\mathbf{u})|c\right)\right)}{-\log\epsilon} = \sum_{i=1}^{T}\left(1 + \frac{\log D_c^{trn} + \log P\left(\alpha_i(\mathbf{u})|c\right)}{-\log\epsilon}\right)$$
$$= \sum_{i=1}^{T}\left(1 + \frac{\log m_{ci\alpha}}{-\log\epsilon}\right) \qquad (3.22)$$

For suitable choices of ϵ and θ, the network responses in (2.5) will approximately satisfy an expression $\sum_{i=1}^{T}\Theta_\theta(m_{ci\alpha})$. As illustrated by figure 3.5, for integer tallies, the approximation becomes arbitrarily accurate for $\theta = 1$ as $\epsilon \to 0$. Hence the standard n-tuple method could be justified this way if the independence assumption were acceptable and the absence of sub-patterns in the training data could be taken as strong evidence that the corresponding probabilities are tiny. Essentially, the method

counts the number of factors of ϵ in (3.22).

3.5 N-tuple Method as a Regression Network

A view of the n-tuple method that provides a consistent statistical framework has been recently proposed by Allinson and Kolcz. They show that a slightly modified standard RAM net can be interpreted as a regression network with tuples collectively acting as kernel functions approximating the expectation $\mathcal{E}[y|\mathbf{x}]$ of a scalar output y for a given input vector $\mathbf{x}$. This architecture can be used not only for pattern classification but any regression problem such as plant control or time series prediction. Before we discuss that important paper, in the following subsection, we review General Regression Networks which constitute a basis of n-tuple regression networks.

3.5.1 General Regression Network

A General Regression Neural Network (Specht, 1990; Specht, 1991) is a memory based method utilising a one-pass learning algorithm. The network's output converges to the (nonlinear in general) regression surface.

The inputs are d-dimensional real-valued vectors $\mathbf{x}$ whereas the output is a real-valued scalar y (the architecture can be easily extended to accommodate multidimensional outputs). The random variable y and random vector $\mathbf{x}$ are distributed according to the density function $p(y, \mathbf{x})$.

The relationship between the input and output variables of the modelled process can be described in terms of the regression of the dependent variable y on input variable $\mathbf{x}$. The regression $\mathcal{E}[y|\mathbf{x}]$ is simply the mean value of y for each value of $\mathbf{x}$ based on a finite sample of data points $\mathcal{D}^{trn} = \{\boldsymbol{x}^i, y^i\}$, $1 \leq i \leq D^{trn}$. For the known probability density function $p(y, \mathbf{x})$ it is defined as

$$g(\mathbf{x}) = \mathcal{E}[y|\mathbf{x}] = \frac{\int_{-\infty}^{+\infty} y\, p(y, \mathbf{x}) dy}{\int_{-\infty}^{+\infty} p(y, \mathbf{x}) dy}. \tag{3.23}$$

In general the analytical form of $p(y, \mathbf{x})$ is not known and one has to approximate it. The regression network uses non-parametric density estimation method (Parzen, 1962; Cacoullos, 1966) with separable kernel function $\mathbf{K}(y, \mathbf{x}) = \mathbf{K}(\mathbf{x}) K(y)$. The density estimator $\hat{p}(y, \mathbf{x})$ in general is given by

$$\hat{p}(y, \mathbf{x}) = \frac{1}{D^{trn}} \sum_{i=1}^{D^{trn}} \mathbf{K}(\mathbf{x} - \mathbf{x}^i) K(y - y^i). \tag{3.24}$$

If a Gaussian kernel is used this estimate becomes

$$\hat{p}(y, \mathbf{x}) = \frac{1}{(2\pi\sigma^2)^{(d+1)/2}} \frac{1}{D^{trn}} \sum_{i=1}^{D^{trn}} \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}^i\|^2}{2\sigma^2}\right) \exp\left(-\frac{(y - y^i)^2}{2\sigma^2}\right) \tag{3.25}$$

where d is the dimensionality of the input vector $\mathbf{x}$. This estimate can now be substituted into the regression equation 3.23 instead of $p(y, \mathbf{x})$ to obtain an estimator $\hat{g}(\mathbf{x})$ of the regression function:

$$\hat{g}(\mathbf{x}) = \frac{\sum_{i=1}^{D^{trn}} \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}^i\|^2}{2\sigma^2}\right) \int_{-\infty}^{+\infty} y \exp\left(-\frac{(y-y^i)^2}{2\sigma^2}\right) dy}{\sum_{i=1}^{D^{trn}} \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}^i\|^2}{2\sigma^2}\right) \int_{-\infty}^{+\infty} \exp\left(-\frac{(y-y^i)^2}{2\sigma^2}\right) dy}. \tag{3.26}$$

Because the $(d+1)$-variate Gaussian kernel function $\mathbf{K}(y, \mathbf{x})$ is normalised and separable with respect to variables $\mathbf{x}$ and y integration over y can be accomplished analytically yielding

$$\hat{g}(\mathbf{x}) = \frac{\sum_{i=1}^{D^{trn}} y^i \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}^i\|^2}{2\sigma^2}\right)}{\sum_{i=1}^{D^{trn}} \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}^i\|^2}{2\sigma^2}\right)}. \tag{3.27}$$

This estimate is essentially a weighted average of all of the training outputs y^i. The weight decreases exponentially with the Euclidean distance of the input $\mathbf{x}$ from the training input $\mathbf{x}^i$.

The determination of the optimal value of the smoothing parameter σ is important for obtaining a good approximation to the true density. For a very large σ, $\hat{g}(\mathbf{x})$ becomes the sample mean of the training outputs y^i. As σ approaches 0, $\hat{g}(\mathbf{x})$ equals the value of y^i associated with the $\mathbf{x}^i$ closest to $\mathbf{x}$. For an intermediate value of σ all training points y^i are taken into account but those closer to $\mathbf{x}$ are given more weight.

Specht suggests the determination of the optimal kernel width σ by the hold out method. In this method one training sample at a time is excluded from the training set $\mathcal{D}^{trn}$. The network uses the held out sample to evaluate its output. Repeating this process for each training sample enables one to calculate the mean squared error between the actual outputs y^i and their estimates. The procedure can be repeated for different values of the parameter σ and the optimal one, in terms of the error function, can be selected.

Various other Parzen window functions can be used in equation 3.25 estimating the joint density $p(y, \mathbf{x})$. Specht suggests a number of them (Specht, 1990) that can be computed more efficiently than the Gaussian kernel. In particular a multivariate, separable kernel

$$\mathbf{K}(\mathbf{x}, \mathbf{x}^i) = \exp(-\frac{1}{\sigma} \sum_{j=1}^{d} |x_j - x_j^i|_M) \tag{3.28}$$

results in the regression estimator

$$\hat{g}(\mathbf{x}) = \frac{\sum_{i=1}^{D^{trn}} y^i \mathbf{K}(\mathbf{x}, \mathbf{x}^i)}{\sum_{i=1}^{D^{trn}} \mathbf{K}(\mathbf{x}, \mathbf{x}^i)} = \frac{\sum_{i=1}^{D^{trn}} y^i \exp\left(-\frac{1}{\sigma} \sum_{j=1}^{d} |x_j - x_j^i|_M\right)}{\sum_{i=1}^{D^{trn}} \exp\left(-\frac{1}{\sigma} \sum_{j=1}^{d} |x_j - x_j^i|_M\right)} \tag{3.29}$$

that involves Manhattan distance rather than Euclidean distance.

The properties of the General Regression Network can be summarised as follows

1. Learning is fast because it only requires storing the training set in the memory,
2. The network is suitable for sparse datasets as the regression surface is defined even with one sample,
3. The smoothing parameter σ can be easily determined from the data.

The major problem with this technique is that the amount of computation required to estimate new output is proportional the size of the training set. Consequently if the training set is large, clustering has to be performed (Sebastyen, 1966). As discussed in the next section, the n-tuple regression network overcomes this problem by representing the data in terms of features. The amount of calculations in this network is independent of the training set size and depends on the number of tuples used to sample the input space.

3.5.2 N-tuple Regression Network

A simple modification (Allinson & Kolcz, 1995b) of a binary RAMnet architecture allows it to operate as a kernel regression estimator. The modification consists of adding an integer tally $r_{ck\alpha}$ (a normalisation factor) for each real-valued memory location $m_{ck\alpha}$ (a weight). The resulting network has just one "discriminator", therefore we drop the subscript c in the discussion below.

Before training commences all weights and tallies are set to zero. For each data point $\{\mathbf{x}^i, y^i\} \in \mathcal{D}^{trn}$ the real-valued input vector $\mathbf{x}^i$ is transformed into a corresponding binary vector $\mathbf{u}^i$ using a suitable encoding technique (these are discussed in section 2.6). Bit string $\mathbf{u}^i$ is then processed by a set of T n-tuples and a tuple (an address) vector $\boldsymbol{\alpha}(\mathbf{u}^i)$ is generated. For each memory location addressed by the k-th tuple with the value $\alpha_k(\mathbf{u}^i)$, the corresponding output value y^i is added to the cell $m_{k\alpha_k(\mathbf{u}^i)}$ and tally $r_{k\alpha_k(\mathbf{u}^i)}$ is incremented

$$\forall k = 1, 2, \ldots, T, \quad \forall i = 1, 2, \ldots D^{trn}, \quad m_{\alpha_k(\mathbf{u}^i)} = m_{\alpha_k(\mathbf{u}^i)} + y^i, \quad r_{\alpha_k(\mathbf{u}^i)} = r_{\alpha_k(\mathbf{u}^i)} + 1.$$

The network's output $y(\mathbf{v})$ for a test vector $\mathbf{v}$ is calculated as a ratio of the sum of the weights $m_{\alpha_k(\mathbf{v})}$ and the sum of corresponding tallies $r_{\alpha_k(\mathbf{v})}$

$$y(\mathbf{v}) = \frac{\sum_{k=1}^{T} m_{k\alpha_k(\mathbf{v})}}{\sum_{k=1}^{T} r_{k\alpha_k(\mathbf{v})}}. \tag{3.30}$$

If the denominator in equation 3.30 is zero the output is set to zero as well. The notion of Tuple distance (discussed in detail in section 3.3.2) allows us to derive a number of relationships that can be used to show the connection of this model with Regression Networks. The following equations

describe the memory contents $r_{k\alpha_k(\mathbf{v})}$, $m_{k\alpha_k(\mathbf{v})}$ and tuple proximity as a function of the training data

$$\sum_{k=1}^{T} \delta_{\alpha_k(\mathbf{u}),\alpha_k(\mathbf{v})} = T\left(1 - \frac{\rho(\mathbf{u},\mathbf{v})}{T}\right), \tag{3.31}$$

$$m_{k\alpha_k(\mathbf{v})} = \sum_{i=1}^{D^{trn}} y^i \delta_{\alpha_k(\mathbf{v}),\alpha_k(\mathbf{u}^i)} \qquad \forall k = 1, 2, \ldots, T, \tag{3.32}$$

$$r_{k\alpha_k(\mathbf{v})} = \sum_{i=1}^{D^{trn}} \delta_{\alpha_k(\mathbf{v}),\alpha_k(\mathbf{u}^i)} \qquad \forall k = 1, 2, \ldots, T. \tag{3.33}$$

We use these relationships to expand the numerator and denominator of equation 3.30 as functions of the tuple distances $\rho(\mathbf{v}, \mathbf{u}^i)$ between the test vector $\mathbf{v}$ and the training vectors $\mathbf{u}^i$. The numerator can be rewritten as

$$\begin{aligned} \sum_{k=1}^{T} m_{k\alpha_k(\mathbf{v})} &= \sum_{k=1}^{T}\sum_{i=1}^{D^{trn}} y^i \delta_{\alpha_k(\mathbf{v}),\alpha_k(\mathbf{u}^i)} = \sum_{i=1}^{D^{trn}} y^i \sum_{k=1}^{T} \delta_{\alpha_k(\mathbf{v}),\alpha_k(\mathbf{u}^i)} = \\ &= T\sum_{i=1}^{D^{trn}} y^i \left(1 - \frac{\rho(\mathbf{v},\mathbf{u}^i)}{T}\right) \end{aligned} \tag{3.34}$$

and the denominator as

$$\begin{aligned} \sum_{k=1}^{T} r_{k\alpha_k(\mathbf{v})} &= \sum_{k=1}^{T}\sum_{i=1}^{D^{trn}} \delta_{\alpha_k(\mathbf{v}),\alpha_k(\mathbf{u}^i)} = \sum_{i=1}^{D^{trn}}\sum_{k=1}^{T} \delta_{\alpha_k(\mathbf{v}),\alpha_k(\mathbf{u}^i)} = \\ &= T\sum_{i=1}^{D^{trn}} \left(1 - \frac{\rho(\mathbf{v},\mathbf{u}^i)}{T}\right) \end{aligned} \tag{3.35}$$

yielding

$$y(\mathbf{v}) = \frac{\sum_{i=1}^{D^{trn}} y^i \left(1 - \frac{\rho(\mathbf{v},\mathbf{u}^i)}{T}\right)}{\sum_{i=1}^{D^{trn}} \left(1 - \frac{\rho(\mathbf{v},\mathbf{u}^i)}{T}\right)}. \tag{3.36}$$

When a thermometer code is used for encoding the input vector's components, the Hamming distance can be replaced by Manhattan distance distance (ignoring quantisation effects). The effective kernel function $\mathbf{K}(\mathbf{v}, \mathbf{u}^i)$ approximated by equation 3.16 which involves Hamming distance is then equivalent to the kernel $\mathbf{K}(\mathbf{x}, \mathbf{x}^i)$ operating on the original real valued input $\mathbf{x}$ and using the Manhattan distance:

$$\mathbf{K}(\mathbf{v}, \mathbf{u}^i) = \exp\left(-\frac{n|\mathbf{v} - \mathbf{u}^i|_H}{R}\right) \equiv \exp\left(-\frac{n|\mathbf{x} - \mathbf{x}^i|_M}{R}\right) = \mathbf{K}(\mathbf{x}, \mathbf{x}^i). \tag{3.37}$$

Plugging this expression into equation 3.36 yields

$$y(\mathbf{v}) \equiv y(\mathbf{x}) = \frac{\sum_{i=1}^{D^{trn}} y^i \mathbf{K}(\mathbf{x}, \mathbf{x}^i)}{\sum_{i=1}^{D^{trn}} \mathbf{K}(\mathbf{x}, \mathbf{x}^i)}. \tag{3.38}$$

which is equivalent to the regression equation 3.29 of the General Regression Network. Consequently, the output of the n-tuple regression network $y(\mathbf{v})$, is on average approximately equal to the estimate of the regression $\hat{g}(\mathbf{x}) \approx \mathcal{E}[y|\mathbf{x}]$.

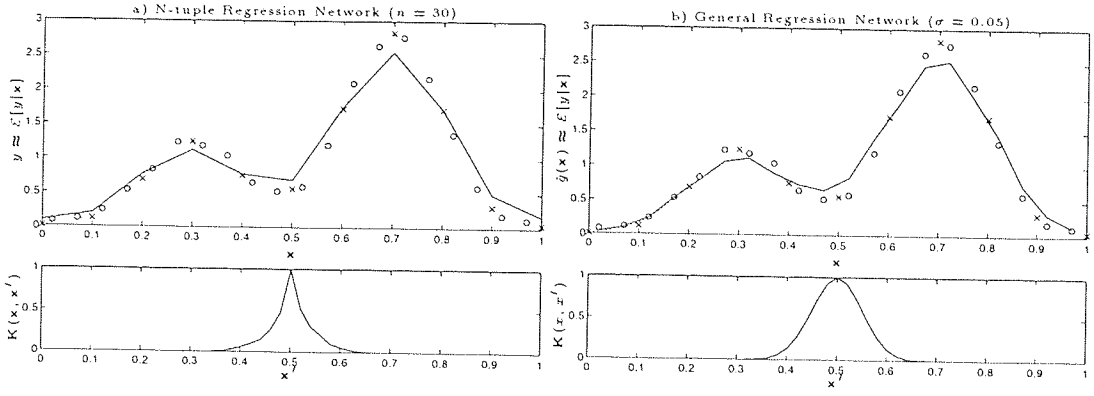


Figure 3.6: **a)** Performance of the n-tuple regression network with $n = 30$, $T = 100$ on a simple regression problem. The lower graph shows the effective tuple kernel $K(x, x')$ centered on a bit string corresponding to $x = 0.5$. Training and test data is shown by circles and crosses respectively. **b)** Performance of the Specht's regression network with a Gaussian kernel for parameter $\sigma = 0.05$ on the same data. The shape of the kernel is shown on the lower plot.

The choice of the kernel width R/n depends on the data and the optimum value can be determined using a cross-validation technique discussed in section 3.5.1. (Note, that it can be carried out very fast with RAMnets because, unlike MLPs, they do not suffer from the credit assignment problem. Instead of holding out one of the training vectors $\mathbf{u}$ and retraining the system with the rest of the dataset we can use the entire dataset for training then compute $\boldsymbol{\alpha}(\mathbf{u})$, decrement the tallies in the cells addressed by $\mathbf{u}$ and compute the network's response $y(\mathbf{u})$. After the calculation of $y(\mathbf{u})$ the tallies are restored allowing us to process another vector. Thus, the time required to perform a cross-validation will be only about twice the test time. Also note, that if k-fold cross-validation is to be performed, we only need integer tallies with the maximum value $k + 1$ in order to determine if a feature will remain set after removing k patterns from the training set.)

The retina size R is determined by the dimensionality of the input vector $\mathbf{x}$, quantisation of its components and the encoding technique selected. The free parameter n can become large (500 is not unusual) for bigger datasets or data localised in the input space. Therefore, the use of hash-tables (Knuth, 1973; Rohwer & Lamb, 1993) to realize efficient storage of weights and counters is necessary.

The properties of the n-tuple regression network are the following:

1. The response time of the RAMnet is independent of the training set size and the clustering techniques, that were needed for Specht's network, are not necessary.

2. The distances between the test point and the training points are computed implicitly by tuple sampling, which accounts for the operational speed of the system.

3. As in the case of the general regression network training requires one pass through the dataset. The determination of the parameter σ can be done quickly and does not require retraining of

the system.

The disadvantage of this modification of binary RAMnet is, however, an increased storage requirement. A floating point number and an integer value must be recorded as opposed to 1 bit for the standard method.

3.6 RAMnets and Radial Basis Functions

The output of the binary RAMnet given by equation 2.4 can expanded in terms of all the memory cells as

$$y_c = \sum_{k=1}^{T} m_{ck\alpha_k(\mathbf{v})} = \sum_{k=1}^{T} \sum_{\alpha=0}^{2^n-1} m_{ck\alpha} \delta_{\alpha,\alpha_k(\mathbf{v})}. \tag{3.39}$$

This form can be regarded as a linear transformation applied to the output of unusual basis functions $\phi_{k,\alpha}^{Ram}(\mathbf{v}) = \delta_{\alpha,\alpha_k(\mathbf{v})}$.

The alternative expression for the RAMnet's output can be given in terms of the training data. Using equation 3.33 and approximating 3.9 with 3.14 we obtain

$$y_c = \sum_{k=1}^{T} \sum_{i=1}^{D_c^{trn}} \delta_{\alpha_k(\mathbf{v}),\alpha_k(\mathbf{u}^i)} \approx T \sum_{i=1}^{D_c^{trn}} e^{\frac{H(\mathbf{v},\mathbf{u}^i)}{R}} = \sum_{i=1}^{D_c^{trn}} w_{ci} e^{\frac{H(\mathbf{v},\mathbf{u}^i)}{R}}. \tag{3.40}$$

Equations 3.39 and 3.40 are equivalent provided that we can find the training patterns $\mathbf{u}^i$ and the weights w_{ci} such that

$$m_{ck\alpha} = \sum_{i=1}^{D_c^{trn}} w_{ci} \delta_{\alpha,\alpha_k(\mathbf{u}^i)}. \tag{3.41}$$

One way to arrange this is to choose all the patterns $\mathbf{u}^i$ so they are separated from each other by tuple distance T, i.e, none of the patterns $\mathbf{u}^i$ match each other in any n-tuple. (This is possible if and only if the number of these patterns is no more than 2^n.) In this situation there is at most one i for any given n-tuple k and address α_k such that $\alpha_k(\mathbf{u}^i) = \alpha$, which may be called $i(k,\alpha)$ when it exists. Then the choice

$$m_{ck\alpha} = \begin{cases} w_{ci(k,\alpha)} & \text{if } i(k,\alpha) \text{ is defined,} \\ 0 & \text{otherwise.} \end{cases} \tag{3.42}$$

satisfies (3.41). Such a network acts like the radial basis function network (3.40) with a Hamming spherically symmetric local basis function of radius roughly R/n centred on each pattern $\mathbf{u}^i$, even though the patterns $\mathbf{u}^i$ do not appear in the implementation (3.39), and to the benefit of computation speed, no distance calculations $H(\mathbf{u}^i, \mathbf{v})$ are ever performed.

If the training patterns meet the separation condition, they can be identified with the basis function centres $\mathbf{u}^i$. To see this, observe that the assumed one-to-one correspondence between pattern i and

address $\alpha_k(\mathbf{u}^i)$ in the k-th n-tuple memory implies that the set of memory locations corresponding to one pattern is disjoint from the set addressed by any other. Therefore, for each i and c one can meaningfully define $m_{ci} = m_{ck\alpha_k(\mathbf{u}^i)}$. As $D_c^{trn} \leq 2^n$, the network response to a test pattern $\mathbf{v}$ (3.39) can then be re-written by replacing the sum over addresses with a sum over training patterns:

$$y_c = \sum_{k=1}^{T} \sum_{i=1}^{D_c^{trn}} m_{ck\alpha_k(\mathbf{u}^i)} \delta_{\alpha_k(\mathbf{u}^i),\alpha_k(\mathbf{u})}. \tag{3.43}$$

Then it is clear from the equation 3.40 that one can identify $m_{ci} = w_{ci}$.

The interpretation of an n-tuple regression network as an effective radial basis function network with a function centre on each training pattern requires the patterns $\mathbf{u}^i$ to be tuple-separated by T. This condition would be valid at least to a good approximation if their Hamming separation were large compared to R/n.

It may not be easy to arrange this *and* ensure good coverage of the pattern space by the local effective basis functions. To ensure good coverage, the data needs to just happen to be arranged so that each training pattern of a class is about R/n bits away from it's nearest neighbour. To do as well as possible on the separability condition as well, all other neighbours should be much further away, but the triangle inequality requires the next nearest neighbour to be within $2R/n$ bits.

3.7 RAMnets and Other Memory Models

N-tuple method is a memory based method. The training phase can be viewed as storing information about features pertinent to a given class and recognition as a retrieval of this information. Note, that a RAMnet implements a distributed storage mechanism because, unlike conventional memories, one training pattern $\mathbf{u}$ can change the contents of many cells.

It is natural to compare RAMnets with Kanerva's sparse distributed memory because both of them address the issue of information storage in the form of distributed activity patterns and use an "overlap" measure to determine similarity of the input vectors. The main idea behind Kanerva's model is that a number of "hard" locations $\boldsymbol{\xi}$ in the space $\mathbb{H}^R$ is chosen randomly (in a similar manner to selecting n-tuples). In order to store a bit string $\mathbf{u}$ we find a set of hard locations which are within a radius r from $\mathbf{u}$ and update their contents. Reading at an address $\mathbf{v}$ is accomplished by averaging the contents of hard locations within r bits of $\mathbf{v}$.

We also demonstrate that RAMnets can be viewed as an Associative Memory if we consider the original binary vectors in the enhanced tuple space for the Frequency Weighted version of the classifier.

3.7.1 The Kanerva Model

Kanerva's Sparse Distributed Memory model (Kanerva, 1988), has been developed theoretically using an "overlap" measure similar to that defined by equation (3.14). Although intended mainly as an associative memory, it is easily generalised for classification problems or function interpolation problems. The interpolation version will be presented here.

Instead of n-tuples, a set of T bit strings of length R is randomly selected from a uniform distribution. These are used as *centres* $\boldsymbol{\xi}_k$ of k-th hard-sphere radial basis functions

$$\phi_k(\mathbf{u}; r) = \begin{cases} 1 & \text{if } H(\boldsymbol{\xi}_k, \mathbf{u}) \leq r \\ 0 & \text{if } H(\boldsymbol{\xi}_k, \mathbf{u}) > r \end{cases} \tag{3.44}$$

of a somewhat carefully chosen radius r. Memory space for a vector in the range of the function to be approximated is associated with each centre. The memory at centre k is set to

$$\mathbf{m}_k^{(\mathrm{K})} = \frac{\sum_{i=1}^{D^{trn}} y^i \phi_k(\mathbf{u}^i; r)}{\sum_{i=1}^{D^{trn}} \phi_k(\mathbf{u}^i; r)} \tag{3.45}$$

during training with a set of vectors $\mathcal{D}^{trn} = \{\mathbf{u}^i\} \quad 1 \leq i \leq D^{trn}$. Here y^i is the training output corresponding to the pattern $\mathbf{u}^i$. The output $y^{(\mathrm{K})}(\mathbf{v})$ and memory $\mathbf{m}_k^{(\mathrm{K})}$ can belong to any space in which a weighted average can be defined. For classification problems, y^i is an indicator function, and for an associative memory, $y^i = \mathbf{y}^i = \mathbf{u}^i$ is a bit string.

The network response to test pattern $\mathbf{v}$ is

$$y^{(\mathrm{K})}(\mathbf{v}) = \frac{\sum_{k=1}^{T} \mathbf{m}_k^{(\mathrm{K})} \phi_k(\mathbf{v}; r)}{\sum_{k=1}^{T} \phi_k(\mathbf{v}; r)}, \tag{3.46}$$

which is

$$y^{(\mathrm{K})}(\mathbf{v}) = \frac{\sum_{i=1}^{D^{trn}} y^i \sum_{k=1}^{T} \phi_k(\mathbf{u}^i; r)\phi_k(\mathbf{v}; r)}{\sum_{i=1}^{D^{trn}} \sum_{k=1}^{T} \phi_k(\mathbf{u}^i; r)\phi_k(\mathbf{v}; r)} \tag{3.47}$$

in terms of the training data. (A further thresholding operation is required if a the output is to be a bit string.) Each training pattern $\mathbf{u}^i$ contributes its desired output $y(\mathbf{v})$ to an average weighted by

$$\sum_{k=1}^{T} \phi_k(\mathbf{u}^i; r)\phi_k(\mathbf{v}; r) \stackrel{def}{=} T\left(1 - \frac{\rho_r^{(\mathrm{K})}(\mathbf{v}, \mathbf{u}^i)}{T}\right) \tag{3.48}$$

the number of centres within Hamming distance r of both the training pattern $\mathbf{u}^i$ and the test pattern

v. Plugging identity 3.48 into equation 3.47 we obtain

$$y^{(\mathrm{K})}(\mathbf{v}) = \frac{\sum_{i=1}^{D^{trn}} y^i \left(1 - \frac{\rho_r^{(\mathrm{K})}(\mathbf{v},\mathbf{u}^i)}{T}\right)}{\sum_{i=1}^{D^{trn}} \left(1 - \frac{\rho_r^{(\mathrm{K})}(\mathbf{v},\mathbf{u}^i)}{T}\right)} \tag{3.49}$$

which is resembles n-tuple regression network output defined earlier by equation 3.36.

Evidently, $\rho_r^{(\mathrm{K})}(\mathbf{v},\mathbf{u}^i)$ plays a role similar to the tuple distance, with centres within distance r of both patterns being counted instead of n-tuples. Due to the random placement of centres, the expectation of $\rho_r^{(\mathrm{K})}(\mathbf{v},\mathbf{u}^i)$ is also a function $\rho_r^{(\mathrm{K})}(H)$ of the Hamming distance $H = H(\mathbf{v},\mathbf{u}^i)$, although it is more complicated than (3.11) or (3.14). The exact form

$$\rho_r^{(\mathrm{K})}(H) \stackrel{\text{def}}{=} T - \sum_k \phi_k(\mathbf{u}^i; r)\phi_k(\mathbf{v}; r) \tag{3.50}$$

is a sum of products of binomial coefficients which can be approximated (Kanerva, 1988) by

$$\rho_r^{(\mathrm{K})}(H) \approx T\left(1 - \int_{H/R}^{1} \frac{dx}{2\pi\sqrt{x(1-x)}} e^{-\frac{(x-R/2)^2}{R/2}\frac{1}{(1-x)}}\right) \tag{3.51}$$

for $0 \ll H \ll R$.

To carry the comparison even further we observe that the interpolative n-tuple network can be regarded as a special case of the Kanerva network (3.45), (3.46), if the tuple distance in 3.36 is replaced with a Hamming distance restricted to a tuple. Specifically, a Kanerva centre $\boldsymbol{\xi}_{\alpha_i}$ can be associated with each memory location α at each n-tuple i by defining all bits of $\boldsymbol{\xi}_{\alpha_i}$ arbitrarily except for those involved in the i-th input mapping. These bits must form subpattern α: $\sum_{j=0}^{n-1} \xi_{\eta(i,j)} 2^j = \alpha, \ \forall i$. This gives a total of $T2^n$ hard locations. The Hamming distance in space $\mathbb{H}^R$ is replaced by a Hamming distance in the i-th subspace $\mathbb{H}_i^n$ (different for each tuple) defined by a mapping η of the i-th tuple: $\sum_{j=0}^{n-1}(1 - \delta_{\boldsymbol{\xi}_{\eta(i,j)},\mathbf{u}_{\eta(i,j)}})$. The radius $r = 0$ which ensures that out of the total $T2^n$ locations exactly T of them having a feature α_i will be updated. Furthermore, the tuple distance ρ equals Kanerva distance $\rho_0^{(\mathrm{K})}$ as defined by equation 3.50. Note that approximation 3.51 cannot be used in this case because it was derived for a different similarity measure and space (Hamming distance in hypercube $\mathbb{H}^R$).

