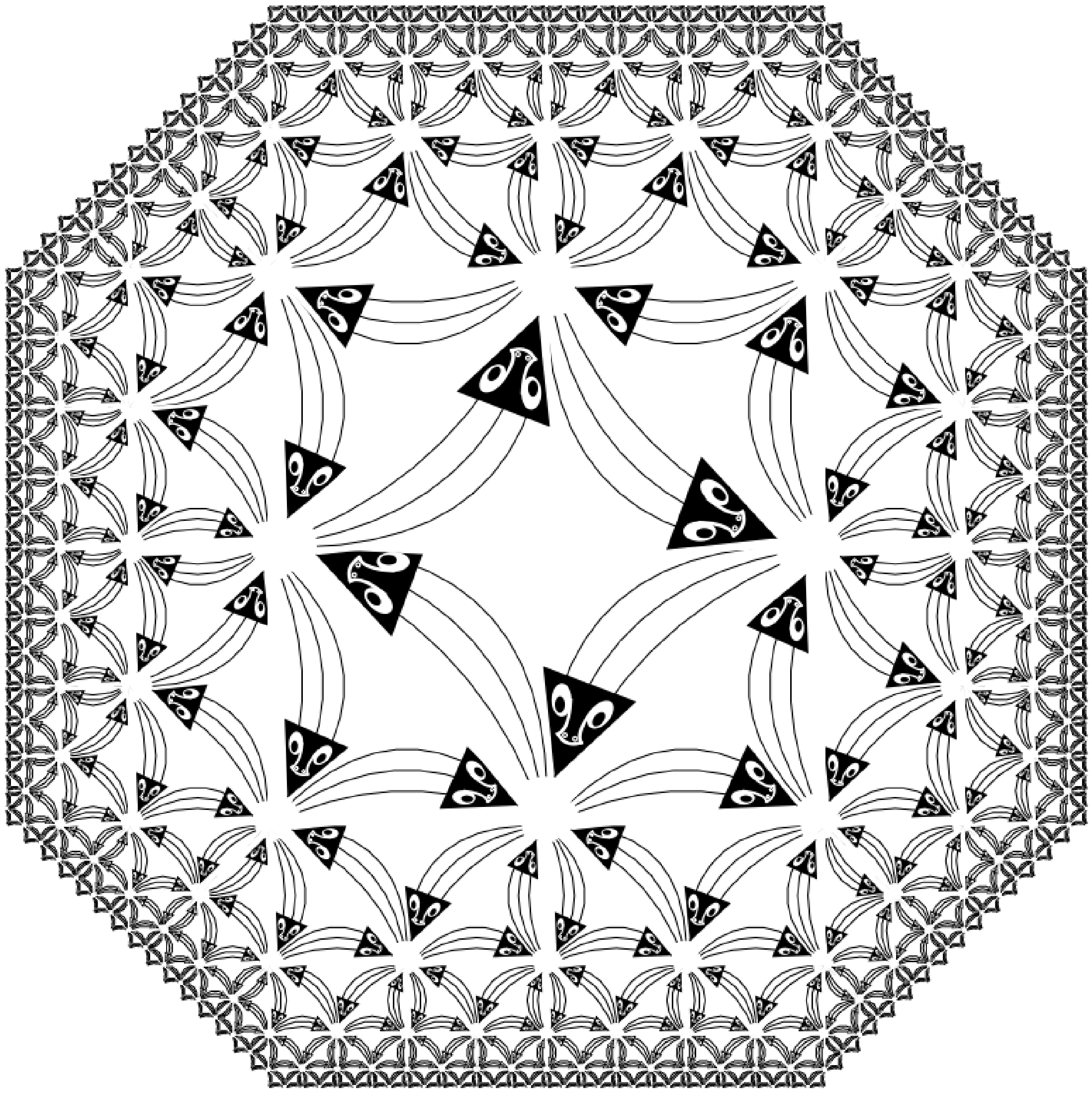
COLLECTED WORKS

— VOLUME 4 —

UNPUBLISHED PAPERS



Ubique Sentio

COLLECTED WORKS

— VOLUME 4 —

UNPUBLISHED PAPERS

STEPHEN LUTTRELL

COLLECTED WORKS: UNPUBLISHED PAPERS
Version 0.9, May 9, 2019
Copyright © Stephen Luttrell 2019.

Published by www.lulu.com.

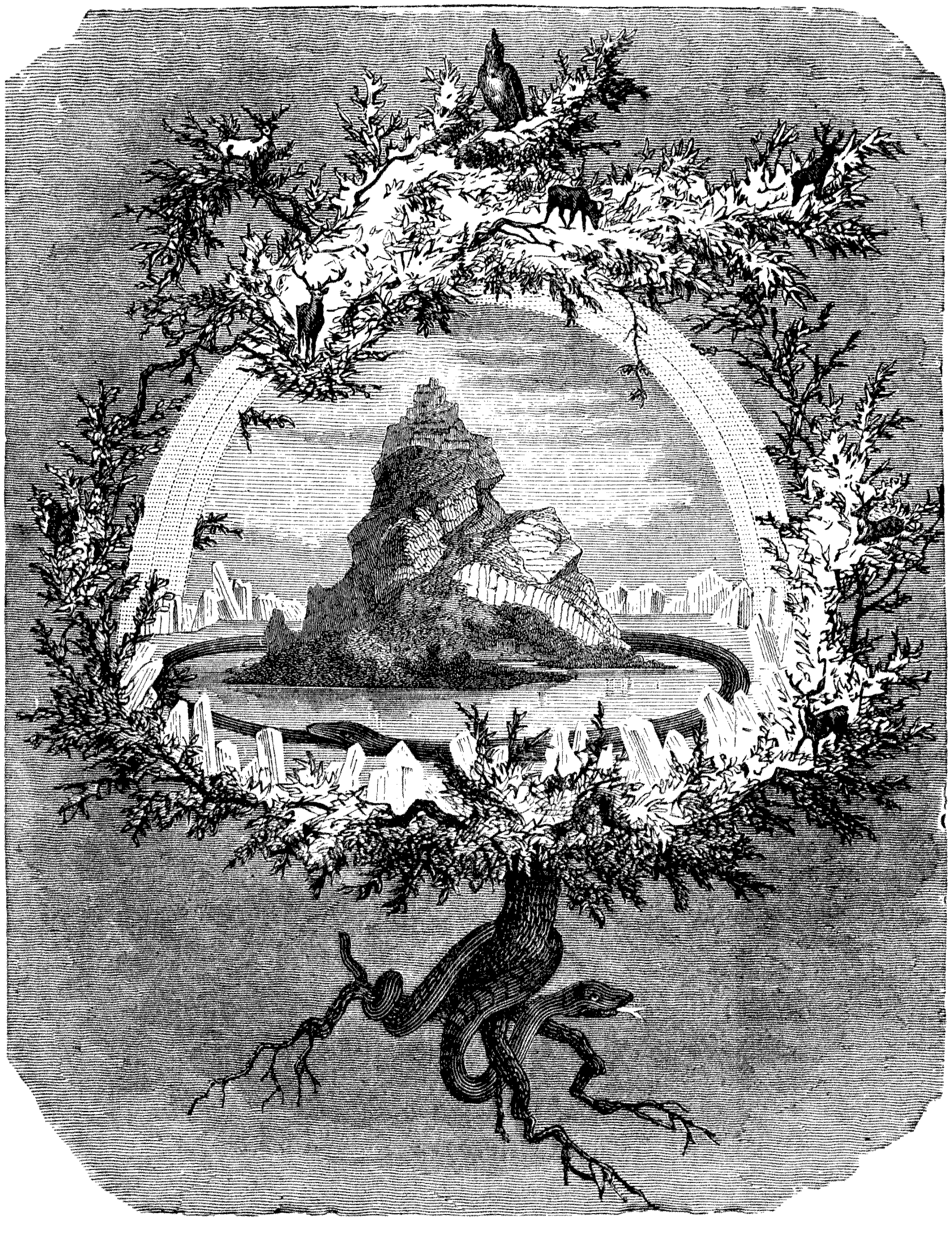
Edited by Stephen Luttrell.

The material in this book was produced from the individual master files (or manuscripts) that were originally submitted for publication, and then the material was cross-checked with the papers as originally published. Many of the graphical illustrations have been recreated from their original source code, and all known typographical errors have been corrected.

First published 2019.

The source text was prepared with LyX (document preparation system), and the source graphics was prepared with Mathematica (technical computing system) and GIMP (graphics editor). All of the source files were then typeset with LaTeX .

ISBN: XXXX



An ash I know there stands,
Yggdrasill is its name,
a tall tree, showered
with shining loam.
From there come the dews
that drop in the valleys.
It stands forever green over
Urðr's well.

Contents

Preface

I have often received comments that my published papers are spread all over the place, and that it is necessary to read a large subset of my papers before they individually begin to make sense. There is also the problem that many of my key papers were published in conference proceedings, or not published at all, which makes it difficult to obtain copies of them. These are the main reasons for the creation of this set of 4 volumes, which collect all of my papers so that they can be conveniently accessed alongside each other. There is some duplication of material, where for instance a technical report is very similar to a subsequently published paper, but I have included everything for completeness. The material is split between the 4 volumes as follows:

1. Published Papers – journal papers, conference papers, and book chapters.
2. Reports : Part 1 – Royal Signals and Radar Establishment (RSRE), Defence Research Agency (DRA), and Defence Evaluation and Research Agency (DERA) technical reports and research notes.
3. Reports : Part 2 – QinetiQ technical reports.
4. Unpublished Papers – mostly arXiv papers.

I worked as a British scientific civil servant for about 20 years, and during that time I published a lot of scientific papers all of which are British Crown copyright. Fortunately, Her Majesty's Stationery Office (HMSO) has very liberal views on allowing the reproduction of Crown copyright material. This has made possible the creation of this collection of papers which contains retypeset versions of all of my publications.

The route by which I produced this set of volumes was long and torturous – starting sometime in the 1990s, and continuing off-and-on (more off than on) up to the publication of this set of volumes. I experimented with many different software applications for processing the retypeset publications, but I ultimately converged on using LaTeX – via the LyX front-end to LaTeX – as being the only document processor that could easily create the high-quality results that I needed. I should have started doing things this way on "day one", but I wrongly believed that there must be a better approach, so I went off to explore the universe of alternative possibilities before finally returning to LaTeX.

Acknowledgements

I would like to thank various people for discussions which led to rapid advances in my understanding during the early stages of my basic research programme into information processing. Ordered chronologically, the key steps were: Chris Oliver (compound statistical modelling), Roy Pike (singular value decomposition), Steve Gull (SVD, Wiener filtering, Bayesian methods) and with John Skilling (maximum entropy method), Andrew Sleigh (artificial intelligence), Jonathan Pritchard (WISARD pattern recognition device, Kohonen self-organising networks), Robert Tough (stochastic processes, stochastic differential equations), Julian Besag (Markov random fields), John Bridle (Boltzmann Machine, hidden Markov models), David MacKay (stimulating discussions about Bayesian methods and information theory), Chris Webber (stimulating discussions about self-organising networks), Andrew Webb (stimulating discussions about many areas of information processing research).

I am indebted to the many people who were involved in various ways with writing software to implement the fruits of my information processing research – they breathed fire into my equations. I would also like to thank Stephen Wolfram for creating Mathematica powered by the Wolfram computer programming language, without which I would not have had the patience to write software myself – this has been my long-term "secret weapon" for making rapid progress in my research.

The Ministry of Defence (United Kingdom) continuously supported my long-term basic research programme into information processing. This started with the MoD's "basic research advisory committee" recruiting me as a civil servant to work at the government's Royal Signals and Radar Establishment. This establishment went through a couple of reorganisations and name-changes as the Defence (Evaluation and) Research Agency, and it was eventually sold off and privatised as QinetiQ.

Introduction

The motivation for the research that is described in these volumes is the wish to explain things in terms of their underlying causes, rather than merely being satisfied with phenomenological descriptions. When this reductionist approach is applied to information processing it allows the internal structure of information to be analysed, so information processing algorithms can then be derived from first principles.

One of the simplest examples of this approach is the diagonalisation of a data covariance matrix – there are many variants of this basic approach, such as singular value decomposition – in which the assumed independent components of high-dimensional data are identified and extracted. The main limitation of this type of information analysis approach is that it is based on linear algebra applied globally to the data space, so it is unable to preserve information about any local data structure in the data space. For instance, if the data lives on a low-dimensional curved manifold embedded in the data space, then only the global properties of this manifold would be preserved by global linear algebra methods.

In practice, data whose high-dimensional structure is non-trivial typically lives on a noisy version of a curved manifold, so techniques for analysing such data must automatically handle this type of structure. For instance, a blurred image of a point source is described by its underlying degrees of freedom – i.e. the position of the source – and as the source moves about it generates a curved manifold that lives in the high-dimensional space of pixel values of the sampled image. The basic problem is then to deduce the internal properties of this manifold by analysing examples of such images. A more challenging problem would be to extend this analysis to images that contain several overlapping blurred images of point sources, and so on. There is no limit to the complexity of the types of high-dimensional data that one might want to analyse.

These methods then need to be automated so that they do not rely on human intervention, which would then allow them to be inserted as "components" into information processing networks. The purpose of the research that is described in these volumes is to develop principled information processing methods that can be used for such analysis. Self-organising information processing networks arise naturally in this context, in which ways of cutting up the original manifold into simpler pieces emerge automatically.

CLUSTER DECOMPOSITION OF PROBABILITY DENSITY FUNCTIONS [*]

S. P. Luttrell
Royal Signals and Radar Establishment, St Andrews Road, Malvern, WORCS, WR14 3PS, UK

We derive a hierarchical cluster decomposition of joint probability density functions. This is realised in a multi-layer topographic neural network structure. Possible applications in image processing include clutter modelling and target detection.

Summary

A problem which is frequently encountered in the Bayesian analysis of images (or, more generally, patterns) is the choice of a suitable representation of probability density functions (PDF). We shall concentrate on network representations which are acquired as a result of a training process. Adaptive Markov random fields (MRF) provide one such approach, and the Boltzmann machine is a familiar example of an adaptive MRF. A limitation of the MRF approach to image analysis is the need to perform extensive numerical simulations in order to extract the simplest results. We therefore seek an alternative approach which substantially reduces the amount of computation which is involved without at the same time compromising the quality of the PDF representation too much.

We shall thus present a hierarchical cluster decomposition scheme. This is equivalent to a multilayer feedforward unsupervised neural network which forms a suitable hierarchical feature space for decomposing the input image. The node functions and the learning algorithm which we use are related to the self-organising scheme which has been proposed by Kohonen [1]. The PDF representation is completed by measuring a set of co-occurrence matrices on the network and combining them to produce the maximum entropy estimate of the true PDF. This representation is closely related to that used in the WISARD pattern recognition device.

This approach is much cheaper computationally than the MRF approach. During training we improve Kohonen's algorithm by introducing a renormalisation scheme in which we progressively increase the number of "neurons" in order to optimise the use of computational resources. During testing an image propagates forwards through the network in one pass, and the appropriate cooccurrence matrix elements are picked up. This type of network could find use in Bayesian image processing applications which currently use an MRF. The speed of use of the cluster decomposition scheme (both training and testing) is such that real-time adaptive PDFs might also be possible.

We shall present some simple demonstrations of this type of neural network, and where possible we shall compare the theoretical with the actual PDF.

[1] Kohonen T, 1984, "Self-organisation and associative memories", Springer, Berlin

[*] Typeset in LaTeX on May 3, 2019.
This summary was submitted to the IEEE Conference on Neural Information Processing Systems (1988) on 9 May 1988. Paper #295. It was not accepted for presentation, but it underpins several subsequently published papers.

Adaptive Cluster Expansion (ACE): A Hierarchical Bayesian Network [*]

S P Luttrell
Room EX21, QinetiQ, Malvern Technology Centre

Using the maximum entropy method, we derive the "adaptive cluster expansion" (ACE), which can be trained to estimate probability density functions in high dimensional spaces. The main advantage of ACE over other Bayesian networks is its ability to capture high order statistics after short training times, which it achieves by making use of a hierarchical vector quantisation of the input data. We derive a scheme for representing the state of an ACE network as a "probability image", which allows us to identify statistically anomalous regions in an otherwise statistically homogeneous image, for instance. Finally, we present some probability images that we obtained after training ACE on some Brodatz texture images – these demonstrate the ability of ACE to detect subtle textural anomalies.

I. INTRODUCTION

The purpose of this paper is to train probabilistic network models of images of homogeneous textures for use in Bayesian decision making. In our past work in this area [11–14, 16] we successfully used entropic methods to design Markov random field (MRF) models to reproduce the observed statistical properties of textured images. We now wish to formulate a novel MRF structure that requires much less effort to train and use. There are two essential ingredients in our simplification: we do not use hidden variables, and we restrict our attention to hierarchical transformations of the data.

The use of hidden variables is a flexible way of modelling high order correlations in data [1], but it leads to lengthy Monte Carlo simulations to estimate averages over the hidden variables. An MRF without hidden variables is specified by a set of transformation functions, each of which extracts some statistic from the data, and together they provide sufficient information to compute the probability density function (PDF) of the data [14, 16].

We can obtain a wealth of statistical information about the data by restricting our attention to a finite number of well-defined transformation functions. For instance, in [5] a number of useful textural features are presented, which may be used to model and discriminate between various textures that occur in images. However, we wish to design our transformation functions adaptively in a data-driven manner, so that the resulting set is optimised to capture the statistical properties of the data. We choose to use adaptive hierarchical transformation functions, because these not only capture statistical properties at many length scales, but are also very easy to train.

We briefly discussed hierarchical transformation functions in [21], where we conjectured that topographic mappings [8] might be appropriate for connecting together the layers of the hierarchy. We investigated topographic mappings in [15, 17, 19, 22, 23] and found that they could be rapidly trained to produce useful multiscale representations of data. We therefore use multilayer topographic mappings to adaptively design hierarchical transformation functions of data for use in MRF models. In this type of model different layers of the hierarchy measure statistical structure on different length scales, and shorter length scale structures are clustered together and correlated to produce longer length scale structures. We therefore frequently refer to this type of scheme as an adaptive cluster expansion (ACE). By interpreting ACE as a multilayered n-tuple processor we can relate ACE to a multilayered version of WISARD [2].

We demonstrate the ability of ACE to learn the statistical structure of texture by training an adaptive pyramid image processor. There are many ways of displaying the statistical information extracted from the data by such a processor, but we prefer to use what we call a "probability image", which is generated from the estimated local PDF of the data.

The layout of this paper is as follows. In Section II we use the maximum entropy method to estimate the PDF of the data, subject to a set of marginal probability constraints measured using hierarchical transformation functions, to yield an MRF model in closed form (i.e. no undetermined Lagrange multipliers). In Section III we extend this result to remove some of the limitations of its hierarchical structure, such a translation non-invariance, and describe the ACE system for producing probability images. In Section IV we present the result of applying ACE to some textured images taken from the Brodatz set [3].

II. MAXIMUM ENTROPY PDF ESTIMATION

In this section we present a derivation of a hierachical maximum entropy estimate $Q_{mem}(\boldsymbol{x})$ of an observed true PDF $P(\boldsymbol{x})$, where we constrain $Q_{mem}(\boldsymbol{x})$ so that certain marginal PDFs agree with observation. Although we consider only the case of a binary tree, we also present

[*]Typeset in LaTeX on May 3, 2019.
This is archived at http://arxiv.org/abs/cs/0410020, 10 Oct 2004. This paper was submitted to IEEE Trans. PAMI on 2 May 1991. Paper PAMI No. 91-05-18. It was not accepted for publication, but it underpins several subsequently published papers. © Crown Copyright 1991

a simple diagrammatic representation of this result that allows us easily to extend it to general trees.

A. Basic maximum entropy method

For completeness, we first of all outline the basic principles [6, 7] of the maximum entropy method of assigning estimates of PDFs. Introduce the entropy functional H

$$H = -\int d\boldsymbol{x}\, Q(\boldsymbol{x}) \log\left(\frac{Q(\boldsymbol{x})}{Q_0(\boldsymbol{x})}\right) \tag{2.1}$$

in which the PDF $Q_0(\boldsymbol{x})$ is used to introduce prior knowledge about $P(\boldsymbol{x})$. Loosely speaking, H measures the extent to which $Q(\boldsymbol{x})$ is non-committal about the value that $\boldsymbol{x}$ might take. The maximum entropy method consists of maximising H subject to the following set of constraints

$$\begin{aligned} C_{1,i} &= \int d\boldsymbol{x}\, Q(\boldsymbol{x})\, y_i(\boldsymbol{x}) - \int d\boldsymbol{x}\, P(\boldsymbol{x})\, y_i(\boldsymbol{x}) \\ &= 0 \end{aligned} \tag{2.2}$$

where the $y_i(\boldsymbol{x})$ are the components of a vector $\boldsymbol{y}(\boldsymbol{x})$ of sampling functions. These constraints ensure that certain average values are the same whether they are measured using $Q(\boldsymbol{x})$ (i.e. our estimated PDF) or using $P(\boldsymbol{x})$ (i.e. the observed true PDF). By carefully selecting the $\boldsymbol{y}(\boldsymbol{x})$ we can optimise the agreement between $Q(\boldsymbol{x})$ and $P(\boldsymbol{x})$ as appropriate.

$Q_{mem}(\boldsymbol{x})$ may be found by introducing a vector $\boldsymbol{\lambda}$ of Lagrange multipliers, and functionally differentiating $H - \sum_i \lambda_i\, C_{1,i}$ with respect to $Q(\boldsymbol{x})$ to yield eventually

$$Q_{mem}(\boldsymbol{x}) = \frac{Q_0(\boldsymbol{x}) \exp(-\boldsymbol{\lambda}.\boldsymbol{y}(\boldsymbol{x}))}{\int d\boldsymbol{x}'\, Q_0(\boldsymbol{x}') \exp(-\boldsymbol{\lambda}.\boldsymbol{y}(\boldsymbol{x}'))} \tag{2.3}$$

The undetermined Lagrange vector $\boldsymbol{\lambda}$ must be chosen in such a way that the constraints are satisfied – this is usually a non-trivial problem.

Now we shall consider a special case of the maximum entropy problem in which we carefully design the $y_i(\boldsymbol{x})$ so that they constrain a set of marginal probabilities [16]. Thus we make the following replacements

$$\begin{aligned} y_i(\boldsymbol{x}) &\longrightarrow \delta(y - y(\boldsymbol{x})) \\ \lambda_i &\longrightarrow \lambda(y) \end{aligned} \tag{2.4}$$

where $\delta(y - y(\boldsymbol{x}))$ is a Dirac delta function. In the $\{y_i(\boldsymbol{x}), \lambda_i\}$ version of the maximum entropy problem, by varying the value of an index i we could scan through the set of constraint functions $y_i(\boldsymbol{x})$ and Lagrange multipliers λ_i. However, in the $\{\delta(y - y(\boldsymbol{x})), \lambda(y)\}$ version of the maximum entropy problem, by varying the value of a variable y we can scan through the set of constraint functions $\delta(y - y(\boldsymbol{x}))$ and Lagrange multipliers $\lambda(y)$.