3.7.2 RAMnets and the Associative Memory

There is a connection between RAMnets, especially a Frequency Weighted version, and the Associative Memories (Pao, 1989; Kohonen, 1978). Let $\mathbf{x}_i$ be the i-th binary pattern vector in R-dimensional space and let $\mathbf{a}_i$ be a binary key vector in c-dimensional space associated with $\mathbf{x}_i$. The storage of the association pairs $\{\mathbf{x}_i, \mathbf{a}_i\}_{i=1}^{i=D^{trn}}$ is achieved by the superposition of the outer-products

$$\mathrm{M} = \sum_i \mathbf{a}_i \mathbf{x}_i^T. \tag{3.52}$$

The recall of the key $\mathbf{a}_i$ upon the presentation of the pattern $\mathbf{x}_i$ occurs by a simple matrix multiplication. The perfect recall is in general not possible because of the "cross-talk" between the different association pairs. In general,

$$\begin{aligned} \mathbf{M}\mathbf{x}_k &= \left(\sum_i \mathbf{a}_i \mathbf{x}_i^{\mathrm{T}}\right) \mathbf{x}_k \\ &= \mathbf{x}_k^{\mathrm{T}}\mathbf{x}_k \mathbf{a}_k + \sum_{j \neq k} \mathbf{x}_j^{\mathrm{T}}\mathbf{x}_k \mathbf{a}_j \end{aligned} \tag{3.53}$$

but if the patterns are orthogonal ($\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j = \delta_{i,j}$) the recall is perfect and $\mathbf{M}\mathbf{x}_i = \mathbf{a}_i$.

The n-tuple sampling can be viewed a mapping $\mathbb{H}^R \rightarrow \mathbb{H}^{2^n T}$ between the binary input vector $\mathbf{x}$ in R-dimensional space and the tuple vector $\mathbf{u}(\mathbf{x})$ in an enhanced $2^n T$-dimensional space (Austin, 1989b; Austin & Stonham, 1987). The vectors in the enhanced space are increasingly orthogonal as the tuple size increases. This is demonstrated in figure 3.7 where we plotted the angle (calculated as $\gamma = \mathrm{acos}\left(\frac{\mathbf{u}_i^{\mathrm{T}}\mathbf{u}_j}{\|\mathbf{u}_i\|\|\mathbf{u}_j\|}\right)$) between every pair $\{\mathbf{u}_i, \mathbf{u}_j\}$ of the tuple vectors. The dimensionality of the original input vector $\mathbf{x}$ was $R = 5$. The x and y axis are labelled with input vectors $\{\mathbf{x}_i\}$ (there are $2^R = 32$ of them) sorted in the Gray order. The angle γ between the pairs of the tuple vectors approaches 90° as n increases, i.e., the vectors in the enhanced space can be made more orthogonal than in the original space. Consequently, if the n-tuple sampling is used, the cross-talk term $\sum_{j \neq k} \mathbf{x}_j^{\mathrm{T}}\mathbf{x}_k \mathbf{a}_j$ in equation 3.53 can be reduced improving the recall from the memory.

Note the similarity of the storage equation 3.52 (expressed in terms of the correlation matrix $\mathbf{M}$) with the training equation 2.3 (expressed a RAM node or partition of $\mathbf{M}$) for the RAMnets. If the tally truncation threshold θ in 2.3 is set to infinity (which yields the Frequency Weighted RAMnet version) and if the key vector $\mathbf{a}$ in 3.52 is a 1-out-of-c code if the pattern class then the columns of the matrix $\mathbf{M}$ are equivalent to the discriminators of the RAMnet. Moreover, if we maintain the above constraint on $\mathbf{a}$ it is obvious that the binary RAMnet is equivalent to the Willshaw's Associative Memory (Beale & Jackson, 1990) and the first stage correlation matrix in ADAM (Austin, 1989a).

3.7.3 Other Work Viewing RAMnets as Memory Models

Kanerva's sparse distributed memory had been used in n-tuple networks (Guoqing *et al.*, 1992) in order to reduce the storage cost and allow large values of the parameter n. This sparse RAMnet was applied to Chinese handwriting recognition, and good results reported in experiments on a small (3000 patterns) scale. The architecture is identical with the binary RAMnet but each RAM node is implemented as a Kanerva memory. During training the i-th tuple address α_i is compared with T hard locations $\boldsymbol{\xi}^i$ and the contents of the locations within Hamming radius r is set. When an unknown pattern $\mathbf{v}$ is input, the contents of all selected hard locations in each discriminator is accumulated to form a score vector $\mathbf{y}$. The pattern is assigned to a class c with the maximum score component y_c.

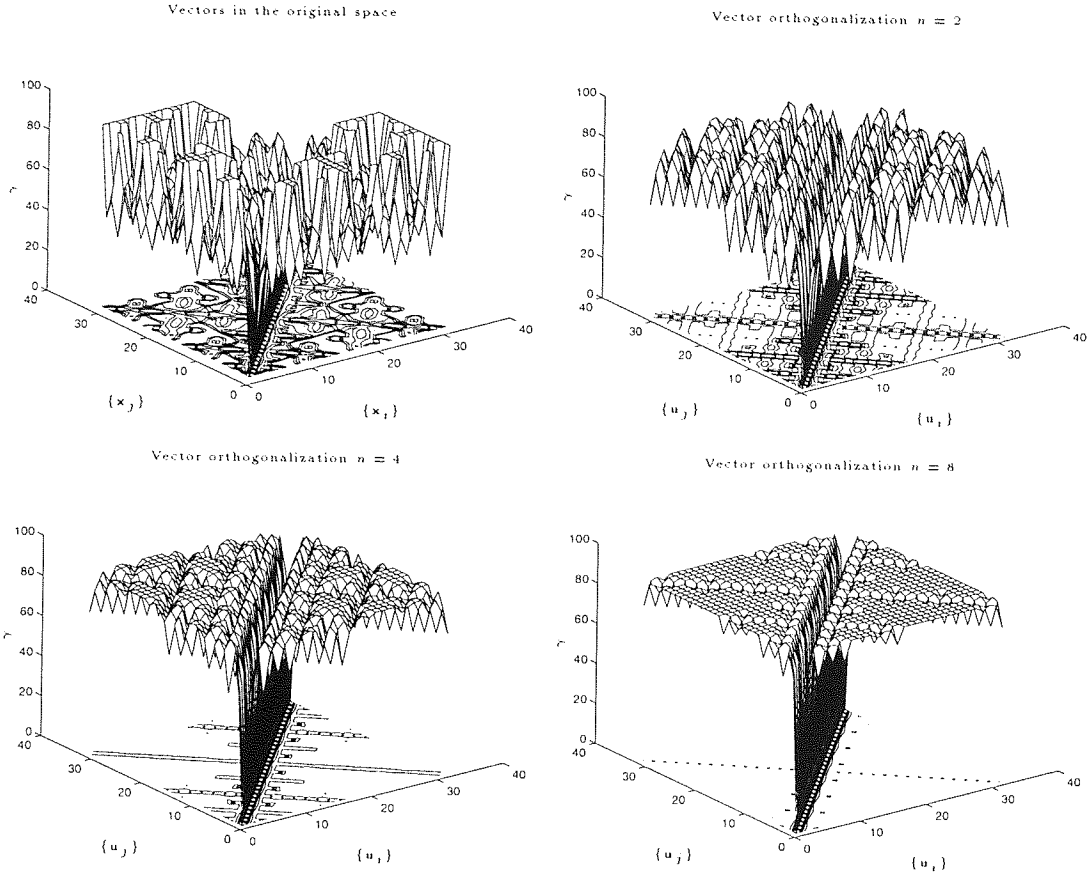


Figure 3.7: Orthogonalization of the tuple vectors $\mathbf{u}_i$ in the enhanced space for different tuple sizes n. The original inputs $\mathbf{x}_i$ are binary vectors in 5-dimensional space. They were Gray sorted. For each of the $2^R = 32$ input vectors we obtained a corresponding tuple vector $\mathbf{u}_i$ by n-tuple sampling all $\mathbf{x}_i$ with 50 tuples. The angle γ between all the pairs of the tuple vectors was calculated as $\gamma = \text{acos}\left(\frac{\mathbf{u}_i^T \mathbf{u}_j}{\|\mathbf{u}_i\| \|\mathbf{u}_j\|}\right)$. The orthogonality of the tuple vectors in the enhanced space increases with the parameter n.

The properties of RAMnets viewed as an associative memory have also been studied (Wong & Sherrington, 1988). In the limit of many nodes and under the assumption that the patterns to be stored are uncorrelated the storage capacity, retrieval error and radius of attraction can be determined. It turns out that RAM nets have lower retrieval error rates than Hopfield networks of the same capacity indicating their potential as neural memories.

Geometrical properties of the n-dimensional boolean space, which are relevant to RAMnets (pattern overlap) have been recently studied (Braga, 1995) and an improved approximation for the Kanerva's distance has been proposed.

3.8 Summary

The early exploratory work on RAMnets investigated the system's performance as a function of its parameters n, T and η. The major contributions include the calculation of the generalisation set $\mathcal{G}_\theta$ and identification of a zero tally problem in the Frequency Weighted version of the classifier. The research in the 1960s and early 1970s was somewhat limited by the lack of cheap storage. Bledsoe and Browning, for example, were strongly in favour of the Frequency Weighted version because it performed better than a binary RAMnet for the values of parameter n ($n \leq 6$) that they were able to implement. In fact, ten years later, the work of Ullmann demonstrated that for larger tuple sizes binary RAMnets outperform the former model, which suffers form the lack of data and poor probability estimates.

RAMnets can be understood by Hamming or Tuple distance analysis. The first method allows us to calculate the expected output $\mathcal{E}[y]$ given the Hamming distance from the test vector $\mathbf{v}$ to the training vectors $\mathbf{u}^i$ and the Hamming inter-distances within the training set itself. It turns out, however, that the formula is not tractable due to its combinatorial complexity.

The Tuple distance approach, on the other hand, sidesteps this problem by considering a RAMnet trained on just one training bit string $\mathbf{u}$. It allows one to calculate the response y to the test vector $\mathbf{v}$ and view the system as a kernel smoother. This idea can be traced back to Aleksander's work on RAMnet windows (compare equations 3.2 and 3.12).

The discovery of an effective kernel function in RAMnets led directly to n-tuple regression networks. Their significant advantage over Specht's regression networks with an explicitly computed Gaussian kernel is twofold. The tuple kernel never needs to be calculated and the network's response does not depend on the size of the training set but on the number of tuples employed.

It is well known that the Bayesian approach of assigning an unknown pattern $\mathbf{v}$ to the class c with maximum posterior $P(c|\boldsymbol{\alpha})$ yields minimum classification error. The Frequency Weighted version of the classifier can be made look like a Bayesian classifier if we introduce class priors $P(c)$, assume independence of tuple addresses in order to compute the likelihood $P(\boldsymbol{\alpha}|c)$ and invert it to obtain the posterior. The independence assumption, however, lacks plausibility and there are further technical problems (zero tally problem) that make those modifications *ad hoc* and unprincipled.

A closer look at the relationship of a binary RAMnet to the Walsh expansion, the latter being a principled method for discrete probability estimation, shows that there is a serious problem with the method itself. It leaves out a number of terms in the expansion and we cannot hope to view the output y_c of a binary RAMnet as a conditional class probability estimate. The potential lack of universality of RAMnets is confirmed by viewing them as a Φ-machine with n-th order polynomial functions. Again, certain terms of the expansion will be missing and in some cases the discrimination

surface will not be able to separate patterns into their respective classes.

N-tuple system can be related to Kanerva's Sparse Distributed Memory. The selection of hard centres ξ_i in RAMnets is not entirely random, as n bits of the centre must correspond to the tuple address α_i. Furthermore, the metric used to calculate the similarity of patterns is Hamming distance restricted to the n-tuple. These arbitrary modifications allow Kanerva's Memory to operate as a RAMnet.

The Frequency Weighted version of the classifier is equivalent to the Associative Memory storing the input vectors mapped into the larger tuple space. The tuple vectors are more orthogonal and therefore during the retrieval phase the cross-talk (tuple overlap) error term is significantly reduced.

N-Tuple pre-processing is used in ADAM which, as the Willshaw's Associative Memory, is closely related to binary RAMnets.

3.9 Conclusions

The review uncovers a number of issues and problems that should be investigated. It determined the direction of my research work and consequently the shape of the chapters to follow.

Most of the experimental work on RAMnets used small datasets of binary data coming almost entirely from the domain of image recognition. This data did not allow an objective comparison of the n-tuple system's performance with other methods. With efficient preprocessing algorithms available (section 2.6) real valued vectors can be mapped into the binary space and allow one to use larger number of datasets for benchmarking. Chapter 4 is devoted to a large scale experiment with StatLog datasets and the comparison of binary RAMnets' performance to other well established methods.

The Frequency Weighted version of the method is interesting because it can be related to the well researched Bayesian classifier. A number of improvements can be introduced so that it resembles it more closely. One should take class priors into account and propose a principled way of calculating probability estimates for zero tallies. These issues and the relationship of the Frequency Weighted version to the binary RAMnet are covered in chapter 5.

It is clear, that there is no theoretical framework for RAMnets on the scale available to other, well established models. It is possible to show how these well understood methods can be made to operate as an n-tuple system but it is more difficult to modify RAMnets so that they are equivalent to them. On the one hand, a slightly modified Kanerva's Memory can be made to operate as a binary RAMnet if we make some requirements on the metric space and the placement of centres. On the other hand, the same RAMnet cannot be shown to perform probability estimation and the modifications and assumptions made to link of the Frequency Weighted n-tuple network to the Bayesian classifier can

be difficult to justify.

It has been demonstrated that n-tuple networks are not universal approximators. Furthermore, they use a training regime which does not minimise a cost function. Knowing that there are sources of potential sub-optimality of the model, we would like to be able to quantify how well (or badly) a RAMnet with given parameters will perform on a given dataset. Chapter 6 is concerned with efforts to quantify the cost of sub-optimality that one has to pay for the speed of the model.

Chapter 4

Benchmarking RAMnets

4.1 Introduction

The n-tuple method of classification is based on memorising randomly selected features. The amount of computation involved is minimal, especially when iterative cost driven methods are used in comparison. The operational speed and ease of hardware implementation are the features of RAMnets that make them an attractive alternative to more sophisticated algorithms.

However, is prudent to suspect that relatively poor performance will accompany the speed and simplicity of the n-tuple algorithm. There are many reports of satisfactory results with the method (see section 3.2) but few studies involving comparisons with other models (Rohwer & Cressy, 1989). Furthermore, most studies use just one or two small data sets. Clearly, there is a need for an exhaustive, comparative benchmarking analysis of RAMnets' performance that would prove or disprove their viability.

Therefore, a large experiment was carried out, in which the n-tuple method was tested on 11 large real-world data sets which had been previously used by the European Community ESPRIT StatLog project (Michie *et al.*, 1994) to test 23 other classification algorithms including the most popular neural network methods. The significant advantage of the StatLog data is the diversity of the algorithms studied and the fact that most of it is in the public domain (available via ftp from `ftp.strath.ac.uk` or `ics.uci.edu`).

The results, presented in section 4.7.2, show no systematic performance gap between the n-tuple method and the others, on 7 of the 11 data sets tested. It is easy to recognise the other 4, without referring to any competing methods, because the n-tuple method performed no better than random guessing. The experiments therefore suggest, that in most cases the n-tuple method gives competitive performance, and the cases when it does not are clearly recognisable.

When a fast and simple method proves to be a competitive performer in a large set of experiments, one would like to know why. Although, the n-tuple method has not yet yielded to theoretical analysis as well as the optimisation-based approaches which can be embedded in Bayesian statistical theory (MacKay, 1992b), the theoretical tools reviewed and discussed in the previous chapter prove helpful in developing a semi-quantitative account of why the method failed on 4 of the 11 datasets. The main theoretical stumbling blocks are also indicated.

4.2 Description of the Classification Procedures

The classification algorithms used in the StatLog project are described in detail in (Michie *et al.*, 1994). They fall into the three categories: discriminants, decision trees and rule-based methods and density estimators.

Linear discriminants divide the sample space with a number of hyperplanes so that they (ideally) separate the data points generated by different classes. The orientation of the hyperplanes is determined by the shape of the data clusters.

The discriminant models come in a number of flavours depending on the type of the (nonlinear) transformation applied to the data vectors. The following methods belong to this type: linear, logistic and quadratic discriminants, multi-layer perceptrons and projection pursuit. In general, this group consists of statistical methods.

Decision trees and rule-based methods come from the area of Machine Learning. This type of classification procedure is based on a recursive partitioning of the sample space into hyper-rectangles. Each one of them may be split further into a number of smaller volumes. The boundaries between the hyper-rectangles are usually parallel to the attribute axis.

Density estimation methods are concerned with local probability estimation at each training point. A number of statistical models are of this type: naive Bayes, kernel density estimator, probabilistic decision tree and k-nearest neighbours. Neural network methods such as LVQ and the Kohonen map also belong to this group.

The n-tuple method was reviewed but not used in the StatLog experiments, presumably because most of the datasets comprised real-valued, high dimensional data and suitable pre-processing techniques geared towards RAMnets were not available at the time of the project. Section 4.4 describes the details of these techniques and their effect on generalisation properties of the system.

4.3 Description of the Datasets

Originally, 20 commercial and scientific datasets were used in the StatLog project. Each of the larger data sets (with many more than 1000 samples) was randomly split into training and testing partitions. Different methodologies (cross-validation and bootstrap) were applied to the smaller data sets.

There were two constraints that determined the final selection of the datasets for our benchmarking study involving RAMnets: the dataset availability and size. Most of the StatLog data can be downloaded via `ftp` and is either free or the permission to use it in the research can be easily arranged. Large data sets had been selected to demonstrate the speed advantage of RAMnets, and to facilitate an extensive test of the method. Furthermore, the data could be split into training and testing partitions avoiding practical complications of bootstrap and cross-validation. These considerations led to the selection of the following datasets:

Shuttle This data originated from NASA and concerns the position of radiators within the Space Shuttle. It was split into a train and a test set with 43500 and 14500 patterns respectively. There are 7 classes but almost 80% of the examples belong just to class 1. On the other hand, the training set contains only 6 patterns generated by class 6. Each pattern is described by 9 real-valued attributes. The data is noise free and arbitrarily small error rates can be obtained given enough of the training data. Furthermore, the data-points form multiple non overlapping clusters containing examples coming only from one class and the boundaries between clusters are parallel to the coordinate axis (labelled with the attributes). This makes the decision trees ideal for this dataset.

DNA The DNA dataset is concerned with a partitioning problem. Given a sequence of the DNA, the task is to find the boundaries between exons and introns. Exons are the part of the DNA retained and introns are the DNA sequence which is not copied, during protein creation. Extensive preprocessing has been carried out on this data. Each pattern originally consisted of 60 nucleotides, each represented by one of four symbolic attributes (a,c,g,t). The attribute sequence was binary encoded resulting in 180 binary attributes. The training set and the test set contains 2000 and 1186 examples respectively. The class proportions are not equal: there are 3 classes (intron–extron, extron–intron junction and neither of them) and 52% of the data belongs just to one class. Decision tree procedures as well as the statistical algorithms performed well on this dataset.

Technical This dataset is confidential and was supplied by Daimler–Benz AG. It contains an unusually high number of classes (91) and real valued attributes (56), most of which have value zero. The data apparently has been processed by a decision tree process before it was handed over to StatLog participants for evaluation. By considering the four most frequent classes ($c_{69}, c_{72}, c_{77}, c_{78}$) and

tabulating the value of the attribute x^i_{52} it becomes apparent that the correct classifications can be made by placing the decision boundaries at values ± 0.055 and ± 0.085. The classes in this dataset are thus defined by the values of the attributes and good results can be obtained with the decision trees.

BelgianI This dataset was confidential and concerned with stability determination for power systems. The state of the system is described by a 28 component real vector of voltages, power flows and injections which is labelled either stable or unstable. The training and test set contain 1250 patterns each. Classes 1 and 2 appear to have two clusters each, indicating that there may be two types of "stable" state. The statistical algorithms SMART and Logdisc produced best results for this dataset.

BelgianII This data describes the same problem as BelgianI but was generated by a larger simulation. The number of attributes was increased to 57. The training and test data sets comprise 2000 and 1000 patterns respectively. The SMART algorithm again performed well as did the machine learning algorithms: IndCART, NewID and AC^2. Naive Bayes, Kohonen and ITrule gave results worse than the default error rate.

Tsetse The Tsetse dataset is concerned with the distribution of tsetse flies in Zimbabwe. The environmental conditions of the regions in which this insect is present were encoded into 14 real-valued attributes. The two classes (tsetse's presence or absence) are equally well represented in the sample. The training set has 3500 examples whereas the test set contains 1499 patterns. The machine learning algorithms based on decision trees performed best, followed by modern statistical algorithms. The MLP, LVQ and k-nearest neighbours obtained comparable results on this dataset.

Cut20, Cut50 The problem to be solved here is segmentation of words into letters. Each example in the data set contains 50 (20 in Cut20) attributes describing some measurements made on the text along a potential segmentation point. Cut20 was obtained from the original data by selecting the most informative 20 attributes using stepwise regression on the entire dataset. Both versions contain 11220 training patterns and 7480 test examples. The class priors are highly skewed and 94% of the data describes just one class. The attributes are continuous with little correlation, which is suitable for the k-nearest neighbours method which performed very well in this case.

SatIm The data was gathered by a Landsat satellite recording images in four spectral bands from the green to infra-red. The pixelated image of size 82×100 corresponds to an actual area of 80×80m. Because the information given by the neighbouring pixels may help the classification, the data pattern was constructed using a 3×3 pixel area for each spectral range, which yielded 36 integer-valued attributes. The task was to classify each pixel into one of the seven classes corresponding to the

Name	Classes	Largest Prior	Attributes	Training Patterns	Testing Patterns	Description
BelgianII	2	0.924	57 real	2000	1000	Classify measurements on simulated large scale power system as leading to stable or unstable behaviour.
Cut50	2	0.941	50 real	11220	7480	50 measurements from a candidate segmentation point in joined handwritten text. Classify as suitable cut point or not. Commercially confidential data.
Cut20	2	0.941	20 real	11220	7480	Best 20 attributes (by stepwise regression) from Cut50.
Technical	91	0.230	56	4500	2580	Commercially confidential. Appears to be generated by a decision tree. Most attribute values are 0.
DNA	3	0.525	180 Boolean	2000	1186	Sequences of 60 nucleotides (4-valued) classified into 3 categories.
SatIm	6	0.242	36 integer	4435	2000	3x3 pixel regions of Landsat images. Intensities in 4 spectral bands. Classified into 6 land uses at central pixel.
Chromo	24	0.044	16	20000	20000	Images of Chromosomes, reduced to 16 features.
BelgianI	2	0.5664	28 real	1250	1250	As Belgian II with a smaller simulation. Attributes thought to be least informative omitted from simulation.
Tsetse	2	0.508	14 real	3500	1499	Classify environmental attributes for presence of Tsetse flies.
Letter	26	0.045	16 16-valued	15000	5000	Images of typed capital letters, described by 16 real numbers discretised into 16 integers.
Shuttle	7	0.784	9 real	43500	14500	Classification problem concerning position of radiators on the Space Shuttle. Noise-free data.

Table 4.1: Descriptions of Datasets Used.

type of the soil of the area represented by the pixel. The training set contains 4435 examples and the test set 2000 patterns. For this data k-nearest neighbours obtained the best results. Algorithms like RBFs that perform local feature extraction also did well on this data set. However, because the attributes are highly correlated (as we expect from neighbouring pixels), statistical methods assuming independence, like Naive Bayes, were not successful.

Chromo This data was obtained from the MRC Human Genetics Unit, Edinburgh. There are 40000 examples of chromosomes belonging to 24 classes. The data is split evenly into the training and the test set. Each chromosome is described by 16 attributes (features measured directly from the chromosome image, e.g., axis, length). The discriminant algorithms performed best here with the quadratic discriminant being the best. All other methods follow closely with three (NewID, AC^2, and ALLOC80) lagging behind. The probable reason why ALLOC80 (a kernel density estimator) failed, is the non-normality of the distribution of attributes.

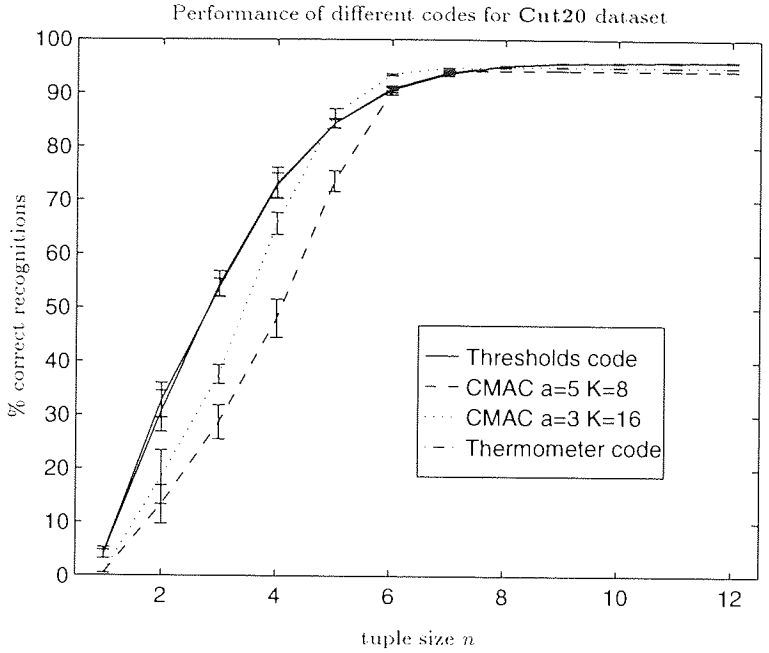


Figure 4.1: RAMnet's performance on Cut20 as a function of tuple size n for CMAC code with $a = 5$, $K = 8$ and $a = 3$, $K = 16$, Thermometer code with resolution 100 bits and Random Thresholds code with 100 thresholds. The error bars of size $\pm$ 1 standard deviation are centred around the mean performance measured for 10 runs. The Random Thresholds code and Thermometer code curves overlap.

The 11 datasets described above were used to benchmark the RAMnet against other algorithms. The summary of the datasets appears in table 4.1.

4.4 Preprocessing of Scalar Attributes

The RAMnet classifies bit strings, but the attributes of the patterns in the StatLog data sets are mostly real numbers or integers. Given that generalisation in RAMnets is related to Hamming distances, it is important to transform numbers into bit strings in such a way that the Manhattan distance is proportionally transformed into the Hamming distance. As discussed in section 2.6.1 the thermometer code preserves exactly the topology of the data points. However, a number of StatLog datasets has high dimensional pattern vectors (50 attributes is not uncommon), which after conversion would lead to large retina sizes R and would call for an impractical number of n-tuples to ensure a generous oversampling of the input. (The comparison of RAMnet with Walsh expansion in section 3.4.1 made it plain that the more terms (tuples) are taken, the better approximation to the underlying class probability density and consequently the better discrimination between the pattern classes.)

Therefore, the CMAC preprocessing scheme was used. It has a number of advantages. It can handle binary, integer or real-valued attributes so it can be applied uniformly to all datasets. Furthermore,

by selecting the parameters a and K, one can trade off the code length R against the size of the subspace where the distance relationships are preserved.

The CMAC code expects positive integers in range $[0, (2^a - 1)K]$. Consequently, all the data was rescaled into this range and rounded down. Each attribute was rescaled separately. The scaling coefficients were calculated using the training data and used for rescaling of the training and test vectors. If, after scaling, the test attribute value was found to be outside the valid range, it was assigned to the nearest valid integer value. After rescaling of the A attributes the pattern vector was CMAC encoded resulting in a bit string of length $R = AKa$.

The preprocessing scheme and the choice of parameters a and K has an influence on the performance of the classifier. Figure 4.1 shows the test set classification accuracy as a function of n for a 100-bit thermometer code, a 48-bit ($a = 3$, $K = 16$) and a 40-bit ($a = 5$, $K = 8$) CMAC code and a 100-bit random thresholds code for one of the datasets studied. For small values of n, the longer codes perform better, presumably because they are linear over a larger fraction of the dynamic range. But with increasing n, the generalisation set shrinks, so more and more of the linear region of the longer codes is ignored by the RAMnet, eliminating their advantage. Meanwhile, accuracy improves with n, eventually levelling out. This is probably due to making higher-order correlations available, as well as reducing saturation effects. It seems that the parameters a and K have no significant effect on the system's performance if n is increased high enough to take advantage of the bit correlation information.

For all the datasets, the parameter values chosen were $a = 5$, and $K = 8$. In the Letter data set (see table 4.1), where the attributes can take on only 16 values, it would be more reasonable to use a one-out-of-N encoding with strings of 16 bits, but the CMAC/Gray procedure was used anyway for the convenience of uniformity.

The maximum code length $R = 2280$ is reached for the BelgianII dataset. This corresponds approximately to the four-fold oversampling of the retina assuming that 1000 8-tuples are used.