The modification in Equation 2.4 causes the constraints in Equation 2.2 to become

$$\begin{aligned} C_2(y) &\equiv \int d\boldsymbol{x}\, Q(\boldsymbol{x})\, \delta(y - y(\boldsymbol{x})) - \int d\boldsymbol{x}\, P(\boldsymbol{x})\, \delta(y - y(\boldsymbol{x})) \\ &= Q(y) - P(y) \\ &= 0 \end{aligned} \tag{2.5}$$

where we have defined the PDFs over y as

$$\begin{aligned} Q(y) &= \int d\boldsymbol{x}\, Q(\boldsymbol{x})\, \delta(y - y(\boldsymbol{x})) \\ P(y) &= \int d\boldsymbol{x}\, P(\boldsymbol{x})\, \delta(y - y(\boldsymbol{x})) \end{aligned} \tag{2.6}$$

Thus the delta function constraints force $Q(y) = P(y)$. Note that we have used a rather loose notation for our PDFs – $P(\boldsymbol{x})$ and $P(y)$ are in fact different functions of their respective arguments. We have made this choice of notation for simplicity, because the context will always indicate unambiguously which PDF is required.

By analogy with the previous maximum entropy derivation, $Q_{mem}(\boldsymbol{x})$ may be found by functionally differentiating $H - \int dy\, \lambda(y)\, C_2(y)$ with respect to $Q(\boldsymbol{x})$ to yield

$$\begin{aligned} Q_{mem}(\boldsymbol{x}) &= \frac{Q_0(\boldsymbol{x}) \exp(-\lambda(y(\boldsymbol{x})))}{\int d\boldsymbol{x}'\, Q_0(\boldsymbol{x}') \exp(-\lambda(y(\boldsymbol{x}')))} && (2.7) \\ &\longrightarrow Q_0(\boldsymbol{x})\, f(y(\boldsymbol{x})) && (2.8) \end{aligned}$$

where $\lambda(y(\boldsymbol{x}))$ is an undetermined Lagrange function of $y(\boldsymbol{x})$. In Equation 2.8 we present a simpler notation by introducing an undetermined function $f(y(\boldsymbol{x}))$ to absorb the exponential function and the denominator term that appeared in Equation 2.7. We may impose the constraints in Equation 2.5, and use the definitions of $Q(y)$ and $P(y)$ in Equation 2.6 to obtain $f(y)$ in the form

$$f(y) = \frac{P(y)}{\int d\boldsymbol{x}'\, Q_0(\boldsymbol{x}')\, \delta(y - y(\boldsymbol{x}'))} \tag{2.9}$$

and $Q_{mem}(\boldsymbol{x})$ in the form

$$Q_{mem}(\boldsymbol{x}) = \frac{Q_0(\boldsymbol{x})\, P(y(\boldsymbol{x}))}{\int d\boldsymbol{x}'\, Q_0(\boldsymbol{x}')\, \delta(y(\boldsymbol{x}) - y(\boldsymbol{x}'))} \tag{2.10}$$

Note that this result is a closed form solution because it contains no undetermined Lagrange functions, unlike Equation 2.3 which contains an undetermined Lagrange vector $\boldsymbol{\lambda}$. The normalisation of this solution can be verified as follows

$$\begin{aligned}\int d\boldsymbol{x}\, Q_{mem}(\boldsymbol{x}) &= \int d\boldsymbol{x}\, dy\, \delta(y - y(\boldsymbol{x}))\, \frac{Q_0(\boldsymbol{x})\, P(y)}{\int d\boldsymbol{x}'\, Q_0(\boldsymbol{x}')\, \delta(y - y(\boldsymbol{x}'))} \\ &= \int dy\, P(y) \\ &= 1\end{aligned} \tag{2.11}$$

where we use the identity $\int dy\, \delta(y - y(\boldsymbol{x})) = 1$ to create a dummy integral over y.

B. Hierarchical maximum entropy method

The purpose of this subsection is to present a generalisation of Equation 2.10 that uses hierarchical transformation functions.

In practice the result in Equation 2.10 has a limited usefulness. Firstly, we would like to impose many simultaneous constraints, each using its own constraint function $\delta(y_i - y_i(\boldsymbol{x}))$ in Equation 2.5, but this cannot in general be done without sacrificing our closed form solution in Equation 2.10. Secondly, we would like to impose higher order constraints, using a constraint function $\delta(\boldsymbol{y} - \boldsymbol{y}(\boldsymbol{x}))$. This may easily be done by making the replacement $y \longrightarrow (y_1, y_2, \cdots)$ in Equation 2.10. However, there is a hidden problem, because the greater the dimensionality of $\boldsymbol{y}$, the less easy is it to make the necessary measurements to establish the form of $P(\boldsymbol{y})$. Fortunately, there is a solution to both of these problems, which we shall describe below.

We shall apply the maximum entropy method with constraints of the form shown in Equation 2.5 to a hierarchy of transformed versions of the input vector $\boldsymbol{x}$. In order to make our calculation tractable we introduce the notation shown in Figure 1. The $\boldsymbol{x}_{ijk...}$ are various partitions of the input vector $\boldsymbol{x}$, the $y_{ijk...}$ are various transformed versions $y_{ijk...}(\boldsymbol{x}_{ijk...})$ of the input $\boldsymbol{x}_{ijk...}$, and the $f_{ijk\cdots,i'j'k'\cdots}$ are the Lagrange functions $f_{ijk\cdots,i'j'k'\cdots}(y_{ijk\cdots}, y_{i'j'k'\cdots})$ that appear in the generalised version of the maximum entropy solution $Q_{mem}(\boldsymbol{x})$ in Equation 2.7.

We choose to write the dependence of $y_{ijk...}$ directly on the input $\boldsymbol{x}_{ijk...}$, even though the value of $y_{ijk...}$ is obtained via a number of intermediate transformations leading from the leaf nodes of the tree up to node $ijk...$, because this leads to a transparent hierarchical maximum entropy derivation. It is convenient to define $\Pi_{ijk\cdots}(\boldsymbol{x}_{ijk\cdots})$ as the product of the Lagrange functions that appear beneath node $ijk...$ of the tree. $\Pi_{ijk\cdots}(\boldsymbol{x}_{ijk\cdots})$ has the following recursion property

$$\Pi_{ijk\cdots}(\boldsymbol{x}_{ijk\cdots}) = f_{ijk\cdots 1,ijk\cdots 2}(y_{ijk\cdots 1}(\boldsymbol{x}_{ijk\cdots 1}), y_{ijk\cdots 2}(\boldsymbol{x}_{ijk\cdots 2}))\, \Pi_{ijk\cdots 1}(\boldsymbol{x}_{ijk\cdots 1})\, \Pi_{ijk\cdots 2}(\boldsymbol{x}_{ijk\cdots 2}) \tag{2.12}$$

Also introduce a normalisation (or Jacobian) factor defined as

$$Z_{ijk\cdots}(y_{ijk\cdots}) = \int d\boldsymbol{x}_{ijk\cdots}\, \delta(y_{ijk\cdots} - y_{ijk\cdots}(\boldsymbol{x}_{ijk\cdots}))\, \Pi_{ijk\cdots}(\boldsymbol{x}_{ijk\cdots}) \tag{2.13}$$

which is a sum of $\Pi_{ijk\cdots}(\boldsymbol{x}_{ijk\cdots})$ over all the states $\boldsymbol{x}_{ijk\cdots}$ of the leaf nodes beneath node $ijk...$ that are consistent with $y_{ijk...}$ emerging at node $ijk...$.

The proof of the general hierarchical maximum entropy result proceeds inductively. Firstly, we generalise Equation 2.4 to become

$$\begin{aligned} y_i(\boldsymbol{x}) &\longrightarrow \delta(y_{ijk\cdots 1} - y_{ijk\cdots 1}(\boldsymbol{x}_{ijk\cdots 1}))\, \delta(y_{ijk\cdots 2} - y_{ijk\cdots 2}(\boldsymbol{x}_{ijk\cdots 2})) \\ \lambda_i &\longrightarrow \lambda_{ijk\cdots}(y_{ijk\cdots 1}, y_{ijk\cdots 2}) \end{aligned} \tag{2.14}$$

Secondly, we generalise Equation 2.8 to become

$$Q_{mem}(\boldsymbol{x}) = Q_0(\boldsymbol{x})\, f_{1,2}(y_1(\boldsymbol{x}_1), y_2(\boldsymbol{x}_2))\, \Pi_1(\boldsymbol{x}_1)\, \Pi_2(\boldsymbol{x}_2) \tag{2.15}$$

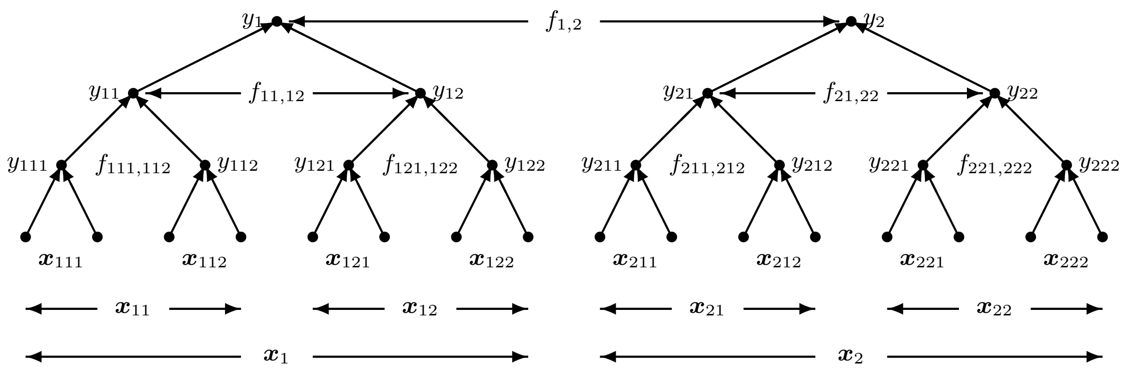


Figure 1: Notation used in the hierarchical maximum entropy derivation.

where we display the Lagrange function $f_{1,2}(y_1, y_2)$ that connects the topmost node-pair (i.e. node-pair $(1,2)$) in the tree, but conceal the other Lagrange functions by using the $\Pi_{ijk\cdots}$ notation.

We may determine the exact form of $f_{1,2}(y_1, y_2)$ independently of the rest of the Lagrange functions (which are hidden inside the $\Pi_1(\boldsymbol{x}_1)$ and $\Pi_2(\boldsymbol{x}_2)$ functions) by imposing the constraint shown in Equation 2.5 and Equation 2.6 (as applied to node-pair $(1,2)$) to obtain

$$\begin{aligned} P_{1,2}(y_1, y_2) &= \int d\boldsymbol{x}_1\, d\boldsymbol{x}_2\, \delta(y_1 - y_1(\boldsymbol{x}_1))\, \delta(y_2 - y_2(\boldsymbol{x}_2))\, Q_{mem}(\boldsymbol{x}) \\ &= f_{1,2}(y_1, y_2)\, Z_1(y_1)\, Z_2(y_2) \end{aligned} \tag{2.16}$$

which yields

$$f_{1,2}(y_1, y_2) = \frac{P_{1,2}(y_1, y_2)}{Z_1(y_1)\, Z_2(y_2)} \tag{2.17}$$

Substituting this result back into Equation 2.15 yields

$$Q_{mem}(\boldsymbol{x}) = P_{1,2}(y_1(\boldsymbol{x}_1), y_2(\boldsymbol{x}_2))\, \frac{\Pi_1(\boldsymbol{x}_1)}{Z_1(y_1(\boldsymbol{x}_1))}\, \frac{\Pi_2(\boldsymbol{x}_2)}{Z_2(y_2(\boldsymbol{x}_2))} \tag{2.18}$$

which correctly obeys the constraint on the joint PDF $P_{1,2}(y_1, y_2)$ of the topmost pair of nodes in the tree.

We now marginalise $Q_{mem}(\boldsymbol{x})$ in order to concentrate our attention on the left-hand main branch of the tree. Thus

$$\begin{aligned} Q_{1,mem}(\boldsymbol{x}_1) &= \int d\boldsymbol{x}_2\, Q_{mem}(\boldsymbol{x}) \\ &= \int d\boldsymbol{x}_2\, dy_2\, \delta(y_2 - y_2(\boldsymbol{x}_2))\, Q_{mem}(\boldsymbol{x}) \\ &= P_1(y_1(\boldsymbol{x}_1))\, \frac{\Pi_1(\boldsymbol{x}_1)}{Z_1(y_1(\boldsymbol{x}_1))} \end{aligned} \tag{2.19}$$

We now use the recursion property given in Equation 2.12 to extract the Lagrange function associated with node-pair $(11,12)$. Thus $Q_{1,mem}(\boldsymbol{x}_1)$ becomes

$$Q_{1,mem}(\boldsymbol{x}_1) = P_1(y_1(\boldsymbol{x}_1))\, f_{11,12}(y_{11}(\boldsymbol{x}_{11}), y_{12}(\boldsymbol{x}_{12}))\, \frac{\Pi_{11}(\boldsymbol{x}_{11})\, \Pi_{12}(\boldsymbol{x}_{12})}{Z_1(y_1(\boldsymbol{x}_1))} \tag{2.20}$$

As before, we may determine the exact form of $f_{11,12}(y_{11}, y_{12})$ independently of the rest of the Lagrange functions by applying the constraints to node-pair $(11,12)$ to obtain

$$f_{11,12}(y_{11}, y_{12}) = \frac{P_{11,12}(y_{11}, y_{12})}{P_1(y_1)}\, \frac{Z_1(y_1)}{Z_{11}(y_{11}) Z_{12}(y_{12})} \tag{2.21}$$

where the value of y_1 is to be understood to be obtained directly from the values of y_{11} and y_{12} via the mapping which connects node-pair $(11,12)$ to node 1. Substituting this result into Equation 2.20 yields

$$Q_{1,mem}(\boldsymbol{x}_1) = P_{11,12}(y_{11}(\boldsymbol{x}_{11}), y_{12}(\boldsymbol{x}_{12}))\, \frac{\Pi_{11}(\boldsymbol{x}_{11})}{Z_{11}(y_{11}(\boldsymbol{x}_{11}))}\, \frac{\Pi_{12}(\boldsymbol{x}_{12})}{Z_{12}(y_{12}(\boldsymbol{x}_{12}))} \tag{2.22}$$

By inspection, we see that Equation 2.18 and Equation 2.22 are identical in form once we have accounted for their different positions in the tree, so we may use induction to obtain all of the rest of the Lagrange functions in the form

$$f_{ijk\cdots 1,ijk\cdots 2}(y_{ijk\cdots 1}, y_{ijk\cdots 2}) = \frac{P_{ijk\cdots 1,ijk\cdots 2}(y_{ijk\cdots 1}, y_{ijk\cdots 2})}{P_{ijk\cdots}(y_{ijk\cdots})} \frac{Z_{ijk\cdots}(y_{ijk\cdots})}{Z_{ijk\cdots 1}(y_{ijk\cdots 1})\, Z_{ijk\cdots 2}(y_{ijk\cdots 2})} \tag{2.23}$$

which is analogous to Equation 2.21, and where $y_{ijk\cdots}$ is obtained directly from the values of $y_{ijk\cdots 1}$ and $y_{ijk\cdots 2}$. The $\frac{\Pi}{Z}$ factors may be discarded once we reach the leaf nodes of the tree, because the integral in Equation 2.13 then reduces to $Z = \Pi$.

Finally, by starting with Equation 2.15 and recursively simplifying the $\Pi_{ijk\cdots}$ using Equation 2.12 and substituting for the Lagrange functions $f_{ijk\cdots,i'j'k'\cdots}$ using Equation 2.23 we obtain eventually for an n-layer tree

$$\begin{aligned} Q_{mem}(\boldsymbol{x}) &= \left[\prod_{k=0}^{n-2} \prod_{i_1,i_2,\cdots,i_k=1}^{2} \frac{P_{i_1 i_2\cdots i_k 1, i_1 i_2\cdots i_k 2}(y_{i_1 i_2\cdots i_k 1}(\boldsymbol{x}_{i_1 i_2\cdots i_k 1}), y_{i_1 i_2\cdots i_k 2}(\boldsymbol{x}_{i_1 i_2\cdots i_k 2}))}{P_{i_1 i_2\cdots i_k 1}(y_{i_1 i_2\cdots i_k 1}(\boldsymbol{x}_{i_1 i_2\cdots i_k 1}))\, P_{i_1 i_2\cdots i_k 2}(y_{i_1 i_2\cdots i_k 2}(\boldsymbol{x}_{i_1 i_2\cdots i_k 2}))}\right] \\ &\quad \times \left[\prod_{i_1,i_2,\cdots,i_k=1}^{2} P_{i_1 i_2\cdots i_n}(\boldsymbol{x}_{i_1 i_2\cdots i_n})\right] \end{aligned} \tag{2.24}$$

where we have rearranged the terms to collect together the factors that each node-pair $(i_1 i_2 \cdots i_k 1, i_1 i_2 \cdots i_k 2)$ contributes.

Although we have concentrated on deriving $Q_{mem}(\boldsymbol{x})$ for a binary tree, the principle of the derivation carries over unchanged to arbitrary tree structures, and Equation 2.24 may easily be generalised. In Appendix B we explain the relationship of the single layer version of Equation 2.24 to the random access memory network that is known as WISARD [2].

C. Diagrammatic notation

We now present the steps in the inductive derivation leading from Equation 2.18 to Equation 2.22 as a diagram in Figure 2. We use a triangle to represent a subtree, and we indicate its apex node, its associated Π or $\frac{\Pi}{Z}$ factor, and its dependence on $\boldsymbol{x}$. Figure 2a represents Equation 2.18, which is a pair of trees connected by the joint PDF of their apex nodes. By integrating over $\boldsymbol{x}_2$ we remove the right hand tree to obtain Figure 2b, which corresponds to Equation 2.19. We then explicitly display the two daughter nodes to obtain Figure 2c, which corresponds to Equation 2.20, although we have grouped the terms together slightly differently, for simplicity. This exposes one of the Lagrange functions which we determine explicitly to obtain Figure 2d, which corresponds to Equation 2.22. One cycle of the inductive proof is completed by noting the correspondence between Figure 2a and Figure 2d.

We represent Equation 2.24 in diagrammatic form in Figure 3. The tree structure represents the flow of the transformations of the original input data $\boldsymbol{x}$. Each square cornered rectangle represents the marginal PDF of the enclosed node-pair (i.e. one $P_{i_1 i_2\cdots i_n}$ term from the second factor in Equation 2.24). Each round cornered rectangle represents the normalised marginal PDF of the encosed node-pair (i.e. one $\frac{P_{i_1 i_2\cdots i_k 1, i_1 i_2\cdots i_k 2}}{P_{i_1 i_2\cdots i_k 1} P_{i_1 i_2\cdots i_k 2}}$ term from the first factor in Equation 2.24). Overall, we obtain Equation 2.24 as the product of the rectangles in Figure 3.

This notation makes it easy to generalise the result in Equation 2.24 in a purely diagrammatic fashion, by firstly constructing an arbitrary (i.e. not necessarily binary) tree-like transformation of the input data, and secondly using as maximum entropy constraints the marginal PDF of each set of sister nodes in the tree. This prescription permits many possible ACE structures, including those in which different constraints effectively operate between different layers of the hierarchy (by mapping one or more node values directly from layer to layer).

Each rectangle representing a marginal PDF in Figure 3 contributes to the maximum entropy estimate of the PDF of a cluster of nodes in the input data. Because of the tree structure, clusters at each length scale are built out of clusters at smaller length scales. Equation 2.24 tells us exactly how to incorporate into $Q_{mem}(\boldsymbol{x})$ any additional statistical properties that might be observed when forming larger clusters out of smaller clusters in this way.

Finally, Figure 3 suggests an informal derivation of Equation 2.24. Thus the expression for the maximum entropy estimate of the joint PDF of the input data $\boldsymbol{x}$ in Equation 2.18 can be viewed as the joint PDF of the pair of nodes at the top of the tree in Figure 3 *times* corrective Jacobian factors that compensate for effects of the many-to-one mapping that the input data undergoes before it reaches the top of the tree. The final maximum entropy expression in Equation 2.24 merely enumerates these corrective Jacobian factors explicitly in terms of marginal PDFs measured at various levels of the tree. This makes it clear that the maximum entropy method gives a result that is consistent with simple counting ar-

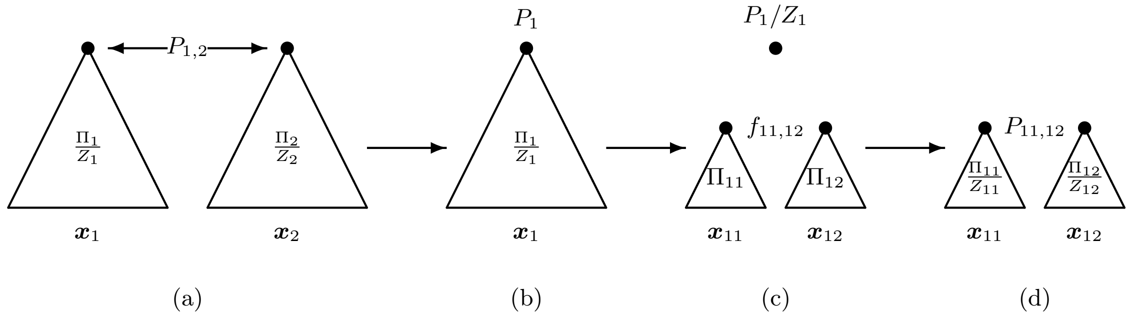


Figure 2: The individual steps of the inductive hierarchical maximum entropy derivation.

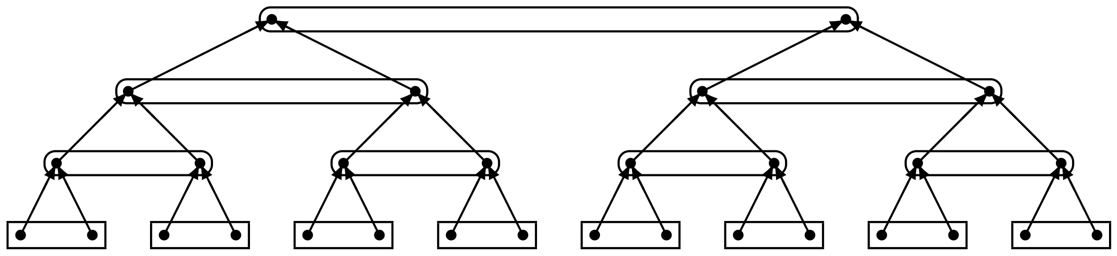


Figure 3: A diagrammatic representation of the hierarchical maximum entropy result.

guments, which could therefore be used in place of the rather involved maximum entropy derivation.

III. IMPLEMENTATION OF AN ANOMALY DETECTOR

Henceforth we shall refer to our hierarchical maximum entropy method as an adaptive cluster expansion (ACE). In this section we describe how to implement Equation 2.24 in software. We assume that the ACE transformation functions have already been optimised using the unsupervised network training algorithm that we describe in Appendix A and in [19], so the purpose of this section is to explain how to manipulate Equation 2.24 into a form that produces a useful output from the network. For concreteness, we produce an output in the form of an image that represents the degree to which each local patch of an input data is statistically anomalous, when compared to the global statistical properties of the input data.

A. Two-dimensional array of inputs

In Section II we represented ACE as if it were operating on a 1-dimensional arrays of inputs (e.g. time series). In practice this might indeed be the case, but in this paper we choose to study 2-dimensional arrays of inputs (e.g. images). There is no difficulty in applying ACE to an image, provided that we appropriately assign the leaf nodes to pixels of the image. In Figure 4 we show

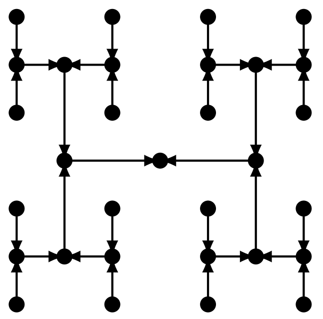


Figure 4: ACE connectivity for processing a 2-dimensional array of inputs.

the simplest possibility in which the image is alternately compressed in the north-south and east-west directions. A priori, the choice of whether to start with north-south or east-west compression is arbitrary, but if we knew, for instance, that the image had stronger short range correlations in the east-west direction than the north-south direction, then it would be better to compress east-west first of all. Note that in Figure 4 the topology of the tree is the same as in Figure 3, but the way in which the leaf nodes are identified with the data samples is different.

More generally, we could identify the leaf nodes of the tree with the image pixels in any way that we please, provided that no pixel is used more than once (to guarantee that the tree-like topology is preserved). The problem of optimising the identification of leaf nodes with pixels is extremely complicated, so we shall not pursue it in this paper.

B. Histograms

The maximum entropy PDF in Equation 2.24 is a product of (normalised) marginal PDFs. In a practical implementation of ACE the $y_{ijk\cdots}$ are discrete-valued quantities (for instance, integers in the interval $[0,255]$), and the $P_{ijk\cdots,i'j'k'\cdots}(y_{ijk\cdots},y_{i'j'k'\cdots})$ are probabilities (not PDFs). We estimate the $P_{ijk\cdots,i'j'k'\cdots}(y_{ijk\cdots},y_{i'j'k'\cdots})$ by constructing 2-dimensional histograms

$$P_{ijk\cdots,i'j'k'\cdots}(y_{ijk\cdots},y_{i'j'k'\cdots}) \simeq \frac{1}{N_{ijk\cdots,i'j'k'\cdots}} h_{ijk\cdots,i'j'k'\cdots}(y_{ijk\cdots},y_{i'j'k'\cdots}) \quad (3.1)$$

where $h_{ijk\cdots,i'j'k'\cdots}(y_{ijk\cdots},y_{i'j'k'\cdots})$ is the number of counts in the histogram bin $(y_{ijk\cdots},y_{i'j'k'\cdots})$, and N is the total number of histogram counts given by

$$N_{ijk\cdots,i'j'k'\cdots} = \sum_{y_{ijk\cdots}} \sum_{y_{i'j'k'\cdots}} h_{ijk\cdots,i'j'k'\cdots}(y_{ijk\cdots},y_{i'j'k'\cdots}) \quad (3.2)$$

Note that the estimate in Equation 3.1 suffers from Poisson noise due to the finite number of counts in each histogram bin.

In order to build up this estimate we first of all train the ACE transformation functions as explained in Appendix A. The histogram bins are then initialised to zero, and subsequently filled with counts by exposing the trained ACE to many examples of input vectors (possibly, the set used to train the transformation functions). Thus each vector is propagated up through the ACE-tree, and we then inspect each node-pair $(ijk\cdots,i'j'k'\cdots)$ for which a marginal probability needs to be estimated, and increment its corresponding histogram bin thus

$$h_{ijk\cdots,i'j'k'\cdots}(y_{ijk\cdots},y_{i'j'k'\cdots}) \longrightarrow h_{ijk\cdots,i'j'k'\cdots}(y_{ijk\cdots},y_{i'j'k'\cdots}) + 1 \quad (3.3)$$

When the training set has been exhausted, histogram bin $(ijk\cdots,i'j'k'\cdots)$ records the number of times that state $(y_{ijk\cdots},y_{i'j'k'\cdots})$ occurred.

A major disadvantage of using histograms is that they have a large number of adjustable parameters (i.e. the number of counts in each bin) that have to be determined by the training data, so they do not generalise very well. However, for the purpose of this paper, we do not need to resort to using more sophisticated ways of estimating PDFs.

C. Translation invariant processing

We wish to detect statistical anomalies in images which have otherwise spatially homogeneous statistics, such as textures. An invariance of the statistical properties of the true PDF $P(\boldsymbol{x})$ can be expressed as

$$P(\mathcal{G}\boldsymbol{x}) = P(\boldsymbol{x}) \quad (3.4)$$

where $\mathcal{G}$ is any element of the invariance group, which we shall assume to be the group of translations of the image pixels. In Equation 2.24 $Q_{mem}(\boldsymbol{x})$ does not respect translation invariance for two reasons. Firstly, we use transformations $y_{ijk\cdots}(\boldsymbol{x}_{ijk\cdots})$ that are explicitly translation variant, because the functional form depends on the $ijk\cdots$ indices. Secondly, we connect together these transformations in translation variant way, because the tree structure in Figure 1 and Figure 4 does not treat all of its leaf nodes equivalently. We shall therefore modify the cluster expansion procedure that we derived in Section II B to guarantee translation invariance. This will lead to a much improved maximum entropy estimate $Q_{mem}(\boldsymbol{x})$ of the true $P(\boldsymbol{x})$.

Firstly, use the same transformation function at each position within a single layer of ACE. Thus in Equation 2.24 we make the replacement

$$\begin{aligned} y_{i_1 i_2\cdots i_k 1}(\boldsymbol{x}_{i_1 i_2\cdots i_k 1}) &\longrightarrow y^k(\boldsymbol{x}_{i_1 i_2\cdots i_k 1}) \\ y_{i_1 i_2\cdots i_k 2}(\boldsymbol{x}_{i_1 i_2\cdots i_k 2}) &\longrightarrow y^k(\boldsymbol{x}_{i_1 i_2\cdots i_k 2}) \end{aligned} \quad (3.5)$$

where we indicate that the transformation is associated with the k-th layer of ACE by attaching a superscript k to each function. This yields

$$\begin{aligned} Q_{mem}(\boldsymbol{x}) \;=\; & \left[\prod_{k=0}^{n-2} \prod_{i_1,i_2,\cdots,i_k=1}^{2} \frac{P_{i_1 i_2\cdots i_k 1, i_1 i_2\cdots i_k 2}(y^k(\boldsymbol{x}_{i_1 i_2\cdots i_k 1}), y^k(\boldsymbol{x}_{i_1 i_2\cdots i_k 2}))}{P_{i_1 i_2\cdots i_k 1}(y^k(\boldsymbol{x}_{i_1 i_2\cdots i_k 1}))\, P_{i_1 i_2\cdots i_k 2}(y^k(\boldsymbol{x}_{i_1 i_2\cdots i_k 2}))}\right] \\ & \times \left[\prod_{i_1,i_2,\cdots,i_k=1}^{2} P_{i_1 i_2\cdots i_n}(\boldsymbol{x}_{i_1 i_2\cdots i_n})\right] \end{aligned} \quad (3.6)$$

Equation 3.6 guarantees translation invariance (in the sense of a "single-instruction-multiple-data" computer) of the processing that occurs when the input data is propagated upwards through the overlapping trees.

Secondly, assume that Equation 3.4 holds for all image translations, so that the marginal PDFs are independent of position. We may make this explicit in our notation by making the following replacement in Equation 3.6

$$\begin{aligned}\frac{P_{i_1 i_2 \cdots i_k 1, i_1 i_2 \cdots i_k 2}(\cdot)}{P_{i_1 i_2 \cdots i_k 1}(\cdot)\, P_{i_1 i_2 \cdots i_k 2}(\cdot)} &\longrightarrow \frac{P^k_{1,2}(\cdot)}{P^k_1(\cdot)\, P^k_2(\cdot)} \\ P_{i_1 i_2 \cdots i_n}(\cdot) &\longrightarrow P^{n-1}(\cdot)\end{aligned} \tag{3.7}$$

where we use the same superscript notation as in Equation 3.5. This yields

$$\begin{aligned}Q_{mem}(\boldsymbol{x}) &= \left[\prod_{k=0}^{n-2} \prod_{i_1, i_2, \cdots, i_k=1}^{2} \frac{P^k_{1,2}(y^k(\boldsymbol{x}_{i_1 i_2 \cdots i_k 1}), y^k(\boldsymbol{x}_{i_1 i_2 \cdots i_k 2}))}{P^k_1(y^k(\boldsymbol{x}_{i_1 i_2 \cdots i_k 1}))], P^k_2(y^k(\boldsymbol{x}_{i_1 i_2 \cdots i_k 2}))}\right] \\ &\times \left[\prod_{i_1, i_2, \cdots, i_k=1}^{2} P^{n-1}(\boldsymbol{x}_{i_1 i_2 \cdots i_n})\right]\end{aligned} \tag{3.8}$$

Equation 3.8 guarantees not only translation invariance of the transformations that propagate the data through the tree, but also translation invariance of the marginal PDFs of $P(\boldsymbol{x})$ that are used to construct $Q_{mem}(\boldsymbol{x})$.

Both of the simplifications in Equation 3.5 and Equation 3.7 reduce the total number of unknowns that have to be determined. For a given amount of training data we can thus construct a better maximum entropy estimate $Q_{mem}(\boldsymbol{x})$ of the true P(x). The transformation functions may be optimised better, and the histogram bins have a reduced Poisson noise.

We usually apply ACE to such large input arrays that it is not appropriate to build a single binary tree whose leaf nodes encompass the entire input array. Instead, we divide the input array (which we shall assume is a $2^M \times 2^M$ array of image pixels) into a set of contiguous $2^{m_1} \times 2^{m_2}$ arrays, each of which we analyse using Equation 3.8. There are no constraint functions to measure the mutual dependencies between these subarrays, so the maximum entropy joint PDF of the set of subarrays is a product of terms of the form shown in Equation 3.8.

$$\begin{aligned}\log(Q_{mem}(\boldsymbol{x})) &= \sum_{k=0}^{n-2} \sum_{a_1=1}^{2^{M-m_1}} \sum_{a_2=1}^{2^{M-m_2}} \sum_{i_1, i_2, \cdots, i_k=1}^{2} \log\left(\frac{P^k_{1,2}(y^k(\boldsymbol{x}^{a_1,a_2}_{i_1 i_2 \cdots i_k 1}), y^k(\boldsymbol{x}^{a_1,a_2}_{i_1 i_2 \cdots i_k 2}))}{P^k_1(y^k(\boldsymbol{x}^{a_1,a_2}_{i_1 i_2 \cdots i_k 1}))\, P^k_2(y^k(\boldsymbol{x}^{a_1,a_2}_{i_1 i_2 \cdots i_k 2}))}\right) \\ &+ \log\left(\sum_{a_1=1}^{2^{M-m_1}} \sum_{a_2=1}^{2^{M-m_2}} \sum_{i_1, i_2, \cdots, i_k=1}^{2} \log\left(P^{n-1}(\boldsymbol{x}^{a_1,a_2}_{i_1 i_2 \cdots i_n})\right)\right)\end{aligned} \tag{3.9}$$

The summation over (a_1, a_2) ranges over the $2^{2M-m_1-m_2}$ contiguous subarrays in the overall $2^M \times 2^M$ array, and the a_1, a_2 superscript on each $\boldsymbol{x}_{ijk\cdots}$ vector indicates that it belongs to subarray (a_1, a_2). Note that we have transformed $Q_{mem}(\boldsymbol{x}) \longrightarrow \log(Q_{mem}(\boldsymbol{x}))$ for convenience.

The final step in constructing a fully translation invariant PDF is to modify the sum over subarrays so that it includes all possible placements of the $2^{m_1} \times 2^{m_2}$ subarray within the overall $2^M \times 2^M$ array. There are $2^{2M-m_1-m_2}$ possible positions when the placement of the subarray is restricted as in Equation 3.9, whereas there are $(2^M - 2^{m_1} + 1)(2^M - 2^{m_2} + 1)$ possible positions when all placements of the subarray are permitted. We therefore make the replacement

$$\begin{aligned}\sum_{a_1=1}^{2^{M-m_1}} \sum_{a_2=1}^{2^{M-m_2}} (\cdot) &\longrightarrow \frac{2^{2M-m_1-m_2}}{(2^M - 2^{m_1} + 1)\,(2^M - 2^{m_2} + 1)} \sum_{p_1=1}^{2^M - 2^{m_1}+1} \sum_{p_2=1}^{2^M - 2^{m_2}+1} (\cdot) \\ &\simeq 2^{-m_1-m_2} \sum_{p_1=1}^{2^M} \sum_{p_2=1}^{2^M} (\cdot)\end{aligned} \tag{3.10}$$

in Equation 3.9, where (p_1, p_2) is the coordinate of the pixel in the top left hand corner of the $2^{m_1} \times 2^{m_2}$ subarray. If we ignore edge effects, then we may use the approximation in the final line of Equation 3.10, which is the average of $2^{m_1+m_2}$ separate contributions of the form shown in Equation 3.9. Equation 3.10 effectively replaces the original maximum entropy PDF $Q_{mem}(\boldsymbol{x})$ by the geometric mean of a set of maximum entropy PDFs. This averaging reduces the problems caused by Poisson noise on the histogram bin contents to yield a greatly improved maximum entropy PDF estimate.

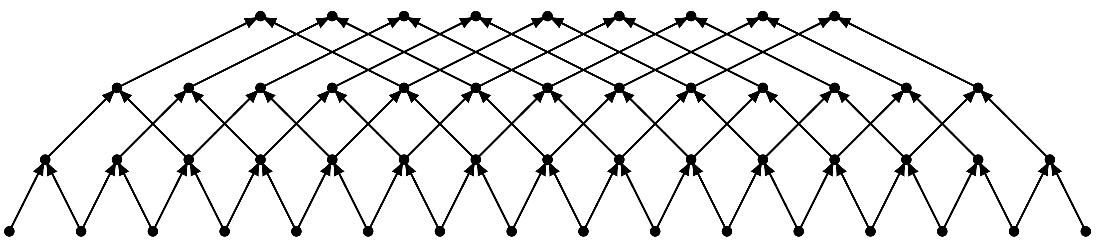


Figure 5: Connectivity for multiple overlapping binary trees.

In practice, we would implement each layer of ACE as a frame store, and the transformation between each pair of adjacent layers as a look-up table. The translation invariant ACE that we derived in Equation 3.9 (with the replacement given in Equation 3.10) may be implemented using the connectivity shown in Figure 5. Ignoring edge effects, we may write Equation 3.9 symbolically as

$$\log(Q_{mem}(\boldsymbol{x})) \simeq \sum_{k=0}^{n-2} \frac{1}{2^{n-k}} \sum \log\left(\frac{P_{1,2}^k}{P_1^k\, P_2^k}\right) + \frac{1}{2}\sum \log(P^{n-1}) \tag{3.11}$$

where the inner summations range over all positions within a single layer of Figure 5. We omit all of the functional dependencies, because they are easy to obtain from Figure 3. Each $\frac{P_{1,2}^k}{P_1^k P_2^k}$ term is represented by a rectangle with rounded corners in Figure 3, and each P^{n-1} term is represented by a rectangle with square corners in Figure 3. We have not drawn these rectangles in Figure 5 because they would overlap, and thus confuse the diagram.

D. Forming a probability image

Equation 3.11 is the fundamental result that we use to construct useful image processing schemes. However, it would not be very useful simply to calculate the value of $\log(Q_{mem})$ as a single global measure of the logarithmic probability associated with an image. We choose instead to break up Equation 3.11 into smaller pieces, and to examine their contribution to the overall $\log(Q_{mem})$. In effect, we look at how $\log(Q_{mem})$ is built up from the information in each layer of ACE, which in turn we break down into contributions from different areas of the image.

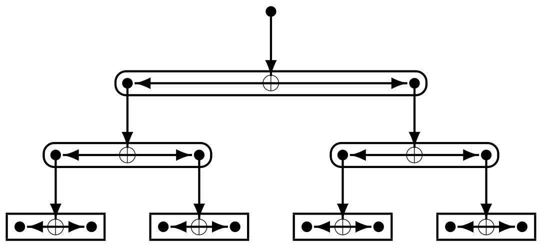


Figure 6: Backpropagation scheme for constructing a probability image.

In order to ensure that our decomposition of $\log(Q_{mem})$ can be easily computed, we use the backpropagation scheme shown in Figure 6 to control the data flow through a translation invariant network of an identical connectivity to the one shown in Figure 5. Each node of this backpropagation network records a logarithmic probability, and is cleared to zero before starting the backpropagation computations. The rectangles in Figure 6 represent exactly the same logarithmic probability terms that appeared in Figure 3, which we now use as sources of logarithmic probability that we inject into the backpropagating data flow.

The detailed operation of Equation 3.6 is as follows. Each addition symbol takes as input a contribution recorded at a node in the next layer above, adds its own logarithmic probability source $\log\left(\frac{P_{1,2}^k}{P_1^k P_2^k}\right)$, scales the result by $\frac{1}{4}$, and it finally adds a copy of this result to the value stored at each of its own pair of associated nodes, as shown. The values that accumulate at the leaf nodes represent various contributions to the sum in Equation 3.11. If the translation invariant version of Figure 6 is applied to the translation invariant network shown in Figure 5, then the sum of the values that accumulate at the leaf nodes reproduces Equation 3.11 precisely.

This method of computing $\log(Q_{mem})$ might seem to be circuitous, but it has the great advantage of both being computationally cheap and forming an image-like representation of $\log(Q_{mem})$, which we call a "probability image". Each $\log\left(\frac{P_{1,2}^k}{P_1^k P_2^k}\right)$ term in Equation 3.11 will contribute equally to 2^{n-k} pixels in the probability image. These pixels will be arranged as either a square or a 2-to-1 aspect ratio rectangle according to whether there is an odd number or even number of backpropagation steps from the k-th layer to the leaf nodes. The probability image is therefore a superposition of square and rectangular tiles of logarithmic probability. Each tile corresponds to a node of the network shown in Figure 5.

It is useful to display as an image the contributions of a single layer of the network to the probability image, because different layers contribute to the structure of $\log(Q_{mem})$ at different length scales. This image may be displayed in the conventional way, with small probabilities mapped to black, large probabilities mapped to white, and intervening probabilities mapped to shades of grey, in which case we call it a "probability image". It is also useful to invert the grey scale so that small probabilities map to black, in which case we call it an "anomaly image", because regions which have statistical properties that occur infrequently show up as bright peaks in the image. We find that the use of probability images and/or anomaly images is an extremely effective way of visually interpreting $\log(Q_{mem})$ in Equation 3.11.

E. Modular implementation

For completeness we now present a brief description of a complete system for producing probability and/or anomaly images. This system consists of two tightly coupled subsystems – an ACE subsystem for decomposing the image data, and a probability image subsystem for forming the output image. Figure 7 combines in one diagram all of the results that we have discussed so far. The upper part of Figure 7 is a pure translation invariant ACE subsystem, whereas the lower half is a backpropagating probability image subsystem operating as shown in Figure 6. The backpropagating subsystem takes input information from various layers of ACE, as shown. Modules "I" are framestores that record the various transformed images. Modules "M" are look-up tables that record the inter-layer mappings. Modules "T" represent the training algorithm that we explain in Appendix A, which we enclose in a dashed box because the "T" modules are switched out of the circuit once the mappings "M" have been determined. Modules "H" are accumulators that record the 2-dimensional histograms, and then regularise and normalise them appropriately. Modules "P" are framestores that record the various backpropagated probability images. Modules "log" are look-up tables (in fact only one such table is needed) that implement a logarithm function. Modules "$\oplus$" and "$\otimes$" perform the addition and scaling operations that we discussed earlier in connection with Figure 6. "N" is scaling factor (which is $\frac{1}{4}$ if we wish to reproduce the result in Equation 3.11). The lines that are annotated "G" represent a ganging together of the (pointers to) pixels in adjacent layers of the ACE subsystem and in the probability image subsystem. These ensure that the entire system works in lockstep, as required.

The simplest mode of operation of this system can be broken down into three stages Firstly, train each layer (from left to right) of the ACE subsystem on a training image. Secondly, propagate a test image (from left to right) through the layers of ACE. Finally, construct a probability image by backpropagating (from right to left) contributions from the various layers of ACE. Furthermore, it is useful to display separately the probability (or anomaly) images that emerge from each layer of ACE, as we shall see in Section IV.

There is a variety of methods of optimising "T", and hence "M". The method that we describe in Appendix A trains each layer in sequence, which takes 2.3 second per layer (using a VAXstation 3100, and assuming 6 bits per pixel), which gives a full training time of 20 seconds for the 8 layer network that we use in our numerical simulations. We do not make use of more sophisticated schemes in which different layers are simultaneously trained, whilst communicating information with each other to improve the global performance of ACE.

F. Relationship to co-occurrence matrix methods

Both the basic maximum entropy PDF $Q_{mem}(\boldsymbol{x})$ in Equation 2.24, and the translation invariant version of $\log(Q_{mem}(\boldsymbol{x}))$ in Equation 3.11 that we implement in practice, depend on various PDFs that are measured in an ACE-tree. The second term of Equation 3.11 may be written as

$$Q_{mem}(\boldsymbol{x}) = \sqrt{\prod P^{n-1}} \qquad (3.12)$$

Each P^{n-1} factor is the spatial average of the marginal PDF of pairs of adjacent pixel values, assuming that we use the identification of leaf nodes with pixels that we show in Figure 4. The square root in Equation 3.12 compensates for the fact that the product of P^{n-1} factors generates the product of two maximum entropy PDFs shifted by one pixel relative to each other.