4.5 Generalisation Distance and CMAC Constraint

Because tuple score decays exponentially with Hamming distance 3.14, there should be relatively little ill effect if a training pattern further than R/n bits away from a test pattern is replaced by another training pattern further than R/n bits away (although figure 3.1 indicates that R/n is a *very* approximate estimate). Thus if there are A scalar attributes, one can expect the non-linearity of the CMAC/Gray mapping to do little harm if $K > \frac{R/A}{n}$. There are aK bits per attribute, so this

condition is

$$a < n. \tag{4.1}$$

Scalar differences up to $\pm K$ fall within the linear region of the mapping. This represents a fraction $\frac{2K}{(2^a-1)K}$ or about 2^{1-a} of the largest separation allowed. With $a < n$, the "generalisation Hamming distance" $\frac{R/A}{n} = \frac{aK}{n}$ corresponds to a scalar separation of $\pm\frac{aK}{n}$, which is the fraction $\frac{2a}{n(2^a-1)} \approx \frac{a}{n}2^{1-a}$ (for $a > 1$) of the largest possible scalar separation.

For $a = 1$, the mapping becomes the "thermometer code", in which integer x is mapped to a bit string with the last x bits set and the remaining $K - x/K$ unset. If K is adjusted to preserve the input interval, then larger a values give shorter codes, which should be similarly effective as long as $a < n$ and scalar attributes separated by more than fraction 2^{1-a} of their dynamic range can be regarded as dissimilar as far as generalisation is concerned. Note, that if the constraint 4.1 is to be met for the values of a and K selected ($a = 5$, $K = 8$), the tuple size n should have a value $n > 5$.

4.6 Practical Experience for Setting n and T

Simple theoretical considerations and practical experience provide fairly strong guidance for setting the architectural parameters, n, T, and θ. To begin with, the fact that the network response to an arbitrary pattern is essentially an average over the n-tuples (see equations 2.4, 2.5) means that the results should become increasingly consistent with increasing T. Because the n-tuple method can process an entire dataset within seconds, there is little need for any data analysis method other than explicit measurement of the variation of performance as the input mapping is re-randomised a few times for a given T. T is increased if the variation is unacceptably high. Practical experience indicates that T should be large enough to ensure a generous oversampling of the retina. Usually, values of 100 to 1000 turn out to be adequate. It is commonly observed that the network's output for the winning class exceeds that of the second runner up by an uncomfortably small margin, such as 3 n-tuples out of 1000, but that the correct class nevertheless wins consistently. Perhaps most n-tuples give a constant response to most patterns, so effectively only a fraction of those in the network are contributing to the decisions. This suggests that many n-tuples could be trimmed from the network, but such variations on the method are vulnerable to over-training and complicate the theory.

Practical experience tends to favour small values of the threshold θ, particularly $\theta = 1$. A possible rationale for this was given in section 3.4.2.

Many considerations apply to the choice of n-tuple size n. Experimentally it usually turns out that bigger is better, up to an impractically large size (Rohwer & Lamb, 1993), which requires an unreasonable amount of training data, but $n = 8$ is usually enough, and $n = 3$ is sometimes adequate.

This can be explained qualitatively by observing that information about the correlations among up to n bits is available to the classifier. It never hurts to take account of higher-order correlations, but it is plausible that 8-th order correlations contain all that is needed for most binary data sets. Another intuition is that the training process should write to neither too small nor too great a proportion of the 2^n addresses at each node. If n is too large, the sub-patterns occuring in the training data will be unlikely to recur in the test data, whereas if n is too small, the memory can saturate, in which case $m_{cia} = \theta$ for most memory locations, so most discriminative power is lost (see section 2.7).

Moreover, results of the previous section indicate that for real valued datasets processed with the CMAC algorithm the constraint 4.1 should be satisfied by choosing $n > a$.

These issues are further complicated if the class priors are highly skewed, so that one class has far more training data than another. Although a precise theory is not available, there are strong enough theoretical tools to gain considerable insight, as is demonstrated in the discussion of the experimental results.

4.7 The Experiments

The threshold θ was set to 1 in all the experiments reported here, the n-tuple size n was set to 8, and T was set to 1000 n-tuples. These parameter values yield good classification results for all the datasets and have been determined experimentally. They are also supported by the arguments given in section 4.6. The results reported are averages over 10 different random input mappings η. Plots of the binary RAMnet's performance as a function of the tuple size n appear in figure 4.2. The solid line indicates the percentage of correct classifications. Ties were counted as misclassifications. The dotted line shows the RAMnet's performance with class prior information used to resolve the score ties. The latter procedure yields improved performance for small values of n. This advantage tends to disappear as n increases because the saturation drops down and the score ties are less likely. The results for the comparison with other algorithms include ties resolution which makes a significant difference only for the Technical dataset.

The advantage of the random mapping over the excusive sampling scheme is demonstrated by the lack of a drop off in recognition performance for large n. Exclusive mapping severly reduces the number of tuples as the tuple size n increases, whereas the random mapping does not impose any constraint on T.

As a technical point, we note that the experimental procedure for determining n involves using test data to set an architectural parameter. However, the subsequent re-randomisation of the input mapping η completely scrambles the network connectivity, re-defining the random features used for

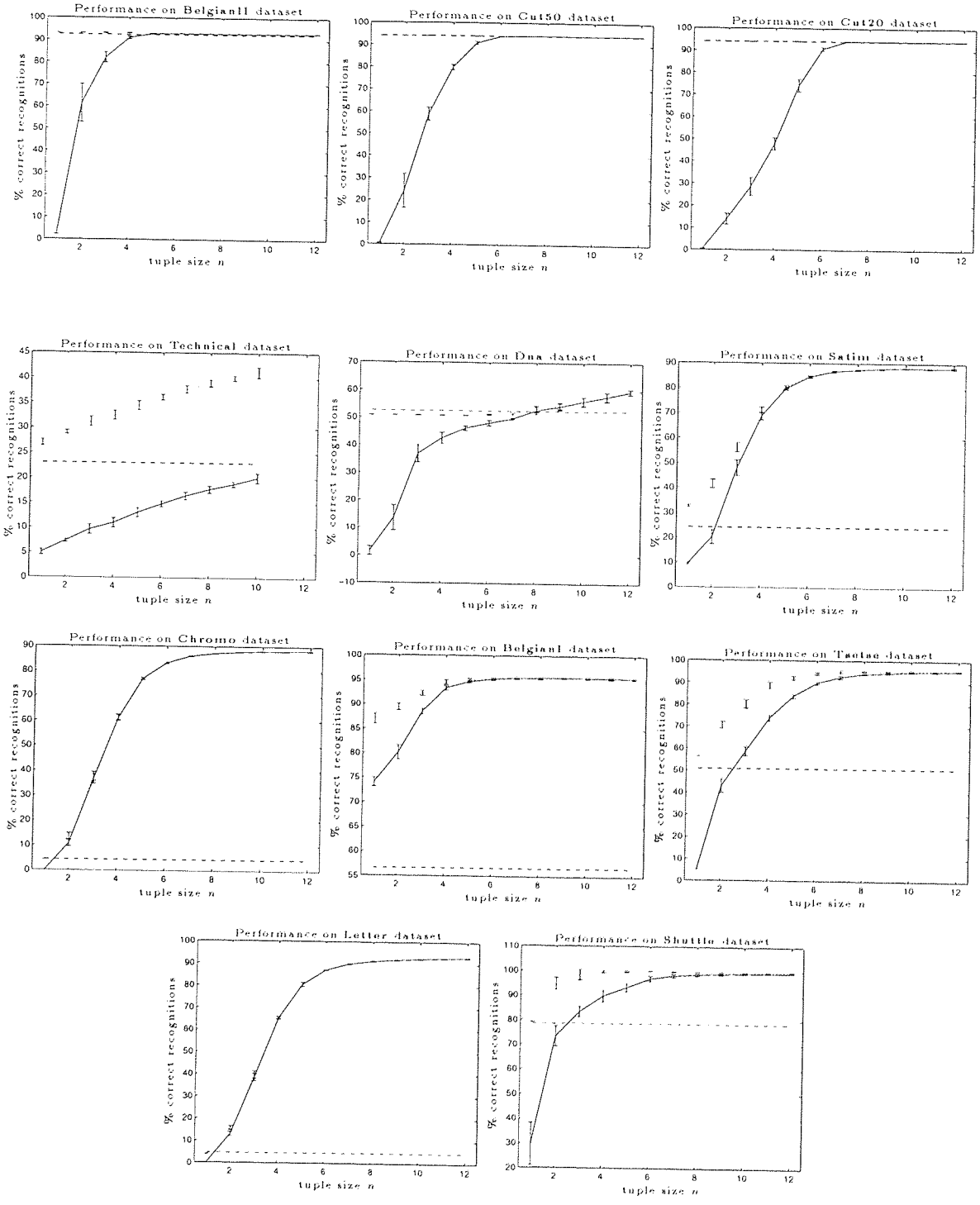


Figure 4.2: Performance of the binary system with 1000 n-tuples as a function of n for StatLog datasets. The solid line represents the performance of the system where tied patterns are rejected. The dotted line shows the performance for the system which chooses the class with a maximum prior when a tie occurs. The dashed horizontal line shows the default performance, i.e., performance that can be obtained by always selecting a class that has maximum prior probability.

discrimination. Thus, the experiments do demonstrate generalisation from one input mapping to another, for a given n, and it is difficult to argue that new test data randomly drawn from the same distribution will have a more severe effect than selecting new features randomly from the same data. Hence this procedural expedient was felt justified.

4.7.1 Time and Memory Requirements

Computation time requirements were insignificant in these experiments, which were carried out with a C++ program on a SUN Sparc workstation. For example, an 8-tuple network can be trained on the 2000 57-attribute training patterns of the BelgianII data set in about 49 seconds. Sixteen of these seconds are needed just to read in the data; another 4 to do the CMAC/Gray conversion of the floating point attributes; and the final 29 to train the RAMnet itself. Testing the same 2000 patterns takes slightly longer, 37 seconds instead of 29, because a loop over classes is needed within the loop over n-tuples. Detailed timing statistics are not published for the algorithms used in the StatLog project, but it is clear that popular neural network algorithms such as Back Propagation and even the relatively fast Radial Basis Functions are slow by comparison. The algorithm is highly parallelisable, so if it were important for the RAMnet to be even faster, special purpose parallel hardware could be designed or purchased (Aleksander *et al.*, 1984). It would be feasible for a biological system to implement a highly parallel but otherwise trivial calculation along these lines.

The storage requirements for a fixed n and T are determined by the number of classes (we need as many discriminators as classes). For the parameters chosen the highest memory requirement occured for the Technical dataset, calling for $2^8 \times 1000 \times 91$ bits which is 2.8Mbytes of RAM.

4.7.2 Results

The classification results for each algorithm attempted with each data set are presented in figure 4.3. Table 4.2 gives a brief description of each algorithm with the symbol used to represent it in the figure. The classification error rates increase from left to right, and are scaled separately for each data set, so that they equal 1 at the error rate of the trivial method of always guessing the class with the highest prior probability, ignoring the input pattern.

As remarked in section 4.7, the results plotted for the n-tuple recognition algorithm are averages over 10 randomly selected input mappings. If the corresponding standard deviations were plotted as error bars in figure 4.3, they would be obscured by the dots representing the means.

RAMnets.

(●) n-tuple recogniser.

Discriminators.

(♣) Back Propagation in a 1-hidden-layer MLP.

(♠) Radial Basis Functions.

(♡) Cascade Correlation.

(⊕) SMART (Projection pursuit).

(⊗) Dipol92 (based on pairwise linear discriminators).

(⊖) Logistic discriminant.

(⊙) Quadratic discriminant.

(⊙) Linear discriminant.

Methods related to density estimation.

(α) CASTLE (Probabilistic decision tree).

(β) k-NN (k nearest neighbours).

(γ) LVQ (Learning Vector Quantisation).

(δ) Kohonen topographic map.

(ε) Naive-Bayes (Estimate assuming independent attributes).

(ζ) ALLOC80 (Kernel function density estimator)

Decision trees.

(a) NewID (Decision Tree)

(b) AC^2 (Decision Tree)

(c) Cal5 (Decision Tree)

(d) CN2 (Decision Tree)

(e) C4.5 (Decision Tree)

(f) CART (Decision Tree)

(g) IndCART (CART variation)

(h) BayesTree (Decision Tree)

(i) ITrule (Decision Tree)

Table 4.2: Synopsis of Algorithms with symbols used in Figure 4.3.

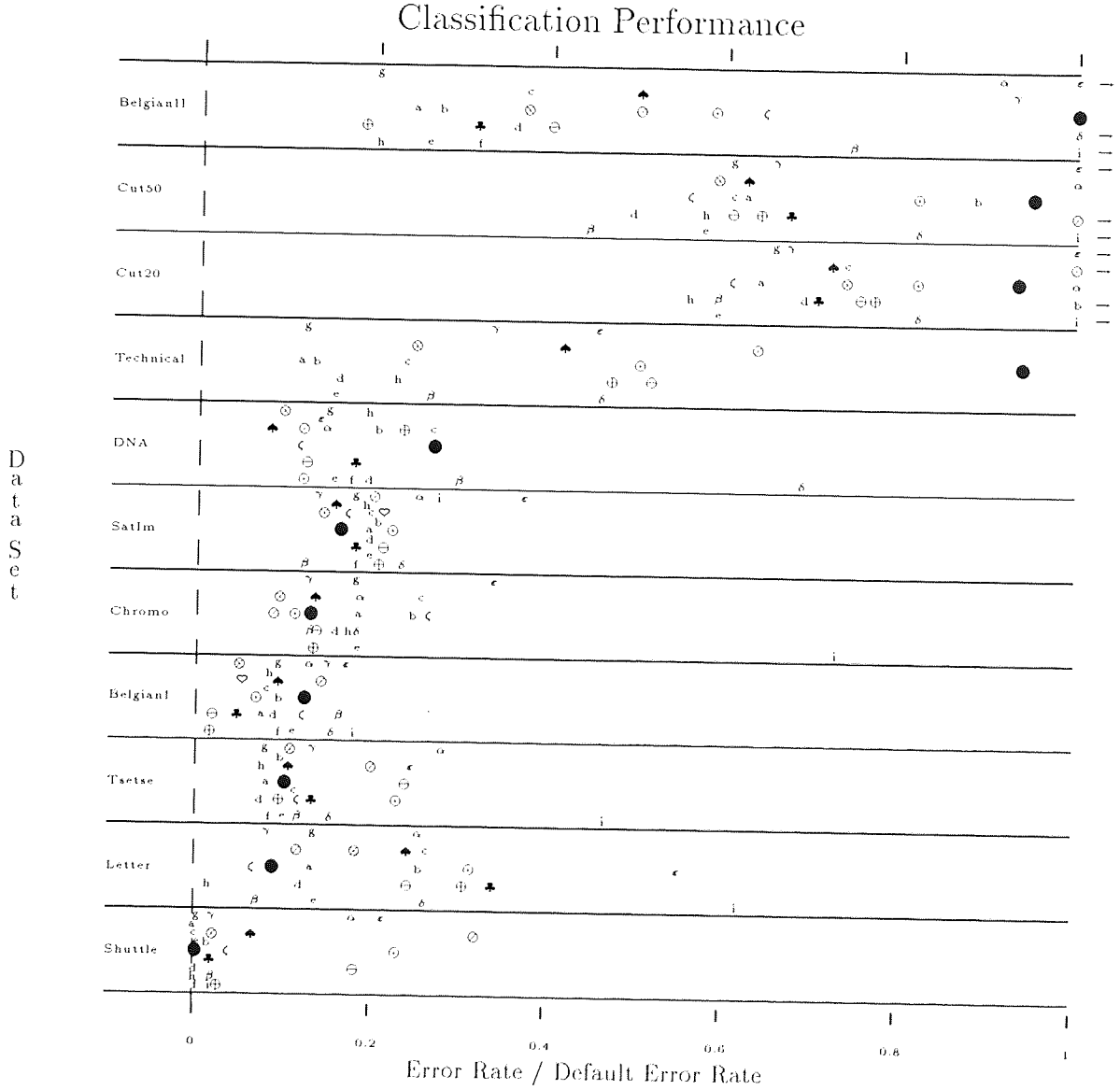


Figure 4.3: Results for n-tuple (●) and other algorithms. Algorithm codes appear in Table 4.2. Classification error rates increase from left to right, and are scaled separately for each data set, so that they equal 1 at the error rate of the trivial method of always guessing the class with the highest prior probability, ignoring the input pattern. The arrows indicate the few cases in which performance was worse than this.

4.8 Analysis of Results

The n-tuple method delivered competitive accuracy on 6 of the data sets tested (Shuttle, Letter, Tsetse, BelgianI, Chromo, SatIm), performed modestly on 1 (DNA) and failed entirely on the other 4 (Belgian II, Cut50, Cut20, Technical). Further experimental and theoretical analysis was carried out to explain the failures.

The available tuple distance theory is not amenable to treating overlap effects, even though they determine the actual outputs y_c of the RAMnet. Subsection 4.8.1 discusses the role of overlap in the classification decision made by the system and sheds some light on the failure of the method for the datasets with highly skewed priors.

The tuple distance theory indicates that the classification of an unknown pattern $\mathbf{v}$ is influenced by the class identity of its nearest neighbours $\mathbf{u}^i$. With the CMAC preprocessing scheme this leads to the idea of "generalisation hypercubes" centered on the training data points $\mathbf{u}^i$. From this viewpoint, presented in subsection 4.8.2, the datasets which the RAMnet failed to process correctly do not contain enough training examples to let the n-tuple network generalise correctly.

4.8.1 Training Data Overlap on Tuples

A network trained on a set of patterns $\{\mathbf{v}^i\}$ could respond to a test pattern $\mathbf{u}$ by any amount between $T\min(1, \max_i e^{-n\frac{H(\mathbf{u},\mathbf{v}_i)}{R}})$ and $T\min(1, \sum_i e^{-n\frac{H(\mathbf{u},\mathbf{v}_i)}{R}})$, depending on the correlations between the training patterns, as manifest in "overlap effects". Unfortunately, this circumstance limits the usefulness of the Tuple distance for explaining the standard n-tuple method. Because of a combinatorial explosion, there is no feasible method of measuring tuple correlations. Similar problems crippling an attempt of full formal analysis of the method (see section 3.3.1) for datasets of arbitrary size have been reported. However, some insight into the mechanism of the RAMnet can be gained by analysis of the experimental data.

Overlap effects are displayed in figure 4.4. Figures 4.4a and 4.4b show the network output for a test pattern as training patterns are added in Hamming distance order from the test pattern. For figure 4.4a, the plots show the actual tuple score, and for figure 4.4b it is accumulated for all the training patterns, effectively ignoring the overlap.

Figure 4.4b shows that distant patterns of the incorrect class match the test pattern on many n-tuples, but figure 4.4a shows that most of these subpatterns had already turned up in closer training patterns. Figure 4.4c shows the number of new RAM locations accessed as training patterns are accumulated, regardless of whether these are accessed by the test pattern. One class is disadvantaged by a smaller prior probability, and correspondingly fewer training samples, but it has the advantage

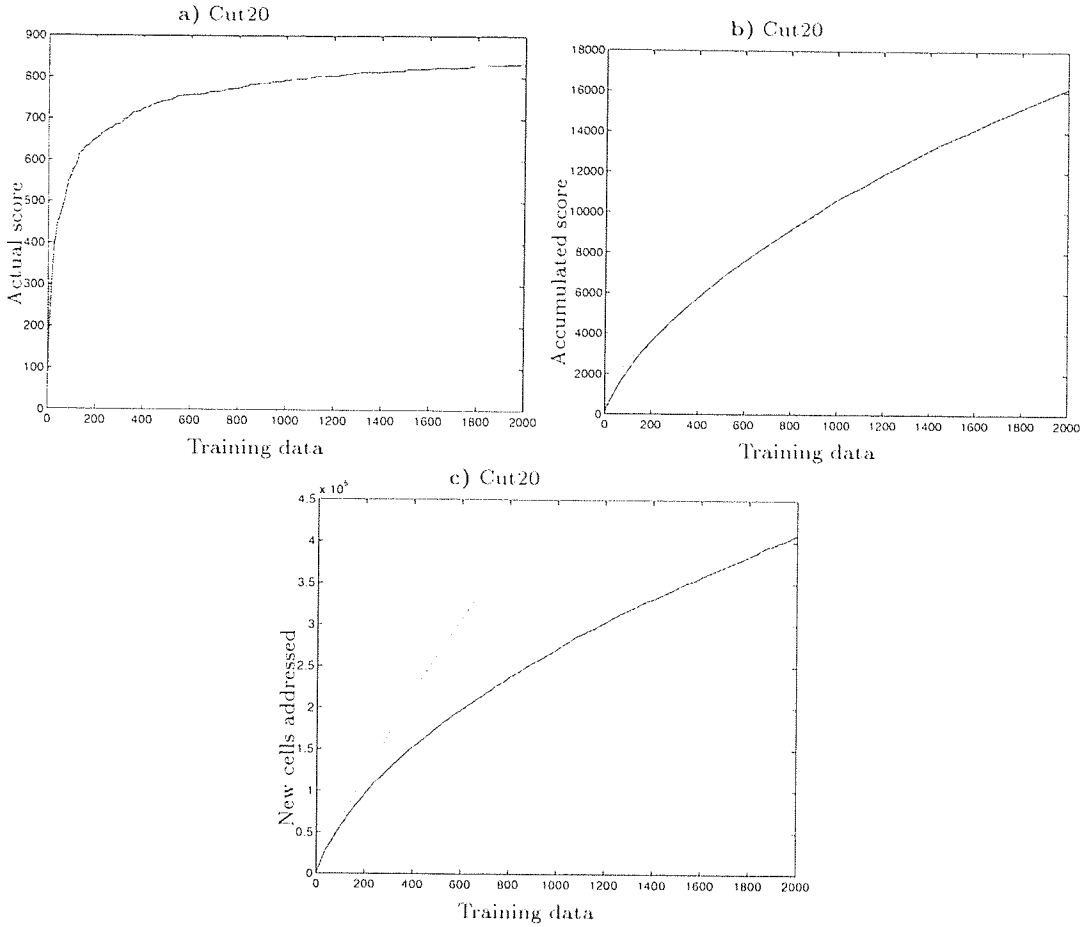


Figure 4.4: **a)** actual network response to a test pattern $\mathbf{u}$; **b)** accumulated network response (as though patterns never overlapped on any tuples) and **c)** the number of new RAM cells addressed as a function of training patterns. Light lines are used to indicate the discriminator associated with the class that generated $\mathbf{u}$, dark lines denote the other class. The training data is sorted by the Hamming distance to the test pattern.

that it populates new RAM locations more rapidly. Whereas the more probable class is showing signs of levelling off in this respect, the less probable class is not, so it seems likely that if more data were available (in the same proportions), then the test pattern would be more likely to be classified correctly with the less probable class. This scenario was frequently observed in datasets with skewed priors, and motivates the hypothesis that errors would have been reduced if more data were available.

4.8.2 The Generalisation Hypercubes

The tuple distance theory assumes that only one training vector $\mathbf{u}$ was used and thus ignores the overlap effects which affect the RAMnet's output.

In order to gain some insight into classification decisions made by a network trained with many training patterns $\mathbf{u}^i$, many detailed Hamming distance vs. tuple score plots like Figure 3.1 were generated and inspected.

Ignoring overlap effects, a test pattern should be assigned to the class of most of the training patterns that lie nearer to it, in Hamming distance, than about R/n. A glance at figure 4.5 shows that the distribution of tuple distances bears no systematic relationship to R/n, but nevertheless it was found that the closest patterns did tend to determine the decision, at least when the class priors were roughly equal. Figures 4.5a and 4.5b show examples of this, which was by far the most common situation encountered. Figure 4.5c shows an error due to overlap effects, abetted by a highly skewed prior (this is the same type of error as illustrated previously in figures 4.4a-c). The 4 troublesome datasets had highly skewed priors and predominately showed this pattern, although the very easy Shuttle dataset also had highly skewed priors. Figure 4.5 shows a relatively rare situation in which overlap effects rescued a pattern which would have been misclassified, judging by its Hamming-near neighbours. It would appear that although overlap effects are quantitatively important, they tend not to alter the conclusion that the near neighbours determine the decision, at least when the priors are relatively uniform.

Given that Hamming neighbours tend to determine the classification outcome, it seems sensible to suspect that test patterns in the 4 problematic data sets have a shortage of good neighbours. It turns out that they simply don't have enough neighbours at all, within the distance scales relevant to RAMnet generalisation. To generalise properly, a test pattern must have at least 1 training pattern within a Hamming distance of about R/n. Distributed evenly over A CMAC/Gray-mapped scalar attributes, this is a scalar difference of about $\frac{a}{n}2^{1-a}$, with the attributes scaled to lie between 0 and 1, as explained in section 4.5. Therefore, each training pattern can provide information about any test pattern which falls within a hypercube of volume roughly $\left(\frac{a}{n}2^{1-a}\right)^A$. The number of such cubes required to cover the region of attribute space where test data is likely to appear can be crudely estimated by approximating this region as a hyper-rectangle with edge lengths given by the eigenvalues of the sample covariance matrix of the training data. Any eigenvalues smaller than $\frac{a}{n}2^{1-a}$ should be rounded up to this value, because the covering cubes must be at least this thick. The number of "generalisation hypercubes" required to cover the data region is therefore roughly $\prod_{i=1}^{A}\max(1, \lambda_i \frac{n}{a}2^{a-1})$ for $1 < a \leq n$, where the λ_i are the eigenvalues. Figure 4.6 shows this lower bound on the number of training samples required, for each dataset studied, taking $a = 5$ and $n = 8$ as in the experiments.

Aside from Technical and DNA, the problematic datasets stand out as several orders of magnitude more deficient in training data than the others, some of which are mildly deficient according to this crude estimate. DNA is special in that its Boolean attributes were treated as integers, so it's data distribution will be highly non-Gaussian and therefore poorly described by the covariance matrix. The Technical data set turned out to be coverable by just 1 hypercube, according to this estimate. Presumably then, each of its patterns looks the same to the RAMnet, and this accounts for its failure.

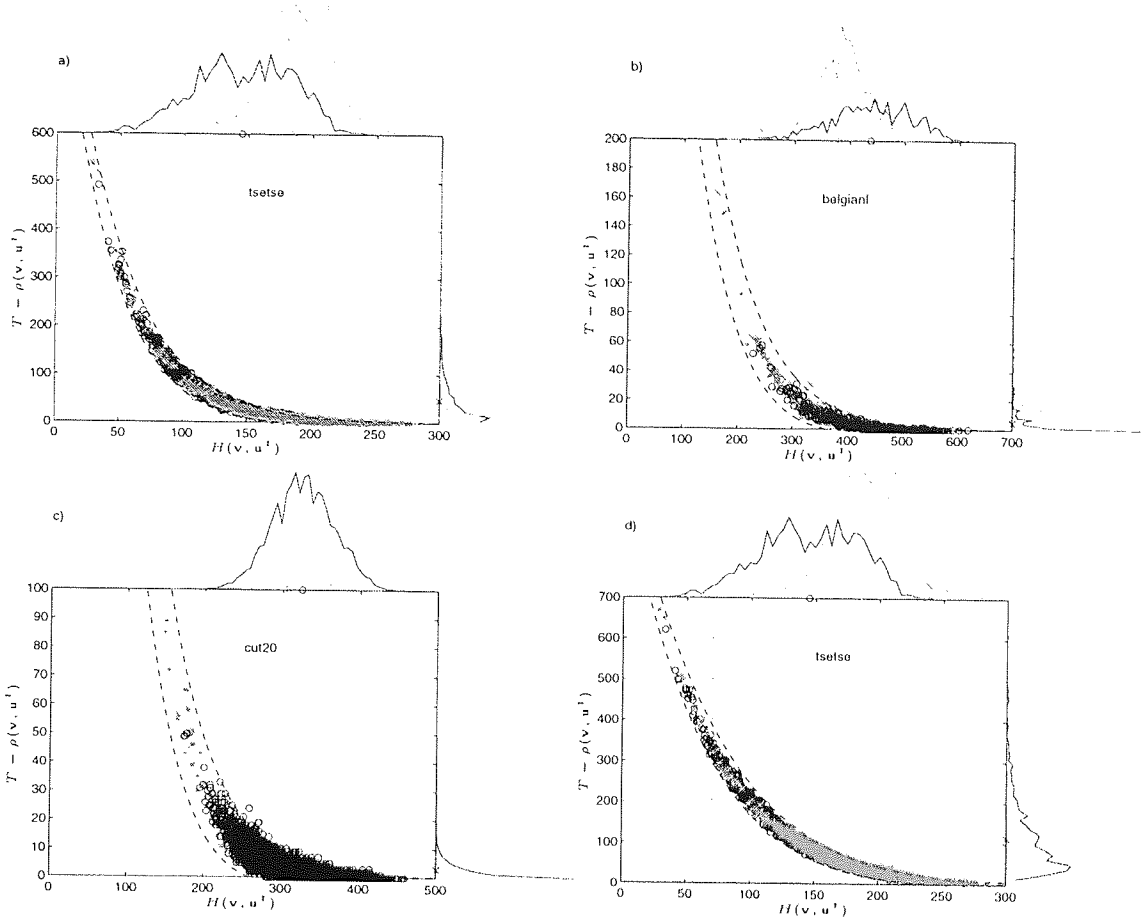


Figure 4.5: Tuple score vs. Hamming Distance: **a)** Classification error (∗=952, ○=970) due to high number of NN from the incorrect class; **b)** Correct classification (∗=803, ○=564) resulting from the domination of NN patterns of the correct class; **c)** Classification error (∗= 805, ○=884) caused by exceptionally large number of patterns from the incorrect class in the tail of the distribution (skewed priors); **d)** Correct classification (∗=996, ○=994) thanks to the smaller tuple overlap on patterns from the correct class.