By using Equation 3.1 we may approximate Equation 3.12 as a product of histograms. In this case each histogram is the spatial average of the co-occurrence matrix of pairs of adjacent pixel values, as commonly used in image processing [5]. Thus we may use conventional co-occurrence matrix methods to construct a simple form of maximum entropy PDF, which corresponds to using only one layer of ACE.

This co-occurrence matrix result can be generalised, using Equation 2.24 or Equation 3.11, to model higher order statistical behaviour. Although these results depend on co-occurrence matrices measured at various places in the ACE-tree, the contributions which do not depend directly on the input data (i.e. the first term of Equation 3.11) actually model higher order statistics of the input data. This is because the value $y_{ijk\cdots}$ that emerges from node $ijk\cdots$ of the ACE-tree depends on $\boldsymbol{x}_{ijk\cdots}$, so the joint PDF $P_{ijk\cdots1,ijk\cdots2}(y_{ijk\cdots1}, y_{ijk\cdots2})$ depends on the statistics of the pair $(\boldsymbol{x}_{ijk\cdots1}, \boldsymbol{x}_{ijk\cdots2})$. Thus ACE is a very convenient way of combining together the various

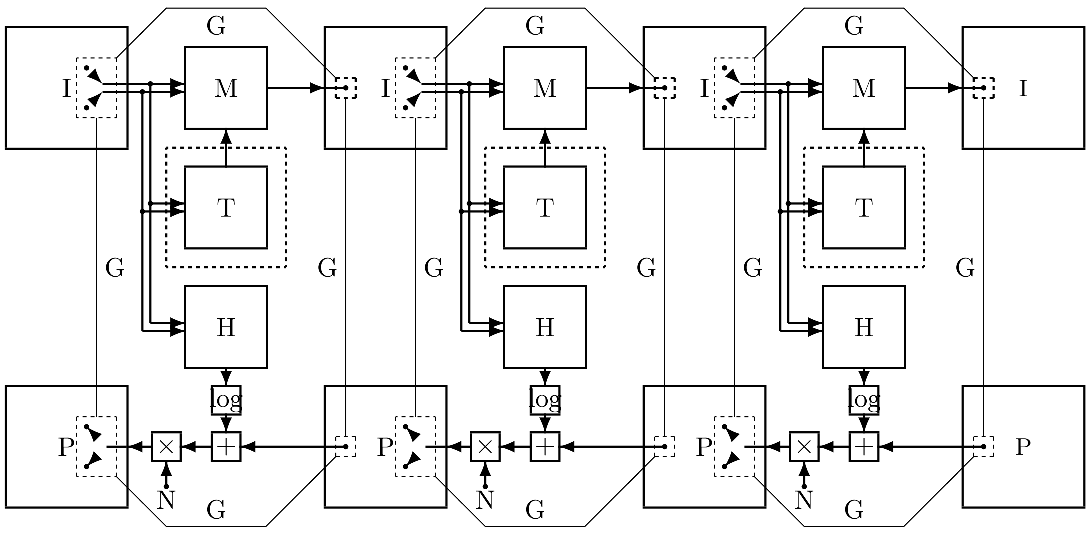


Figure 7: Three layer translation invariant ACE system.

orders of statistical information that are contained in co-occurrence matrices at various places in the ACE-tree, as shown in Figure 3.

IV. NUMERICAL RESULTS

In this section we explain the finer details of how to implement Figure 7 in software, and we present the results of applying the system to four 256 × 256 images of textiles taken from the Brodatz texture set [3].

A. Experimental procedure

We compensated for some of the effects of non-uniform illumination by adding to each image a grey scale wedge whose gradient was chosen in such a way as to remove the linear component of the non-uniformity. Not only does this improve the translation invariance of the image statistics, but it also improves the quality of the hierarchical coding of the image, because we reduce the need to develop redundant codes which differ only in their overall grey level.

Throughout our experiments we generate optimal inter-layer mappings using the training methods that we explain in Appendix A. These are known as topographic mappings in the neural network literature, and we showed in [18] why they are appropriate for building multistage vector quantisers. We choose to compress the image in alternate directions using the following sequence: north/south, east/west, north/south, east/west, etc. This compression sequence leads to the following sequence of rectangular image regions that influence the state of each pixel in each stage of ACE: 1×2, 2×2, 2×4, 4×4, etc, using (east/west, north/south) coordinates. In all of our experiments we use an 8 stage ACE.

The number of bits per pixel that we use in each layer of ACE determines the quality of the hierarchical vector quantisation that emerges. Increasing the number of bits improves the quality of the vector quantisation but increases the training time: we need to compromise between these two conflicting requirements. In our work on simple Brodatz texture images we have found that 6-8 bits per pixel is sufficient.

It is important to note that for a given number of bits (after compression) there is an upper limit on the allowed entropy that the input data can have. This problem becomes more severe the greater the data compression factor (i.e. the further we progress through the layers of ACE). For instance, if the input image is very noisy then 6-8 bits will be sufficient only to give good vector quantisation performance in the first few layers of ACE. This problem arises because ACE does not have much prior knowledge of the statistical properties of the input data, so each node of ACE encodes its input without assuming a prior model. A prior model would allow us to reduce the bit rate. This is a fundamental limitation to the capabilities of the current version of ACE.

The choice of the size of the 2-dimensional histogram bins is also important. A property of the topographic mappings that we use to to connect the layers of ACE is that adjacent histogram bins derive from input vectors that are close to each other (in the Euclidean sense), so it is sensible to rebin the histogram by combining together adjacent bins. Thus we control the histogram bin size by truncating the low order bits of each binary vector that represents a pixel value. If we do not truncate any bits, then the 2-dimensional histogram faithfully records the number of times that a pair of pixel values has occured. However, if we truncate b low order bits of each pixel value then effectively we sum together the histogram bins in groups of 2^{2b} $(= 2^b \times 2^b)$ adjacent bins, which smooths the histogram. The more smoothing that we impose the less Poisson noise the histogram suffers. However, as we smooth the histogram we run the danger of smoothing away significant structure that might usefully be used to characterise the input image: so we need to make a compromise. In our Brodatz texture work we use only 4-6 bits of each pixel value to generate the histograms in each stage of ACE. Note that we use more bits for vector quantisation than for histogramming because the vector quantisation needs to be good enough to preserve information for encoding by later layers of the hierarchy, whereas the histogramming information is not passed to later layers.

In Equation 3.11 we need to estimate the logarithm of various probabilities from the histograms. We do this in two stages. Firstly, we regularise the histograms by placing a lower bound on the permitted number of counts. One possible prescription is to ensure that each histogram bin has a number of counts at least as large as the average number of counts in all the histogram bins (as determined before regularising the histogram). Thus

$$h_{ijk\cdots,i'j'k'\cdots}(y_{ijk\cdots},y_{i'j'k'\cdots}) \longrightarrow \begin{cases} h_{ijk\cdots,i'j'k'\cdots}(y_{ijk\cdots},y_{i'j'k'\cdots}) & h \geq \langle h \rangle \\ \langle h_{ijk\cdots,i'j'k'\cdots}(y_{ijk\cdots},y_{i'j'k'\cdots}) \rangle & h < \langle h \rangle \end{cases} \tag{4.1}$$

where the angle brackets $\langle \cdots \rangle$ denote an average over histogram bins, rounded up to the next largest integer to avoid setting histogram bins to zero. Secondly, we estimate the probabilities $P_{ijk\cdots,i'j'k'\cdots}(y_{ijk\cdots},y_{i'j'k'\cdots})$ by inserting the regularised histograms into Equation 3.1. We use a marginalised version of Equation 3.1 to estimate the marginal probabilities $P_{ijk\cdots}(y_{ijk\cdots})$. Finally, we compute the logarithmic probabilities in Equation 3.11 by using a table of logarithms of integers, up to the maximum possible number of counts that could occur in a histogram bin – it suffices to tabulate logarithms up to $\log(N)$.

The prescription in Equation 4.1 is crude but effective. We could improve the performance by introducing prior knowledge of the statistical properties of the input data. Our histogram smoothing prescription already implicity makes use of prior knowledge of the properties of the Posson noise process that affects the histogram counts, and prior knowledge of the fact that adjacent histogram bins correspond to similar input vectors. Additional prior knowledge would further enhance the performance, especially in cases where there is a limited amount of training data (such as small images, or small segments of larger images).

A pitfall that must be avoided is using histogram bins that are too small when one intends to train ACE on one image and then use a different image to generate a probability image. Effectively, the large number of small bins records the details of the statistical fluctuations of the training image (as particular realisations of a Poisson noise process in each bin), which thus acts as a detailed record of the structure in the training image. The histograms thus look very spiky, and in an extreme case there may be a counts recorded in only a few bins with zeros in all of the remaining bins. If this situation occurs then the training image records a large $\log(Q_{mem}(\boldsymbol{x}))$, whereas a test image having the same statistical properties records a small $\log(Q_{mem}(\boldsymbol{x}))$. Effectively, the spikes in the training and test image histograms are not coincident. This problem can be solved by choosing a large enough histogram bin size.

Finally, we display the logarithmic probability image as follows. We determine the range of pixel values that occurs in the image, and we translate and scale this into the range $[0, 255]$. This ensures that the smallest logarithmic probability appears as black, and the largest logarithmic probability appears as white, and all other values are linearly scaled onto intermediate levels of grey. This prescription has its dangers because each probability image determines its own special scaling, so one should be careful when comparing two different probability images. It can also be adversely affected by pixel value outliers arising from Poisson noise effects, where an extreme value of a single pixel could affect the way in which the whole of an image is displayed. However, we find that the overlapping tree prescription in Figure 5 together with the backpropagation prescription in Figure 6, causes enough effective averaging together of the histogram bins that we do not encounter problems with pixel value outliers.

In all of the images that we present below, we compensate for the uneven illumination by introducing a grey scale wedge as we explained earlier, we use 8 bits per pixel for vector quantisation, we use 6 bits per pixel for histogramming, and we invert the $[0, 255]$ scale to produce an anomaly image, in which a white pixel indicates a small (rather than a large) logarithmic probability.

B. Texture 1

In Figure 8 we show the first Brodatz texture image that we use in our experiments. The image is slightly unevenly illuminated and has a fairly low contrast, but nevertheless its statistical properties are almost translation invariant.

In Figure 9 we show the anomaly images that derive from Figure 8. Note how the anomaly images be-

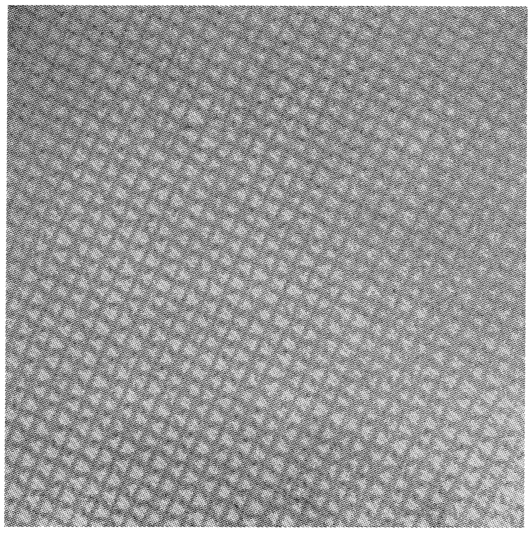


Figure 8: 256 × 256 image of Brodatz fabric number 1.

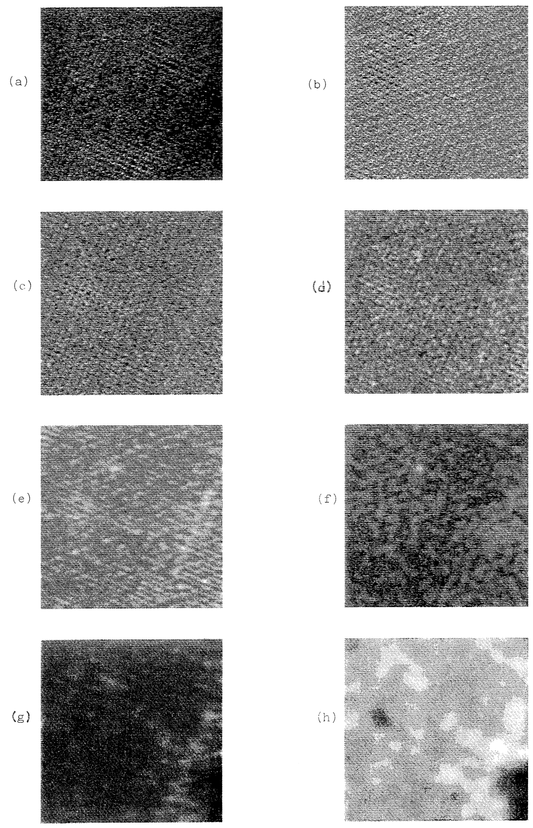


Figure 9: 256×256 anomaly images of Brodatz fabric number 1.

come smoother as we progress from Figure 9a to Figure 9h, due to the increasing amount of averaging that occurs amongst the overlapping backpropagated rectangular tiles that build up each image.

Figure 9e and especially Figure 9f reveal a highly localised anomaly in the original image. Figure 9f corresponds to a length scale of 8×8 pixels, which is the approximate size of the fault that is about $\frac{1}{4}$ of the way down and slightly to the left of centre of Figure 8. The fault does not show up clearly on the other figures in Figure 9 because their characteristic length scales are either too short or too long to be sensitive to the fault.

There is a major feature in the bottom right hand corner of Figure 9h, where the anomaly image is darker than average, indicating that the corresponding part of the original image has a higher than average probability. This is a different type of anomaly to the sort that we have envisaged so far – it occurs because the corresponding part of original image happens to explore only a high probability part of the space that is explored by the whole image. This part of the anomaly image is surrounded by a brighter than average border, which indicates a conventional anomalous region.

From Figure 9 we conclude that ACE can easily pick out localised faults in highly ordered textures.

C. Texture 2

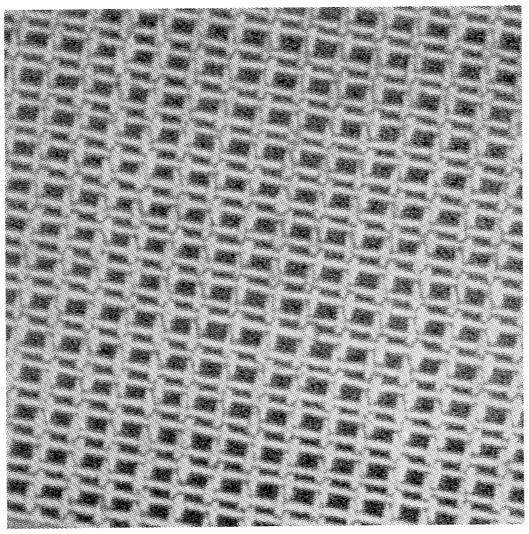


Figure 10: 256 × 256 image of Brodatz fabric number 2.

In Figure 10 we show the second Brodatz texture image that we use in our experiments. The image has a high contrast and translation invariant statistical properties.

In Figure 11 we show the anomaly images that derive from Figure 10. The most interesting anomaly image is Figure 11f which shows several localised anomalies. About halfway down and to the left of centre of the image is an anomaly that corresponds to a dark spot on the thread in Figure 10. The brightest of the anomalies in the cluster just above the centre of the image corresponds to what appears to be a slightly torn thread in Figure 10. The other anomalies in this cluster are weaker, and correspond to slight distortions of the threads. There is another anomaly just below and to the right of the centre

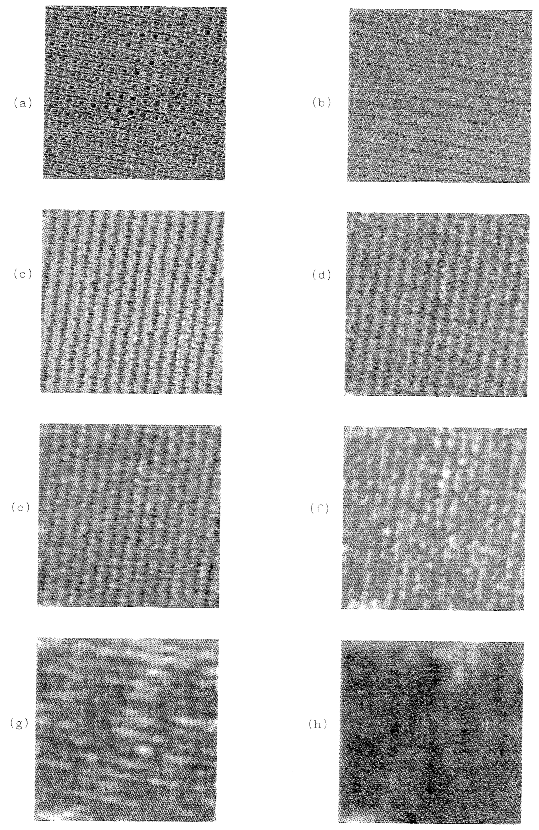


Figure 11: 256×256 anomaly images of Brodatz fabric number 2.

of Figure 11g, which corresponds to what appears to be another slightly torn thread in Figure 10. These anomalies all occur at, or around, a length scale of 8×8 pixels. Several of the anomaly images show an anomaly in the bottom left hand corner of the image, which corrsponds to a small uniform patch of fabric in Figure 10.

The results in Figure 11 corroborate the evidence in Figure 9 that ACE can be trained in an unsupervised fashion to pick out localised faults in highly ordered textures.

D. Texture 3

In Figure 12 we show the third Brodatz texture image that use in our experiments.The image has a very high contrast and statistical properties that are almost translation invariant. However the density of anomalies is much higher than in either Figure 8 or Figure 10.

In Figure 13 we show the anomaly images that derive from Figure 12. The most prominant anomaly is in Figure 13g, at a length scale of 8×16 pixels, which corresponds to region of Figure 12 that is just above and to the left of centre of the image. This region is anomalous because it is both distorted and has slightly thicker threads than elsewhere. The large distorted region in the bottom left hand corner of Figure 12 does not show up very clearly to the naked eye in Figure 13, but Figure 13f and Figure 13h have significant peaks in this region. There are also many other localised peaks in Figure 13 which can be traced back to corresponding faults in Figure 12.

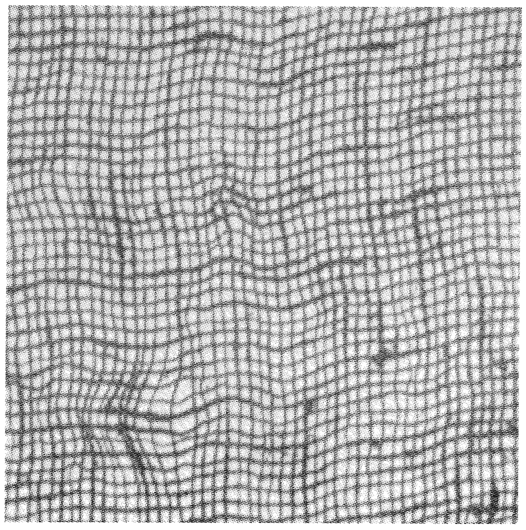


Figure 12: 256 × 256 image of Brodatz fabric number 3.

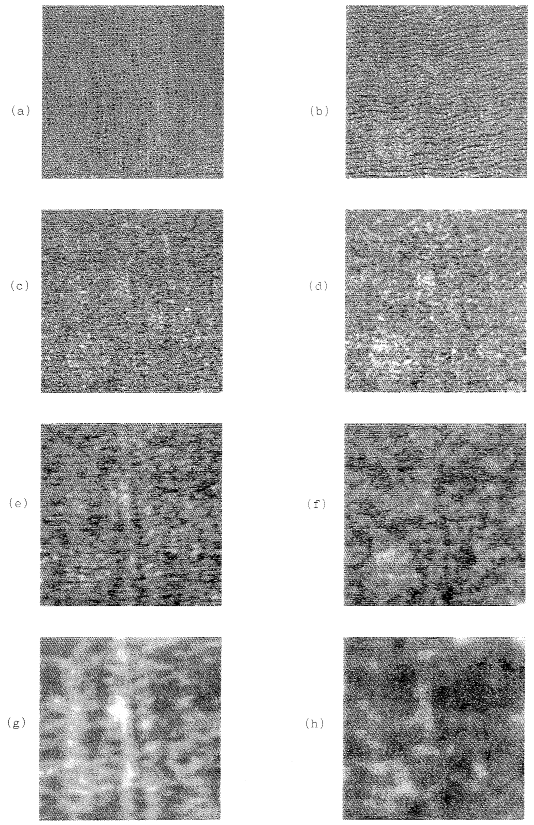


Figure 13: 256×256 anomaly images of Brodatz fabric number 3.

Comparing Figure 13 with Figure 9 and Figure 11 we conclude that the ability of ACE to pick out faults is degraded as the density of faults increases. This is because the faults themselves are part of the statistical properties that are extracted by ACE, and if a particular fault occurs often enough in the image then it is no longer deemed to be a fault.

E. Texture 4

In this section we present a slightly different type of experiment in which we train ACE on one image and test ACE on another image. To create the two images we start with a single 256×256 image of a Brodatz texture, which we divide into a left half and a right half. We then use the left half to build up the training image, and the right half to build up the test image.

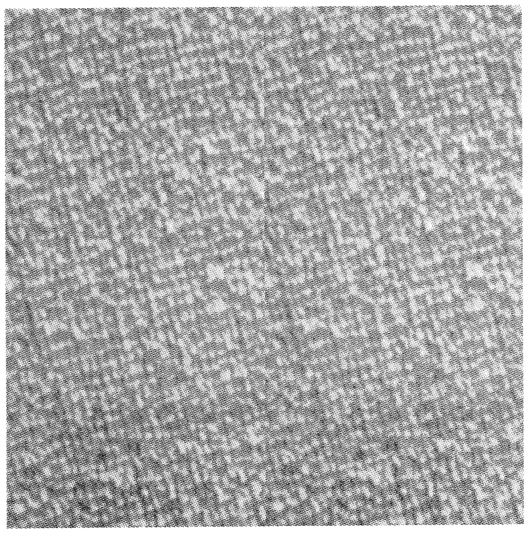


Figure 14: 256×256 image of Brodatz carpet for training.

In Figure 14 we show the training image which is a montage of two copies of the left hand half of a Brodatz texture image. Note that this montage contains only as much information as was present in the original half image from which it was constructed. In Figure 15 we show the test image which is a montage of two copies of the right hand half of a Brodatz texture image, and superimposed on that is a 64×64 patch which we generated by flipping the rows and columns of a copy of the top left hand corner of this image. This patch is a hand crafted anomaly. Note that in constructing these images we have scrupulously avoided the possibility that the training and test images could contain elements deriving from a common source.