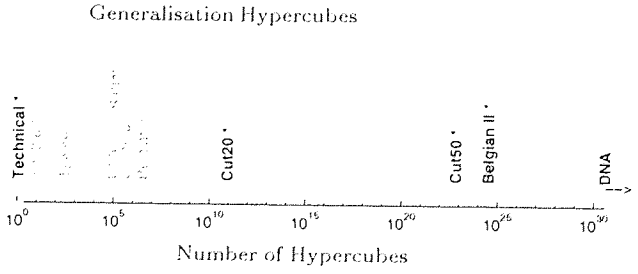


Figure 4.6: The number of hypercubes required to cover the space occupied by data. The datasets on which n-tuple classifier performed poorly are printed in bold face. A star denotes the existence of skewed priors.

It is not possible to address the data deficiencies by supplying more data, especially when several orders of magnitude more samples are needed, but it is possible to tweak the RAMnet parameters to enlarge the "generalisation cubes". However, there is less room to maneuver than one would like. To enlarge the cubes, n must be decreased, but this risks degradation of performance due to loss of high-order correlation information, as indicated in figure 4.1. Decreasing n also requires decreasing a, if the constraint $a \leq n$ is to be respected, keeping R/n within the linear region of the CMAC/Gray mapping. Low a values give less memory-efficient representations of scalars, at any given resolution. Systematic experiments varying the parameters did not produce significant improvements on the 4 problematic data sets (or the others). A more far-reaching improvement in the algorithm is required.

4.9 Summary

Extensive experimental trials, on a scale uncommon for *any* algorithm, were carried out with the n-tuple classifier. The fact that this was possible at all testifies to the method's impressive speed, which derives from its simple principle of learning by 1-shot memorisation of random features. In 6 of the 11 datasets tested, this speed and simplicity can be enjoyed without sacrificing classification accuracy relative to 23 other slower methods, including the most popular neural network methods.

Most of the StatLog data is real-valued and had to be pre-processed in order to obtain binary inputs. The CMAC mapping scheme was used for this purpose because it yields compact codes.

In order to explain what went wrong on the 4 data sets which gave poor results the theoretical tools (presented in section 3.3.2) were used to understand the generalisation properties of the n-tuple classifier (which are defined in the Hamming space of the CMAC code words) in the original real valued attribute space. It turns out that the requirement of proportionality between the Manhattan distance and the Hamming distance imposes a constraint on the CMAC parameter a and the tuple size n.

These considerations were then used to develop the idea of generalisation hypercubes, which give a crude estimate of the number of training samples required for a RAMnet to generalise properly.

An analysis of the experimental data also provided some insight into why skewed class priors pose a problem to the classifier. Training patterns generated by a more probable class populate more memory cells and are more likely to be winners in the simple scoring scheme implemented by the binary RAMnet.

4.10 Conclusions

In spite of its imperfections, the n-tuple method demonstrates that its underlying principle, learning by memorisation of random features, is a powerful one. Competitive results can be obtained for a number of datasets indicating that RAMnets are a viable classification method.

However, because the output of the binary RAMnet is related to the likelihood $P(\boldsymbol{\alpha}|c)$ (see section 3.4.2) rather than to the class posterior $P(c|\boldsymbol{\alpha})$, it has problems with datasets in which the class priors are skewed. This problem is addressed by the Frequency Weighted RAMnets which we discuss and compare with the binary version in the next chapter.

Furthermore, it turns out that one parameter, the n-tuple size n, controls both the distance-scales associated with generalisation behaviour, and the complexity of the random features used to discriminate classes. For some data sets it is not possible to find one setting suitable for both of these considerations.

Therefore, there is a need for a theoretical framework that would allow one to predict, for a given data set, the classification performance of the system. Such framework is proposed in chapter 6.

Chapter 5

Frequency Weighted RAMnets

5.1 Introduction

We recall (see section 3.4), that the Frequency Weighted n-tuple system can be obtained from the original, binary system by setting the tally truncation threshold θ to ∞ instead of the more usual 1. This allows one to use full tallies to estimate low-order conditional feature probabilities and apply a Bayesian framework to the classification problem (Liu, 1964; Duda & Hart, 1973).

The material presented here is an extension of chapter 4. It is discussed separately because the Frequency Weighted RAMnets are viewed from a different perspective than the binary model. Consequently, the theoretical considerations involved are quite different from those of the binary version.

Because the score in the Frequency Weighted n-tuple system can be interpreted as a conditional probability estimate, two fundamental problems arise: how should we use the tallies m_{cia} to calculate the corresponding probability estimates $P(\alpha_i|c)$ and how should we combine these estimates in order to obtain the joint conditional class probability $P(c|\boldsymbol{\alpha})$.

We propose a theoretical solution to the first problem, introducing in the place of the commonly applied Maximum Likelihood estimation an alternative approach proposed by Turing and Good. We do not attempt to attack the second problem, i.e., to provide a theoretical explanation of the apparent lack of correlations between tuple addresses which would justify the independence assumptions used below.

We show that the Good-Turing estimation method (GTE) can, to a certain extent, rectify the Zero frequency problem (see section 5.3) by providing, within a formal framework, improved estimates of small tallies. It is also demonstrated that it leads to better tuple system performance than Maximum Likelihood estimation (MLE).

5.2 Frequency Weighted RAMnet with MLE

The Maximum Likelihood estimation has been routinely (Bledsoe & Bisson, 1962; Sixsmith *et al.*, 1990; Ullmann & Kidd, 1969) applied for the Frequency Weighted n-tuple systems. In this approach the training set $\mathcal{D}^{trn}$ can be viewed as a sample of tuple addresses α_i. Given $P(c|\boldsymbol{\alpha})$, the probability of class c conditioned on a feature vector $\boldsymbol{\alpha}$ (the set of all memory locations addressed by an unknown pattern), optimal classification results can be obtained by assigning the unknown pattern to the most probable class.

The following Maximum Likelihood estimates of probabilities arise naturally in the n-tuple system:

$$\begin{aligned} P(\alpha_i|c) &= \frac{m_{ci\alpha}}{D_c^{trn}} \qquad \text{where } D_c^{trn} = \sum_{\alpha} m_{ci\alpha}, \\ P(\alpha_i) &= \frac{\sum_c m_{ci\alpha}}{D^{trn}} \qquad \text{where } D^{trn} = \sum_c \sum_{\alpha} m_{ci\alpha}, \\ P(c) &= \frac{\sum_{\alpha} m_{ci\alpha}}{D^{trn}}, \qquad \forall i. \end{aligned} \tag{5.1}$$

Bayes' rule can be applied to obtain class probabilities. The likelihood $P(\boldsymbol{\alpha}|c)$ and evidence $P(\boldsymbol{\alpha})$ for the full feature vector are impossible to compute directly, but these can be estimated from low order probabilities $P(\alpha_i|c)$ and $P(\alpha_i)$ using independence assumptions.

The most common approach (Sixsmith *et al.*, 1990) assumes that $P(\alpha_i|c)$ as well as $P(\alpha_i)$ are independent[1], where α_i is the address of the pattern in n-tuple i. The conditional class probability can then be approximated by

$$P(c|\boldsymbol{\alpha}) \approx P(c) \prod_i \frac{P(\alpha_i|c)}{P(\alpha_i)}. \tag{5.2}$$

To simplify the computations, it is convenient to use the logarithms of the probabilities. The class posterior can then be approximated with a sum rather than a product:

$$\log P(c|\boldsymbol{\alpha}) \approx \log P(c) + \sum_i \left(\log P(\alpha_i|c) - \log P(\alpha_i) \right). \tag{5.3}$$

However implausible the independence assumption may appear, there have been reports of reasonable results obtained with this method (see section 3.2). The major advantage of the Frequency Weighted system is that it does not suffer from saturation. This makes it superior for small tuple sizes n, but the advantage tends to disappear as n is increased, due to worsening probability estimates based on diminishing tallies in each of the increasingly numerous memory locations (Ullmann, 1969). It would be desirable to modify the Frequency Weighted system in such a way as to retain its robustness for any tuple size n.

[1] It often goes unnoticed that it turns out to be highly restrictive to demand both of these conditions together, a difficulty we presume to be dwarfed by the inaccuracy of each assumption individually.

5.3 Weakness of the MLE and the Zero Tally Problem

In the Maximum Likelihood approach, an estimate $\hat{P}$ of the true probability P of an event is approximated as the ratio of the event's tally r to the sample size N; $\hat{P} = \frac{r}{N}$. Under the assumption that each tally value is binomially distributed (with unknown probability P that the feature is present in a pattern of class c and $1 - P$ that it is not) the ratio $\frac{r}{N}$ is the Maximum Likelihood estimate of P. The uncertainty of the tally can be defined as its standard deviation, which can be estimated as

$$\delta r = \sqrt{NP(1-P)} \approx \sqrt{r(1-P)}. \tag{5.4}$$

In a tuple with n inputs, P is one of $2^n - 1$ other multinomial parameters which sum to 1. Therefore, P is typically much less than 1, so

$$\delta r \approx \sqrt{r}. \tag{5.5}$$

Equation 5.5 shows that the accuracy of MLE is limited for the events with small tallies. The relative tally uncertainty $\frac{\delta r}{r}$, grows with diminishing tallies and becomes undefined for zero tally.

It should be noted that the fact that a tally $r = 0$ for some event doesn't imply that the probability of the event is also zero. It merely states that the event has not taken place in a finite sample of size N. Moreover, because the probabilities assigned to all events are normalised, the assertion that the zero tally probability is non zero requires that the "excess probability mass" contained in the other estimates should be somehow redistributed. This problem is known in the literature (Witten & Bell, 1991) as the Zero Tally Problem.

Various unprincipled, *ad hoc* techniques exist which try to rectify it. The most common one is to add an arbitrary small constant ϵ to each zero tally. However, the choice of a particular constant is difficult to justify formally. We make some experimental observations concerning this Maximum Likelihood system with zero tally correction (MLZ) in section 5.6.

5.4 Good-Turing Estimate (GTE)

An alternative method of probability estimation was originally proposed by Turing and researched in detail by Good (Good, 1953) in the context of species frequencies in a mixed population. It has also been applied in linguistics for n-gram probability estimation (Church & Gale, 1991), statistical text compression (Witten & Bell, 1991) and speech recognition (Katz, 1987). The advantage of GTE over MLE is improvement of the accuracy of the probability estimates derived from non-zero tallies. Moreover, an estimate for objects not present in the sample can also be provided.

Suppose we draw a random sample of size N from the population of objects. For each object we set up a tally. We record n_r, the number of distinct objects that were represented exactly r times in

the sample, so that

$$N = \sum_{r=1}^{\infty} r n_r. \tag{5.6}$$

Let $\hat{P}_r^{GT}$ denote the Good–Turing estimate of the population probability of an arbitrary object that occured r times in the sample. This entails the assumption that all events which occured r times have the same probability P_r. The Good–Turing theorem states that the expected value of P_r for an event with tally r in one particular sample is r^{GT}/N where the smoothed tally r^{GT} can be approximated as

$$r^{GT} \approx (r+1)\frac{n_{r+1}}{n_r} \qquad r \geq 0. \tag{5.7}$$

The uncertainty of the smoothed tally is defined as its standard deviation $\sigma(r^{GT})$ and can be approximated by

$$\delta r^{GT} = \sigma(r^{GT}) \approx (r+1)\sqrt{\frac{n_{r+1}}{n_r^2}\left(1 + \frac{n_{r+1}}{n_r}\right)} \qquad r \geq 0. \tag{5.8}$$

Note that unlike the uncertainty given by equation 5.4, the uncertainty of the smoothed tally is low for small tallies r. It is also defined for zero tallies, because the sample of size N is typically much smaller than the population from which it is drawn.

The derivation of the formula 5.7 is essentially Bayesian with a uniform prior. Let the number of distinct objects (features, species) in the population be s, which is supposed to be finite. For each object we set up a tally m_μ $(1 \leq \mu \leq s)$. We draw a sample of size N from the infinite population of object instances and for all objects record the tally m_μ of its occurrence in the sample. Assume that we are given the true population probabilities $P_1, \ldots, P_\mu, \ldots, P_s$ of the objects. Good's proof is based on the quantity P_r^{GT} which is the probability of an object that occured r times in the sample of size N. It is defined as the expected value of the probability P_μ of the μ-th object, given that it's tally holds a value r:

$$P_r^{GT} \stackrel{def}{=} \mathcal{E}_N[P_\mu | m_\mu = r] = \sum_{\mu'=1}^{s} P_{\mu'} P(\mu' | m_{\mu'} = r) \tag{5.9}$$

We note that because the objects are selected at random, the prior probability of an object being a μ-th one is $1/s$. Furthermore, the likelihood that the μ-th object will occur r times in the sample of size N is given by

$$P(m_\mu = r | \mu) = \binom{N}{r} P_\mu^r (1 - P_\mu)^{N-r} \tag{5.10}$$

assuming that the tally m_μ is binomially distributed. The posterior can now be calculated using the Bayes' rule and the uniform prior assumption as

$$P(\mu | m_\mu = r) = \frac{P_\mu^r (1 - P_\mu)^{N-r}}{\sum_{\mu'=1}^{s} P_{\mu'}^r (1 - P_{\mu'})^{N-r}} \tag{5.11}$$

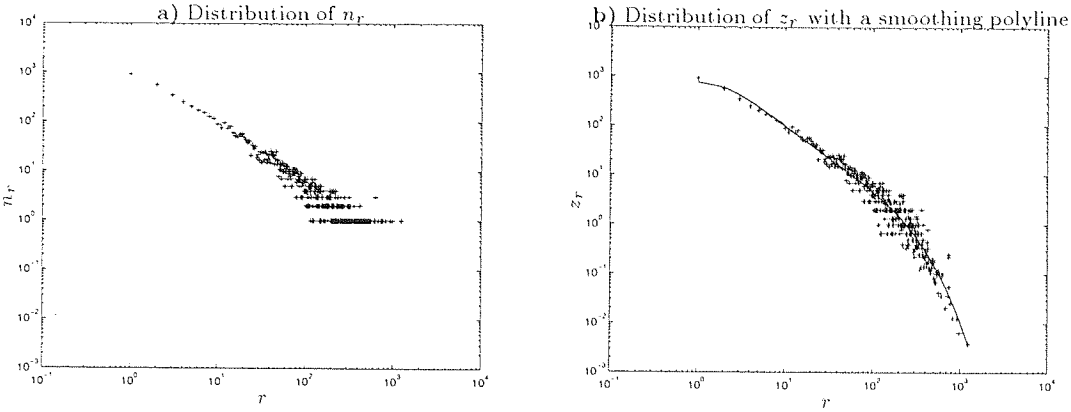


Figure 5.1: **a)** Original distribution of frequencies of tallies in class 0 of the Tsetse database. **b)** distribution after averaging transform has been applied. The solid line denotes the polynomial curve fitted. Note the logarithmic scale of the axis.

and used in equation 5.9 to obtain

$$P_r^{GT} = \frac{\sum_{\mu=1}^{s} P_\mu^{r+1}(1-P_\mu)^{N-r}}{\sum_{\mu=1}^{s} P_\mu^{r}(1-P_\mu)^{N-r}}. \tag{5.12}$$

Observing that the expected frequency of tallies with a value r is of the form

$$\mathcal{E}_N[n_r] = \binom{N}{r} \sum_{\mu=1}^{s} P_\mu^r (1-P_\mu)^{N-r} \tag{5.13}$$

(The random variable n_r is a sum of variables with a value r. The expectation of a sum of variables is equal to a sum of expectations of the variables with a value r, which in turn are assumed to be binomially distributed.) we can express equation 5.12 in terms of $\mathcal{E}_N[n_r]$ and $\mathcal{E}_{N+1}[n_{r+1}]$ obtaining

$$P_r^{GT} = \frac{r+1}{N+1}\frac{\mathcal{E}_{N+1}[n_{r+1}]}{\mathcal{E}_N[n_r]} \approx \frac{(r+1)n_{r+1}}{Nn_r} \approx \frac{r^{GT}}{N} \tag{5.14}$$

where the expectations were replaced with observables and it was assumed that $N \approx N+1$.

Various other derivations (Nadás, 1985) of this theorem exist.

5.5 Smoothing GTEs

In the derivation of the Good–Turing estimate we replaced the expectation of n_r with the observables. Typically, n_1 is large and is the best measured of the n_r values. However, with r increasing this substitution is difficult to justify. Let us consider an example of the frequency distribution of tuple addresses in one discriminator for Tsetse dataset. Part of this distribution appears in table 5.1.

There are 924 distinct tuple addresses that occured just once in the training set, 563 that appeared twice, etc. Note, that small tallies r have larger values of n_r. As r increases it is more probable that

frequency r	frequency of frequency n_r
1	924
2	563
3	350
4	253
5	207
6	171
7	155
8	130
...	...
928	1
979	1
1237	1

Table 5.1: The distribution of feature's frequencies for the Tsetse database.

a given value of r cannot be found in any of the RAM cells. The distribution $\{n_1, n_2, n_r \ldots\}$ tends to be increasingly noisy and requires smoothing.

Moreover, for large values of r there are "gaps" in the distribution of n_r. This suggests that we should average a non-zero n_r value with the zero n_r values surrounding it. We use the transform proposed by Church and Gale (Church & Gale, 1991) given by

$$z_r = \frac{2n_r}{t - q} \tag{5.15}$$

where t, r, q are the successive indices of non-zero n_r. Averaging occurs for larger values of r only, because if there are no "gaps" the transformation has no effect.

After averaging we still have to smooth the z_r. This is accomplished by fitting a log polynomial onto the data. Unlike Gale who used a polynomial of order one (Gale, 1993) we found that polynomials of higher orders are required to obtain a satisfactory fit to the data. Consequently, tally frequency distributions z_r were smoothed with polynomials of order 4, giving a new smoothed tally r^{SGT},

$$r^{SGT} = (r+1)e^{\sum_{i=1}^{4} a_i \ln^i \frac{r+1}{r}} \qquad r \geq 1 \tag{5.16}$$

with parameters $a_1, a_2, \ldots, a_4$ determined from the averaged distribution z_r. Figure 5.1 shows the original n_r, and averaged z_r distributions with the fitted polynomial curve.

The smoothed Good–Turing estimate (SGTE) may be quite different from the original Good–Turing estimate (GTE) for small values of r. We would therefore prefer to use GTE for small r and then switch to SGTE and keep on using this estimate for the remaining tally values.

The new, composite smoothed tally r^* is equal to r^{GT} if $|r^{SGT} - r^{GT}| > 1.65 \times \sigma(r^{GT})$, i.e., if the difference between the Good-Turing tally and its smoothed version is significant (exceeds 90% confidence interval on r^{GT}) we use GTE. Otherwise we use SGTE for the remaining tallies.

The probability estimates computed using the corrected tallies have to be normalised because two different methods (GTE and SGTE) of estimation are employed. The normalised probability $\hat{P}_r^{norm}$

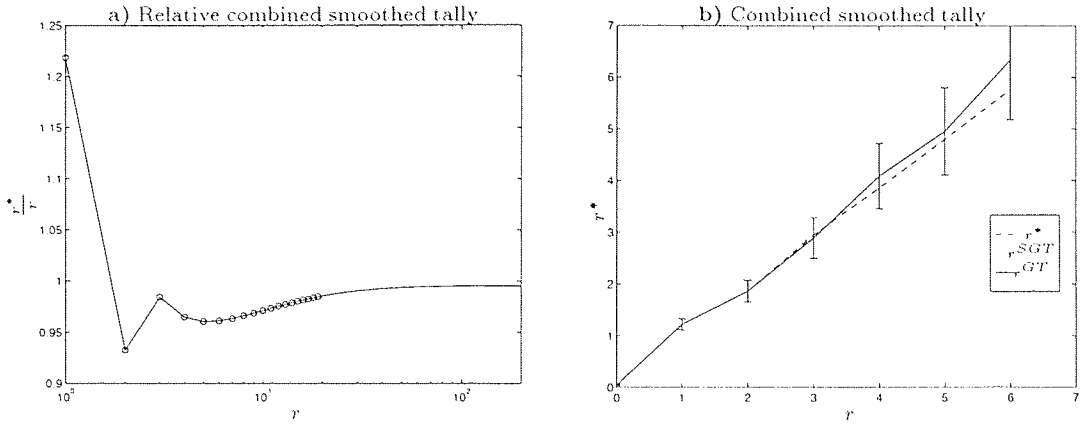


Figure 5.2: **a)** Relative adjusted tally r^* for class 0 of the Tsetse dataset. **b)** Illustration of the smoothed tally r^* combined from tallies r^{GT} and r^{SGT}. The error-bars on r^{GT} are $1.65 \times \sigma(r^{GT})$. The switch from GTE to SGTE takes place at $r = 3$.

for the tally r is computed using the unnormalised probabilities $\hat{P}_r^* = \frac{r^*}{N}$ as

$$\begin{cases} n_0 \neq 0 \quad \hat{P}_r^{norm} = (1 - \frac{n_1}{N}) \frac{\hat{P}_r^*}{\sum_{r' \geq 1} n_{r'} \hat{P}_{r'^*}} \qquad r \geq 1, \qquad \hat{P}_0^{norm} = \frac{n_1}{n_0 N} \\ n_0 = 0 \quad \hat{P}_r^{norm} = \frac{\hat{P}_r^*}{\sum_{r'} n_{r'} \hat{P}_{r'^*}} \qquad r \geq 1 \end{cases} \tag{5.17}$$

5.6 Application of GTE for the Frequency Weighted RAMnet

Most of the memory contents in the Frequency Weighted RAMnet hold zero tallies. Their existence poses difficulties when the logarithm of joint probability $\log P(\boldsymbol{\alpha}|c)$ is approximated with a sum $\sum_i \log P(\alpha_i|c)$ and a typical *ad hoc* solution is to use a small negative constant ϵ instead of $\log 0$. Good–Turing estimation readjusts the tallies in a principled way replacing zero tallies with a ratio $\frac{n_1}{D_c^{trn} T n_0}$.

The tuple addresses α_i and α_j are considered to be different objects even if $\alpha_i = \alpha_j$, unless their mappings are also identical, i.e. if $\eta(i,k) = \eta(j,k) \ \ \forall k$. Therefore, the total number of objects s is $2^n \times T$ and the sample size N is $D_c^{trn} \times T$ because each training pattern of class c is broken down into T tuple addresses. The value n_r is simply the number of memory locations with a tally r.

In order to obtain probabilities $P(\alpha_i|c)$ normalised within a tuple node one would have to apply Good–Turing estimation for each tuple in each class c separately. This is hardly possible because the distribution n_r is very sparse, especially for small tuple sizes n. Therefore, the estimation has been carried out collectively for all T tuples within a class c, i.e., for the population of $T2^n$ features. Consequently, the probabilities $P(\alpha_i|c)$ are normalised within each discriminator c and each zero tally is smoothed by the same amount regardless of the tuple which generated it.

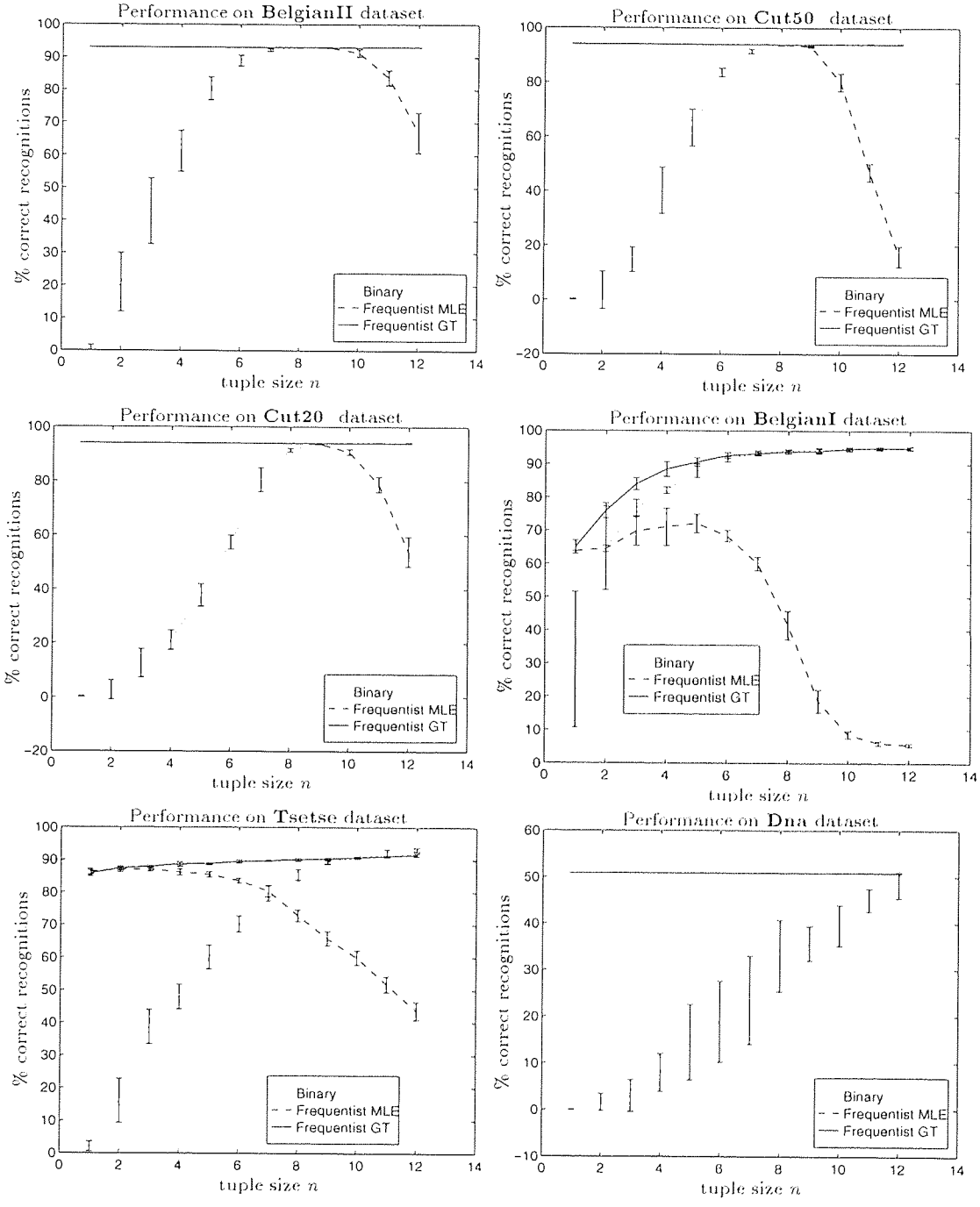


Figure 5.3: The MLE and GTE RAMnet's performance compared to that of binary n-tuple system for various values of tuple size n. All systems comprised of 100 tuples. The error-bars are of size one standard deviation computed for 10 random tuple mappings.

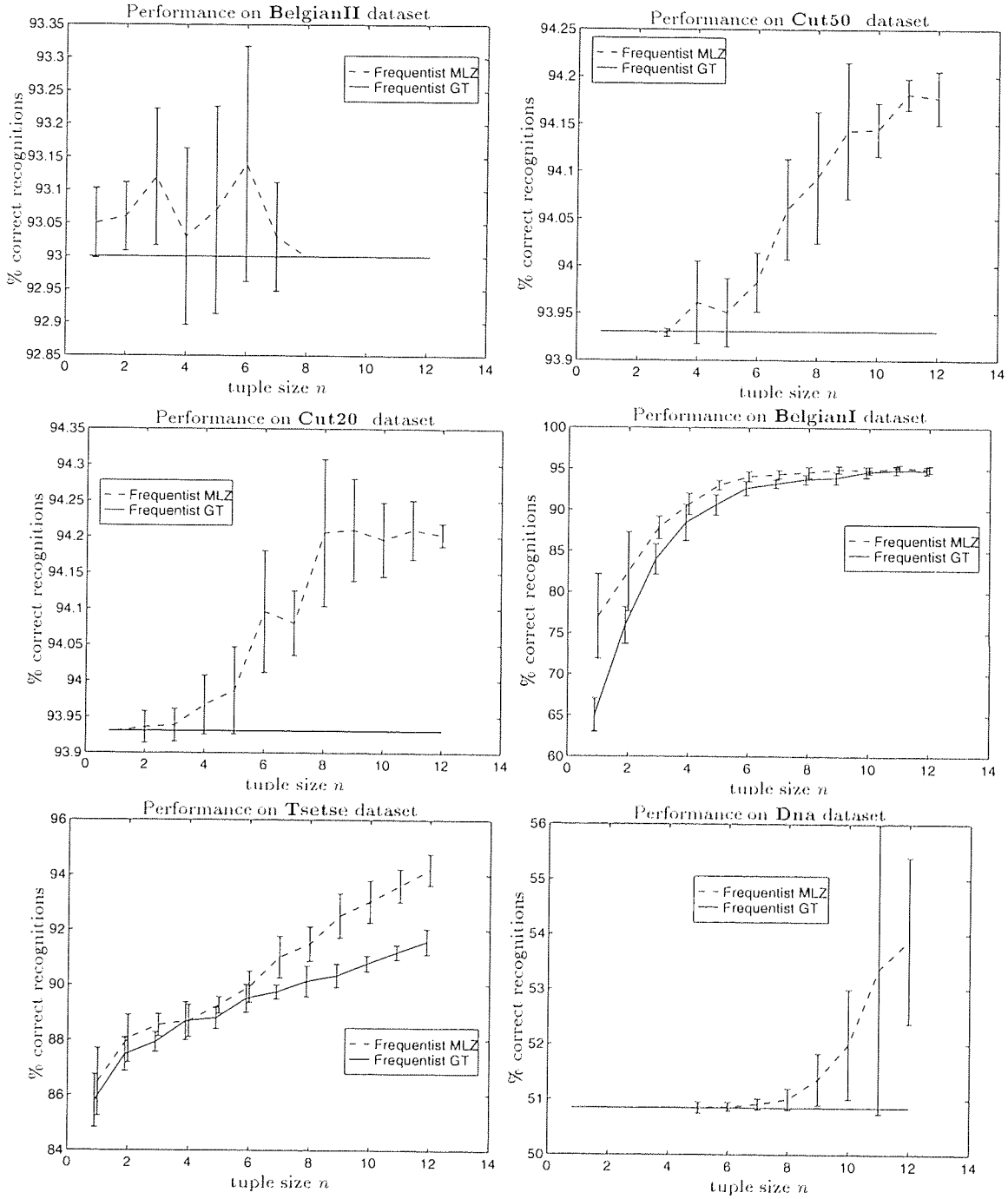


Figure 5.4: Performance of the n-tuple system with Good-Turing estimates (GTE) compared with Maximum Likelihood estimates system (MLZ) with zero tally correction ($\epsilon = 10^{-150}$). The systems comprised of 100 tuples. The error-bars are of size one standard deviation computed for 10 random tuple mappings.

Figure 5.2a shows the relative composite smoothed tally r^*/r computed for the first discriminator of the n-tuple system trained on the Tsetse dataset. The construction of the combined smoothed tally r^* is given in figure 5.2b. We observe that for the first three tallies GTE was chosen whereas SGTE was used for the remaining tallies. The adjusted zero tally was $r_0^* = 0.0452$.

The performance of the three versions of the n-tuple classifier: binary, Maximum Likelihood (ML) and Good-Turing (GT) were compared using a number of StatLog datasets. The Maximum Likelihood version contained the original non-smoothed tallies which were used to compute probabilities (given by equation 5.1) and the scores (given by equation 5.3). All zero tallies were ignored when calculating the score. If the i-th memory location $m_{ci\alpha_i(\mathbf{u})}$ addressed by a test pattern $\mathbf{u}$ contained a zero value, the probability estimate $P(\alpha_i|c)$ was not calculated and did not enter the sum of the logarithms of $P(\alpha_i|c)$ and $P(\alpha_i)$ over i features. This procedure corresponds to replacing a zero tally with the sample size, so that the corresponding probability estimate equals 1.

Figure 5.3 shows the performance of the binary, ML and GT RAMnets on 6 StatLog datasets (BelgianI, BelgianII, Cut20, Cut50, Dna and Tsetse) for different values of the parameter n. The plots show the mean performance for 10 random tuple mappings $\boldsymbol{\eta}$ and the error bars equal to one standard deviation. All systems comprised 100 tuples. This is considerably less than the 1000 tuples used in the benchmarking study described in chapter 4 because of the increased memory requirements for the Frequency Weighted versions which store floating point numbers rather than bits.

Both ML and GT systems perform better than the binary version for small values of n, because they do not suffer from saturation. Unlike the Frequency Weighted system with MLE, the GT version retains the performance with increasing n. However, it eventually becomes inferior to the binary system. It seems that for n large enough, any technique other than zero tally counting (which is equivalent to setting the tally truncation threshold θ to one) is less effective.

We also compared the performance of the GT system to that of MLZ, which is technically an ML system with zero tallies substituted by an arbitrarily chosen, small positive constant ϵ. The zero tally probability estimates result, after taking their logarithms, in large negative values that tend to cancel out the contributions from the non-zero tally estimates to the sums in the equation 5.3. The amount of cancellations increases exponentially with n increasing (because the saturation drops down at this rate) and logarithmically with ϵ decreasing.

The experimental results plotted in Figure 5.4 suggest that if ϵ is small enough then MLZ will outperform the GT system, especially for large n. This can be explained by observing that MLZ with $\epsilon \to 0$ will make exactly the same classification decision as the binary system (because of the cancellations mentioned earlier), except for the patterns that are tied (have the same score) in the binary version. For large n, the saturation is very low, as is the probability of a tie. Consequently, the

performance of MLZ must be equal to the performance of a binary system within a margin $\pm\frac{D^{tied}}{D^{tst}}$ where D^{tst} is test set size and D^{tied} number of tied patterns.

5.7 Summary

The Frequency Weighted version of the n-tuple classifier was examined and two major problems concerning this RAMnet type were identified: joint probability approximation and the estimation of low order probabilities.

The major weakness of Maximum Likelihood estimate for the n-tuple system is that it is not accurate for small tallies. A principled approach to tally smoothing using the Good-Turing formula leads to an improved system performance for larger values of n. However, experiments suggest that replacing zero tallies with a small constant and using a Maximum Likelihood estimate yields even better results.

5.8 Conclusions

The bulk of the experimental evidence seems to indicate, that the binary system will yield better performance that the Frequency Weighted version for large values of the parameter n. With n increasing the amount of data required to construct reasonably accurate probability estimates quickly becomes excessive.

The Maximum Likelihood system is inferior to a version which uses Good-Turing estimates because it delivers inaccurate estimates for low tallies which are most often recorded in the memory. The Good-Turing estimate constitutes a theoretical improvement but it turns out to be inferior to the simple binary RAMnet. There maybe two reasons for the lack of performance: correlations between the tuple addresses and insufficient statistics of the n_r distribution (the lack of data).

The MLZ system in which zero tallies are replaced with a constant lacks a firm theoretical grounding. In practice, however, this system combines the strengths of the Frequency Weighted and binary versions. For small values of n, the MLZ system operates as the former because there are relatively few zero tallies and the constant ϵ does not contribute to the score. Because the number of zero tallies increases exponentially with n for a fixed training set size, the score of the MLZ system will be mostly determined by the contribution from ϵ values for large n. Thus, the MLZ n-tuple system becomes equivalent to the original, binary RAMnet.

The binary system is not concerned with probability estimation and is free from the problems associated with the Frequency Weighted versions. It appears that if the input data is binary and the preprocessing extracts features in a random fashion from the input, then a simple memorisation is

superior to the techniques attempting to approximate a joint class probability based on the estimation of the marginals and an independence assumption.

Chapter 6

Generalisation Cost of RAMnets

6.1 Introduction

In order to predict quantitatively the performance of methods such as the ultra-fast RAMnet, which are not trained by minimising a cost function, we develop a Bayesian formalism for estimating the generalisation cost of a wide class of algorithms. We focus our attention on the generalisation cost for one dimensional problems.

We show (section 6.2), that the regression RAMnet's output is linear in the training output data $\mathbf{y}_{(N)}$ and is completely determined by a matrix $\mathbf{J}$ and the vector $\mathbf{y}_{(N)}$. This observation allows us to express the expected cost as a function of $\mathbf{J}$ and point out explicitly the analytical difficulties that arise for this class of models.

Section 6.3 introduces stochastic processes and their basic properties. We then show how a Gaussian process can be used to make inferences about an unknown function based on a training sample of inputs $\mathbf{x}_{(N)}$ and noisy teaching outputs $\mathbf{y}_{(N)}$ (section 6.4).

We build on this formalism and calculate (section 6.5) the expected cost and its variance. We verify the analytical derivation numerically and apply the formalism to two one dimensional regression problems using regression RAMnets as models.

6.2 Regression RAMnet as a Linear Basis Function Model

In this chapter we will focus on the regression RAMnets. We recall from section 3.5.2 that after training with a sample $\{\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\} = \{x^i, y^i\}_{i=1}^{i=N}$ this model outputs

$$y(x) = \overset{m}{\mathbf{f}}_x = \frac{\sum_{i=1}^{N} y^i \sum_{k=1}^{T} \delta_{\alpha_k(\mathbf{v}^i),\alpha_k(\mathbf{u})}}{\sum_{i=1}^{N}\sum_{k=1}^{T} \delta_{\alpha_k(\mathbf{v}^i),\alpha_k(\mathbf{u})}} = \frac{\sum_{i=1}^{N} y^i \sum_{k=1}^{T}\sum_{\alpha=0}^{2^n-1} \delta_{\alpha,\alpha_k(\mathbf{u})}\delta_{\alpha,\alpha_k(\mathbf{v}^i)}}{\sum_{i=1}^{N}\sum_{k=1}^{T}\sum_{\alpha=0}^{2^n-1} \delta_{\alpha,\alpha_k(\mathbf{u})}\delta_{\alpha,\alpha_k(\mathbf{v}^i)}} \tag{6.1}$$

$$= \frac{\sum_{i=1}^{N} y^i \sum_{k=1}^{T}\sum_{\alpha=0}^{2^n-1} \phi_{k,\alpha}^{Ram}(x)\phi_{k,\alpha}^{Ram}(x^i)}{\sum_{i=1}^{N}\sum_{k=1}^{T}\sum_{\alpha=0}^{2^n-1} \phi_{k,\alpha}^{Ram}(x)\phi_{k,\alpha}^{Ram}(x^i)} = \frac{\sum_{i=1}^{N} y^i K(x,x^i)}{\sum_{i=1}^{N} K(x,x^i)}.$$

where the tuple eigenfunctions $\phi_{k,\alpha}^{Ram}(x) = \delta_{\alpha,\alpha_k(\mathbf{u})}$ (see section 3.4.1), $\mathbf{u}$ is the binary code corresponding to the input x, $\mathbf{v}^i$ is the binary code corresponding to the input x^i, and the kernel function $K(x,x') = \sum_{k=1}^{T}\sum_{\alpha=0}^{2^n-1} \phi_{k,\alpha}^{Ram}(x)\phi_{k,\alpha}^{Ram}(x')$.

Let us now define an infinite dimensional matrix $\mathbf{J}$ with elements

$$J_{xx'} = \frac{K(x,x')}{\sum_{i=1}^{N} K(x,x^i)}. \tag{6.2}$$

If a RAMnet was trained with the dataset $\{\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\} = \{x^i, y^i\}_{i=1}^{i=N}$ then its response to a test vector $\mathbf{x}_{(P)}$ comprising P inputs is linear in the data $\mathbf{y}_{(N)}$:

$$\overset{m}{\mathbf{f}} = \mathbf{J}_{(PN)}\mathbf{y}_{(N)} \tag{6.3}$$

where $\overset{m}{\mathbf{f}}$ denotes the model's output vector of length P, $\mathbf{J}_{(PN)}$ denotes the $P \times N$ submatrix of $\mathbf{J}$ with elements given by equation 6.2.

We indicate in section 6.5.1 that for RAMnets, the expression for the expected generalisation cost involves integrals over $\mathbf{J}_{(PN)}$. These had to be evaluated numerically. Details of the cost derivation for RAMnets can be found in Appendix A.4.

6.3 Definition of the Stochastic Process

Let us introduce stochastic processes and define their basic properties following the formalism of (Papoulis, 1984).

Let ζ denote the outcome of a random experiment. A stochastic process is a rule for assigning to the variable ζ a function $f(x,\zeta)$. Usually, ζ is omitted in the notation. Therefore, $f(x)$ has a number of interpretations depending on which variable is allowed to vary and which one is assumed to be constant:

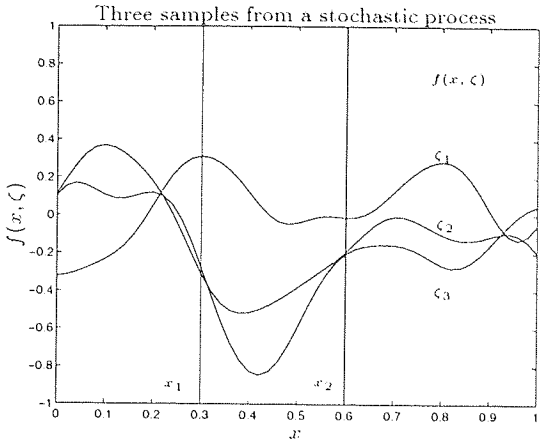


Figure 6.1: Illustration of a stochastic process.

- $f(x)$ is a family of functions of x which depend on the parameter ζ.
- If ζ is fixed, $f(x)$ depends only on x and is called a sample of the given process.
- If x is fixed and ζ is allowed to vary then $f(x)$ is a random variable.
- If both ζ and x are fixed, $f(x)$ is a number.

The domain of ζ is the set of infinitely many experimental outcomes $\mathcal{S} = \{\zeta_1, \zeta_2, \ldots, \zeta_\infty\}$ and the domain of x is a set of real numbers $\mathbb{R}$. If x is discrete then the values of $f(x)$ are countable and the process is discrete-state rather than continuous. These concepts are illustrated on figure 6.1 which shows three samples of a stochastic process.

A stochastic process is an infinity of random variables, because for each x there are infinitely many values of the function $f(x)$. For a fixed x the random variable $f(x)$ has the distribution function

$$F(f, x) = P(f(x) \leq f) \tag{6.4}$$

and the density

$$p(f, x) = \frac{\partial F(f, x)}{\partial f}. \tag{6.5}$$

The second order distribution of the process $f(x)$ is the joint distribution

$$F(f_1, f_2; x_1, x_2) = P(f(x_1) \leq f_1,\ f(x_2) \leq f_2) \tag{6.6}$$

of the random variables $f(x_1)$ and $f(x_2)$. The corresponding second order density is given by

$$p(f_1, f_2; x_1, x_2) = \frac{\partial^2 F(f_1, f_2; x_1, x_2)}{\partial f_1 \partial f_2}. \tag{6.7}$$

The mean $\bar{f}(x)$ of the stochastic process is the expected value of the random variable $f(x)$

$$\bar{f}(x) = \mathcal{E}[f(x)] = \int_{-\infty}^{+\infty} f p(f, x) df \tag{6.8}$$

The autocovariance $V(x_1, x_2)$ of the stochastic process is the covariance of two random variables $f(x_1)$ and $f(x_2)$

$$\begin{aligned} V(x_1, x_2) &= \mathcal{E}\left[\left(f(x_1) - \bar{f}(x_1)\right)\left(f(x_2) - \bar{f}(x_2)\right)\right] \\ &= \mathcal{E}[f(x_1) f(x_2)] - \bar{f}(x_1)\bar{f}(x_2) \\ &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f_1 f_2 p(f_1, f_2; x_1, x_2) df_1 df_2 - \bar{f}(x_1)\bar{f}(x_2) \end{aligned} \tag{6.9}$$

For the discrete stochastic processes the mean $\bar{f}(x)$ can be represented by a vector $\bar{\mathbf{f}}$ with components $\bar{\mathbf{f}}_x = \bar{f}(x)$ and the covariance function $V(x_1, x_2)$ can be represented by a matrix $\mathbf{V}$ with elements $V_{x_1 x_2} = V(x_1, x_2)$. For continuous processes the mean vector $\bar{\mathbf{f}}$ and the covariance matrix $\mathbf{V}$ are infinite dimensional.

Gaussian processes are a particular type of stochastic processes where it is assumed that the random variables $f(x_1), f(x_2), \ldots, f(x_N)$ have a joint normal distribution for any N and $x_1, x_2, \ldots, x_N$. A Gaussian Process $\mathcal{GP}(\bar{f}, V)$ is completely determined by its mean $\bar{f}(x)$ and the covariance function $V(x_1, x_2)$.

6.4 Prediction with a Gaussian Process

In the discussion below we use a matrix notation. We write an infinite dimensional vector $\mathbf{f}$ to denote the function f and replace $f(x)$ with $\mathbf{f}_x$. The domain of inputs x is represented by a vector $\mathbf{x}$. We can visualise the covariance function $V(x, x')$ of the inputs x and x' as an infinite dimensional matrix $\mathbf{V}$ with elements $V_{xx'} = V(x, x')$. The terms on the diagonal of $\mathbf{V}$ denote the variance of the function $f(x)$.

Let us imagine that we are given a sample of N input variables $\mathbf{x}_{(N)}$ and possibly noisy output variables $\mathbf{y}_{(N)}$, generated by a system that implements a true function $\mathbf{f}$, which is unknown to us. Our task is to infer $\mathbf{f}$ using the information provided by the sample $\{\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\}$.

We use Bayesian Inference with a Gaussian Process (O'Hagan, 1994) to specify the optimal approximation $\tilde{\mathbf{f}}$. Firstly, we specify our beliefs about the unknown function $\mathbf{f}$ and define a prior probability distribution $P(\mathbf{f})$ over all possible functions. Secondly, we calculate the joint distribution $P(\mathbf{f}, \mathbf{x}_{(N)}, \mathbf{y}_{(N)})$ of the function $\mathbf{f}$ and the sample of observations $\{\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\}$. Finally, we obtain a formula for the posterior distribution $P(\mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)})$ of the function $\mathbf{f}$ given the sample $\{\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\}$. The mean of this distribution becomes our optimal estimate $\tilde{\mathbf{f}}$.

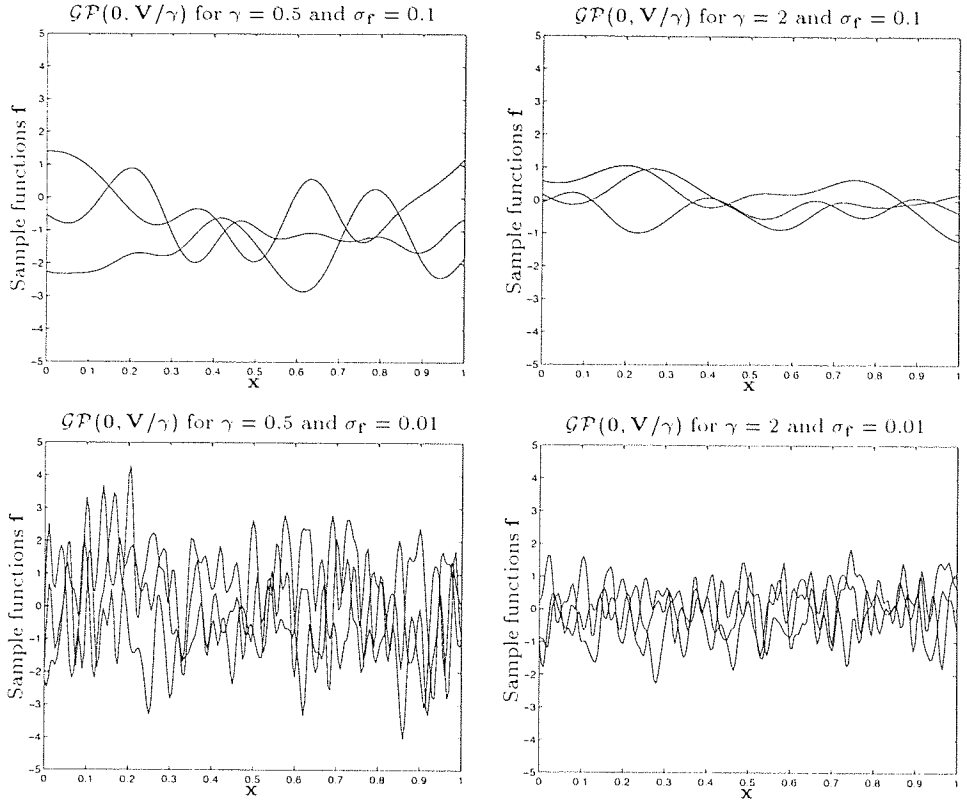


Figure 6.2: Samples from the Gaussian process prior $\mathcal{GP}(\mathbf{0}, \mathbf{V}/\gamma)$ with the covariance matrix of the form $V_{xx'} = \exp\left(-\frac{1}{2}\frac{(x-x')^2}{\sigma_{\mathbf{f}}^2}\right)$ for various choices of the parameters γ and $\sigma_{\mathbf{f}}$.

6.4.1 Prior Distribution

We would like to formulate a prior belief about an unknown function $\mathbf{f}$. We may believe, that the prior expectation (mean) of this function is $\bar{\mathbf{f}}$. Furthermore, we can specify a prior covariance function $V_{xx'}$ (tabulated in the form of the matrix $\mathbf{V}$) between the values of the function at the points x and x'. We introduce the hyperparameter γ which controls the overall scale of the variation of the function. Each element of the covariance matrix $\mathbf{V}$ is divided by γ. If we also assume that the function values have joint normal distribution, then this procedure defines a Gaussian process prior distribution $\mathcal{GP}(\bar{\mathbf{f}}, \mathbf{V}/\gamma)$ of the unknown function $\mathbf{f}$.

The prior probability of $\mathbf{f}$, with mean $\bar{\mathbf{f}}$ and scaled covariance $\mathbf{V}/\gamma$ is defined by

$$P(\mathbf{f}) = (1/Z_\gamma)\exp\left(-\frac{\gamma}{2}(\mathbf{f}-\bar{\mathbf{f}})^{\mathrm{T}}\mathbf{V}^{-1}(\mathbf{f}-\bar{\mathbf{f}})\right) \tag{6.10}$$

where $Z_\gamma = \det\left[\frac{2\pi}{\gamma}\mathbf{V}\right]^{\frac{1}{2}}$.

Let us choose, as an illustration, the covariance function of the form $V_{xx'} = \exp\left(-\frac{1}{2}\frac{(x-x')^2}{\sigma_{\mathbf{f}}^2}\right)$ and define a prior Gaussian process $\mathcal{GP}(0, \mathbf{V}/\gamma)$ with zero mean and covariance matrix $\mathbf{V}$. A number of samples taken from this process for various values of the parameters γ and $\sigma_{\mathbf{f}}$ appears on figure 6.2.

We note that the correlation length of the function can be controlled by the variable $\sigma_{\mathbf{f}}$. The overall scale of variation of $\mathbf{f}$ is controlled by γ so that the standard deviation of the function values is equal to $1/\sqrt{\gamma}$.

More comprehensive illustrations of samples of the functions generated from various choices of covariance are given in (Zhu & Rohwer, 1996).

6.4.2 The Joint and the Posterior Distribution

Our prior beliefs about $\mathbf{f}$ will be updated if we are given a sample of noisy observations $\mathbf{y}_{(N)}$ of the function $\mathbf{f}$ evaluated at N inputs $\mathbf{x}_{(N)}$.

We assume that the observed outputs $\mathbf{y}_{(N)}$ are the sum of the values of the true function $\mathbf{f}_{(N)}$ evaluated at $\mathbf{x}_{(N)}$ and a noise signal ϵ, i.e., $\mathbf{y}_{(N)} = \mathbf{f}_{(N)} + \epsilon$. We also assume the noise ϵ has a Gaussian distribution with zero mean and covariance matrix $\mathbf{Q}/\beta$ where β controls the overall noise magnitude. (In general, noise may be added in such a way that the outputs are correlated and $\mathbf{Q}$ is a general positive definite matrix, but for the cost calculation we will define the noise to be position independent and the outputs to be uncorrelated. This assumption simplifies analytical derivations and will be used in the definition of the cost functional $C(\overset{m}{\mathbf{f}}, \mathbf{f})$ and all the experimental work below.) Hence, we will use a diagonal matrix $\mathbf{Q} = \mathbf{I}$ so that $\sigma_\epsilon = 1/\sqrt{\beta}$).

These assumptions define the likelihood of the outputs $\mathbf{y}_{(N)}$ given the function $\mathbf{f}$ and inputs $\mathbf{x}_{(N)}$

$$P\left(\mathbf{y}_{(N)}|\mathbf{x}_{(N)}, \mathbf{f}\right) = (1/Z_\beta) \exp\left(-\frac{\beta}{2}(\mathbf{f}_{(N)} - \mathbf{y}_{(N)})^{\mathrm{T}} \mathbf{Q}^{-1} (\mathbf{f}_{(N)} - \mathbf{y}_{(N)})\right) \tag{6.11}$$

where $Z_\beta = \det\left[\frac{2\pi}{\beta}\mathbf{Q}\right]^{\frac{1}{2}}$.

We would like to derive the posterior distribution $P\left(\mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\right)$ of the function $\mathbf{f}$, given the sample of input and outputs. However, we first consider a joint distribution $P\left(\mathbf{f}, \mathbf{y}_{(N)}|\mathbf{x}_{(N)}\right)$ of the function $\mathbf{f}$ and the sample outputs $\mathbf{y}_{(N)}$ given the inputs $\mathbf{x}_{(N)}$. Because the likelihood $P\left(\mathbf{y}_{(N)}|\mathbf{x}_{(N)}, \mathbf{f}\right)$ and the prior $P(\mathbf{f})$ are Gaussian distributions, $P\left(\mathbf{f}, \mathbf{y}_{(N)}|\mathbf{x}_{(N)}\right)$ is also a Gaussian distribution.

Let us write the input vector $\mathbf{x}$ as a composition of two vectors: the sample inputs $\mathbf{x}_{(N)}$ and the test inputs $\mathbf{x}_{(P)}$. In general, $\mathbf{x}_{(P)}$ can be equal to $\mathbf{x}$ and contain $\mathbf{x}_{(N)}$, but in practical applications the number P of the test inputs is limited and we are only interested in the values $\mathbf{f}_{(P)}$ of the function at certain specified inputs $\mathbf{x}_{(P)}$. Because the noise ϵ has zero mean the joint distribution $P\left(\mathbf{f}_{(P)}, \mathbf{y}_{(N)}|\mathbf{x}_{(N)}\right)$ has a mean

$$\mathcal{E}\begin{bmatrix}\mathbf{f}_{(P)} \\ \mathbf{y}_{(N)}\end{bmatrix} = \begin{bmatrix}\bar{\mathbf{f}}_{(P)} \\ \bar{\mathbf{f}}_{(N)}\end{bmatrix}. \tag{6.12}$$

The covariance matrix is given by

$$\text{cov}\begin{bmatrix}\mathbf{f}_{(P)}\\ \mathbf{y}_{(N)}\end{bmatrix} = \begin{bmatrix}\frac{1}{\gamma}\mathbf{V}_{(PP)} & \frac{1}{\gamma}\mathbf{V}_{(PN)}\\ \frac{1}{\gamma}\mathbf{V}_{(NP)} & \mathbf{K} = \frac{1}{\gamma}\mathbf{V}_{(NN)} + \frac{1}{\beta}\mathbf{Q}\end{bmatrix} \tag{6.13}$$

where $\mathbf{V}_{(NN)} = V(\mathbf{x}_{(N)}, \mathbf{x}_{(N)})$ is a $N \times N$ covariance matrix involving only the sample inputs $\mathbf{x}_{(N)}$ and $\mathbf{V}_{(PP)} = V(\mathbf{x}_{(P)}, \mathbf{x}_{(P)})$ is a $P \times P$ in the general case) matrix involving only the test points $\mathbf{x}_{(P)}$. The covariance matrices $\mathbf{V}_{(NP)} = V(\mathbf{x}_{(N)}, \mathbf{x}_{(P)})$ of size $N \times P$ and $\mathbf{V}_{(PN)} = \mathbf{V}^{\mathrm{T}}_{(NP)}$ involve a mixture of training and test points.

The conditional distribution $P\left(\mathbf{f}_{(P)}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\right)$ can be found using the following result proved in (von Mises, 1964):

Theorem 1

If the joint distribution of $\begin{bmatrix}\mathbf{f}_{(P)}\\ \mathbf{y}_{(N)}\end{bmatrix}$ *is normal with the mean* $\begin{bmatrix}\bar{\mathbf{f}}_{(P)}\\ \bar{\mathbf{f}}_{(N)}\end{bmatrix}$ *and the covariance matrix* $\begin{bmatrix}\frac{1}{\gamma}\mathbf{V}_{(PP)} & \frac{1}{\gamma}\mathbf{V}_{(PN)}\\ \frac{1}{\gamma}\mathbf{V}_{(NP)} & \mathbf{K}\end{bmatrix}$, *the conditional distribution of* $\mathbf{f}_{(P)}$ *given* $\mathbf{y}_{(N)}$ *is normal with mean*

$$\overset{*}{\mathbf{f}} = \bar{\mathbf{f}}_{(P)} + \frac{1}{\gamma}\mathbf{V}_{(PN)}\mathbf{K}^{-1}[\mathbf{y}_{(N)} - \bar{\mathbf{f}}_{(N)}] \tag{6.14}$$

and covariance matrix

$$\mathbf{A} = \frac{1}{\gamma}\mathbf{V}_{(PP)} - \frac{1}{\gamma^2}\mathbf{V}_{(PN)}\mathbf{K}^{-1}\mathbf{V}_{(NP)}. \tag{6.15}$$

In general, any element $A_{xx'}$ of the posterior covariance matrix can be found by evaluating

$$A_{xx'} = \frac{1}{\gamma}V_{xx'} - \frac{1}{\gamma^2}\mathbf{v}(x)^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{v}(x') \tag{6.16}$$

where the vector $\mathbf{v}(x) = V(\mathbf{x}_{(N)}, x)$ of length N, describes the covariance between a test point x and the training points $\mathbf{x}_{(N)}$.

The posterior distribution $P\left(\mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\right)$ can be written as

$$P\left(\mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\right) = \det\left[2\pi\mathbf{A}\right]^{-\frac{1}{2}} \exp^{-\frac{1}{2}(\mathbf{f}-\overset{*}{\mathbf{f}})^{\mathrm{T}}\mathbf{A}^{-1}(\mathbf{f}-\overset{*}{\mathbf{f}})}. \tag{6.17}$$

and the optimal estimate of the unknown function $\mathbf{f}$ given a sample $\{\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\}$ is the mean $\overset{*}{\mathbf{f}}$ of this distribution.

Note, that $\overset{*}{\mathbf{f}}$ depends on prior mean $\bar{\mathbf{f}}$, the functional form of the prior covariance $\mathbf{V}$ (including the correlation length $\sigma_{\mathbf{f}}$ and the scalar γ) and on the noise covariance $\mathbf{Q}$ and scalar β.