In Figure 16 we show the anomaly images that derive from Figure 15 after having trained on Figure 14. Figure 16f shows the strongest response to the anomalous patch in the centre of the image, corresponding to anomaly detection on a length scale of 8×8 pixels.

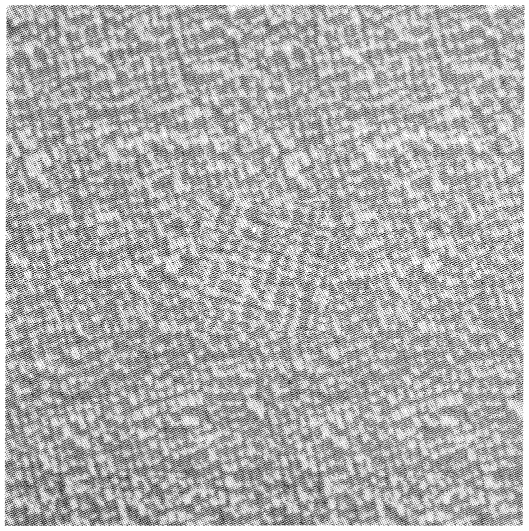


Figure 15: 256×256 image of Brodatz carpet for testing.

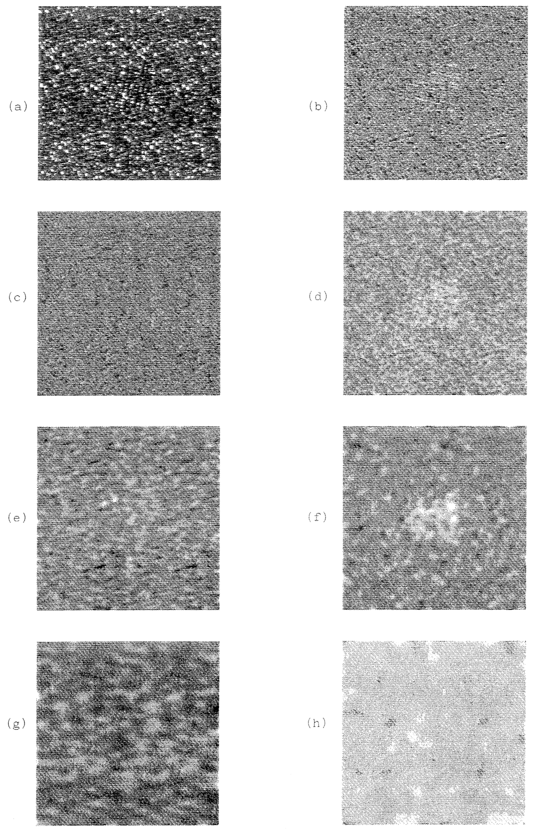


Figure 16: 256×256 anomaly images of Brodatz carpet.

V. CONCLUSIONS

Using maximum entropy methods, we have shown how to construct maximum entropy estimates of PDFs by using adaptive hierarchical transformation functions to record various marginal PDFs of the data, which we call an "Adaptive Cluster Expansion" (ACE). This method is a member of the same family as the trainable MRF known as the Boltzmann Machine, but it uses sophisticated transformations of the input data rather than hidden variables to characterise the high order statistical properties of the training set. The simulations in this paper use hierarchical topographic mappings to build these transformations, but this is a convenience, not a necessity.

We have also shown how to extend ACE so that it can be applied to translation invariant image processing, such as the detection of statistical anomalies in otherwise statistically homogeneous textures. Our methods show great promise, not only because they are amenable to a full theoretical analysis leading to closed-form maximum entropy solutions, but also because they lead directly to a modular system design which can locate anomalies in textures.

We have presented several examples where ACE successfully detects anomalous regions in otherwise statistically homogeneous textures. In all cases ACE adaptively extracts the global statistics of an image at various length scales during the unsupervised training, which takes 20 seconds (on a VAXStation 3100) for the 8 layer ACE network that we applied to this problem. ACE then uses these statistics to form an output image that represents the probability that each local patch of the input image belongs to the ensemble of patches presented during training. We call this a "probability image".

Some possible applications of our results are as follows. Inspection of textiles: this relies on the assumed statistical homogeneity of an unflawed piece of textile, so that faults show up as anomalies, which we have demonstrated successfully in this paper. Detection of targets in noisy background clutter in radar images: this is basically a noisy version of the textile inspection problem, which goes somewhat beyond what we have presented in this paper, because it needs to address the problem of the noise entropy saturating ACE. Texture segmentation: this is an ambitious goal which requires much further analysis in order to derive a computationally cheap method of handling multiple simultaneous textures.

Appendix A: Vector quantisation

In this appendix we summarise the hierarchical vector quantisation method that we presented in detail in [19]. In this paper we use this technique to optimise the inter-layer mappings in Figure 7. We have applied this technique elsewhere to image compression [20], and multilayer self-organising neural networks [17, 18, 22, 23].

1. Standard vector quantisation

This subsection contains those details of the theory of standard vector quantisation that one needs to understand before proceeding to the modified vector quantisation scheme that we present in Section A 2.

The problem is to form a coding y of a vector $\boldsymbol{x}$ in such a way that a good estimate $\boldsymbol{x}'$ of $\boldsymbol{x}$ can be constructed from knowledge of y alone. The sketch derivation in this section is presented in greater detail in [18]. Thus a vector quantiser is constructed by minimising a Euclidean distortion D_1 with respect to the choice of coding function $y(\boldsymbol{x})$ and decoding function $\boldsymbol{x}'(y)$, where

$$D_1 = \int d\boldsymbol{x}\, P(\boldsymbol{x})\, \|\boldsymbol{x} - \boldsymbol{x}'(y(\boldsymbol{x}))\|^2 \qquad \text{(A1)}$$

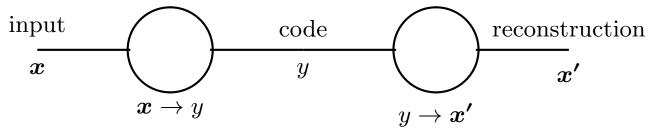


Figure 17: Encoding and decoding in a vector quantiser.

We may represent the encoding and decoding operations diagrammatically as shown in Figure 17. By functionally differentiating D_1 with respect to $y(\boldsymbol{x})$ and $\boldsymbol{x}'(y)$ we obtain

$$\frac{\delta D_1}{\delta y(\boldsymbol{x})} = P(\boldsymbol{x})\, \frac{\partial}{\partial y} \|\boldsymbol{x}'(y) - \boldsymbol{x}\|^2 \Big|_{y=y(\boldsymbol{x})} \qquad \text{(A2)}$$

$$\frac{\delta D_1}{\delta \boldsymbol{x}'(y)} = 2 \int d\boldsymbol{x}\, P(\boldsymbol{x})\, \delta(y - y(\boldsymbol{x}))\, (\boldsymbol{x}'(y) - \boldsymbol{x}) \qquad \text{(A3)}$$

Setting $\frac{\delta D_1}{\delta y(\boldsymbol{x})} = 0$ in Equation A2 yields the optimum encoding function

$$y(\boldsymbol{x}) = \arg \min_y \|\boldsymbol{x} - \boldsymbol{x}'(y(\boldsymbol{x}))\|^2 \qquad \text{(A4)}$$

which is called "nearest neighbour encoding". Setting $\frac{\delta D_1}{\delta \boldsymbol{x}'(y)} = 0$ in Equation A3 yields the optimum decoding function

$$\boldsymbol{x}'(y) = \frac{\int d\boldsymbol{x}\, P(\boldsymbol{x})\, \delta(y - y(\boldsymbol{x}))\, \boldsymbol{x}}{\int d\boldsymbol{x}\, P(\boldsymbol{x})\, \delta(y - y(\boldsymbol{x}))} \qquad \text{(A5)}$$

which is the update scheme derived in [10]. Alternatively, we may use an incremental scheme to optimise the decoding function by following the path of steepest descent which we may obtain from Equation A3 as

$$\delta \boldsymbol{x}'(y) = \epsilon\, \delta(y - y(\boldsymbol{x}))\, (\boldsymbol{x} - \boldsymbol{x}'(y)) \quad \text{where } 0 < \epsilon < 1. \qquad \text{(A6)}$$

An iterative optimisation scheme may be formed by alternately applying Equation A4 and then either Equation A5 or Equation A6. This scheme will alternately improve the encoding and decoding functions until a local minimum distortion is located. Alternating Equation A4 and Equation A5 is commonly called the "LBG" (after the authors of [10]) or "k-means" algorithm.

2. Noisy vector quantisation

This subsection contains the theoretical details of the optimisation of inter-layer mappings that we use in our numerical simulations in Section IV. Thus we generalise the results of Section A 1 to the case where the encoded version of the input vector is distorted by a noise process [4, 9, 19, 23].

Define a modified Euclidean distortion D_2 as

$$D_2 = \int d\boldsymbol{x}\, P(\boldsymbol{x}) \int dn\, \pi(n)\, \|\boldsymbol{x} - \boldsymbol{x}'(y(\boldsymbol{x}) + n)\|^2 \quad \text{(A7)}$$

We may represent the encoding and decoding operations together with the noise process diagrammatically as shown in Figure 18, which is a trivially modified version of Figure 17. By functionally differentiating D_2 with respect to $y(\boldsymbol{x})$ and $\boldsymbol{x}'(y)$ we obtain

$$\begin{aligned}\frac{\delta D_2}{\delta y(\boldsymbol{x})} &= P(\boldsymbol{x}) \int dn\, \pi(n) \frac{\partial}{\partial y} \|\boldsymbol{x}'(y) - \boldsymbol{x}\|^2 \Big|_{y=y(\boldsymbol{x})+n} & \text{(A8)}\\ \frac{\delta D_2}{\delta \boldsymbol{x}'(y)} &= 2 \int d\boldsymbol{x}\, P(\boldsymbol{x})\, \pi(y - y(\boldsymbol{x}))\, (\boldsymbol{x}'(y) - \boldsymbol{x}) & \text{(A9)}\end{aligned}$$

Equation A8 is a "smeared" version of Equation A2, so $\frac{\delta D_2}{\delta y(\boldsymbol{x})} = 0$ does not lead to nearest neighbour encoding because the distances to other code vectors have to be taken into account in order to minimise the damaging effect of the noise process. However, it is usually a good approximation to use the nearest neighbour encoding scheme shown in Equation A4. Setting $\frac{\delta D_2}{\delta \boldsymbol{x}'(y)} = 0$ in Equation A9 yields the optimum decoding function

$$\boldsymbol{x}'(y) = \frac{\int d\boldsymbol{x}\, P(\boldsymbol{x})\, \pi(y - y(\boldsymbol{x}))\, \boldsymbol{x}}{\int d\boldsymbol{x}\, P(\boldsymbol{x})\, \pi(y - y(\boldsymbol{x}))} \quad \text{(A10)}$$

which should be compared with Equation A5. Alternatively, we may obtain a steepest descent scheme in the form

$$\delta \boldsymbol{x}'(y) = \epsilon\, \pi(y - y(\boldsymbol{x}))\, (\boldsymbol{x} - \boldsymbol{x}'(y)) \ \text{ where } 0 < \epsilon < 1 \quad \text{(A11)}$$

which should be compared with Equation A6.

As in Section A 1, iterative optimisation schemes can be constructed in which we alternate the optimisation of the coding and decoding functions. Alternating Equation A4 (which approximately solves $\frac{\delta D_2}{\delta \boldsymbol{x}'(y)} = 0$) and Equation A11 yields the standard topographic mapping training algorithm [8], which is widely used in various forms in neural network simulations.

3. Hierarchical vector quantisation

In Figure 19 we show the simplest type of hierarchical vector quantiser. It consists of an inner quantiser contained in the dashed box, surrounded by a pair of outer quantisers.

If the part of the diagram contained in the dashed box were removed and direct connections made so that ${y_1}' = y_1$ and ${y_2}' = y_2$, then Figure 19 would reduce to a pair of independent vector quantisers of the type shown in Figure 17. The dashed box contains a vector quantiser which encodes (y_1, y_2) to produce a code which it subsequently decodes to obtain $({y_1}', {y_2}')$.

From the point of view of y_1 the effect of being passed through the inner quantiser is to modify y_1 thus $y_1 \longrightarrow {y_1}'$. A similar argument applies to $y_2 \longrightarrow {y_2}'$. The actual distortions ${y_1}' - y_1$ and ${y_2}' - y_2$ will be correlated in practice, but we shall model them as if they were independent processes, and thus reduce Figure 19 to two independent vector quantisers of the type shown in Figure 18.

This procedure can be extended to a hierarchical vector quantiser with any number of levels of nesting. From the point of view of the quantisers at any level, we shall model the effect of the quantisers inwards from that level as independent distortion processes. It turns out not to be critically important what precise distortion model one uses, provided that it approximately represents the overall scale of the distortion due to quantisation.

In [19] we presented in detail a phenomenological distortion model that we used to obtain an efficient training procedure for topographic mappings and their application to hierarchical vector quantisers. Alternatively, the standard topographic mapping training procedure in [8] could be used, but this is a rather inefficient algorithm. The basic training procedure may be obtained from Equation A11 as

1. Select a training vector $\boldsymbol{x}$ at random from the training set.
2. Encode $\boldsymbol{x}$ to produce y ($= y(\boldsymbol{x})$).
3. For all y' do the following:
   (a) Determine the corresponding code vector $\boldsymbol{x}'(y')$.
   (b) Move the code vector $\boldsymbol{x}'(y')$ directly towards the input vector $\boldsymbol{x}$ by a distance $\epsilon\, \pi(y - y(\boldsymbol{x}))\, |\boldsymbol{x} - \boldsymbol{x}'(y)|$.
4. Go to step 1.

This cycle is repeated as often as is required to ensure convergence of the codebook of code vectors.

The standard method [8] specifies that $\pi(y' - y)$ should be an even unimodal function whose width should be gradually decreased as training progresses. This allows coarse-grained organisation of the codebook to occur, followed progressively by ever more fine-grained organisation, until finally the algorithm converges towards an optimum codebook.

In our own modification [19] of the standard method we replace a shrinking $\pi(y' - y)$ function acting on a fixed number of code vectors by a fixed $\pi(y' - y)$ function acting on an increasing number of code vectors. There

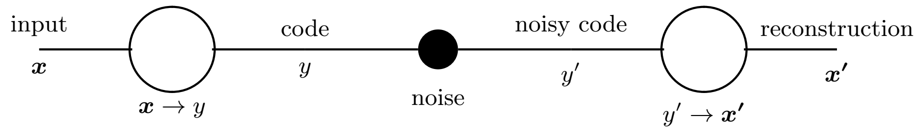


Figure 18: Encoding and decoding in a noisy vector quantiser.

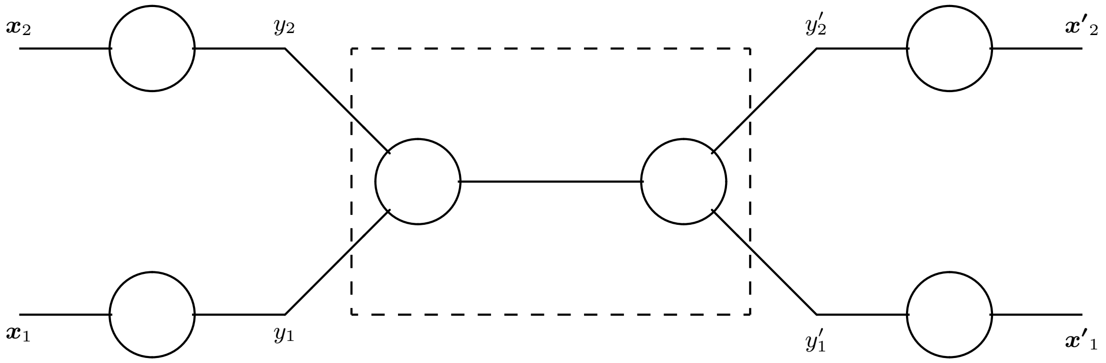


Figure 19: Encoding and decoding in a hierarchical vector quantiser.

are many minor variations on this theme, but we find that it is sufficient to define

$$\pi(y'-y) = \begin{cases} \epsilon & y' = y \\ \epsilon' & |y'-y| = 1 \\ 0 & |y'-y| > 1 \end{cases} \tag{A12}$$

where we have absorbed ϵ in Equation A11 into the definition of $\pi(y'-y)$. We use a binary sequence of codebook sizes $N = 2, 4, 8, 16, 32, \cdots$, where each codebook is initialised by interpolation from the next smaller codebook. We find that the following parameter values yield adequate convergence: $\epsilon = 0.1$, $\epsilon' = 0.05$, and we perform $20N$ training updates before doubling the value of N and progressing to the next larger size of codebook. The $N = 2$ codebook can be initialised using a random pair of vectors from the training set.

Appendix B: Relationship to WISARD

The second bracketed term in Equation 2.24 could be implemented in hardware as shown in Figure 20. This implementation assumes that the state vector $\boldsymbol{x}$ is quantised by representing each of its components using a finite number of binary digits (bits). Note that we have taken advantage of the fact that we are discussing a single layer network in order to simplify the notation in Figure 20 (as compared with Equation 2.24). The i-th block in this circuit is a random access memory (RAM) which records a transformation from $\boldsymbol{x}_i$ (the address of an entry in the RAM) to $\log P_i(\boldsymbol{x}_i)$ (the corresponding entry in the RAM). The data bus at the bottom of Figure 20 carries the components of $\boldsymbol{x}$ (represented bitwise) to the relevant RAM. Note that each bit of $\boldsymbol{x}$ is used exactly once in forming addresses for the RAM, so the mapping from $\boldsymbol{x}$ to the set of addresses is *bijective.* The upper part of Figure 20 shows how the outputs are directed to an accumulator where they are summed to form $\log Q_{mem}(\boldsymbol{x})$.

Figure 20 is a variant of the WISARD pattern recognition network [2]. The elements that our MEM solution and WISARD have in common are: a bijective mapping from the bits of an input state vector onto the address lines of a set of RAMs, and the accumulation of the outputs of the RAMs to form the overall network output.

However, there are some differences between the single layer ACE and the WISARD prescriptions for the contents of the RAMs. ACE specifies a set of functions (i.e. logarithms of marginal probabilities) to tabulate in the RAMs. Suppose that we truncate these table entries to a 1-bit representation, so that we use 0 to represent small logarithmic probabilities and 1 to represent large logarithmic probabilities. Each entry (i.e. 1 or 0) in the table then records whether the configuration of binary digits (i.e. the address of the entry) frequently occurs in the set of patterns corresponding to $P(\boldsymbol{x})$ (the "training set"). The final output is therefore the total number of 1's that the input pattern addresses in the n tables. In effect, this is the total number of coincidences between configurations of bits in the input pattern and those in a predefined category. This 1-bit version of ACE is qualitatively the same as the table look-up and summation operations performed in the simplest WISARD network, which completes the connection that we sought between ACE and WISARD.

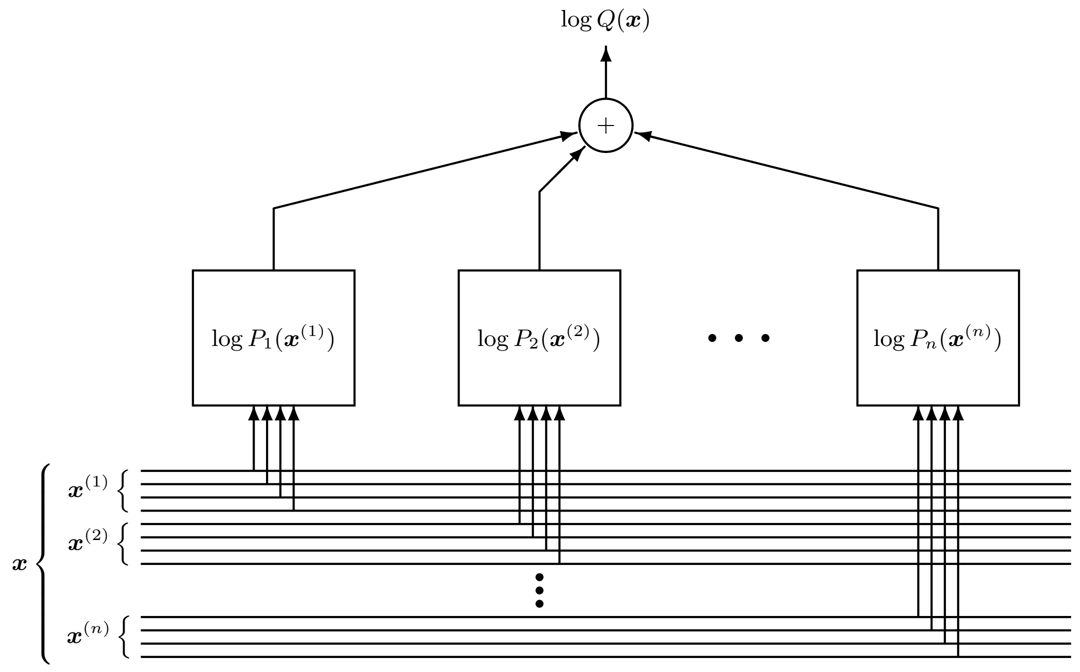


Figure 20: Single layer ACE is WISARD.

[1] D H Ackley, G E Hinton, and T J Sejnowski, *A learning algorithm for Boltzmann machines*, Cognitive Science **9** (1985), no. 2, 147–169.

[2] I Aleksander and T J Stonham, *Guide to pattern recognition using random access memories*, Computers and Digital Techniques **2** (1979), no. 1, 29–40.

[3] P Brodatz, *Textures - a photographic album for artists and designers*, Dover, New York, 1966.

[4] N Farvardin, *A study of vector quantisation for noisy channels*, IEEE Transactions on Information Theory **36** (1990), no. 4, 799–809.

[5] R M Haralick, K Shanmugam, and I Dinstein, *Textural features for image classification*, IEEE Transactions on Systems, Man, and Cybernetics **3** (1973), no. 6, 610–621.

[6] E T Jaynes, *Prior probabilities*, IEEE Transactions on Systems Science and Cybernetics **4** (1968), no. 3, 227–241.

[7] ———, *On the rationale of maximum entropy methods*, Proceedings of the IEEE **70** (1982), no. 9, 939–952.

[8] T Kohonen, *Self-organisation and associative memory*, Springer-Verlag, Berlin, 1984.

[9] H Kumazawa, M Kasahara, and T Namekawa, *A construction of vector quantisers for noisy channels*, Electronics and Engineering in Japan **67** (1984), no. 4, 39–47.

[10] Y Linde, A Buzo, and R M Gray, *An algorithm for vector quantiser design*, IEEE Transactions on Communications **28** (1980), no. 1, 84–95.