Let us illustrate the formalism described above with an example. The true function, taken from (Neal, 1995) is given by

$$\mathbf{f}_x = 0.3 + 0.4x + 0.5\sin(2.7x) + 1.1/(1 + x^2) + \epsilon \tag{6.18}$$

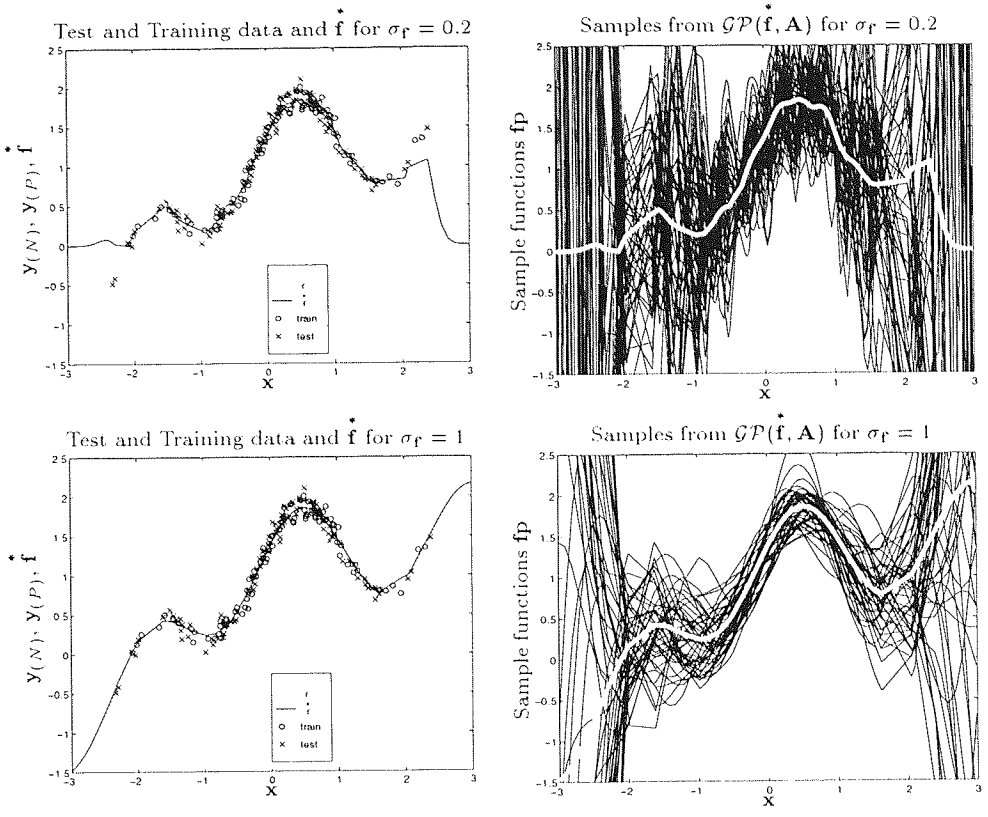


Figure 6.3: The panels on the left show the training sample $\{\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\}$ (circles) and the test sample $\{\mathbf{x}_{(P)}, \mathbf{y}_{(P)}\}$ (crosses) generated from the underlying true function $\mathbf{f}$ plotted with a dotted line. The approximation $\overset{*}{\mathbf{f}}$, for various choices of the parameters $\sigma_\mathbf{f}$, is shown with a solid line. Panels on the right show the corresponding samples form the Gaussian process posterior $\mathcal{GP}(\overset{*}{\mathbf{f}}, \mathbf{A})$ with the mean $\overset{*}{\mathbf{f}}$ and the covariance matrix $\mathbf{A}$.

where the Gaussian noise variable ϵ has mean $\mu_\epsilon = 0$ and a diagonal covariance noise matrix $\mathbf{Q} = \mathbf{I}$ where the standard deviation $\sigma_\epsilon = \sqrt{1/\beta} = 0.1$. The training set $\{\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\}$ and test set $\{\mathbf{x}_{(P)}, \mathbf{y}_{(P)}\}$ each comprise 100 data-points. The inputs were generated by the standard normal distribution ($\mu_x = 0$, $\sigma_x = 1$).

The true function $\mathbf{f}$ is shown by the dotted line in the left panels of figure 6.3. The training and test set are plotted with circles and crosses respectively.

We follow the procedure described in the section 6.4.1 and select a prior $\mathcal{GP}(\mathbf{0}, \mathbf{V})$ with zero mean (The observed values $\mathbf{y}_{(N)}$ vary about 1 and a choice of the prior mean $\bar{\mathbf{f}} = \mathbf{1}$ would also be justified, but we choose $\bar{\mathbf{f}} = 0$ to simplify the calculations.) and a covariance matrix of the form $V_{xx'} = \exp\left(-\frac{1}{2}\frac{(x-x')^2}{\sigma_\mathbf{f}^2}\right)$.

We calculate the posterior mean $\overset{*}{\mathbf{f}}$ according to equation 6.14 which is our approximation to the unknown function $\mathbf{f}$. This approximation is plotted with a solid line for two values of the hyperparameter $\sigma_\mathbf{f}$ in the left panels of figure 6.3. The right panels show a number of samples $\mathbf{fp}$ generated

from the posterior Gaussian process $\mathcal{GP}(\overset{*}{\mathbf{f}}, \mathbf{A})$. We note that the smoothness of the approximation $\overset{*}{\mathbf{f}}$ is controlled by the parameter $\sigma_{\mathbf{f}}$. Its correct setting yields a good approximation even for the inputs x far away from the training inputs $\mathbf{x}_{(N)}$. For small values of $\sigma_{\mathbf{f}}$ (smaller correlation length), the prior prevails more strongly in places where no data is available and consequently $\overset{*}{\mathbf{f}}$ drops to zero (the prior mean) faster.

The hyperparameters can also be selected so as to maximise the logarithm of the likelihood of the data $P\left(\mathbf{x}_{(N)}, \mathbf{y}_{(N)}|\gamma, \sigma_{\mathbf{f}}\right)$ which is defined (Williams, 1995) for the zero mean prior as

$$\log P\left(\mathbf{x}_{(N)}, \mathbf{y}_{(N)}|\gamma, \sigma_{\mathbf{f}}\right) = -\tfrac{1}{2}\log\det\left[\mathbf{K}\right] - \tfrac{1}{2}\mathbf{y}_{(N)}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{y}_{(N)} - \frac{N}{2}\log 2\pi. \tag{6.19}$$

However, we will set $\sigma_{\mathbf{f}}$ and γ using only our prior knowledge of the function $\mathbf{f}$.

6.5 The Expected Cost and Variance for Regression Problems

The framework developed in section 6.4 allows us to calculate the optimal model $\overset{*}{\mathbf{f}}$ for the noisy interpolation problem in which the output data points $\mathbf{y}_{(N)}$ result from adding noise ϵ to the result $\mathbf{f}_{(N)} = \mathbf{f}(\mathbf{x}_{(N)})$ of applying unknown function f to input data points $\mathbf{x}_{(N)}$, which are generated from a distribution $P(x)$. Consequently, for a suboptimal model with a prediction function $\overset{m}{\mathbf{f}}$, we can calculate the expected generalisation cost under the posterior. We present the derivation of the formulae for this cost and its errorbars in section 6.5.1. In order to verify the analytical results we carry out a numerical experiment (see section 6.5.2). We choose a true function $\mathbf{f}$ by selecting it from the prior $\mathcal{GP}(\mathbf{0}, \mathbf{V})$ and use an n-tuple regression network to calculate the suboptimal approximation $\overset{m}{\mathbf{f}}$. We then calculate the posterior $\mathcal{GP}(\overset{*}{\mathbf{f}}, \mathbf{A})$ and generate a number of posterior function samples $\mathbf{f_p}$. For each one of them we evaluate the cost of using the model function $\overset{m}{\mathbf{f}}$. This results in a cost distribution whose mean and variance agrees excellently with the expressions derived analytically.

6.5.1 Derivation of the Expected Cost and its Variance

Let us define the cost of associating an output $\overset{m}{\mathbf{f}}_x$ of the model with an input x that actually produced an output y (containing the noise) as

$$C(\overset{m}{\mathbf{f}}_x, y) = \tfrac{1}{2}(\overset{m}{\mathbf{f}}_x - y)w_x(\overset{m}{\mathbf{f}}_x - y) \tag{6.20}$$

The average of this cost, over all the inputs $\mathbf{x}$ and noisy outputs $\mathbf{y}$, defines a cost functional given by

$$C(\overset{m}{\mathbf{f}}, \mathbf{f}) = \int C(\overset{m}{\mathbf{f}}_x, y)P(x)\,P(y|x, \mathbf{f})\,dxdy, \tag{6.21}$$

for the true function is $\mathbf{f}$. This form is obtained by noting that the function $\mathbf{f}$ carries no information about the input density x, i.e., $\mathbf{f}$ and x are independent. Furthermore, the input data x supplies no information about y beyond that supplied by $\mathbf{f}$.

The knowledge of x is affected by the presence of the training inputs $\mathbf{x}_{(N)}$ so $P(x)$ can be replaced by $P(x|\mathbf{x}_{(N)})$ yielding

$$C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)}) = \int C(\overset{m}{\mathbf{f}}_x, y) P(x|\mathbf{x}_{(N)}) P(y|x, \mathbf{f}) \, dx dy. \tag{6.22}$$

The distributions in (6.22) are unchanged by further conditioning on $\mathbf{y}_{(N)}$ and therefore we could write $C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)}) = C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)})$. With the independence assumptions mentioned earlier, this allows us to write the joint posterior $P(x, y, \mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)})$ as a product

$$\begin{aligned} P(x, y, \mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}) &= P(x|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}, \mathbf{f}) P(y|x, \mathbf{f}, \mathbf{x}_{(N)}, \mathbf{y}_{(N)}) P(\mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}) \\ &= P(x|\mathbf{x}_{(N)}) P(y|x, \mathbf{f}) P(\mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}) \end{aligned} \tag{6.23}$$

The cost functional $C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)})$ has the expectation value under the posterior $P(\mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)})$ given by

$$\begin{aligned} \langle C|\mathbf{x}_{(N)}, \mathbf{y}_{(N)} \rangle &= \int C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)}) P(\mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}) \, d\mathbf{f} \\ &= \int C(\overset{m}{\mathbf{f}}_x, y|\mathbf{x}_{(N)}) P(x, y, \mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}) \, dx dy d\mathbf{f} \end{aligned} \tag{6.24}$$

and variance

$$\mathrm{var}\left[C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)})\right] = \int C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)})^2 P(\mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}) \, d\mathbf{f} - \langle C|\mathbf{x}_{(N)}, \mathbf{y}_{(N)} \rangle^2. \tag{6.25}$$

These expressions are exact and hold for any model. Regression RAMnets, described by the matrix $\mathbf{J}_{(PN)}$ and the training outputs $\mathbf{y}_{(N)}$, belong to the particular class of models linear in the training outputs. We can rewrite 6.24 in terms of $\mathbf{J}_{(PN)}$ and $\mathbf{y}_{(N)}$ and obtain (see appendix A.4) an exact analytical form for this model class:

$$\begin{aligned} \langle C|\mathbf{x}_{(N)}, \mathbf{y}_{(N)} \rangle &= \frac{1}{2\gamma^2} \int \sum_{uvts} y_u V_{xx_t} V_{xx_s} K^{-1}_{sv} K^{-1}_{tu} y_v P(x|\mathbf{x}_{(N)}) \, w_x dx \\ &- \frac{1}{\gamma} \int \sum_{uvs} y_u J_{xx_u} V_{xx_s} K^{-1}_{sv} y_v P(x|\mathbf{x}_{(N)}) \, w_x dx \\ &+ \tfrac{1}{2} \int \sum_{uv} y_u J_{xx_u} J_{xx_v} y_v P(x|\mathbf{x}_{(N)}) \, w_x dx \\ &+ \frac{1}{2\gamma} \int V_{xx} P(x|\mathbf{x}_{(N)}) \, w_x dx - \frac{1}{2\gamma^2} \int \sum_{tu} V_{xx_t} K^{-1}_{tu} V_{x_u x} P(x|\mathbf{x}_{(N)}) \, w_x dx \\ &+ \frac{1}{\beta} \int q_x w_x P(x|\mathbf{x}_{(N)}) \, dx. \end{aligned} \tag{6.26}$$

It turns out that the integration over x is not tractable. Consequently, we resort to the numerical approximations in the experimental work reported below. If we plug in the prior, likelihood and posterior distributions given by equations 6.10, 6.11, 6.17 into 6.24 then after some calculations (see appendix A.2) we obtain the following approximation of the posterior cost:

$$\langle C|\mathbf{x}_{(N)}, \mathbf{y}_{(N)} \rangle \approx \tfrac{1}{2} \mathrm{tr}\left[\mathbf{AR}\right] + \tfrac{1}{2} (\overset{*}{\mathbf{f}} - \overset{m}{\mathbf{f}})^{\mathrm{T}} \mathbf{R} (\overset{*}{\mathbf{f}} - \overset{m}{\mathbf{f}}) + \frac{\mathrm{tr}\left[\mathbf{QR}\right]}{2\beta} \tag{6.27}$$

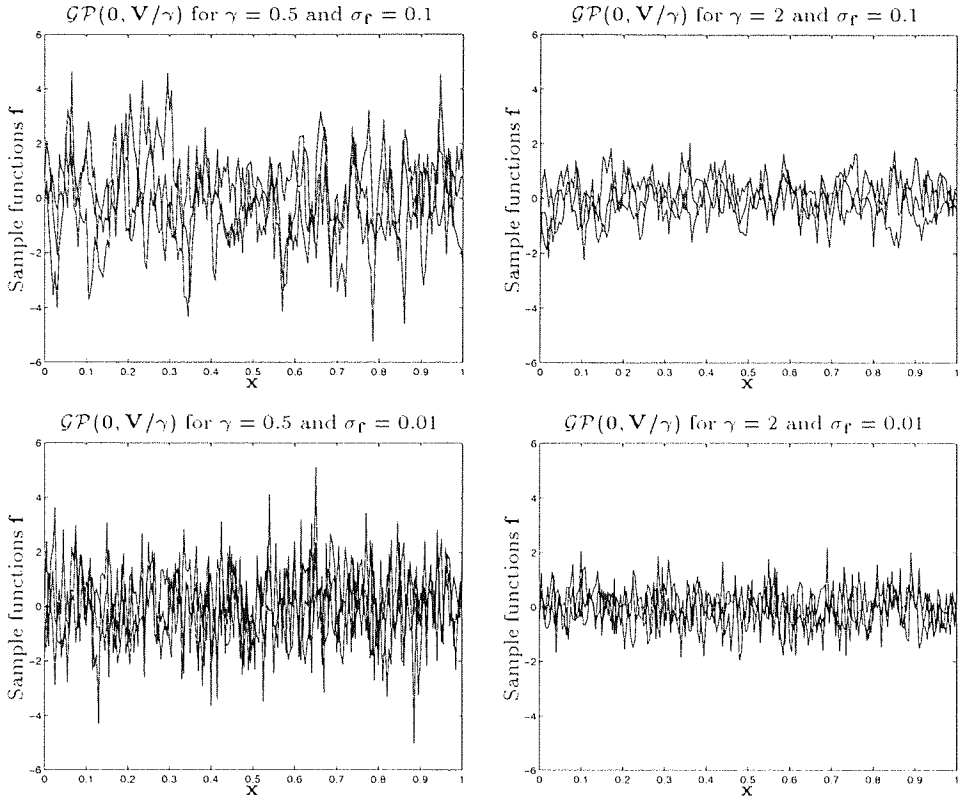


Figure 6.4: Samples from the Gaussian process prior $\mathcal{GP}(\mathbf{0}, \mathbf{V}/\gamma)$ with the covariance matrix of the form $V_{xx'} = \exp\left(-\gamma \frac{|x-x'|}{\sigma_{\mathbf{f}}^2}\right)$ for various choices of the parameters γ and $\sigma_{\mathbf{f}}$.

where the diagonal matrices $\mathbf{R}$ and $\mathbf{Q}$ have the elements $R_{xx'} = \frac{1}{M} P(x|X) w_x \delta_{x,x'}$ and $Q_{xx'} = q_x \delta_{xx'}$ and the x domain was quantised into M bins.

Similar calculations (see appendix A.3) lead to the expression for the variance

$$\mathrm{var}\left[C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)})\right] \approx \tfrac{1}{2}\mathrm{tr}\left[\mathbf{ARAR}\right] + \mathrm{tr}\left[\mathbf{ARRFF}\right]. \tag{6.28}$$

where the elements of $\mathbf{F}$ are $F_{xx'} = (\overset{m}{\mathbf{f}}_x - \overset{*}{\mathbf{f}}_x)\delta_{x,x'}$. The above approximations become arbitrarily accurate as $M \to \infty$. In practice M as little as 50 yields sufficiently good results. See the Appendix for further details.

6.5.2 Numerical Verification of the Formulae

In order to facilitate a verification of the formulae 6.27 and 6.28 which involve approximations of the integrals over x (see appendix A.2) we have to quantize the x domain. We approximate the infinite-dimensional space of functions by a finite-dimensional space of discretised functions, so that function f is replaced by high-dimensional vector $\mathbf{f}$, and $f(x)$ is replaced by $\mathbf{f}_x$, with $f(x) \approx \mathbf{f}_x$ within a volume Δx around x.

Let the input and output variables be one dimensional real numbers. Let the input distribution $P\left(x|\mathbf{x}_{(N)}\right)$ be a Gaussian with mean $\mu_x = 0$ and standard deviation $\sigma_x = 0.2$. Nearly all inputs then fall within the range $[-1, 1]$, which we uniformly quantise into 41 bins. The true function $\mathbf{f}$ is generated from the prior distribution $\mathcal{GP}(\mathbf{0}, \mathbf{V})$ with 41×41 covariance matrix $\mathbf{V}$ with elements $V_{xx'} = e^{-\gamma \frac{|x-x'|}{\sigma_\mathbf{f}}}$. (We chose this covariance function in order to illustrate the influence of the covariance function on the shape of $\mathbf{f}$. Because the true function is generated from the prior it does not matter what covariance function is selected). A number of sample functions $\mathbf{f}$ generated for this choice of covariance function is plotted on figure 6.4. The hyperparameters γ, $\sigma_\mathbf{f}$ were chosen to have the same values as in the plots on figure 6.2.

50 training inputs x were generated from the input distribution and assigned corresponding outputs $y = \mathbf{f}_x + \epsilon$, where ϵ is Gaussian noise with zero mean and standard deviation $\sqrt{1/\beta} = 0.01$. The cost weight $w_x \equiv 1$ and the hyper parameters $\gamma = \sigma_\mathbf{f} = 1$.

The inputs were thermometer encoded over 256 bits, from which 100 tuples of size 30 were randomly selected. 50 training data points were used to train an n-tuple regression network. The response of the RAMnet $\overset{m}{\mathbf{f}}$ was then calculated for all the inputs. The input distribution and functions $\mathbf{f}$, $\overset{m}{\mathbf{f}}$, $\overset{*}{\mathbf{f}}$ are plotted in figure 6.5a.

A Gaussian distribution with mean $\overset{*}{\mathbf{f}}$ and posterior covariance matrix $\mathbf{A}$ was then used to generate 10^4 functions $\mathbf{fp}$ (recollect figure 6.3). Note, that because the input distribution $P(x)$ is known and the entire input domain quantised, we do not need to generate the test input sample in order to compute the test outputs and the model's responses, for each choice of $\mathbf{fp}$. Instead, for each such function $\mathbf{fp}$ we calculate the generalisation cost for the entire input domain as

$$C = \tfrac{1}{2} \sum_x P\left(x|\mathbf{x}_{(N)}\right) (\mathbf{fp}_x - \overset{m}{\mathbf{f}}_x)^2. \tag{6.29}$$

A histogram of these costs appears in figure 6.5b, together with the theoretical and numerically computed average generalisation cost and its variance. Good agreement is evident.

6.5.3 Cost Prediction for Two Regression Problems

We now illustrate the use of the formalism developed in section 6.5.1 with two examples of predicting the generalisation performance of a regression RAMnet when the prior over functions can only be guessed.

In both cases we choose a Gaussian process prior $\mathcal{GP}(\mathbf{0}, \mathbf{V}/\gamma)$. The covariance function also has a Gaussian form $\mathbf{V}_{xx'} = e^{-\frac{1}{2}\frac{(x-x')^2}{\sigma_\mathbf{f}^2}}$ where $\sigma_\mathbf{f}$ is the correlation length of the function. The hyperparameter γ controls the scale of variation of the function values. The standard deviation of the function is equal to $1/\sqrt{\gamma}$. The model used throughout the experiments is an n-tuple regression

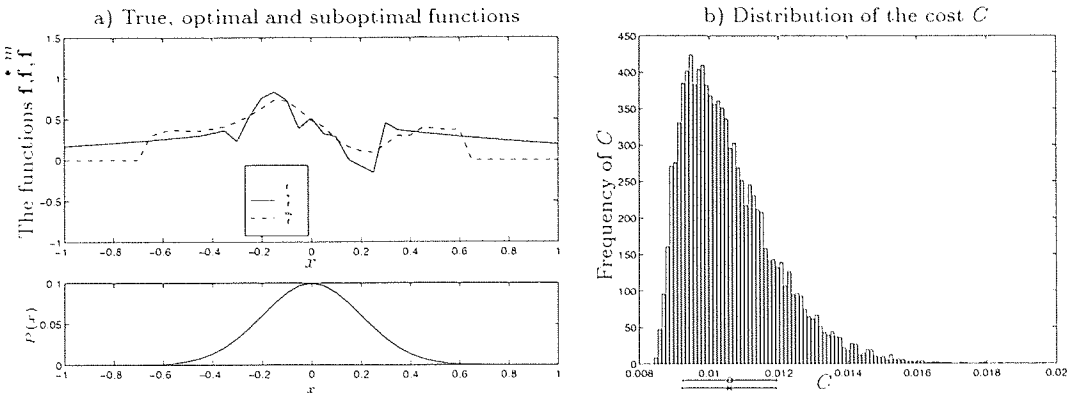


Figure 6.5: **a)** The lower figure shows the input distribution. The upper figure shows the true function $\mathbf{f}$ generated from a Gaussian prior with covariance matrix $\mathbf{V}$ (dotted line), the optimal function $\overset{*}{\mathbf{f}}$ (solid line) and the suboptimal solution $\overset{m}{\mathbf{f}}$ (dashed line). **b)** The distribution of the cost function obtained by generating functions from the posterior Gaussian with covariance matrix $\mathbf{A}$ and calculating the cost according to equation 6.29. The mean and one standard deviation calculated analytically and numerically are shown by the lower and upper error bars respectively.

network with 100 30-tuples.

For each problem we calculate the actual cost using the test set according to the formula

$$C = \tfrac{1}{2} \sum_i (y^i - \overset{m}{\mathbf{f}}_x)^2. \tag{6.30}$$

In order to verify the prior settings of the hyperparameters, we iterate over $\sigma_{\mathbf{f}}$ and γ and plot a family of cost curves that depend on the hyperparameter values. We expect that the actual cost line (constant and independent of $\sigma_{\mathbf{f}}$ and γ) will intersect the predicted cost curve for the values of $\sigma_{\mathbf{f}}$ and γ which agree with our prior assumptions.

The first example is Neal's Regression Problem which was previously used to illustrate the calculation of the posterior mean $\overset{*}{\mathbf{f}}$ (see section 6.4.2, equation 6.18). We recall that the training and test set each comprised 100 data-points. The inputs were generated by the standard normal distribution ($\mu_x = 0$, $\sigma_x = 1$). The input range $[-3, 3]$ was quantised into 61 uniform bins. The bin centres were assigned a 256 bit thermometer code.

The outputs were contaminated with the Gaussian noise with mean $\mu_\epsilon = 0$ and a diagonal noise covariance matrix $\mathbf{Q} = \mathbf{I}\beta$ where the standard deviation $\sqrt{1/\beta} = 0.1$. The cost weight $w_x \equiv 1$.

The training set and the functions $\mathbf{f}$, $\overset{m}{\mathbf{f}}$, $\overset{*}{\mathbf{f}}$ are shown on figure 6.6a for $\gamma = 0.1$. The correlation length of the functions, $\sigma_{\mathbf{f}}$, appears to be of order 1 and the overall scale of variation of the function $1/\sqrt{\gamma}$, is about 3, so γ should be about $1/9$. Figure 6.6b shows the expected cost as a function of γ for various choices of $\sigma_{\mathbf{f}}$, with error bars on the $\sigma_{\mathbf{f}} = 1.0$ curve. The actual cost (equation 6.30) is plotted with a dashed line. There is good agreement around the sensible values of γ and $\sigma_{\mathbf{f}}$.

The second experiment was concerned with another one-dimensional regression problem due to

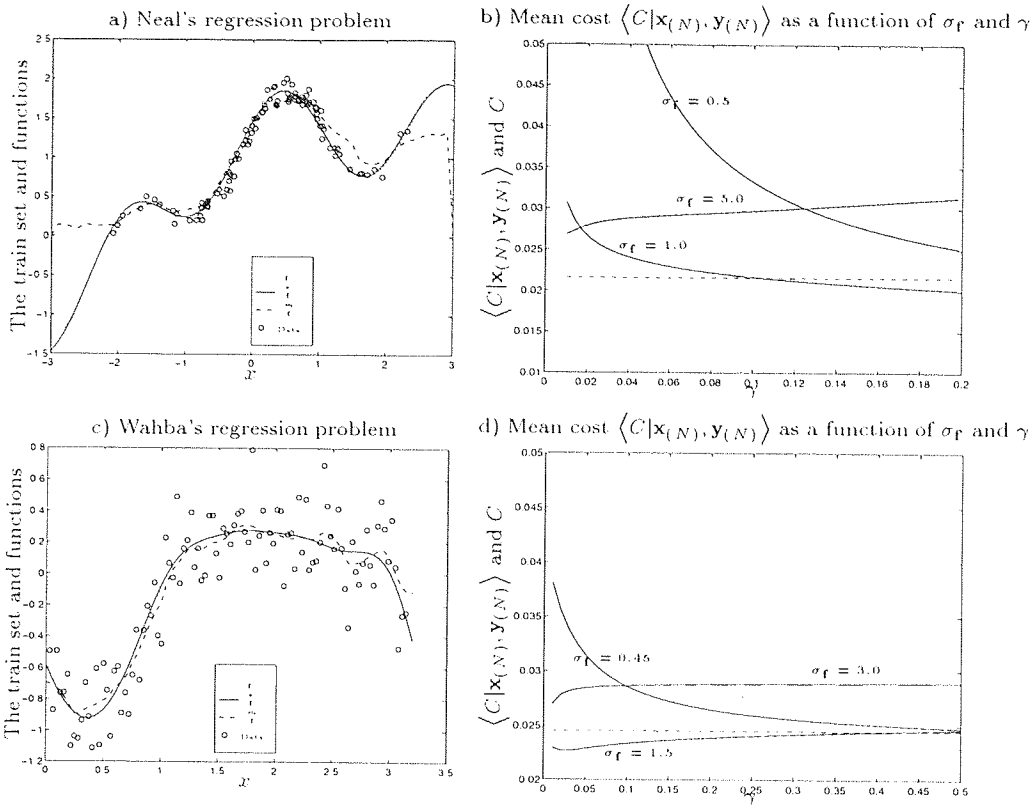


Figure 6.6: **a,c**) Neal's and Wahba Regression problems. The true function $\mathbf{f}$ is indicated by a dotted line, the optimal function $\overset{*}{\mathbf{f}}$ is denoted by a solid line and the suboptimal solution $\overset{m}{\mathbf{f}}$ is indicated by a dashed line. Circles indicate the training data. **b,d**) Dependence of the cost prediction on the values of parameters α and $\sigma_{\mathbf{f}}$. The cost evaluated from the test set is plotted as a dashed line, predicted cost is shown as a solid line with one standard deviation indicated by a dotted line.

Wahba (Wahba & Wold, 1975). The true function, is given by

$$\mathbf{f}_x = 4.26e^{-x} - 4e^{-2x} + 3e^{-3x} + \epsilon \tag{6.31}$$

where the Gaussian noise variable ϵ has mean $\mu_\epsilon = 0$ and a diagonal covariance noise matrix $\mathbf{Q} = \mathbf{I}\beta$ where the standard deviation $\sigma_\epsilon = \sqrt{1/\beta} = 0.2$. The training and test set comprised 100 data-points each. Unlike in Neal's problem, the inputs were generated from the uniform distribution. The input range $[0, 3.2]$ was quantised into 65 bins. The centre of each bin was assigned a 256 bit thermometer code.

The training set and the functions $\mathbf{f}$, $\overset{m}{\mathbf{f}}$, $\overset{*}{\mathbf{f}}$ are shown on figure 6.6c for $\gamma = 0.45$. The correlation length $\sigma_{\mathbf{f}}$, judging from figure 6.6c is somewhere between 0.4 and 1.5 (The function initially changes its value on the small scale, reaching its minimum and then its values are similar for large intervals Δx.).

The overall scale of variation of the function values $1/\sqrt{\gamma}$ is about 1.5, indicating that the value for γ should be about 0.45. We carried out the search in $\sigma_{\mathbf{f}}$ and γ space plotting the expected cost.

Figure 6.6d shows the expected cost as a function of γ for various choices of $\sigma_{\mathbf{f}}$, with error bars on the $\sigma_{\mathbf{f}} = 0.45$ curve. The actual cost computed from the test set is plotted with a dashed line. We note again a good agreement around the sensible values of γ and $\sigma_{\mathbf{f}}$.

6.6 Summary

We provide, for the first time, the means to analyse the generalisation properties of RAMnets in a Bayesian framework. The formalism using Bayesian inference with Gaussian process priors over functions allows us to calculate the expected cost of using a particular model (e.g. n-tuple regression network) for one dimensional regression problems.

We verified the analytical calculations with a numerical experiment and applied the framework in two regression problems using RAMnets as a model. We demonstrated experimentally that for the appropriate choice of the prior hyperparameters $\sigma_{\mathbf{f}}$ and γ we can predict the generalisation cost of an n-tuple regression network for the one dimensional noisy interpolation problem.

6.7 Conclusions

Ultra-fast methods such as RAMnets, which are not trained by directly optimising a cost function, can be analysed in a Bayesian framework to determine their generalisation cost. Because the formalism is constructed in terms of distributions over function space rather than distributions over model parameters, it can be used for model comparison, and in particular to select RAMnet parameters.

The main drawback with this technique, as it stands, is the need to numerically integrate the expression for the expected cost and variance. This difficulty intensifies rapidly as the input dimension increases. Therefore, an interesting research direction would be to search for RAMnet feature sets which allow these integrals to be performed analytically.

Chapter 7

Conclusions and Directions for Further Research

7.1 Conclusions

Our research focused on the one layer n-tuple network applied to classification and regression problems. We introduced a suitable formalism in chapter 2 and showed that efficient mapping algorithms exist for converting reals into bit-strings which allow real-valued data to be processed by RAMnets.

An extensive review was carried out in chapter 3 and indicated that previous research work was relatively self-contained and rarely compared the analytical and experimental results with other models. We validated the Hamming and Tuple distance theories illustrating them with numerical experiments and proceeded to use these analytical tools to show how RAMnets relate to other well established methods. We demonstrated that when the data or model parameters are appropriately constrained binary RAMnet can be directly compared with Bayesian Classifiers, Sparse Distributed Memories and Radial Basis Functions. One reason why the constraints arise is because RAMnets are not universal approximators. We demonstrated this fact viewing n-tuple processing as a projection onto the set of non-orthogonal tuple eigenfunctions making a comparison with the projection onto the Walsh functions. The sub-optimality of the n-tuple network is caused by the tuple mapping which does not admit all terms of the general function expansion and the training procedure which does not estimate the coefficients of the expansion so as to minimise the expected generalisation cost.

In order to determine how this affects the performance of RAMnets, we carried out in chapter 4 a large experimental study of the method benchmarking it against 23 other methods. The StatLog data used in this study is easily accessible allowing other weightless models to be compared in the future. We found that there is no systematic performance gap on 6 out of 11 datasets tested. The

results were modest for one dataset and the method failed entirely on the remaining four. The fact that RAMnets delivered a competitive performance for a considerable proportion of the datasets and that its failures were equally decisive motivated our further efforts to provide an explanation for this behaviour. Limited as it is, we found the Tuple distance tool valuable in developing a semi-quantitive argument based on the Generalisation Hypercubes. We argue, that that the problematic datasets do not have enough of the training data to ensure the correct generalisation. The calculation of the Generalisation Hypercubes is easy and can give a good indication of whether the RAMnets can be expected to perform well on the given data or not.

Because the generalisation in RAMnets is determined by the parameter n which at the same time controls the complexity of the features, it is not possible to eliminate this problem without a serious modification of the method. As the output of the n-tuple network is a crude approximation of the likelihood $P(\boldsymbol{\alpha}|c)$, skewed class priors can also pose a difficulty for the classifier. This depends on the overlap effects on tuples which are difficult to quantify and which we studied experimentally.

We studied the Frequency Weighted classification networks which approximate class posterior $P(c|\boldsymbol{\alpha})$ in chapter 5. We identified two theoretical stumbling blocks: derivation of the marginal probability estimates form the tally distribution and combining the marginals to obtain the joint feature distribution. The latter is very difficult without further knowledge of the correlations between tuples. We therefore assumed tuple independence and tackled only the former problem demonstrating that the routinely used Maximum Likelihood estimation is inaccurate for small tallies which appear most often in RAMnets. We proposed an alternative Good–Turing method which solves the zero tally problem in a principled manner and improves the performance of the Frequency Weighted model. However, because the training set is finite, the binary version outperforms the Frequency Weighted methods, for large values of n, regardless of the probability estimation technique employed. This is because all probabilistic models are based on the unlikely independence assumption and because the estimates become increasingly noisy as n increases. We also showed that the system with the unprincipled zero tally correction (MLZ) is equivalent to the binary system for large n and to the Frequency Weighted system with Maximum Likelihood estimation for the small n combining the strengths of the two versions.

The main source of sub-optimality (and speed) in RAMnets is the training algorithm which does not minimise the expected generalisation cost. We pursued this issue further in chapter 6. We concentrated on the regression RAMnets and a simple noisy interpolation problem. In order to calculate the expected generalisation cost of the method it is necessary to provide a point of reference, i.e., the optimal solution. We computed this optimum using a Bayesian framework with Gaussian process prior assumptions introducing a quadratic cost function to measure the accuracy of the model

for the given dataset.

This approach can be used to determine the optimal parameters of the regression RAMnet for one dimensional interpolation problems. Because RAMnets use feature sets which have a form that cannot be easily integrated analytically under a Gaussian input distribution, numerical approximations have to be used which is the main drawback of the approach.

Our experimental work verified that RAMnets are capable of delivering competitive results for a considerable proportion of the datasets, although since the method is potentially suboptimal, it may not be applicable for certain problems. We are able, however, to calculate analytically the generalisation cost of applying RAMnets to one dimensional regression problems and can semi-quantitatively obtain an indication if the method is likely to fail to classify given data.

The speed of the method means that, in the practical applications, the limited scope of analytical tools for quantifying the generalisation cost of RAMnets is not a major obstacle. The performance of the network can be determined numerically with little computational overhead.

We are of the opinion that, with all their inherent limitations, RAMnets are a powerful method for solving classification and regression problems. Taking into account their simplicity and speed, they should always be considered before more complicated models are applied.

7.1.1 Summary of the Contributions

The research contributions can be summarised as follows:

- Pointing out the relationship of RAMnets to other well established Neural Network Models.
- Verification of the effective tuple kernel theory and numerical work in the review which confirms earlier theories.
- Large scale experimental work that establishes the viability of the method and allows one to compare its performance with 23 other algorithms.
- Introduction of the Good–Turing method of estimation for RAMnets and comparison of the binary n-tuple model with Frequency Weighted versions showing the superiority of the former method over the later for large values of the parameter n.
- Introduction of the Generalisation Hypercubes which allows one to argue semi-quantitatively about the RAMnets performance on the data encoded by CMAC technique.
- Development of the Bayesian framework for the analytical calculation of the generalisation cost of the Regression RAMnets.

7.2 Suggestions for Further Research

Cost Distribution and the χ^2 Distribution. It would be beneficial to have an analytical formula for the generalisation cost distribution rather than to know only its mean and variance. The expected cost distribution is related to the χ^2 distribution as is apparent from the figure 6.5 and equation 6.29 which involves a sum of squares of Gaussian distributed variables $\mathbf{f}_x$. The relation to the χ^2 distribution is nontrivial due to the fact that the variables are correlated, i.e., the posterior covariance matrix $\mathbf{A}$ is not diagonal, and that each variable is weighted by the input distribution. Although we can cope with the former by diagonalising $\mathbf{A}$ and standardising the variables the latter is more awkward. However, it should be possible to derive the probability density function of the cost functional for certain classes of weight functions, e.g., a Gaussian.

Stochastic Processes and Classification. We introduced a Bayesian framework (see chapter 6) for the calculation of generalisation cost of the regression RAMnets. Further work could extend our approach to n-tuple classification networks. It is relatively recently that Gaussian processes have been proposed for Bayesian classification (Barber & Williams, 1997). The main complication of this setting is that the function values have to fall in range $[0, 1]$ (imposing a constraint on the form of the covariance function) and that the prior has to be specified on the transformed, activation function space. Because the integration over the hyperparameters is intractable, Monte Carlo methods have to be used to obtain approximations.

Tuple Kernel. In section 3.3.2 we introduced the tuple kernel function $K(x, x')$ of two inputs x and x' encoded with bit-strings $\mathbf{u}$ and $\mathbf{v}$ which can be approximated (see equation 3.16) by an exponential function of the Hamming distance between the vectors $\mathbf{u}$ and $\mathbf{v}$. This functional form, useful fortunate is not necessarily the most suitable one. It would be interesting to be able to find feature sets that give rise to other kernel functions (e.g. a Gaussian) while still preserving the simplicity of the n-tupling process.

Eigenfunction Expansion and Regularisation Theory. The Tuple kernel function can be viewed (see section 6.2) as an expansion in terms of the binary valued Kronecker delta functions. It is interesting to note that there is a familiar way (Riley, 1974) to expand a kernel into the form $K(x, x') = \sum_k \phi_k(x)\phi_k(x')$, at least when $K(x, x') = K(x - x')$, if the range of ϕ is not restricted to $\{0, 1\}$: an eigenfunction expansion. Indeed, principal component analysis[1] applied to a Gaussian with covariance matrix $\mathbf{K}$ shows that the smallest feature set for a given generalisation cost consists of the (real-valued) projections onto the leading eigenfunctions of $\mathbf{K}$.

[1] $\mathbf{K}^{-1}$ needs to be a compact operator for this to work in the infinite-dimensional limit.

In physics, the function (matrix) which is usually expanded is $\mathbf{K}^{-1}$ (with $K_{xx'} = K(x, x')$), i.e. the differential operator whose corresponding Green's function is $\mathbf{K}$. Because $\mathbf{K}^{-1}$ can be viewed as a differential operator of a certain differential equation there exists a link with the theory of regularisation (Poggio & Girosi, 1990; Girosi *et al.*, 1995). It would be interesting to view regression RAMnets from the point of view of this theory.

The main problem is that the mathematical tools of the calculus are suited for the continuous rather than binary-valued functions. The theory which could help to deal with the latter was introduced by Gibbs (Gibbs, 1970; Harmuth, 1972). He defined the concept of logical derivatives (based on the Walsh expansion) and proposed methods for solution of logical differential equations. It seems, however, that logical differential calculus is still in its infancy. Developments in this area could provide us with the analytical tools applicable to RAMnets in a more natural way than statistical methods used in this work.

References

Al-Alawi, R., & Stonham, T.J. 1992. A Training Strategy and Functionality Analysis of Digital Multi-Layer Neural Networks. *Journal of Intelligent Systems*, **2**, 53–93.

Albus, J.S. 1975. A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC). *Journal of Dynamic Systems Measurement and Control*, **97**(3), 220–227.

Aleksander, I. 1970. Microcircuit Learning Nets–Hamming Distance Behaviour. *Electronic Letters*, **6**(5), 134–136.

Aleksander, I., & Morton, H. 1995. *An Introduction to Neural Computing*. 2 edn. Thomson Publishing Company.

Aleksander, I., & Stonham, T.J. 1979. Guide to Pattern Recognition Using Random-Access Memories. *IEE Proceedings on Computers and Digital Techniques*, **2**(1), 29–40. part E.

Aleksander, I., & Wilson, M.J.D. 1985. Adaptive Windows for Image Processing. *IEE Proceedings - Pt.E*, **132**(5), 233–245.

Aleksander, I., Thomas, W.V., & Bowden, P.A. 1984. WISARD a Radical Step Forward in Image Recognition. *Sensor Review*, 120–124.

Allinson, N.M., & Kołcz, A. 1993. Enhanced N-tuple Approximators. *Weightless Neural Network Workshop*, 38–45.

Allinson, N.M., & Kołcz, A. 1994a. Application of the CMAC Input Encoding Scheme in the N-Tuple Approximation Network. *IEE Proceedings on Comput. Digit. Tech.*, **141**(3), 177–183.

Allinson, N.M., & Kołcz, A. 1994b (October). Euclidean Input Mapping in a N-Tuple Approximation Network. *Pages 285–289 of: 1994 Sixth IEEE Digital Signal Processing Workshop*. The Institute of Electrical and Electronics Engineers.

Allinson, N.M., & Kołcz, A. 1994c. The Theory and Practice of N-Tuple Neural Networks. *In:* Taylor, J.G. (ed), *Adaptive Computing and Information Processing:Neural Networks*. Unicom Publishing.

Allinson, N.M., & Kołcz, A. 1995a. *Distance Relationships in the N-Tuple Mapping.* submitted to Pattern Recognition.

Allinson, N.M., & Kołcz, A. 1995b. *N-tuple Regression Network.* to be published in Neural Networks.

Austin, J. 1988. Gray Scale N-Tuple Processing. *Pages 110–119 of:* Kittler, Josef (ed), *4th International Conference on Pattern Recognition*, vol. 301. Cambridge: Berlin Springer Verlag.

Austin, J. 1989a. ADAM: An Associative Neural Architecture For Invariant Pattern Classification. *Pages 196–200 of: IEE Conference Publication 313.* IEE. First International Conference On Artificial Neural Networks.

Austin, J. 1989b. High Speed Invariant Pattern Recognition Using Adaptive Neural Networks. *Pages 28–32 of: IEE Conference Publication 307*, vol. 307. IEE, Woolwich.

Austin, J. 1991 (February). Neural Networks for Safe Knowledge based Systems. *Pages 4.1–4.3 of: Colloquium on Expert Systems and Safety.* IEE.

Austin, J. 1992. Uncertain Reasoning with RAM Neural Networks. *Journal of Intelligent Systems*, **2**(1–4), 121–154.

Austin, J. 1995. A Review of RAM Based Neural Networks. *Pages 58–66 of: Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems.* Turin: IEEE Computer Society Press.

Austin, J., & Stonham, T.J. 1987. Distributed Associative Memory for Use in Scene Analysis. *Image and Vision Computing*, **5**(4), 251–260.

Austin, J., Jackson, T., & Wood, A. 1991. *VLSI for Artificial Intelligence and Neural Networks.* Plenum Press. Chap. 5.4 Efficient Implementation of Massive Neural Networks, pages 387–397.

Austin, J., Buckle, M. Brown S., & Kelly, I. 1993. ADAM Neural Networks for Parallel Vision. *Pages 173–180 of: JFIT Technical Conference.*

Badr, A. 1993. N-Tuple Classifier for ECG Signals. *Pages 29 – 32 of:* Allinson, N. (ed), *Proceedings of the Weightless Neural Network Workshop '93, Computing with Logical Neurons.*

Barber, D., & Williams, C.K.I. 1997. Gaussian Processes for Bayesian Classification via Hybrid Monte Carlo. *In:* Jordan, M.C. Mozer M.I., & Petsche, T. (eds), *Advances in Neural Information Processing Systems*, vol. 9. Morgan Kaufman. To appear.

Beale, R., & Jackson, T. 1990. *Neural Computing. An Introduction.* Adam Hilger.

REFERENCES

Beauchamp, K. G. 1975. *Walsh Functions and Their Applications.* Academic Press.

Berger, J.O. 1993. *Statistical Decision Theory and Bayesian Analysis.* Springer Verlag.

Bishop, C.M. 1995. *Neural Networks for Pattern Recognition.* Oxford University Press.

Bishop, J.M., Minchinton, P.R., & Mitchell, R.J. 1990. The Minchinton Cell–Analogue Input to the N-Tuple Net. *In: Proceedings of INNC.*

Bledsoe, W.W. 1961. Further Results on the n-tuple Pattern Recognition Method. *IRE Transactions on Electronic Computers*, **EC–10**(March), 96.

Bledsoe, W.W., & Bisson, C.L. 1962. Improved Memory Matrices for the N-Tuple Recognition Method. *IRE Joint Computer Conference*, **11**, 414–415.

Bledsoe, W.W., & Browning, I. 1959. Pattern Recognition and Reading by Machine. *Proceedings of the Eastern Joint Computer Conference*, 225–232.

Braga, A.P. 1995. *Design Models for Recursive Binary Neural Networks.* Ph.D. thesis, Imperial College London.

Cacoullos, T. 1966. Estimation of a Multivariate Density. *Annals of the Institute of Statistical Mathemetics*, **18**(2), 179–189.

Church, K.W., & Gale, W.A. 1991. A Comparison of the Enhanced Good-Turing and Deleted Estimation Methods for Estimating Probabilities of English Bigrams. *Computer Speech and Language*, **5**, 55–64.

Duda, R.O., & Hart, P.E. 1973. *Pattern Classification and Scene Analysis.* John Wiley and Sons.

Filho, E., Fairhist, M., & Bisset, D. 1991. Adaptive Pattern Recognition Using the Goal Seeking Neuron. *Pattern Recognition Letters*, **12**, 131–138.

Gale, W.A. 1993. *Good-Turing Smoothing without Tears.* AT & T Bell Laboratories, P.O. Box 636, Murray Hill, NJ 07974-0636, `gale@research.att.com`.

Gelman, A., Carlin, J.B., Stern, H.S, & Rubin, D.B. 1995. *Bayesian Data Analysis.* Chapman and Hall.

Gibbs, J.E. 1970. Discrete Walsh Functions. *Pages 106–122 of: Proceedings of the 1970 Walsh Functions Symposium.*

Girosi, F., Jones, M., & Poggio, T. 1995. Regularisation Theory and Neural Networks Architecture. *Neural Computation*, **7**, 219–269.

Good, I.J. 1953. The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika*, **40**(3), 237–263.

Gorse, D., & Taylor, J.G. 1993. Review of the Theory of pRAMs. *Pages 13–17 of:* Allinson, N.M. (ed), *Weightless Neural Network Conference.* University of York.

Guoqing, Y., Songcan, Ch., & Jun, L. 1992. Mulitilayer Parallel Distributed Pattern Recognition System Model Using Sparse RAM Nets. *IEE Proceedings-E*, **139**(2), 144–146.

Harmuth, H.F. 1972. *Transmission of Information by Orthogonal Functions.* Springer Verlag.

Haykin, S. 1994. *Neural Networks. A Comprehensive Foundation.* Macmillan.

Hertz, J., Krogh, A., & Palmer, R. 1991. *Introduction to the Theory of Neural Computation.* Addison Wesley.

Highleyman, W. H., & Kamentsky, L.A. 1960. Comments on a Character Recognition Method of Bledsoe and Browning. *IRE Transactions on Electronic Computers*, **EC–9**(June), 263.

Hughes, G.H. 1968. On the Mean Accuracy of Statistical Pattern Recognizers. *IEEE Transactions on Information Theory*, **14**(1), 55–63.

Hurst, S.L., Miller, D.M., & Muzio, J.C. 1985. *Spectral Techniques in Digital Logic.* Academic Press.

Kanerva, P. 1988. *Sparse Distributed Memory.* MIT Press.

Katz, S.M. 1987. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustics Speech and Signal Processing*, **35**(3), 400–401.

Knuth, D.E. 1973. *The Art of Computer Programming.* Vol. 3. Addison Wesley.

Kohonen, T. 1978. *Associative Memory.* Springer Verlag.

Lees, K., Austin, J., & Kennedy, J. 1995 (February). Neural Networks for Real-Time Knowledge Based Applications. *Pages 3/1–3/3 of: IEE Colloquium on Real Time Knowledge Based Systems.* IEE.

Liu, C.N. 1964. A Programmed Algorithm for Designing Multifont Character Recognition Logics. *IEEE Transactions on Electronic Computers*, **139**(2), 586–593.

MacKay, D. J. C. 1992a. Bayesian Interpolation. *Neural Computation*, **4**(3), 415–447.

MacKay, D. J. C. 1992b. A Practical Bayesian Framework for Back-propagation Networks. *Neural Computation*, **4**(3), 448–472.

Marr, D. 1982. *Vision*. San Francisco: W.H. Freeman and Co.

Michie, D., Spiegelhalter, D.J., & Taylor, C.C. (eds). 1994. *Machine Learning, Neural and Statistical Classification*. Prentice-Hall.

Nadás, A. 1985. On Turing's Formula for Word Probabilities. *IEEE Transactions on Acoustics, Speech and Signal Processing*, **33**(6), 1414–1416.

Neal, R. 1995. *Introductory Documentation for Software Implementing Bayesian Learning for Neural Networks Using Markov Chain Monte Carlo Techniques*. Tech. rept. Dept of Computer Science, University of Toronto. Distributed with the software available from http://www.cs.toronto.edu/ radford.

Nilsson, N. 1965. *Learning Machines*. New York: McGraw-Hill.

O'Hagan, A. 1994. *Kendall's Advanced Theory of Statistics*. Arnold. Pages 300–301.

O'Keefe, S.E.M., & Austin, J. 1994. Application of an Associative Memory to the Analysis of Document Fax Images. *Pages 315–325 of: Proceedings of the British Machine Vision Conference*.

Pao, Y. H. 1989. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley. Chap. 6. Recognition and Recall on the Basis of Partial Cues: Associative Memories, pages 141–170.

Papoulis, A. 1984. *Probability, Random Variables and Stochastic Processes*. McGraw Hill. Chap. 9 Stochastic Processes–General Concepts, pages 205–262.

Parzen, E. 1962. On Estimation of a Probability Density Function and Mode. *The annals of mathematical statistics*, **33**, 1065–1076.

Poggio, T., & Girosi, F. 1990. Networks for Approximation and Learning. *Proceedings of the IEEE*, **78**(9), 1481–1497.

Press, W. H., W.T., S.A. Teukolsky, Vetterling, & Flannery, B.P. 1992. *Numerical Recipies in C*. Cambridge University Press. Chap. 7. Random Numbers, pages 304–309.

Riley, K.F. 1974. *Mathematical Methods for the Physical Sciences*. Cambridge University Press. Chap. 7. Superposition methods.

Ripley, B.D. 1996. *Pattern Recognition and Neural Networks*. Cambridge University Press.

Rohwer, R., & Cressy, D. 1989. Phoneme Classification by Boolean Networks. *Proceedings of the European Conference on Speech Communication and Technology*, **2**, 557–560.

Rohwer, R., & Lamb, A. 1993. An Exploration of the Effect of Super Large N-Tuples on Single Layer RAMnets. *Pages 33 - 37 of:* Allinson, N. (ed), *Proceedings of the Weightless Neural Network Workshop '93, Computing with Logical Neurons.*

Rohwer, R., & Tarling, R. 1993. Efficient Use of Training Data in the N-Tuple Recognition Method. *Electronic Letters*, **29**(24), 2093–2094.

Roy, B.J., & Sherman, J. 1967. Two Viewpoints of k-Tuple Pattern Recognition. *IEEE Transactions on Systems, Science and Cybernetics*, **3**(2), 117–120.

Sebastyen, G. 1966. An Algorithm for Non-Parametric Pattern Recognition. *IEEE Transactions on Electronic Computers*, **EC–15**(6), 908–915.

Sixsmith, M.J., Tattersall, G.D., & Rollett, J.M. 1990. Speech Recognition Using N-Tuple Techniques. *British Telecom Technology Journal*, **8**(2), 50–60.

Sollich, Peter. 1994. Finite-Size Effects in Learning and Generalization in Linear Perceptrons. *J. Phys. A*, **27**, 7771–7784.

Specht, D.F. 1990. Probabilistic Neural Network. *Neural Networks*, **3**, 109–118.

Specht, D.F. 1991. A General Regression Neural Network. *IEEE Transactions on Neural Networks*, **2**(6), 568–576.

Stonham, T.J. 1977. Improved Hamming–Distance Analysis for Digital Learning Networks. *Electronic Letters*, **13**(6), 155–156.

Tattersall, I., Foster, S., & Johnston, R.D. 1991. Single Layer Lookup Perceptron. *IEE Proceedings-F*, **138**(1), 46–54.

Ullmann, J.R. 1969. Experiments with the n-tuple Method of Pattern Recognition. *IEEE Transactions on Computers*, **18**(12), 1135–1137.

Ullmann, J.R. 1971. Reduction of Storage Requirements of Bledsoe and Browning's n-tuple Method of Pattern Recognition. *Pattern Recognition*, **3**, 297–306.

Ullmann, J.R., & Kidd, P.A. 1969. Recognition Experiments with Typed Numerals from Envelopes in the Mail. *Pattern Recognition*, **1**, 273–289.

von Mises, R. 1964. *Mathematical Theory of Probability and Statistics.* Academic Press. Chap. 8D. Multivariate Normal Distribution, pages 416–430.

Wahba, G., & Wold, S. 1975. A Completely Automatic French Curve. *Communications in Statistics*, 1–17.

Williams, C.K.I. 1995 (July). *Regression with Gaussian Processes.* To appear in Annals of Mathematics and Artificial Intelligence.

Witten, I.H., & Bell, T.C. 1991. The Zero-Frequency Problem: Estimating the Probabilities of Novel Events in Adaptive Text Compression. *IEEE Transactions on Information Theory*, **37**(4), 1085–1094.

Wong, K.Y.M., & Sherrington, D. 1988. Storage Properties of Randomly Connected Boolean Networks for Associative Memory. *Europhysics Letters*, **7**(3), 197–202.

Zhu, H., & Rohwer, R. 1996. Bayesian Regression Filters and the Issue of Priors. *Neural Computing and Applications*, **4**(3), 130–142.

Zurada, J.M. 1992. *Artificial Neural Systems.* West Publishing Company.

Appendix A

Mathematical Calculations

This appendix is concerned with the integration of the expressions for the expected cost and variance.

In section A.1 we consider Gaussian integrals of certain forms that repeatedly appear in the calculations. We carry out the integration of the expected cost (equation 6.24) and obtain the approximation 6.27 in section A.2. The cost variance (equation 6.25) is integrated in section A.3 and its approximation 6.28 obtained. We note, that the approximations 6.28 and 6.27 are model independent.

Section A.4 contains further details of the expected cost integration. We consider there a class of models linear in the output training data which, as demonstrated in section 6.2, includes the regression RAMnets.

A.1 Gaussian Integrals Involving Quadratic Forms

Let $\mathbf{R}$ be an arbitrary matrix and $\mathbf{V}$ denote the (symmetric) covariance matrix. We are concerned with the Gaussian integrals involving powers of a quadratic form $Q = (\mathbf{x}^{\mathrm{T}}\mathbf{R}\mathbf{x})^n$ for $n = 1$ and $n = 2$.