[11] S P Luttrell, *Markov random fields: a strategy for clutter modelling*, Proceedings of AGARD conference on scattering and propagation in random media (Rome), AGARD, 1987, pp. 7.1–7.8.

[12] ———, *Radar 87*, ch. Designing Markov random field structures for clutter modelling, pp. 222–226, IEE, London, 1987.

[13] ———, *The use of Markov random field models in sampling scheme design*, Proceedings of SPIE international symposium on inverse problems in optics (New York), SPIE, 1987.

[14] ———, *The use of Markov random field models to derive sampling schemes for inverse texture problems*, Inverse Problems **3** (1987), no. 2, 289–300.

[15] ———, *Image compression using a neural network*, Proceedings of IGARSS conference on remote sensing (Edinburgh), ESA, 1988, pp. 1231–1238.

[16] ———, *A maximum entropy approach to sampling function design*, Inverse Problems **4** (1988), no. 3, 829–841.

[17] ———, *Self organising multilayer topographic mappings*, Proceedings of IEEE conference on neural networks (San Diego), IEEE, 1988, pp. 93–100.

[18] ———, *Hierarchical self-organising networks*, Proceedings of IEE Conference on Artificial Neural Networks (London), IEE, 1989, pp. 2–6.

[19] ———, *Hierarchical vector quantisation*, Proceedings of the IEE I **136** (1989), no. 6, 405–413.

[20] ———, *Image compression using a multilayer neural network*, Pattern Recognition Letters **10** (1989), no. 1, 1–7.

[21] ———, *Maximum entropy and Bayesian methods*, ch. The use of Bayesian and entropic methods in neural network theory, pp. 363–370, Kluwer, Dordrecht, 1989.

[22] ———, *Self-organisation: a derivation from first principles of a class of learning algorithms*, Proceedings of IJCNN international joint conference on neural networks (Washington DC), IEEE, 1989, pp. 495–498.

[23] ———, *Derivation of a class of training algorithms*, IEEE Transactions on Neural Networks **1** (1990), no. 2,

229–232.

Adaptive Cluster Expansion (ACE): A Multilayer Network for Estimating Probability Density Functions [*]

S P Luttrell
Defence Research Agency, Malvern

We derive an adaptive hierarchical method of estimating high dimensional probability density functions. We call this method of density estimation the "adaptive cluster expansion", or ACE for short. We present an application of this approach, based on a multilayer topographic mapping network, that adaptively estimates the joint probability density function of the pixel values of an image, and presents this result as a "probability image". We apply this to the problem of identifying statistically anomalous regions in otherwise statistically homogeneous images.

I. INTRODUCTION

The purpose of this paper is to develop a novel type of adaptive network for estimating probability density functions (PDF) for use in Bayesian analysis [3, 5]. We consider only techniques that scale well for use in high dimensional spaces, such as the analysis of large arrays of pixels in image processing. There are many attempts to solve this type of density estimation problem. For instance, the Boltzmann machine [1] is essentially a trainable Gibbs distribution, which permits arbitrarily complicated statistical structure to be modelled via hidden variables. Unfortunately, this generality must be paid for by performing lengthy Monte Carlo simulations. There are various extensions to this technique, such as the higher order Boltzmann machine [13], which capture higher order statistical behaviour more economically, but none of these variations has been shown to be suitable for high-dimensional image processing problems. Using maximum entropy techniques [4], we develop a number of variations on the Gibbs distribution approach [10], and propose a scheme in which we replace simple interactions between a large number of hidden variables (as in the Boltzmann machine) by complicated interactions which directly model the statistical structure of the data; this is an extreme form of the approach taken in [13].

The novel adaptive density estimator that we develop in [10] is based on a multilayer network, in which we choose the layer-to-layer connections to be hierarchical, and the layer-to-layer transformations to be topographic mappings [6]; this adaptively transforms the input data into a multiscale "pyramid-like" format. In [10] we further propose that the joint PDF's of adjacent nodes in each layer should be combined to form an estimate of the joint PDF of the nodes in the input layer. By analogy with the standard derivation of Gibbs distributions, we can also derive our joint PDF estimate by applying the maximum entropy method [4]. However, our result is computationally much cheaper to implement than a standard Gibbs distribution, because we do not need to perform Monte Carlo simulations in order to integrate over the states of hidden variables. We suggest the name "adaptive cluster expansion" (ACE) for this type of network estimate of high-dimensional joint PDF's. Other literature on this approach can be found in [7–9, 11, 12], where we further develop multilayer topographic mapping networks, and their relationship to vector quantisers.

The purpose of this paper is to present a complete account of ACE, and to demonstrate its effectiveness when applied to the problem of density estimation. We do not dwell on the details of how to implement the topographic mapping training algorithm (we review this in the appendix). In Section II we develop the ACE method of density estimation by appealing to simple counting arguments. In Section III we demonstrate the power of ACE by applying it to the problem of estimating the joint PDF of the pixels of textured images selected from the Brodatz album [2].

II. PROBABILITY DENSITY FUNCTION ESTIMATION

In this section we present a derivation of the ACE estimate $Q(\boldsymbol{x})$ of a PDF $P(\boldsymbol{x})$. We develop this result by appealing to simple counting arguments and by using a diagrammatic language.

A. Derivation of the ACE Estimate of a PDF

We show a simple network in Figure 1, where the input space is 4-dimensional, the output space is 2-dimensional, and we factorise the feedforward transformation as $\boldsymbol{y}(\boldsymbol{x}) = (y_1(x_1, x_2), y_2(x_3, x_4))$. Suppose we estimate the joint PDF $P_{\text{out}}(y_1, y_2)$ of the outputs, and the joint PDFs $P_{\text{in},12}(x_1, x_2)$ and $P_{\text{in},34}(x_3, x_4)$ of each pair of inputs, by measuring their histograms, for instance. Using this information alone we now wish to construct an estimate $Q(\boldsymbol{x})$ of the true joint PDF $P(\boldsymbol{x})$ of the 4-dimensional input. There are two alternative, but equivalent, ways of writing $Q(\boldsymbol{x})$, each of which has its own

[*]Typeset in LaTeX on May 9, 2019.
This is archived at http://arxiv.org/abs/1012.3656, 16 Dec 2010. This paper was submitted to IEEE Trans. NN on 16 December 1991. Paper TNN #1115. It was not accepted for publication, but it underpins several subsequently published papers. © Crown Copyright 1991

interesting interpretation.

Firstly, we may write

$$Q(\boldsymbol{x}) = P_{\text{out}}(y_1(x_1,x_2), y_2(x_3,x_4)) \frac{P_{\text{in},12}(x_1,x_2)}{P_{\text{out}}(y_1(x_1,x_2))} \frac{P_{\text{in},34}(x_3,x_4)}{P_{\text{out}}(y_2(x_3,x_4))} \tag{2.1}$$

In Equation 2.1 we construct $Q(\boldsymbol{x})$ as follows. We use $P_{\text{out}}(y_1,y_2)$ directly to estimate the joint PDF of the outputs, and indirectly to estimate the joint PDF of the inputs. In order to convert a PDF in output space (i.e. $P_{\text{out}}(y_1,y_2)$) into a PDF in input space we must divide $P_{\text{out}}(y_1,y_2)$ by a compression factor equal to the number of input values that can produce the observed output value. Because we obtain y_1 and y_2 *separately* from the pairs (x_1,x_2) and (x_3,x_4), respectively, this compression factor is the product of two separate factors. For instance, the compression factor corresponding to y_1 is the ratio $\frac{P_{\text{out}}(y_1(x_1,x_2))}{\langle P_{\text{in},12}(x_1,x_2)\rangle}$, where $\langle P_{\text{in},12}(x_1,x_2)\rangle$ is the average value of $P_{\text{in},12}(x_1,x_2)$ over all the (x_1,x_2) that produce the same value of y_1. However, we may refine this compression factor by using $P_{\text{in},12}(x_1,x_2)$ instead of $\langle P_{\text{in},12}(x_1,x_2)\rangle$ in the denominator, to yield the ratio $\frac{P_{\text{out}}(y_1(x_1,x_2))}{P_{\text{in},12}(x_1,x_2)}$. An analogous argument may be applied to obtain the compression factor corresponding to y_2, and the results combined to obtain the final expression for $Q(\boldsymbol{x})$ as shown in Equation 2.1.

$$Q(\boldsymbol{x}) = P_{\text{in},12}(x_1,x_2) P_{\text{in},34}(x_3,x_4) \frac{P_{\text{out}}(y_1(x_1,x_2), y_2(x_3,x_4))}{P_{\text{out}}(y_1(x_1,x_2))\, P_{\text{out}}(y_2(x_3,x_4))} \tag{2.2}$$

which is trivially the same as Equation 2.1, but we have arranged its terms in a new way. This furnishes us with an alternative interpretation of $Q(\boldsymbol{x})$. Thus, imagine that we are provided only with $P_{\text{in},12}(x_1,x_2)$ and $P_{\text{in},34}(x_3,x_4)$, and no information about the correlations between the pair (x_1,x_2) and the pair (x_3,x_4). This is sufficient for us to construct $Q(\boldsymbol{x})$ as the product $P_{\text{in},12}(x_1,x_2)\, P_{\text{in},34}(x_3,x_4)$. Now, we admit that in fact we also know $P_{\text{out}}(y_1,y_2)$, which is a source of information about correlations between the pair (x_1,x_2) and the pair (x_3,x_4). We make use of this information by forming the dimensionless ratio $\frac{P_{\text{out}}(y_1,y_2)}{P_{\text{out}}(y_1)\, P_{\text{out}}(y_2)}$, which differs from unity when y_1 and y_2 are correlated random variables (i.e. $P_{\text{out}}(y_1,y_2) \neq P_{\text{out}}(y_1)\, P_{\text{out}}(y_2)$). This ratio is greater (or less) than unity when the pair (y_1,y_2) is more (or less) likely to occur than would have been estimated from knowledge of the marginal PDF's $P_{\text{out}}(y_1)$ and $P_{\text{out}}(y_2)$ alone. Finally, we use this dimensionless ratio as a correction factor to obtain the expression for $Q(\boldsymbol{x})$ shown in Equation 2.2. This derivation is heuristic, but it leads to the same result as shown in Equation 2.1.

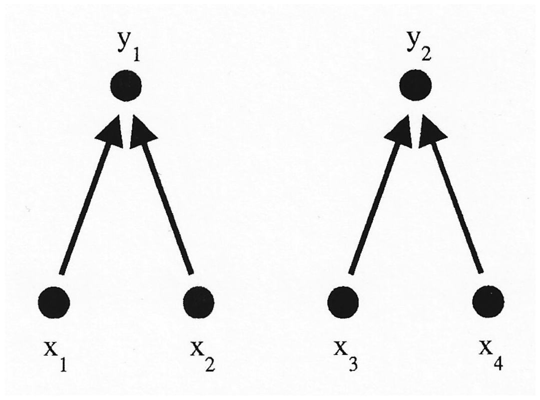


Figure 1: Basic 2-layer network. The input space (layer 0) is 4-dimensional and the output space (layer 1) is 2-dimensional. The layer 0-to-1 transformation factorises into two independent transformations: y_1 depends only on (x_1,x_2), and y_2 depends only on (x_3,x_4).

In Figure 2 we present an alternative representation of the network in Figure 1, in which we emphasise the PDFs that we use to construct $Q(\boldsymbol{x})$. Thus we introduce a shorthand notation in which we use an oval to highlight each clique of nodes in the network. We define the word "clique" to mean "complete set of nodes having the same parent node". As is conventional when discussing tree-like networks, we regard the higher layers of the network as being the ancestors of the lower layers, regardless of the fact that the direction of information flow is in the opposite direction through the tree. We then construct $Q(\boldsymbol{x})$ as the product of the three clique PDF's shown, whilst ensuring that the clique in layer 1 is appropriately normalised to render its contribution dimensionless. This leads to the form of $Q(\boldsymbol{x})$ in Equation 2.2.

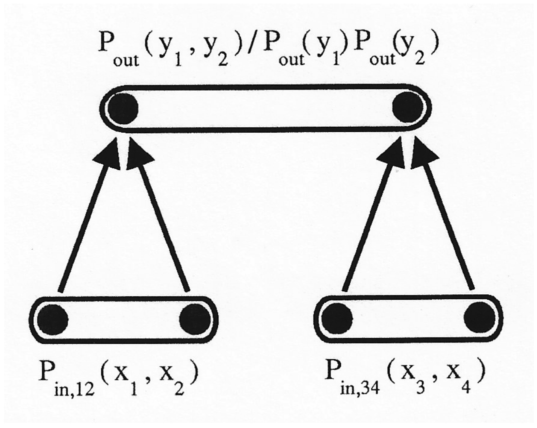


Figure 2: The clique PDFs that we use in the basic 2-layer network. $P_{\text{out}}(y_1, y_2)$ is the joint PDF of the pair of network outputs, and $P_{\text{out}}(y_1)$ and $P_{\text{out}}(y_2)$ are its two marginal PDFs. $\frac{P_{\text{out}}(y_1,y_2)}{P_{\text{out}}(y_1)\,P_{\text{out}}(y_2)}$ is a dimensionless ratio which records correlations between y_1 and y_2. $P_{\text{in},12}(x_1, x_2)$ and $P_{\text{in},34}(x_3, x_4)$ are the joint PDFs of the pairs of inputs from which y_1 and y_2 derive, respectively.

This diagrammatic approach to constructing $Q(\boldsymbol{x})$ may be readily extended to any tree-like feedforward network. We favour this approach, because the basic strategy for deriving $Q(\boldsymbol{x})$ by invoking compression factors remains the same, but the burden of notational detail becomes somewhat heavy, so diagrams provide an ideal shortcut. For convenience, we summarise the prescription for constructing $Q(\boldsymbol{x})$ from a tree-like diagram as follows:

1. Estimate all of the clique PDFs, as histograms, for instance.
2. Deduce all of the single-node marginal PDFs from the clique PDFs estimated in the previous step. For instance this would create $P_{\text{out}}(y_1)$ and $P_{\text{out}}(y_2)$ from $P_{\text{out}}(y_1, y_2)$. This step is not needed in in layer 0.
3. From the results estimated in the previous two steps, for each clique compute a clique factor as follows:
   (a) In the input layer the factor is the clique PDF itself.
   (b) In other layers the factor is the clique PDF divided by the product of its marginal PDF's (e.g. $\frac{P_{\text{out}}(y_1,y_2)}{P_{\text{out}}(y_1)\,P_{\text{out}}(y_2)}$).
4. Finally, to construct $Q(\boldsymbol{x})$, form the product of all of the clique factors estimated in the previous step.

B. Translation Invariance

A disadvantage of the above prescription for constructing $Q(\boldsymbol{x})$ is that it does not treat the components of $\boldsymbol{x}$ on an equal footing. For instance, in Equation 2.2 we see that the pair (x_1, x_2) is treated differently from the pair (x_2, x_3), even though both of these are pairs of adjacent components in the data. In order to solve this problem we construct a number of different tree-like networks, each of which breaks symmetry in its own peculiar way, and then we combine the results from each network to construct a composite $Q(\boldsymbol{x})$ which respects the required symmetry.

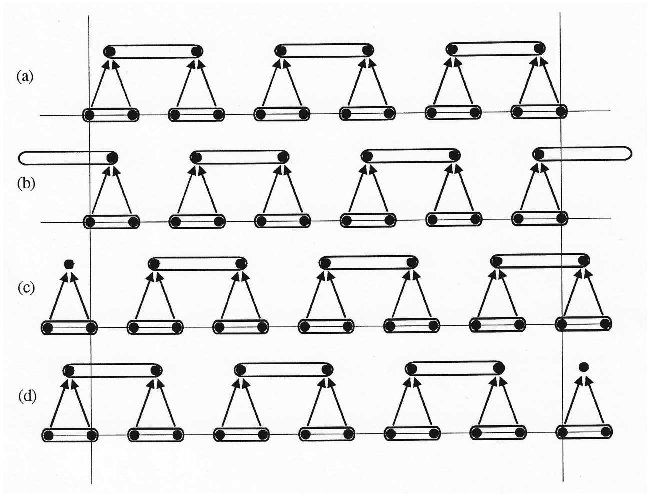


Figure 3: An example of the 4 separate 2-layer networks that we need to combine in order to produce a $Q(\boldsymbol{x})$ that treats each component of $\boldsymbol{x}$ on an equal footing. Figure 3a shows the basic 2-layer network, Figure 3b shows the same network with the layer 1 clique PDF's translated. Figure 3c and Figure 3d derive from Figure 3a and Figure 3b by simultaneously translating the clique PDF's in both network layers.

In Figure 3 we show an example of the set of 4 different 2-layer networks which we need to combine in order to construct a composite $Q(\boldsymbol{x})$. In this example we assume that the input is a high dimensional vector, so we can ignore edge effects. We replicate the basic network structure of Figure 2 across the input vector, as shown. Each of the 4 networks has its own set of clique PDFs (drawn as ovals in Figure 2), each of which leads to its own estimate $Q(\boldsymbol{x})$ which breaks symmetry. However, a symmetric combination (such as the arithmetic or geometric mean) of these 4 results treats each component of $\boldsymbol{x}$ on an equal footing. We can verify this by noting that the set of cliques that contributes in the spatial neighbourhood of each component of $\boldsymbol{x}$ does not depend (apart from a trivial overall translation) on which component we select.

We must select a prescription for forming the composite $Q(\boldsymbol{x})$. It needs only to be a symmetric combination of the 4 individual estimates that we show in Figure 3; the arithmetic mean and geometric mean are obvious choices. On pragmatic grounds, we choose to use the geometric mean, because it corresponds to the arithmetic mean of $\log Q(\boldsymbol{x})$, which is more convenient to perform in limited precision hardware ($\log Q(\boldsymbol{x})$ has a much smaller dynamic range than $Q(\boldsymbol{x})$, assuming that we avoid the logarithmic singularity).

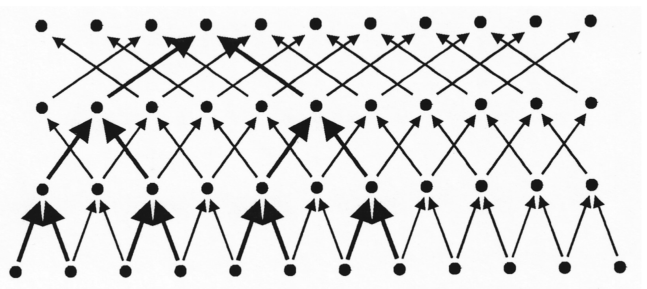


Figure 4: Example of the composite network connectivity that we require in order for a single network to compute a composite $Q(\boldsymbol{x})$, which treats each component of the input on an equal footing. This connectivity is the union of all of the binary trees can be generated from a reference binary tree (which we highlight in bold).

In Figure 4 we show the connectivity of part of a 4-layer composite network that can be used to process the input data in preparation for constructing a composite $Q(\boldsymbol{x})$. This connectivity contains all possible embedded tree-like networks, and in Figure 4 we highlight one such embedded tree for illustrative purposes.

For an n-layer network, we form the composite $Q(\boldsymbol{x})$ as the geometric mean over the $Q(\boldsymbol{x})$ derived from all tree-like networks that are embedded in this composite network, to yield the geometric mean PDF $Q_{\text{gm}}(\boldsymbol{x})$ in the form

$$\log Q_{\text{gm}}(\boldsymbol{x}) = \sum_{L=0}^{n-1} \frac{1}{2^{L+1}} \sum_{k} \log P_k^L \tag{2.3}$$

where L sums over layers 0 to $n-1$ of the network, k sums over cliques within a layer of the network, and P_k^L is the clique PDF at position k in layer L. It is important to note that the cliques are not simply adjacent nodes in each layer of the network. We must select pairs of nodes that form a "complete set of nodes having the same parent node". In layer 0 this means that the nodes are adjacent. In layer 1 the nodes in a pair are separated by 1 intervening node. In layer 2 there are 3 intervening nodes, and so on. For $L \geq 1$ we must ensure that the P_k^L are dimensionless by dividing out the marginal PDFs, as in Equation 2.2. The $\frac{1}{2^{L+1}}$ factor ensures that we include each tree-like network exactly once, and that the final result is indeed the geometric mean of these contributions. Figure 3 shows the terms that Equation 2.3 generates when we set $n = 2$.

There are two further assumptions that we could make in order to simplify our result even further. Firstly, we could assume that the layer-to-layer transformations in Figure 4 were independent of position k within each layer L. Secondly, we could assume that the clique PDFs were independent of position k within each layer L. We can make both of these assumptions if the statistical properties of the input data are known to be translationally invariant (such as might be the case for an image of a texture, for instance). In all of our numerical simulations we make these two simplifying assumptions.

C. Modular Implementation

We now describe a practical implemention of Equation 2.3 in the context of image processing (i.e. 2-dimensional arrays of pixels of data). There are three basic operations to perform. We must use a training set to determine suitable layer-to-layer transformations, then estimate the clique PDFs in each network layer, and then construct $\log Q_{\text{gm}}(\boldsymbol{x})$ from these estimates. Ideally we should optimise the layer-to-layer transformations directly so that the constructed $Q_{\text{gm}}(\boldsymbol{x})$ is "close to" $P(\boldsymbol{x})$ in some sense (e.g. relative entropy), but we have not yet found a computationally cheap way of doing this. Instead, we tackle the problem indirectly, by using our existing multilayer topographic mapping network technique [8]. There are two main reasons for this choice. Firstly, this type of network is computationally cheap to train; we typically train such a network at the rate of 2.3 second per layer on a VAXstation 3100 workstation (assuming 6 bit data values). Secondly, the network encodes the input in such a way as to be able to reconstruct it approximately from the state of any network layer. Although this second property does not in general imply that the encoded input is the optimal one for constructing an estimate of the input PDF, it turns out that it does produce useful results.

In Figure 5 we show a system for constructing $Q_{\text{gm}}(\boldsymbol{x})$, which consists of two interconnected subsystems - a multilayer topographic mapping subsystem for transforming the input image, and a PDF estimation subsystem for forming an output image which contains the contributions to $Q_{\text{gm}}(\boldsymbol{x})$, each recorded in its spatially correct location in the image. For obvious reasons, we call the output image a "probability image". The flow from left to right across the top half of Figure 5 implements the network structure in Figure 4, and the flow from right to left across the bottom half of Figure 5 progressively constructs the probability image.

In Figure 5 the input image becomes layer 0 of a multilayer network. In layer 0 we extract a pair of adjacent pixels, and then pass it through a look-up table (or mapping) to yield a single value which we write into the appropriate pixel location in layer 1 (in Figure 1 this corresponds to transforming (x_1, x_2) to become y_1). We repeat this operation all over layer 0, to yield a whole array of transformed values in layer 1. There is an arbitrariness in our choice of the relative position of the pairs of pixels that we use (e.g. north-south, or east-west, etc). In our simulations we use a north-south relative position in the layer 0-to-1 transformation, east-west in the layer 1-to-2 transformation, and alternate these two choices thereafter as we progress from layer to layer of the network. Note also that the separation of the pairs of pixels

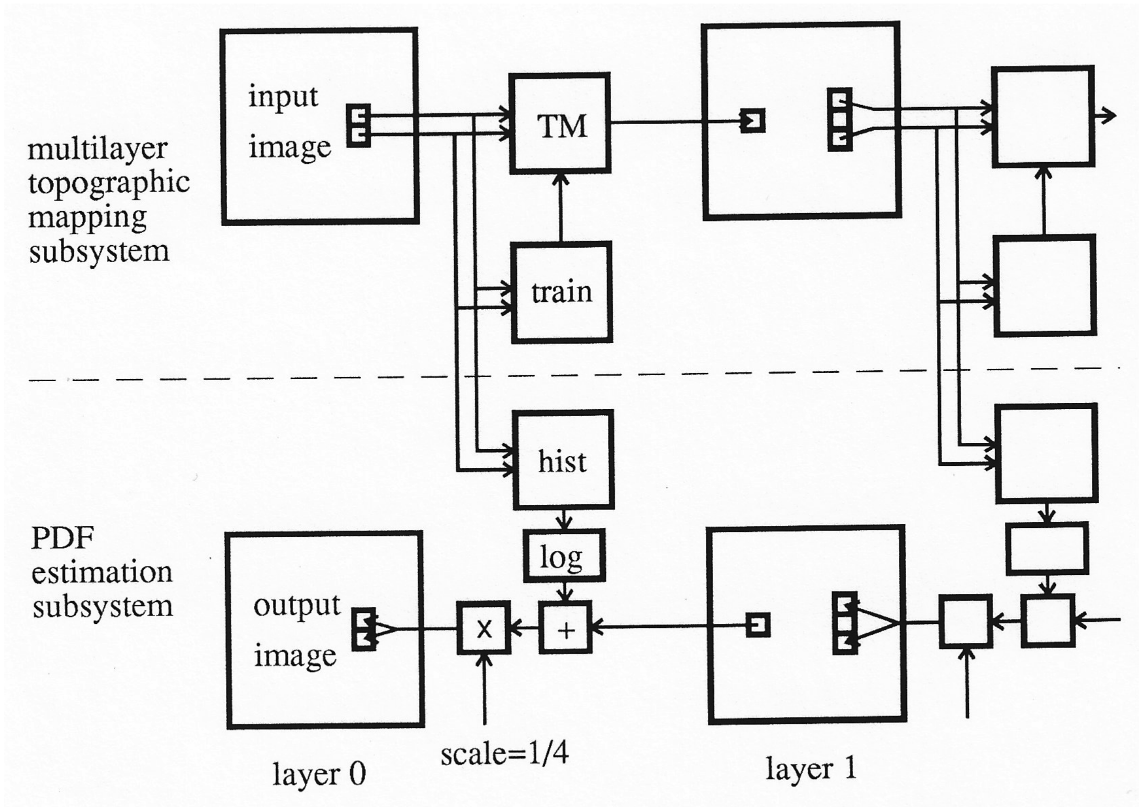


Figure 5: First two layers of a modular system for constructing $Q_{\mathrm{gm}}(\boldsymbol{x})$. The top half of the diagram is a multilayer network subsystem (actually, a multilayer topographic mapping in this case), which operates from left to right. The bottom half of the diagram is the PDF estimation subsystem, which operates from right to left. We connect the two systems by feeding logarithmic clique PDF's measured in the multilayer network through to be added together in the PDF estimator.

is not the same in each layer. In Figure 4 the separation doubles as we progress from layer to layer, but in Figure 5 the separation doubles after every two layers, because we must allow both the east-west and the north-south orientations to be processed at all separations (this is a consequence of processing 2-dimensional data through a 1-dimensional tree-structured network). If we concentrate only on the topology of the network that results from this prescription in Figure 5, we discover that it is identical to the topology in Figure 4. Thus, the only difference between these two cases is the way in which we identify the pixels of the input data array with the layer 0 nodes.

We may use any transformation that we wish in the look-up table. We have not yet discovered a computationally cheap way of optimising the network in order to construct a $Q(\boldsymbol{x})$ that best approximates the required $P(\boldsymbol{x})$. Instead, we optimise the network in such a way that each layer could be used to reconstruct approximately the state of the previous layer. This is not the same optimisation problem, but it is computationally very cheap, and empirically it leads to useful results for $Q(\boldsymbol{x})$. We choose to train the network as a multilayer topographic mapping, which we implement in a look-up table after the training schedule has ended. Typically, the largest number of bits per pixel that we use is 8, which corresponds to a look-up table with 65536 ($= 2^{2\times 8}$) separate addresses, each containing an 8 bit output value.

When we have trained a sufficient number of layers, we may estimate the clique PDF's in each layer. We simply record these as histograms, without making any attempt to interpolate or smooth these estimates; later on we shall mention a number of caveats. This completes the left-to-right pass in the top half of Figure 5.

In order to construct our geometric mean estimate $Q_{\mathrm{gm}}(\boldsymbol{x})$ of $P(\boldsymbol{x})$, we must combine the estimates of the clique PDFs. We may obtain the result in Equation 2.3 by appropriately scaling and summing the logarithms of the histograms (and their marginal histograms) in Figure 5. The method that we use depends on the following rearrangement of Equation 2.3

$$\log Q_{\mathrm{gm}}(\boldsymbol{x}) = \left(\cdots \left(\frac{1}{2} \sum_{k_{n-3}} \log P^{n-3}_{k_{n-3}} + \frac{1}{2} \left(\sum_{k_{n-2}} \log P^{n-2}_{k_{n-2}} + \frac{1}{2} \sum_{k_{n-1}} \log P^{n-1}_{k_{n-1}} \right) \right) \right) \tag{2.4}$$

in which we successively compute the contributions starting at network layer $n-1$, and then work outwards towards layer 0. First of all we initialise all of the images in the PDF estimation subsystem to some constant value (say zero), and then commence at layer $n-1$ (i.e. the righthandmost layer in Figure 5). Using the notation of Figure 2, each clique in the multilayer topographic mapping subsystem contributes a term of the form $\log P_{\mathrm{out}}(y_1, y_2) - \log P_{\mathrm{out}}(y_1) - \log P_{\mathrm{out}}(y_2)$, which we add to the values stored in the two pixels that are located at the same clique position in the PDF estimation subsystem. In order to compensate for this double counting, and in order to account for the $\frac{1}{2}$ factors that appear in Equation 2.4, we scale the logarithmic value by a factor $\frac{1}{4}$ $(= \frac{1}{2} \times \frac{1}{2})$. We then progress layer by layer towards the left in Figure 5. At each layer we generate its logarithmic contribution as above, but now we add to this the contribution from the layer on its right, as shown in Figure 5 and Equation 2.4. By cascading the results backwards from layer to layer of the network, we iteratively construct $\log Q_{\mathrm{gm}}(\boldsymbol{x})$ in the form shown in Equation 2.4. Note that the layer 0 cliques are slightly different, because they contribute terms of the form $\log P_{\mathrm{in},12}(x_1, x_2)$.

When all of these stages are complete, the output image in Figure 5 contains pixel values whose sum equals the required $\log Q_{\mathrm{gm}}(\boldsymbol{x})$. The contribution to $\log Q_{\mathrm{gm}}(\boldsymbol{x})$ that is recorded in an output pixel derives from a (rectangular) region in the input image that surrounds the location of the output pixel, so the output image can be interpreted as an image of correctly spatially registered logarithmic probability contributions to $\log Q_{\mathrm{gm}}(\boldsymbol{x})$.

In our simulations we investigate how each individual layer of the multilayer network contributes to $\log Q_{\mathrm{gm}}(\boldsymbol{x})$, so we switch off all except one of the sources of logarithmic probability in Figure 5, which permits only a single layer of the network to contribute to the construction of the output image. Because each layer of the network typically is sensitive to statistical structure in the input image at only one length scale, the output image then typically reveals contributions to $\log Q_{\mathrm{gm}}(\boldsymbol{x})$ at only one length scale.

We should remark in passing that there are many other possible ways in which Figure 5 could be configured. Our results depend on an underlying tree-like structure, which we replicate to produce the translation invariant network in Figure 4, which we then use directly to produce the design in Figure 5. In the case of a non-binary tree we must be careful to produce the correct generalisation of Figure 4 and Figure 5, but there are no new difficulties in principle.

D. Algorithmic Details

We compensate for some of the effects of non-uniform illumination of the scene in the input image by adding a grey scale wedge whose gradient we choose in such a way as to remove the linear component of the non-uniformity. This improves the assumed translation invariance of the image statistics. We do not attempt to perform a histogram equalisation on the input image, because the transformation from network layer 0 to layer 1 tends to perform this function anyway. In order not to disrupt the discussion, we review the details of the topographic mapping training algorithm in the appendix.

We choose to process the image in alternate directions using the following sequence: north/south, east/west, north/south, east/west, etc. This sequence leads to the following sequence of rectangular regions of the input image that influence the value in each pixel in each layer of the network: 1×2, 2×2, 2×4, 4×4, etc, using (east/west, north/south) coordinates. In all of our experiments we use a 6-layer network, so the value in each pixel in the final layer is sensitive to an 8×8 region of the input image.

The number of bits per pixel B that we use in each layer of the network determines the quality of the topographic mappings (the B bit output from a topographic mapping is the index of the winner from amongst 2^B competing "neurons"). Increasing B improves the quality of the mapping but increases the training time; we need to compromise between these two conflicting requirements. In our work on simple Brodatz texture images we find that choosing B to lie between 6 and 8 proves to be sufficient. Note that we choose to use the same number of bits per pixel in each layer of the network. In general this restriction is not necessary.

It is important to note that for a given value of B there is an upper limit on the allowed entropy (per unit area) that the input data should have. A hierarchically connected multilayer topographic mapping network progressively squeezes the input data through an ever smaller bottleneck (in fact there are multiple parallel bottlenecks due to the overlapping tree structure) as we pass through the layers of the network. There is a upper limit to the number of network layers beyond which it simply cannot preserve information that is useful in estimating the joint density of the input data, which limits the capabilities of our current method.

The choice of the size of the histogram bins is also important. A property of the multilayer topographic mapping network is that adjacent histogram bins derive from input vectors that are close to each other (in the Euclidean sense), so it makes sense to rebin the histogram by adding together the contents of adjacent bins. We may easily control the histogram bin size by truncating the low order bits of each pixel value. If we truncate b low order bits of each pixel value, then effectively we smooth the histogram over 2^b adjacent bins (for each dimension of the histogram). As we smooth the histogram it will suffer from less noise, but we run the danger of smoothing away significant structure that might usefully be used to characterise the statistics of the input image; so we need to make a compromise.

It is most important not to use histogram bins that are too small. A large number of small histogram bins would record the details of the statistical fluctuations of the training image (as particular realisations of a Poisson noise process in each bin), and would act as a detailed record of the structure in the training image, and thus be unable to generalise very well. Such histograms would look very spiky, and in extreme cases there might be counts recorded in only a few bins, with zeros in all of the remaining bins. If this situation were to occur, then the training image would have a large $Q_{\mathrm{gm}}(\boldsymbol{x})$, whereas a test image (having the same statistical properties) would have a small $Q_{\mathrm{gm}}(\boldsymbol{x})$. The cause of this problem is the absence of a significant overlap between the spikes in the training and test image histograms, which could be avoided by ensuring that the histogram bins are not too small. Generally, we find that a little experimentation can be used to determine a robust histogram binning strategy, so we do not attempt to implement a more sophisticated technique here.

Finally, we display the contributions to $\log Q_{\mathrm{gm}}(\boldsymbol{x})$ as follows. We determine the range of pixel values that occurs in the image, and we translate and scale this into the range $[0, 255]$. This ensures that the smallest logarithmic probability appears as black, and the largest logarithmic probability appears as white, and all other values are linearly scaled onto intermediate levels of grey. This prescription has its dangers because each image determines its own special scaling, so one should be careful when comparing two different images. It can also be adversely affected by pixel value outliers arising from Poisson noise effects, where an extreme value of a single pixel could affect the way in which the whole of an image is displayed. However, we find that the overlapping tree structure of our multilayer network causes enough averaging together of individual contributions $Q(\boldsymbol{x})$ that the composite result $Q_{\mathrm{gm}}(\boldsymbol{x})$ does not suffer from problems due to pixel value outliers.

III. APPLICATION TO THE DETECTION OF ANOMALIES IN TEXTURES

In this section we present the results of applying the system shown in Figure 5 to four 256×256 images of textures taken from the Brodatz texture album [2]. In all cases we compensate for uneven illumination by introducing a grey scale wedge as we explained earlier, we use 8 bits per pixel for the topographic mappings, we use 6 bits per pixel for histogramming, and we invert the $[0, 255]$ scale to represent the contributions to $\log Q_{\mathrm{gm}}(\boldsymbol{x})$ in such a way that white pixels indicate a small (rather than a large) contribution to $\log Q_{\mathrm{gm}}(\boldsymbol{x})$. Thus white pixels in the output image correspond to regions of the input image whose statistical properties differ markedly from the statistics averaged over the whole image. We usually call this representation of the contributions to $\log Q_{\mathrm{gm}}(\boldsymbol{x})$ an "anomaly image".

We do not present these results as necessarily being an efficient way of detecting texture anomalies. Rather, we merely apply our novel method of estimating PDF's, as expressed in Equation 2.3 and in Figure 5, to the particular problem of texture analysis, because this is an effective way of demonstrating some of the more interesting properties of $\log Q_{\mathrm{gm}}(\boldsymbol{x})$.

A. Texture 1

In Figure 6 we show the first Brodatz texture image that we use in our experiments. The image is slightly unevenly illuminated and has a fairly low contrast, but nevertheless its statistical properties are almost translation invariant.

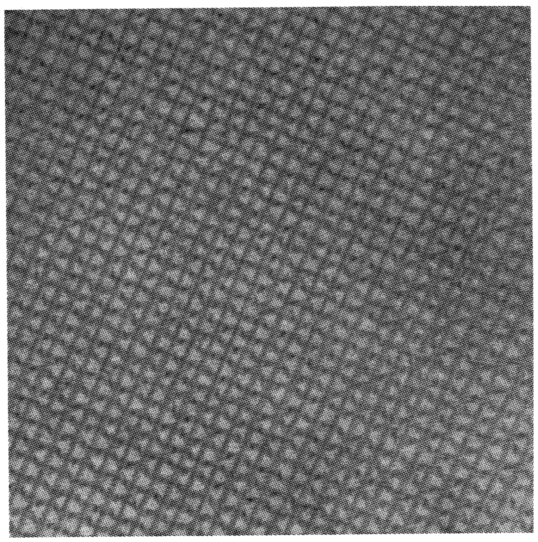


Figure 6: 256×256 image of Brodatz image 1.

In Figure 7 we show the anomaly images that derive from Figure 6. Note how the anomaly images become smoother as we progress from Figure 7a to Figure 7f, due to the increasing amount of averaging that occurs amongst the overlapping trees in the network.

Figure 7e and Figure 7f reveal a highly localised anomaly in the original image. Figure 7f corresponds to a length scale of 8×8 pixels, which is the approximate size of the fault that is about $\frac{1}{4}$ of the way down

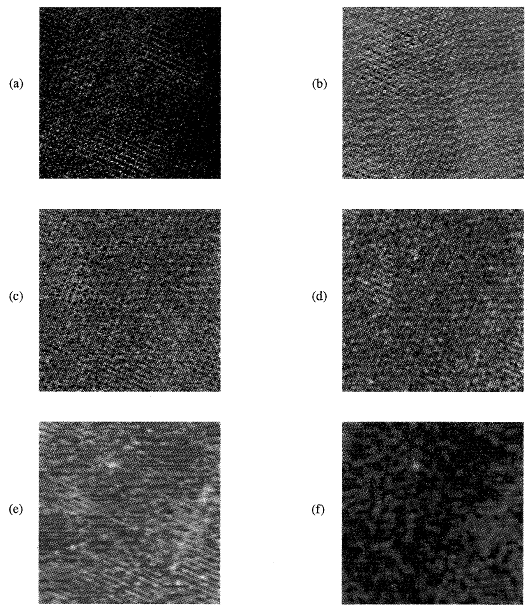


Figure 7: 256×256 anomaly images of Brodatz image 1.

and slightly to the left of centre of Figure 6. The fault does not show up clearly on the other figures in Figure 7 because their characteristic length scales are either too short or too long to be sensitive to the fault.

From Figure 7 we conclude that ACE can easily pick out localised faults in highly ordered textures.

B. Texture 2

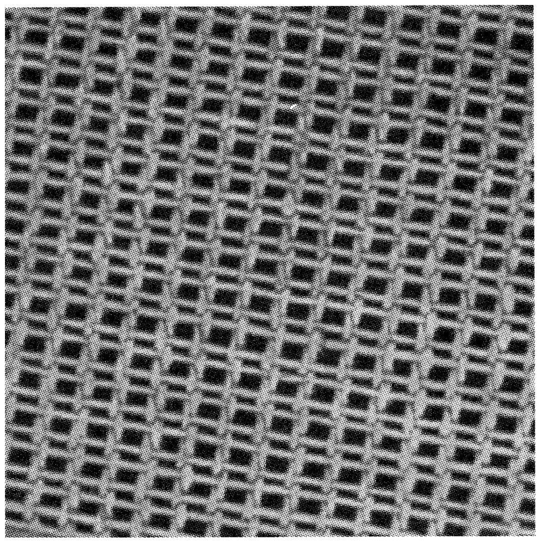


Figure 8: 256×256 image of Brodatz image 2.

In Figure 8 we show the second Brodatz texture image that we use in our experiments. The image has a high contrast and translation invariant statistical properties.

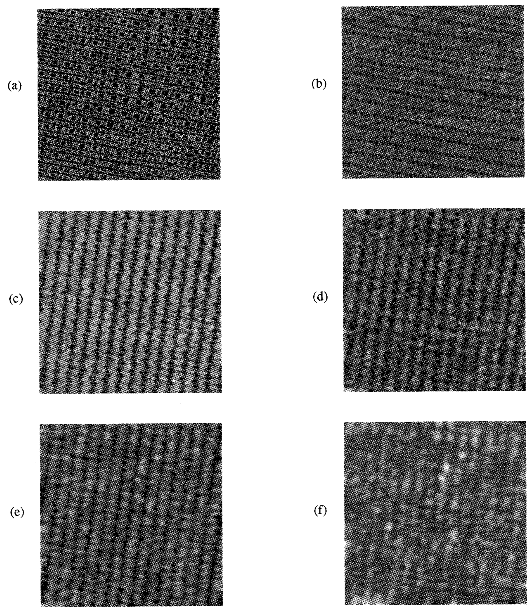


Figure 9: 256×256 anomaly images of Brodatz image 2.

In Figure 9 we show the anomaly images that derive from Figure 8. The most interesting anomaly image is Figure 9f which shows several localised anomalies. About halfway down and to the left of centre of the image is an anomaly that corresponds to a dark spot on the thread in Figure 8. The brightest of the anomalies in the cluster just above the centre of the image corresponds to what appears to be a slightly torn thread in Figure 8. The other anomalies in this cluster are weaker, and correspond to slight distortions of the threads. There is another anomaly just below and to the right of the centre of Figure 9f, which corresponds to what appears to be another slightly torn thread in Figure 8. These anomalies all occur at, or around, a length scale of 8×8 pixels. Several of the anomaly images show an anomaly in the bottom left hand corner of the image, which corresponds to a small uniform patch of fabric in Figure 8.

The results in Figure 9 corroborate the evidence in Figure 7 that we can train ACE to pick out localised anomalies in highly structured textures. This type of texture could be analysed much more simply by model-based techniques that took advantage of their near-periodicity. However, that does not detract from the fact that, by making use of its adaptability, ACE succeeds in modelling these textures without prior knowledge of their near-periodicity. We seek a general purpose approach to density estimation; not a toolkit of different (usually model-based) techniques, each tuned to its own type of problem.

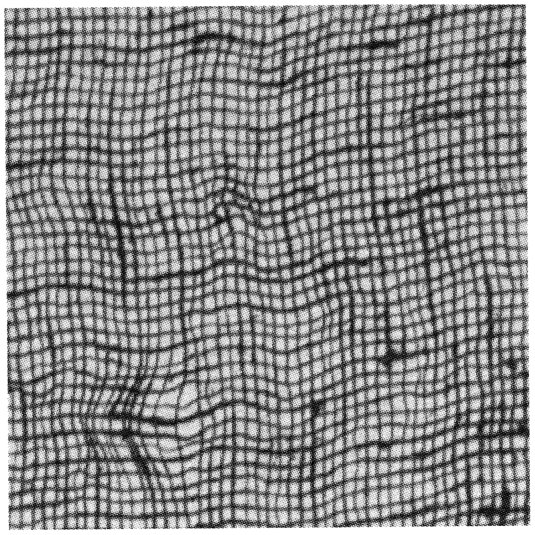


Figure 10: 256×256 image of Brodatz image 3.

C. Texture 3

In Figure 10 we show the third Brodatz texture image that use in our experiments. The image has a very high contrast and statistical properties that are almost translation invariant. However the density of anomalies is much higher than in either Figure 6 or Figure 8.

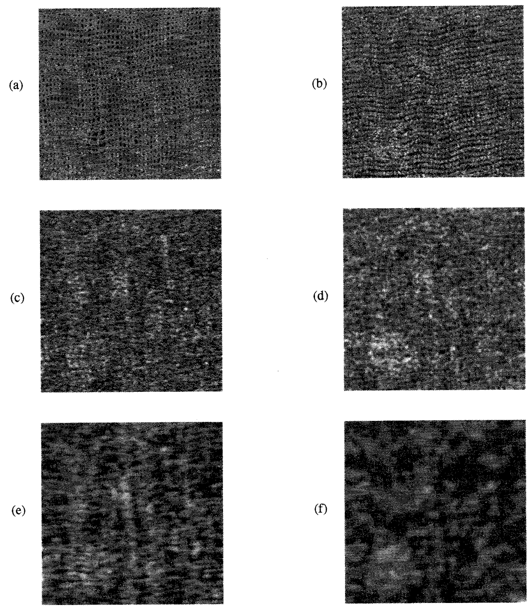


Figure 11: 256×256 anomaly images of Brodatz image 3.

In Figure 11 we show the anomaly images that derive from Figure 10. At the lower left hand corner of Figure 11f there is a large anomaly that corresponds to a region of Figure 10 that is distorted to the left. Figure 11f is sensitive to a length scale of 8×8 pixels, so it does not respond to this leftward distortion (which occurs on a length scale of around 32×32 pixels), rather it responds to localised variations in the separations of the threads.