For $n = 1$ the Gaussian integral of Q has the form

$$\mathcal{I}_1 = \int e^{-\frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{x}}\mathbf{x}^{\mathrm{T}}\mathbf{R}\mathbf{x} \tag{A.1}$$

Note, that we can rewrite it as

$$\begin{aligned}\mathcal{I}_1 &= \frac{d}{d\gamma}\int e^{-\frac{1}{2}\mathbf{x}^{\mathrm{T}}(\mathbf{V}^{-1}-2\gamma\mathbf{R})\mathbf{x}} \qquad |\gamma = 0 \\ &= \frac{d}{d\gamma}\det\left[2\pi(\mathbf{V}^{-1}-2\gamma\mathbf{R})^{-1}\right]^{\frac{1}{2}} \quad |\gamma = 0\end{aligned} \tag{A.2}$$

The differentiation of the determinant is possible if we rewrite it using the identity $\ln\det[\mathbf{A}] = \mathrm{tr}[\ln\mathbf{A}]$. We also note that $(\mathbf{V}^{-1} - 2\gamma\mathbf{R})^{-1} = \mathbf{V}(\mathbf{I} - 2\gamma\mathbf{V}\mathbf{R})^{-1}$. This allows us to evaluate the derivative at

$\gamma = 0$ as follows

$$\begin{aligned}
\mathcal{I}_1 &= \det\left[2\pi\mathbf{V}\right]^{\frac{1}{2}} \frac{d}{d\gamma} \det\left[(\mathbf{I} - 2\gamma\mathbf{V}\mathbf{R})^{-1}\right]^{\frac{1}{2}} \qquad |\gamma = 0 \qquad (A.3)\\
&= \tfrac{1}{2} \det\left[2\pi\mathbf{V}\right]^{\frac{1}{2}} \det\left[(\mathbf{I} - 2\gamma\mathbf{V}\mathbf{R})^{-1}\right]^{\frac{1}{2}} \frac{d}{d\gamma} e^{\mathrm{tr}\left[\ln\left((\mathbf{I}-2\gamma\mathbf{V}\mathbf{R})^{-1}\right)\right]} \quad |\gamma = 0\\
&= \det\left[2\pi\mathbf{V}\right]^{\frac{1}{2}} \det\left[(\mathbf{I} - 2\gamma\mathbf{R}\mathbf{V})^{-1}\right]^{\frac{1}{2}} e^{\mathrm{tr}\left[\ln\left((\mathbf{I}-2\gamma\mathbf{V}\mathbf{R})^{-1}\right)\right]}\\
&\quad \mathrm{tr}\left[(\mathbf{I} - 2\gamma\mathbf{V}\mathbf{R})^{-1}\mathbf{V}\mathbf{R}\right] \qquad |\gamma = 0\\
&= \det\left[2\pi\mathbf{V}\right]^{\frac{1}{2}} \mathrm{tr}\left[\mathbf{V}\mathbf{R}\right] \qquad (A.4)
\end{aligned}$$

The integrals involving higher powers of the quadratic form Q can be treated in a similar way by a repeated differentiation of equation A.2. In particular, the Gaussian integral of Q for $n = 2$ can be calculated as

$$\begin{aligned}
\mathcal{I}_2 &= \int e^{-\frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{x}} \left(\mathbf{x}^{\mathrm{T}}\mathbf{R}\mathbf{x}\right)^2 \qquad (A.5)\\
&= \det\left[2\pi\mathbf{V}\right]^{\frac{1}{2}} \frac{d^2}{d\gamma^2} \int e^{-\frac{1}{2}\mathbf{x}^{\mathrm{T}}(\mathbf{V}^{-1}-2\gamma\mathbf{R})\mathbf{x}} \qquad |\gamma = 0\\
&= \det\left[2\pi\mathbf{V}\right]^{\frac{1}{2}} \{-\det\left[(\mathbf{I} - 2\gamma\mathbf{V}\mathbf{R})^{-1}\right]^{-\frac{3}{2}} e^{2\mathrm{tr}\left[\ln\left((\mathbf{I}-2\gamma\mathbf{V}\mathbf{R})^{-1}\right)\right]}\\
&\quad \mathrm{tr}\left[(\mathbf{I} - 2\gamma\mathbf{V}\mathbf{R})^{-1}\mathbf{V}\mathbf{R}\right]^2 +\\
&\quad 2e^{\mathrm{tr}\left[\ln\left((\mathbf{I}-2\gamma\mathbf{V}\mathbf{R})^{-1}\right)\right]} \mathrm{tr}\left[(\mathbf{I} - 2\gamma\mathbf{V}\mathbf{R})^{-2}(\mathbf{V}\mathbf{R})^2\right]\} \qquad |\gamma = 0\\
&= \det\left[2\pi\mathbf{V}\right]^{\frac{1}{2}} \{2\mathrm{tr}\left[(\mathbf{V}\mathbf{R})^2\right] - \mathrm{tr}^2\left[\mathbf{V}\mathbf{R}\right]\}. \qquad (A.6)
\end{aligned}$$

Therefore, if the random variable $\mathbf{x}$ has a Gaussian distribution with zero mean and the covariance matrix $\mathbf{V}$ then following identities hold:

$$\begin{aligned}
\mathcal{E}\left[\mathbf{x}\mathbf{R}\mathbf{x}\right] &= \mathrm{tr}\left[\mathbf{V}\mathbf{R}\right] \qquad (A.7)\\
\mathcal{E}\left[(\mathbf{x}\mathbf{R}\mathbf{x})^2\right] &= 2\mathrm{tr}\left[(\mathbf{V}\mathbf{R})^2\right] - \mathrm{tr}^2\left[\mathbf{V}\mathbf{R}\right] \qquad (A.8)\\
\mathrm{var}\left[\mathbf{x}\mathbf{R}\mathbf{x}\right] &= 2\left(\mathrm{tr}\left[(\mathbf{V}\mathbf{R})^2\right] - \mathrm{tr}^2\left[\mathbf{V}\mathbf{R}\right]\right) \qquad (A.9)\\
\mathcal{E}\left[\mathbf{R}\mathbf{x}^n\right] &= 0 \quad \text{for odd } n. \qquad (A.10)
\end{aligned}$$

A.2 Integration of the Expected Cost

We first consider the integration (over y and then over x) of the cost functional $C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)})$. It is given by equation 6.22 which with $C(\overset{m}{\mathbf{f}}_x, y)$ expanded yields

$$C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)}) = \iint \tfrac{1}{2}(\overset{m}{\mathbf{f}}_x - y) w_x (\overset{m}{\mathbf{f}}_x - y) P\left(x|\mathbf{x}_{(N)}\right) P\left(y|x, \mathbf{f}\right) dx dy. \qquad (A.11)$$

We assume that the Gaussian noise is uniform and position independent with variance q_x/β. In this case, the likelihood of an output y for a given input x and a function $\mathbf{f}$ has the form

$$P(y|x,\mathbf{f}) = \det\left[\tfrac{2\pi}{\beta}q_x\right]^{-\frac{1}{2}} e^{-\frac{\beta}{2}(\mathbf{f}_x-y)q_x^{-1}(\mathbf{f}_x-y)} \tag{A.12}$$

Plugging the likelihood distribution A.12 into A.11 we have

$$\begin{aligned} C(\overset{m}{\mathbf{f}},\mathbf{f}|\mathbf{x}_{(N)}) &= \iint \det\left[\tfrac{2\pi}{\beta}q_x\right]^{-\frac{1}{2}} e^{-\frac{\beta}{2}(\mathbf{f}_x-y)q_x^{-1}(\mathbf{f}_x-y)} \\ &\quad \tfrac{1}{2}(\overset{m}{\mathbf{f}}_x-\mathbf{y})w_x(\overset{m}{\mathbf{f}}_x-y)P\left(x|\mathbf{x}_{(N)}\right)dxdy \\ &\quad \text{substituting } z=y-\mathbf{f}_x,\ dz=dy \\ &= \iint \left(\tfrac{2\pi}{\beta}q_x\right)^{-\frac{1}{2}} e^{-\frac{1}{2}\beta z q_x^{-1}z}\,\tfrac{1}{2}(\overset{m}{\mathbf{f}}_x-\mathbf{f}_x-z)w_x(\overset{m}{\mathbf{f}}_x-\mathbf{f}_x-z)P\left(x|\mathbf{x}_{(N)}\right)dxdz \\ &\quad \text{expanding the square and using A.10} \\ &= \iint \left(\tfrac{2\pi}{\beta}q_x\right)^{-\frac{1}{2}} e^{-\frac{1}{2}\beta z q_x^{-1}z}\,\tfrac{1}{2}\left((\overset{m}{\mathbf{f}}_x-\mathbf{f}_x)w_x(\overset{m}{\mathbf{f}}_x-\mathbf{f}_x)+zw_xz\right)P\left(x|\mathbf{x}_{(N)}\right)dxdz \\ &\quad \text{integrating out } z \text{ and using A.7} \\ &= \tfrac{1}{2}\int(\overset{m}{\mathbf{f}}_x-\mathbf{f}_x)w_x(\overset{m}{\mathbf{f}}_x-\mathbf{f}_x)P\left(x|\mathbf{x}_{(N)}\right)dx + \tfrac{1}{2}\int\frac{q_xw_x}{\beta}P\left(x|\mathbf{x}_{(N)}\right)dx \end{aligned} \tag{A.13, A.14}$$

The second integral involves the cost weight w_x and noise variance q_x. If w_x and q_x are uniform and the input distribution is Gaussian then this integral could be evaluated analytically. The first integral, however, involves an output of a model $\overset{m}{\mathbf{f}}_x$ which in general can be a complicated function of x.

Therefore, we will have to approximate the integral over x with a discrete sum, which implies quantisation of the input domain. There are at least two ways of doing this. One is to have variable width bins centered on the test inputs and the other is to use a fixed number, M of bins with a constnt width Δx. The first approach is data dependent and has the disadvantage that in the areas where there are few inputs the approximation will be terrible. The later method is data independent and the quality of the approximation can be controled by the parameter M. We feel that the second approach gives us more flexibility and we impose a uniform quantisation of the input domain. Observe that the consequence of this approach is that any function of x is assumed to be constant within Δx. Therefore, if the we are given P test points for which the expected cost should be calculated, in the resulting approximation we will actually use at most M points. This results in a numerical error but we verified experimetally that, for reasonable choices of M, it is not significant.

We define a diagonal $M \times M$ matrix $\mathbf{R}$ holding the maximum likelihood estimates of the input distribution $P\left(x|\mathbf{x}_{(N)}\right)$ and the cost weights w_x:

$$R_{xx'} = \frac{1}{M}P\left(x|\mathbf{x}_{(N)}\right)w_x\delta_{x,x'} \tag{A.15}$$

and a diagonal $M \times M$ matrix $\mathbf{Q}$ with elements $Q_{xx'} = \delta_{x,x'}q_x$.

When the integrals over x are replaced with the sums over M bins, the cost functional $C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)})$ becomes

$$C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)}) = \tfrac{1}{2}(\overset{m}{\mathbf{f}} - \mathbf{f})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}} - \mathbf{f}) + \frac{\mathrm{tr}\,[\mathbf{QR}]}{2\beta} \tag{A.16}$$

We note, that the factor $1/M$ in $R_{xx'}$ plays the role of dx in the integration:

$$(\overset{m}{\mathbf{f}} - \mathbf{f})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}} - \mathbf{f}) = \sum_x (\overset{m}{\mathbf{f}_x} - \mathbf{f}_x) w_x (\overset{m}{\mathbf{f}_x} - \mathbf{f}_x) P\left(x|\mathbf{x}_{(N)}\right) \frac{1}{M} \approx \int (\overset{m}{\mathbf{f}_x} - \mathbf{f}_x) w_x (\overset{m}{\mathbf{f}_x} - \mathbf{f}_x) P\left(x|\mathbf{x}_{(N)}\right) dx.$$

Having calculated $C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)})$ we are ready to compute its posterior expectation given by the equation 6.24:

$$\begin{aligned}
\left\langle C|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\right\rangle &= \int C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)}) P\left(\mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\right) d\mathbf{f} && \text{(A.17)} \\
&\approx (\det[2\pi\mathbf{A}])^{-\frac{1}{2}} \int e^{-\frac{1}{2}(\mathbf{f}-\overset{*}{\mathbf{f}})^{\mathrm{T}}\mathbf{A}(\mathbf{f}-\overset{*}{\mathbf{f}})} \tfrac{1}{2}(\overset{m}{\mathbf{f}} - \mathbf{f})\mathbf{R}(\overset{m}{\mathbf{f}} - \mathbf{f}) d\mathbf{f} + \frac{\mathrm{tr}\,[\mathbf{QR}]}{2\beta} \\
&\quad \text{substituting } \mathbf{z} = \mathbf{f} - \overset{*}{\mathbf{f}},\ d\mathbf{z} = d\mathbf{f} \\
&= (\det[2\pi\mathbf{A}])^{-\frac{1}{2}} \int e^{-\frac{1}{2}\mathbf{z}^{\mathrm{T}}\mathbf{A}\mathbf{z}} \tfrac{1}{2}(\overset{m}{\mathbf{f}} - \mathbf{z} - \overset{*}{\mathbf{f}})\mathbf{R}(\overset{m}{\mathbf{f}} - \mathbf{z} - \overset{*}{\mathbf{f}}) d\mathbf{f} + \frac{\mathrm{tr}\,[\mathbf{QR}]}{2\beta} \\
&\quad \text{integrating out } \mathbf{z} \text{ and using A.7, A.10} \\
&= \tfrac{1}{2}\mathrm{tr}\,[\mathbf{AR}] + \tfrac{1}{2}(\overset{m}{\mathbf{f}} - \overset{*}{\mathbf{f}})\mathbf{R}(\overset{m}{\mathbf{f}} - \overset{*}{\mathbf{f}}) + \frac{\mathrm{tr}\,[\mathbf{QR}]}{2\beta}. && \text{(A.18)}
\end{aligned}$$

We note, that this expression can be evaluated for any model. It is not exact because we approximated the integrals over x with sums. In section A.4 we carry out the analysis of the expression A.18 further. We focus there on the regression RAMnets and express $\overset{m}{\mathbf{f}}_x$ in terms of the $\mathbf{J}$ matrix (see section 6.2).

A.3 Integration of the Cost Variance

The analytical expression for the cost variance (equation 6.25) is a difference of two terms: the posterior expectation of the square of the cost and the square of the expected cost which was calculated in the previous section. We deal with the former term first:

$$\begin{aligned}
&\int C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)})^2 P\left(\mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\right) d\mathbf{f} = && \text{(A.19)} \\
&\quad \text{using A.16 and expanding the square} \\
&= \int P\left(\mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\right) \left[\tfrac{1}{2}(\overset{m}{\mathbf{f}} - \mathbf{f})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}} - \mathbf{f})\right]^2 d\mathbf{f} \\
&+ \frac{\mathrm{tr}\,[\mathbf{QR}]}{\beta} \int P\left(\mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)}\right) \tfrac{1}{2}(\overset{m}{\mathbf{f}} - \mathbf{f})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}} - \mathbf{f}) d\mathbf{f} \\
&+ \left[\frac{\mathrm{tr}\,[\mathbf{QR}]}{2\beta}\right]^2 \\
&= \mathcal{I}_1 + \frac{\mathrm{tr}\,[\mathbf{QR}]}{\beta}\mathcal{I}_2 + \left[\frac{\mathrm{tr}\,[\mathbf{QR}]}{2\beta}\right]^2 && \text{(A.20)}
\end{aligned}$$

We expand the posterior $P\left(\mathbf{f}|\mathbf{x}_{(N)},\mathbf{y}_{(N)}\right)$ and the first integral $\mathcal{I}_1$ becomes

$$\begin{aligned}
\mathcal{I}_1 &= \det\left[2\pi\mathbf{A}\right]^{-\frac{1}{2}} \int e^{-\frac{1}{2}(\mathbf{f}-\overset{*}{\mathbf{f}})^{\mathrm{T}}\mathbf{A}^{-1}(\mathbf{f}-\overset{*}{\mathbf{f}})} \left[\tfrac{1}{2}(\overset{m}{\mathbf{f}}-\mathbf{f})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\mathbf{f})\right]^2 d\mathbf{f} \\
&\quad \text{substituting } \mathbf{z}=\mathbf{f}-\overset{*}{\mathbf{f}},\ d\mathbf{z}=d\mathbf{f} \text{ and expanding the square} \\
&= \det\left[2\pi\mathbf{A}\right]^{-\frac{1}{2}} \int e^{-\frac{1}{2}\mathbf{z}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{z}} \left\{ \left[\tfrac{1}{2}(\overset{m}{\mathbf{f}}-\overset{*}{\mathbf{f}})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\overset{*}{\mathbf{f}})\right]^2 \right. \qquad (A.21) \\
&+ \tfrac{1}{2}(\overset{m}{\mathbf{f}}-\overset{*}{\mathbf{f}})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\overset{*}{\mathbf{f}})\cdot\mathbf{z}^{\mathrm{T}}\mathbf{R}\mathbf{z} + \left[\tfrac{1}{2}\mathbf{z}^{\mathrm{T}}\mathbf{R}\mathbf{z}\right]^2 \\
&- \left. \mathbf{z}^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\overset{*}{\mathbf{f}}) \left[\tfrac{1}{2}(\overset{m}{\mathbf{f}}-\overset{*}{\mathbf{f}})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\overset{*}{\mathbf{f}}) + \mathbf{z}^{\mathrm{T}}\mathbf{R}\mathbf{z}\right] + \left[\mathbf{z}^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\overset{*}{\mathbf{f}})\right]^2 \right\} d\mathbf{f}
\end{aligned}$$

We integrate each term of the sum in the curly brackets separately and use results A.7, A.8 and A.10 to obtain

$$\begin{aligned}
\mathcal{I}_1 &= \left[\tfrac{1}{2}(\overset{m}{\mathbf{f}}-\mathbf{f})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\mathbf{f})\right]^2 + \tfrac{1}{2}(\overset{m}{\mathbf{f}}-\mathbf{f})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\mathbf{f})\cdot\mathrm{tr}\left[\mathbf{A}\mathbf{R}\right] \qquad (A.22) \\
&+ \mathrm{tr}\left[\mathbf{A}\mathbf{R}\mathbf{R}\mathbf{F}\mathbf{F}\right] + \tfrac{1}{2}\mathrm{tr}\left[\mathbf{A}\mathbf{R}\mathbf{A}\mathbf{R}\right] + \left[\tfrac{1}{2}\mathrm{tr}\left[\mathbf{A}\mathbf{R}\right]\right]^2
\end{aligned}$$

where the diagonal matrix $\mathbf{F}$ with elements $F_{xx'}=(\overset{m}{\mathbf{f}}_x-\overset{*}{\mathbf{f}}_x)\delta_{x,x'}$ was introduced in order to carry out the integration of the term involving a quadratic form $\left[\mathbf{z}^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\overset{*}{\mathbf{f}})\right]^2$.

Integral $\mathcal{I}_2$ was calculated before (see equation A.17) and is simply

$$\mathcal{I}_2 = \tfrac{1}{2}\mathrm{tr}\left[\mathbf{A}\mathbf{R}\right] + \tfrac{1}{2}(\overset{m}{\mathbf{f}}-\mathbf{f})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\mathbf{f}) \qquad (A.23)$$

Plugging in the equations A.22, A.23 into A.20 we finally obtain an expression for the second posterior moment of the cost functional $C(\overset{m}{\mathbf{f}},\mathbf{f}|\mathbf{x}_{(N)})$:

$$\begin{aligned}
&\int C(\overset{m}{\mathbf{f}},\mathbf{f}|\mathbf{x}_{(N)})^2 P\left(\mathbf{f}|\mathbf{x}_{(N)},\mathbf{y}_{(N)}\right) d\mathbf{f} \qquad (A.24) \\
&= \left[\tfrac{1}{2}(\overset{m}{\mathbf{f}}-\mathbf{f})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\mathbf{f})\right]^2 + \tfrac{1}{2}(\overset{m}{\mathbf{f}}-\mathbf{f})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\mathbf{f})\cdot\mathrm{tr}\left[\mathbf{A}\mathbf{R}\right] \qquad (A.25) \\
&+ \mathrm{tr}\left[\mathbf{A}\mathbf{R}\mathbf{R}\mathbf{F}\mathbf{F}\right] + \tfrac{1}{2}\mathrm{tr}\left[\mathbf{A}\mathbf{R}\mathbf{A}\mathbf{R}\right] + \left[\tfrac{1}{2}\mathrm{tr}\left[\mathbf{A}\mathbf{R}\right]\right]^2 \\
&+ \frac{\mathrm{tr}\left[\mathbf{Q}\mathbf{R}\right]}{\beta}\left(\tfrac{1}{2}\mathrm{tr}\left[\mathbf{A}\mathbf{R}\right] + \tfrac{1}{2}(\overset{m}{\mathbf{f}}-\mathbf{f})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\mathbf{f})\right) \\
&+ \left[\frac{\mathrm{tr}\left[\mathbf{Q}\mathbf{R}\right]}{2\beta}\right]^2
\end{aligned}$$

The square of the expected cost can easily be found using A.18

$$\begin{aligned}
&\left\langle C|\mathbf{x}_{(N)},\mathbf{y}_{(N)}\right\rangle^2 \qquad (A.26) \\
&= \left[\tfrac{1}{2}\mathrm{tr}\left[\mathbf{A}\mathbf{R}\right] + \tfrac{1}{2}(\overset{*}{\mathbf{f}}-\overset{m}{\mathbf{f}})^{\mathrm{T}}\mathbf{R}(\overset{*}{\mathbf{f}}-\overset{m}{\mathbf{f}}) + \frac{\mathrm{tr}\left[\mathbf{Q}\mathbf{R}\right]}{2\beta}\right]^2 \\
&= \left[\tfrac{1}{2}(\overset{*}{\mathbf{f}}-\overset{m}{\mathbf{f}})^{\mathrm{T}}\mathbf{R}(\overset{*}{\mathbf{f}}-\overset{m}{\mathbf{f}})\right]^2 + \tfrac{1}{2}(\overset{*}{\mathbf{f}}-\overset{m}{\mathbf{f}})^{\mathrm{T}}\mathbf{R}(\overset{*}{\mathbf{f}}-\overset{m}{\mathbf{f}})\cdot\mathrm{tr}\left[\mathbf{A}\mathbf{R}\right] \qquad (A.27) \\
&+ \frac{\mathrm{tr}\left[\mathbf{Q}\mathbf{R}\right]}{\beta}\left(\tfrac{1}{2}\mathrm{tr}\left[\mathbf{A}\mathbf{R}\right] + \tfrac{1}{2}(\overset{m}{\mathbf{f}}-\mathbf{f})^{\mathrm{T}}\mathbf{R}(\overset{m}{\mathbf{f}}-\mathbf{f})\right) \\
&+ \left[\tfrac{1}{2}\mathrm{tr}\left[\mathbf{A}\mathbf{R}\right]\right]^2 + \left[\tfrac{1}{2}\frac{\mathrm{tr}\left[\mathbf{Q}\mathbf{R}\right]}{\beta}\right]^2
\end{aligned}$$

After the subtraction of A.27 from A.25 of most of the terms disappear and the expression for variance is simply

$$\mathrm{var}\left[C(\overset{m}{\mathbf{f}}, \mathbf{f}|\mathbf{x}_{(N)}, \mathbf{y}_{(N)})\right] = \tfrac{1}{2}\mathrm{tr}\left[\mathbf{ARAR}\right] + \mathrm{tr}\left[\mathbf{ARRFF}\right]. \tag{A.28}$$

A.4 More Details of the Cost Expression

We recall from the section A.2 that in the expression for the expected cost A.14 the integrals over x had to be approximated with sums and that a matrix $\mathbf{R}$ was introduced to that purpose. In this section we consider again the expected cost for the models linear in the training outputs $\mathbf{y}_{(N)}$. We start of with the approximate form given by equation A.18, substitute $\mathbf{J}_{(PN)}\mathbf{y}_{(N)}$ for the model output $\overset{m}{\mathbf{f}}$ and work backwards replacing approximate terms involving the matrix $\mathbf{R}$ with the appropriate integrals over x. As a result of this procedure we arrive at an exact expression for the expected cost for linear models.

The approximation A.18 involves the model output $\overset{m}{\mathbf{f}}$ and the optimal approximation $\overset{*}{\mathbf{f}}$. If the Gaussian process prior has zero mean and if the model is linear then using 6.14 and 6.3 we can write their difference in terms of the training outputs $\mathbf{y}_{(N)}$ and matrices $\mathbf{K}$, $\mathbf{J}$ and $\mathbf{V}$ as

$$\begin{aligned} \overset{*}{\mathbf{f}} - \overset{m}{\mathbf{f}} &= \frac{1}{\gamma}\mathbf{V}_{(PN)}\mathbf{K}^{-1}\mathbf{y}_{(N)} - \mathbf{J}_{(PN)}\mathbf{y}_{(N)} \\ &= \left(\frac{1}{\gamma}\mathbf{V}_{(PN)}\mathbf{K}^{-1} - \mathbf{J}_{(PN)}\right)\mathbf{y}_{(N)} \end{aligned} \tag{A.29}$$

Consequently,

$$\tfrac{1}{2}(\overset{*}{\mathbf{f}} - \overset{m}{\mathbf{f}})^{\mathrm{T}}\mathbf{R}(\overset{*}{\mathbf{f}} - \overset{m}{\mathbf{f}}) = \tfrac{1}{2}\mathbf{y}_{(N)}^{\mathrm{T}}\mathbf{L}\mathbf{y}_{(N)} \tag{A.30}$$

where the matrix $\mathbf{L}$

$$\mathbf{L} = \left(\frac{1}{\gamma}\mathbf{V}_{(PN)}\mathbf{K}^{-1} - \mathbf{J}_{(PN)}\right)^{\mathrm{T}}\mathbf{R}\left(\frac{1}{\gamma}\mathbf{V}_{(PN)}\mathbf{K}^{-1} - \mathbf{J}_{(PN)}\right) \tag{A.31}$$

has elements L_{uv}

$$L_{uv} = \left(\frac{1}{\gamma}\sum_{t} V_{xx_t}K^{-1}_{tu} - J_{xx_u}\right)R_{xx'}\left(\frac{1}{\gamma}\sum_{s} V_{x'x_s}K^{-1}_{sv} - J_{x'x_v}\right) \tag{A.32}$$

where the indices x_t, x_u, x_v and x_s select the components of the training input vector $\mathbf{x}_{(N)}$, indices t, u, v and s vary between 1 and the number of training samples N and indices x and x' select the test inputs.

We can now expand equation A.30 in terms of L_{uv} yielding

$$\begin{aligned}
\tfrac{1}{2}\mathbf{y}_{(N)}^{\mathrm{T}}\mathbf{L}\mathbf{y}_{(N)} &= \tfrac{1}{2}\sum_{uv} y_u L_{uv} y_v \qquad \text{(A.33)}\\
&\quad \text{plugging in A.32}\\
&= \tfrac{1}{2}\sum_{uv} y_u \left(\frac{1}{\gamma}\sum_t V_{xx_t}K_{tu}^{-1} - J_{xx_u}\right) R_{xx'} \left(\frac{1}{\gamma}\sum_s V_{x'x_s}K_{sv}^{-1} - J_{x'x_v}\right) y_v\\
&\quad \text{expanding } R_{xx'} \text{ using A.15}\\
&= \tfrac{1}{2}\sum_{uv} y_u \left(\frac{1}{\gamma}\sum_t V_{xx_t}K_{tu}^{-1} - J_{xx_u}\right) \frac{1}{M} P\left(x|\mathbf{x}_{(N)}\right) w_x \delta_{x,x'}\\
&\quad \left(\frac{1}{\gamma}\sum_s V_{x'x_s}K_{sv}^{-1} - J_{x'x_v}\right) y_v\\
&\quad \text{multiplying and recovering the integrals over } x\\
&= \frac{1}{2\gamma^2}\int \sum_{uvts} y_u V_{xx_t} V_{xx_s} K_{sv}^{-1} K_{tu}^{-1} y_v P\left(x|\mathbf{x}_{(N)}\right) w_x dx \qquad \text{(A.34)}\\
&- \frac{1}{\gamma}\int \sum_{uvs} y_u J_{xx_u} V_{xx_s} K_{sv}^{-1} y_v P\left(x|\mathbf{x}_{(N)}\right) w_x dx\\
&+ \tfrac{1}{2}\int \sum_{uv} y_u J_{xx_u} J_{xx_v} y_v P\left(x|\mathbf{x}_{(N)}\right) w_x dx
\end{aligned}$$

where we used the fact that $\mathbf{K}$ is symmetric and the order of summation and integration is irrelevant. These integrals cannot in general be calculated because they involve the elements of the matrix $\mathbf{J}_{(PN)}$ and inverse of $\mathbf{K}$ which are complicated functions of x.

The remaining terms of the equation A.18 are

$$\begin{aligned}
\tfrac{1}{2}\mathrm{tr}\left[\mathbf{A}\mathbf{R}\right] &= \frac{1}{2M}\sum_x A_{xx'} P\left(x|\mathbf{x}_{(N)}\right) w_x \delta_{x,x'} \qquad \text{(A.35)}\\
&= \tfrac{1}{2}\int A_{xx} P\left(x|\mathbf{x}_{(N)}\right) w_x dx\\
&\quad \text{expanding } A_{xx} \text{ with 6.16}\\
&= \frac{1}{2\gamma}\int V_{xx} P\left(x|\mathbf{x}_{(N)}\right) w_x dx - \frac{1}{2\gamma^2}\int \mathbf{v}(x)^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{v}(x) P\left(x|\mathbf{x}_{(N)}\right) w_x dx\\
&\quad \text{expanding the vector } \mathbf{v} \text{ in terms of } V_{xx'}\\
&= \frac{1}{2\gamma}\int V_{xx} P\left(x|\mathbf{x}_{(N)}\right) w_x dx - \frac{1}{2\gamma^2}\int \sum_{tu} V_{xx_t} K_{tu}^{-1} V_{x_u x} P\left(x|\mathbf{x}_{(N)}\right) w_x dx \qquad \text{(A.36)}
\end{aligned}$$

and

$$\tfrac{1}{2}\mathrm{tr}\left[\mathbf{Q}\mathbf{R}\right] = \frac{1}{M}\sum_x \frac{q_x}{\beta}\delta_{x,x'} w_{x'} P\left(x'|\mathbf{x}_{(N)}\right)\delta_{x,x'} = \frac{1}{\beta}\int q_x w_x P\left(x|\mathbf{x}_{(N)}\right) dx. \qquad \text{(A.37)}$$

Equation A.37 could be easily integrated if q_x and r_x were uniform and the covariance matrix as well as the inputs density $P\left(x|\mathbf{x}_{(N)}\right)$ had a Gaussian form. The integration of A.36 again poses a difficulty because of the inverse of $\mathbf{K}$ appearing in the integrand.

Gathering together the results given by A.34, A.36 and A.37 we obtain an expression for the

expected cost for the models linear in the output data:

$$\begin{aligned}\left\langle C|\mathbf{x}_{(N)},\mathbf{y}_{(N)}\right\rangle &= \frac{1}{2\gamma^2}\int\sum_{uvts} y_u V_{xx_t} V_{xx_s} K^{-1}_{su} K^{-1}_{tu} y_v P\left(x|\mathbf{x}_{(N)}\right) w_x dx \\ &- \frac{1}{\gamma}\int\sum_{uvs} y_u J_{xx_u} V_{xx_s} K^{-1}_{sv} y_v P\left(x|\mathbf{x}_{(N)}\right) w_x dx \\ &+ \tfrac{1}{2}\int\sum_{uv} y_u J_{xx_u} J_{xx_v} y_v P\left(x|\mathbf{x}_{(N)}\right) w_x dx \\ &+ \frac{1}{2\gamma}\int V_{xx} P\left(x|\mathbf{x}_{(N)}\right) w_x dx - \frac{1}{2\gamma^2}\int\sum_{tu} V_{xx_t} K^{-1}_{tu} V_{x_u x} P\left(x|\mathbf{x}_{(N)}\right) w_x dx \\ &+ \frac{1}{\beta}\int q_x w_x P\left(x|\mathbf{x}_{(N)}\right) dx.\end{aligned} \tag{A.38}$$

In general, this exact form it is not tractable analytically. It is, however, plausible that for certain non trivial forms of $\mathbf{J}$, $\mathbf{V}$ and $P\left(x|\mathbf{x}_{(N)}\right)$ the integration could be accomplished.

A.5 Conclusions

The analysis of the expression for the expected generalisation cost shows that the closed form cannot be obtained. The major problem concerns the integration of terms involving the model output $\overset{m}{\mathbf{f}}_x$ which in general is a complicated function of x.

We also considered a class of models which are linear in the output training data. It turns out that for this class, which includes the regression RAMnet, the integration over inputs is also not possible in general.

Therefore, we approximate the integrals over the input domain with discrete sums, i.e., we impose a quantisation on x. Although numerical approximations can be easily obtained (with equation A.18) for one dimensional inputs, the problem becomes intractable as the input dimensionality increases. Monte Carlo methods (Press *et al.*, 1992) evaluate the difficult integrals in A.38 could be applied but they are bound to be time consuming for large datasets because the integrand scales badly with the training set size. However, similar integrals have been carried out in the thermodynamic limit (high input dimension) (Sollich, 1994), so the investigation of these techniques in the current setting could be a promising research direction. An alternative possibility would be to find such feature sets (tuple eigenfunctions) that yield forms of $\mathbf{J}$ that could be integrated under a given input distribution.