There are numerous other anomalies in Figure 10; some are detected in Figure 11, and some are not. The ability of ACE to pick out anomalies degrades as the density of anomalies increases. This is because the anomalies themselves are part of the statistical properties that are extracted by ACE from the training image, and if a particular type of anomaly occurs often enough in the image then it is no longer deemed to be an anomaly. In extreme cases there is also the possibility that the entropy (per unit area) of the input image can saturate ACE and thus degrade its performance, as we discussed earlier.

D. Texture 4

In this section we present a slightly different type of experiment in which we train ACE on one image and test ACE on another image. To create the two images we start with a single 256×256 image of a Brodatz texture, which we divide into a left half and a right half. We then use the left half to construct a training image, and the right half to construct a test image. Note that in constructing these images we scrupulously avoid the possibility that the training and test images contain elements deriving from a common source, although there are some small residual correlations between the two images along their common edge.

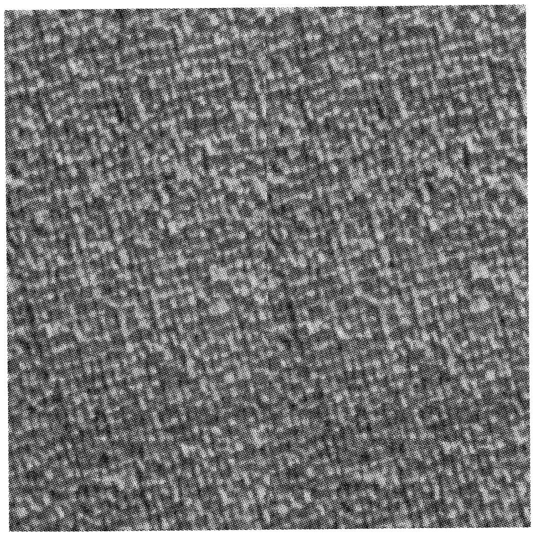


Figure 12: 256×256 image of a Brodatz image of a carpet for training.

In Figure 12 we show the training image which is a montage of two copies of the left hand half of a Brodatz texture image. Note that we use square, rather than rectangular, images because our software is restricted to processing this type of image.

In Figure 13 we show the test image which is a montage of two copies of the right hand half of a Brodatz texture

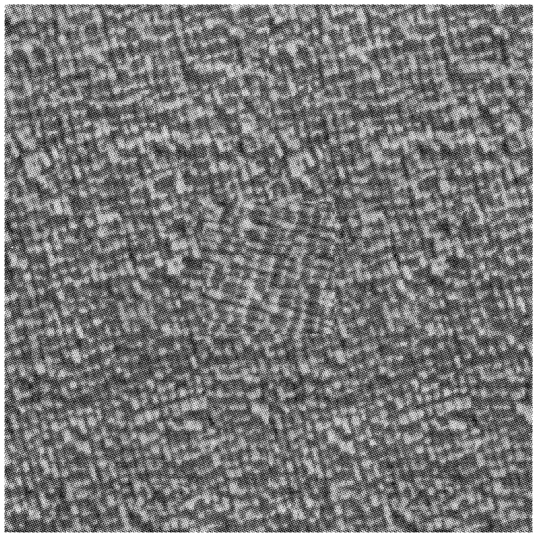


Figure 13: 256×256 image of a Brodatz image of a carpet for testing.

image, and superimposed on that is a 64×64 patch which we generated by flipping the rows and columns of a copy of the top left hand corner of this image. This patch is a hand-crafted anomaly.

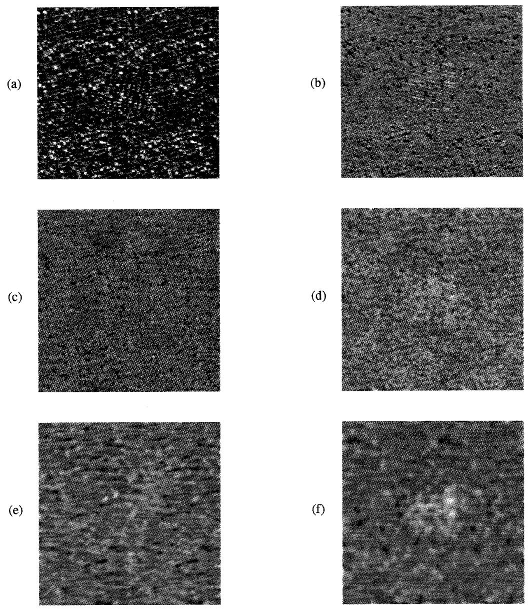


Figure 14: 256×256 anomaly images of a Brodatz image of a carpet.

In Figure 14 we show the anomaly images that derive from Figure 13 after we train on Figure 12. Figure 14f shows the strongest response to the anomalous patch in the centre of the image, corresponding to anomaly detection on a length scale of 8×8 pixels.

IV. CONCLUSIONS

We present a novel method of density estimation in high-dimensional spaces, such as images. In Bayesian data processing there is a pressing need for a flexible way of constructing such estimates, because the basic objects that we manipulate in Bayesian analysis are joint PDFs, which we must somehow construct in the first place. We call the hierarchical network structure that emerges from our analysis an "Adaptive Cluster Expansion", or ACE for short.

ACE is computationally very cheap: we can train a multilayer topographic mapping network to estimate the joint PDF of its input data at the rate of 1 network layer every 2.3 second (on a VAXstation 3100, and assuming 6 bits per pixel), where each layer analyses one length scale (power of 2) in the input data. We find, in our experiments with Brodatz textures, that 6 network layers allows the detection of statistical anomalies in the textures. This result is not universal, because it must depend strongly on the scale at which the anomalous statistical structure in the data is to be found. Although we demonstrate ACE only in a texture anomaly detection rôle, its scope is far greater than this. ACE is a general purpose, and computationally cheap, network for estimating densities in high-dimensional spaces.

For completeness, we should mention that the performance of ACE in its current form has two fundamental limitations. Firstly, we assume that the network connectivity is fixed, and that its functionality is determined by a training algorithm. This restricts the possible statistical properties of the input data that could be estimated. Secondly, ACE is based upon a hierarchically connected multilayer topographic mapping network, which progressively squeezes the input data through an ever smaller bottleneck as we pass through the layers of the network. There is a upper limit to the number of network layers beyond which ACE simply cannot preserve information that is useful in estimating the statistics of the input data. For instance, the statistics of an extremely noisy image of a texture can not be successfully estimated by ACE, because the noise entropy would saturate ACE before the statistics of the underlying texture could be investigated. This problem can be solved by introducing explicit noise models into ACE, which we shall report elsewhere.

The standard topographic mapping training procedure in [6] is a rather inefficient algorithm. In [8] we present in detail an efficient training procedure for topographic mappings, and explain how to use it to train multilayer topographic mappings.

For convenience, we introduce some notation.

$\boldsymbol{x}$	=	input vector
y	=	index of the winning "neuron"
$\boldsymbol{x}(y)$	=	reference vector associated with y
$\pi(y'-y)$	=	topographic neighbourhood function (normalised to unit total mass)
ε	=	update parameter used during training
N	=	number of reference vectors

1. Standard Topographic Mapping Training Algorithm

The standard topographic mapping training procedure is essentially as follows [6]:

1. Select a training vector $\boldsymbol{x}$ at random from the training set.
2. Map $\boldsymbol{x}$ to y by using a nearest neighbour prescription applied to the distance of $\boldsymbol{x}$ from each of the current set of reference vectors.
3. For all y', move the reference vector $\boldsymbol{x}(y')$ directly towards the input vector $\boldsymbol{x}$ by a distance $\varepsilon\,\pi(y'-y)\,\|\boldsymbol{x}-\boldsymbol{x}(y')\|$.
4. Go to step 1.

Repeat this loop as often as is required to ensure convergence of the reference vectors.

The standard training method specifies that $\pi(y'-y)$ should be an even unimodal function whose width should be gradually decreased as training progresses. This allows coarse-grained organisation of the reference vectors to occur, followed progressively by ever more fine-grained organisation, until finally the algorithm converges to an optimum set of reference vectors. In a similar vein, the relative size of the update step ε should also be steadily decreased as training progresses.

2. Modified Topographic Mapping Training Algorithm

In our own modification [8] of the standard topographic mapping training we replace a shrinking $\pi(y'-y)$ function acting on a fixed number of reference vectors, by a fixed $\pi(y'-y)$ function acting on an increasing number of reference vectors. There are many minor variations on this theme, but we find that it is sufficient to define

$$\pi(y'-y) = \begin{cases} \varepsilon & y' = y \\ \varepsilon' & |y'-y| = 1 \\ 0 & |y'-y| > 1 \end{cases}$$

where we absorb the ε into the definition of $\pi(y'-y)$. We increase the number of reference vectors in a binary sequence (i.e. $N = 2, 4, 8, 16, 32, \cdots$), and we initialise each generation of reference vectors by interpolation from the previous generation. We find that the following parameter values yield adequate convergence: $\varepsilon = 0.1$, $\varepsilon' = 0.05$, and we perform $20N$ training updates before doubling the value of N, as above. We initialise the $N = 2$ pair of reference vectors as a random pair of vectors chosen from the training set.

In numerous experiments, we find that this modified form of the topographic mapping training algorithm converges much more rapidly than the standard method. Furthermore, the binary sequence of N values lends itself well to implementing the trained topographic mapping using a look-up table.

[1] D H Ackley, G E Hinton, and T J Sejnowski, *A learning algorithm for Boltzmann machines*, Cognitive Science **9** (1985), no. 2, 147–169.

[2] P Brodatz, *Textures - a photographic album for artists and designers*, Dover, New York, 1966.

[3] R T Cox, *Probability, frequency and reasonable expectation*, American Journal of Physics **14** (1946), no. 1, 1–13.

[4] E T Jaynes, *On the rationale of maximum entropy methods*, Proceedings of the IEEE **70** (1982), no. 9, 939–952.

[5] H Jeffreys, *Theory of probability*, Clarendon Press, Oxford, 1939.

[6] T Kohonen, *Self-organisation and associative memory*, Springer-Verlag, Berlin, 1984.

[7] S P Luttrell, *Hierarchical self-organising networks*, Proceedings of IEE Conference on Artificial Neural Networks (London), IEE, 1989, pp. 2–6.

[8] ———, *Hierarchical vector quantisation*, Proceedings of the IEE I **136** (1989), no. 6, 405–413.

[9] ———, *Image compression using a multilayer neural network*, Pattern Recognition Letters **10** (1989), no. 1, 1–7.

[10] ———, *Maximum entropy and Bayesian methods*, ch. The use of Bayesian and entropic methods in neural network theory, pp. 363–370, Kluwer, Dordrecht, 1989.

[11] ———, *Derivation of a class of training algorithms*, IEEE Transactions on Neural Networks **1** (1990), no. 2, 229–232.

[12] ———, *Self-supervised training of hierarchical vector quantisers*, Proceedings of IEE conference on artificial neural networks (Bournemouth), IEE, 1991, pp. 5–9.

[13] T J Sejnowski, *Higher order Boltzmann machines*, Proceedings of conference on neural networks for computing (Snowbird), vol. 151, American Institute of Physics, pp. 398–403.

A Self-Organising Neural Network for Processing Data from Multiple Sensors [*]

S P Luttrell
Defence Research Agency, St Andrews Road, Malvern, Worcs, WR14 3PS, United Kingdom

This paper shows how a folded Markov chain network can be applied to the problem of processing data from multiple sensors, with an emphasis on the special case of 2 sensors. It is necessary to design the network so that it can transform a high dimensional input vector into a posterior probability, for which purpose the partitioned mixture distribution network is ideally suited. The underlying theory is presented in detail, and a simple numerical simulation is given that shows the emergence of ocular dominance stripes.

I. THEORY

A. Neural Network Model

In order to fix ideas, it is useful to give an explicit "neural network" interpretation to the theory that will be developed. The model will consist of 2 layers of nodes. The input layer has a "pattern of activity" that represents the components of the input vector $\mathbf{x}$, and the output layer has a pattern of activity that is the collection of activities of each output node. The activities in the output layer depend only on the activities in the input layer. If an input vector $\mathbf{x}$ is presented to this network, then each output node "fires" discretely at a rate that corresponds to its activity. After n nodes have fired the probabilistic description of the relationship between the input and output of the network is given by $\Pr(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n|\mathbf{x})$, where $\mathbf{y}_i$ is the location in the output layer (assumed to be on a rectangular lattice of size $\mathbf{m}$) of the i^{th} node that fires. In this paper it will be assumed that the order in which the n nodes fire is not observed, in which case $\Pr(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n|\mathbf{x})$ is a sum of probabilities over all $n!$ permutations of $(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n)$, which is a symmetric function of the $\mathbf{y}_i$, by construction.

The theory that is introduced in section I B concerns the special case $n = 1$. In the $n = 1$ case the probabilistic description $\Pr(\mathbf{y}|\mathbf{x})$ is proportional to the firing rate of node $\mathbf{y}$ in response to input $\mathbf{x}$. When $n > 1$ there is an indirect relationship between the probabilistic description $\Pr(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n|\mathbf{x})$ and the firing rate of node $\mathbf{y}$, which is given by the marginal probability

$$\Pr(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{y}_2, \cdots, \mathbf{y}_n}^{\mathbf{m}} \Pr(\mathbf{y}, \mathbf{y}_2, \cdots, \mathbf{y}_n|\mathbf{x}) \tag{1.1}$$

It is important to maintain this distinction between events that are observed (i.e. $(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n)$ given $\mathbf{x}$) and the probabilistic description of the events that are observed (i.e. $\Pr(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n|\mathbf{x})$). The only possible exception is in the $n \to \infty$ limit, where $\Pr(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n|\mathbf{x})$ has all of its probability concentrated in the vicinity of those $(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n)$ that are consistent with the observed long-term average firing rate of each node. It is essential to consider the $n > 1$ case to obtain the results that are described in this paper.

B. Probabilistic Encoder/Decoder

A theory of self-organising networks based on an analysis of a probabilistic encoder/decoder was presented in [1]. It deals with the $n = 1$ case referred to in section I A. The objective function that needs to be minimised in order to optimise a network in this theory is the Euclidean distortion D defined as

$$D \equiv \sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \int d\mathbf{x}\, d\mathbf{x}'\, \Pr(\mathbf{x})\, \Pr(\mathbf{y}|\mathbf{x})\, \Pr(\mathbf{x}'|\mathbf{y})\, \left\|\mathbf{x} - \mathbf{x}'\right\|^2 \tag{1.2}$$

where $\mathbf{x}$ is an input vector, $\mathbf{y}$ is a coded version of $\mathbf{x}$ (a vector index on a d-dimensional rectangular lattice of size $\mathbf{m}$), $\mathbf{x}'$ is a reconstructed version of $\mathbf{x}$ from $\mathbf{y}$, $\Pr(\mathbf{x})$ is the probability density of input vectors, $\Pr(\mathbf{y}|\mathbf{x})$ is a probabilistic encoder, and $\Pr(\mathbf{x}'|\mathbf{y})$ is a probabilistic decoder which is specified by Bayes' theorem as

$$\Pr(\mathbf{x}|\mathbf{y}) = \frac{\Pr(\mathbf{y}|\mathbf{x})\, \Pr(\mathbf{x})}{\int d\mathbf{x}'\, \Pr(\mathbf{y}|\mathbf{x}')\, \Pr(\mathbf{x}')} \tag{1.3}$$

D can be rearranged into the form [1]

$$D = 2 \sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \int d\mathbf{x}\, \Pr(\mathbf{x})\, \Pr(\mathbf{y}|\mathbf{x})\, \left\|\mathbf{x} - \mathbf{x}'(\mathbf{y})\right\|^2 \tag{1.4}$$

where the reference vectors $\mathbf{x}'(\mathbf{y})$ are defined as

$$\mathbf{x}'(\mathbf{y}) \equiv \int d\mathbf{x}\, \Pr(\mathbf{x}|\mathbf{y})\, \mathbf{x} \tag{1.5}$$

Although equation 1.2 is symmetric with respect to interchanging the encoder and decoder, equation 1.4 is not. This is because Bayes' theorem has made explicit the dependence of $\Pr(\mathbf{x}|\mathbf{y})$ on $\Pr(\mathbf{y}|\mathbf{x})$. From a neural network viewpoint $\Pr(\mathbf{y}|\mathbf{x})$ describes the feed-forward transformation from the input layer to the output layer, and

[*]Typeset in LaTeX on May 9, 2019.
This is archived at http://arxiv.org/abs/1012.4173, 19 Dec 2010. This unpublished draft paper accompanied a talk that was given at the Conference on Neural Networks for Computing, 4-7 April 1995, Snowbird.

$\mathbf{x}'(\mathbf{y})$ describes the feed-back transformation that is implied from the output layer to the input layer. The feed-back transformation is necessary to implement the objective function that has been chosen here.

Minimisation of D with respect to all free parameters leads to an optimal encoder/decoder. In equation 1.4 the $\Pr(\mathbf{y}|\mathbf{x})$ are the only free parameters, because $\mathbf{x}'(\mathbf{y})$ is fixed by equation 1.5. However, in practice, both $\Pr(\mathbf{y}|\mathbf{x})$ and $\mathbf{x}'(\mathbf{y})$ may be treated as free parameters [1], because $\mathbf{x}'(\mathbf{y})$ satisfy equation 1.5 at stationary points of D with respect to variation of $\mathbf{x}'(\mathbf{y})$.

C. Posterior Probability Model

The probabilistic encoder/decoder requires an explicit functional form for the posterior probability $\Pr(\mathbf{y}|\mathbf{x})$. A convenient expression is

$$\Pr(\mathbf{y}|\mathbf{x}) = \frac{Q(\mathbf{x}|\mathbf{y})}{\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} Q(\mathbf{x}|\mathbf{y}')} \tag{1.6}$$

where $Q(\mathbf{x}|\mathbf{y}) > 0$ can be regarded as a node "activity", and $\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} P(\mathbf{y}|\mathbf{x}) = 1$. Any non-negative function can be used for $Q(\mathbf{x}|\mathbf{y})$, such as a sigmoid (which satisfies $0 \leq Q(\mathbf{x}|\mathbf{y}) \leq 1$)

$$Q(\mathbf{x}|\mathbf{y}) = \frac{1}{1+\exp(-\mathbf{w}(\mathbf{y})\cdot\mathbf{x} - b(\mathbf{y}))} \tag{1.7}$$

where $\mathbf{w}(\mathbf{y})$ and $b(\mathbf{y})$ are a weight vector and bias, respectively.

A drawback to the use of equation 1.6 is that it does not permit it to scale well to input vectors that have a large dimensionality. This problem arises from the restricted functional form allowed for $Q(\mathbf{x}|\mathbf{y})$. A solution was presented in [2]

$$\Pr(\mathbf{y}|\mathbf{x}) = \frac{1}{M} Q(\mathbf{x}|\mathbf{y}) \sum_{\mathbf{y}'\in\tilde{N}(\mathbf{y})} \frac{1}{\sum_{\mathbf{y}''\in N(\mathbf{y}')} Q(\mathbf{x}|\mathbf{y}'')} \tag{1.8}$$

where $M \equiv m_1 m_2 \cdots m_d$, and $N(\mathbf{y})$ is a set of lattice points that are deemed to be "in the neighbourhood of" the lattice point $\mathbf{y}$, and $\tilde{N}(\mathbf{y})$ is the inverse neighbourhood defined as the set of lattice points that have lattice point $\mathbf{y}$ in their neighbourhood. This expression for $\Pr(\mathbf{y}|\mathbf{x})$ satisfies $\sum_{\mathbf{y}=1}^{\mathbf{m}} P(\mathbf{y}|\mathbf{x}) = 1$ (see appendix A). It is convenient to define

$$\Pr(\mathbf{y}|\mathbf{x};\mathbf{y}') \equiv \frac{Q(\mathbf{x}|\mathbf{y})}{\sum_{\mathbf{y}''\in N(\mathbf{y}')} Q(\mathbf{x}|\mathbf{y}'')} \tag{1.9}$$

which is another posterior probability, by construction. It includes the effect of the output nodes that are in the neighbourhood of node $\mathbf{y}'$ only. $\Pr(\mathbf{y}|\mathbf{x};\mathbf{y}')$ is thus a localised posterior probability derived from a localised subset of the node activities. This allows equation 1.8 to be written as $\Pr(\mathbf{y}|\mathbf{x}) = \frac{1}{M}\sum_{\mathbf{y}'\in\tilde{N}(\mathbf{y})} \Pr(\mathbf{y}|\mathbf{x};\mathbf{y}')$, so $\Pr(\mathbf{y}|\mathbf{x})$ is the average of the posterior probabilities at node $\mathbf{y}$ arising from each of the localised subsets that happens to include node $\mathbf{y}$.

D. Multiple Firing Model

The model may be extended to the case where n output nodes fire. $\Pr(\mathbf{y}|\mathbf{x})$ is then replaced by $\Pr(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n|\mathbf{x})$, which is the probability that $(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)$ are the first n nodes to fire (in that order). With this modification, D becomes

$$D = 2\sum_{\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n=\mathbf{1}}^{\mathbf{m}} \int d\mathbf{x}\,\Pr(\mathbf{x})\,\Pr(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n|\mathbf{x})\,\left\|\mathbf{x}-\mathbf{x}'(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)\right\|^2 \tag{1.10}$$

where the reference vectors $\mathbf{x}'(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)$ are defined as

$$\mathbf{x}'(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n) \equiv \int d\mathbf{x}\,\Pr(\mathbf{x}|\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)\,\mathbf{x} \tag{1.11}$$

The dependence of $\Pr(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n|\mathbf{x})$ and $\mathbf{x}'(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)$ on n output node locations complicates this result. Assume that $\Pr(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n|\mathbf{x})$ is a symmetric function of its $(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)$ arguments, which corresponds to ignoring the order in which the first n nodes choose to fire (i.e. $\Pr(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n|\mathbf{x})$ is a sum over all permutations of $(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)$). For simplicity, assume that the nodes fire independently so that $\Pr(\mathbf{y}_1,\mathbf{y}_2|\mathbf{x}) = \Pr(\mathbf{y}_1|\mathbf{x})\Pr(\mathbf{y}_2|\mathbf{x})$ (see appendix B for the general case where $\Pr(\mathbf{y}_1,\mathbf{y}_2|\mathbf{x})$ does not factorise). D may be shown to satisfy the inequality $D \leq D_1 + D_2$ (see appendix B), where

$$D_1 \equiv \frac{2}{n}\sum_{\mathbf{y}=1}^{\mathbf{m}} \int d\mathbf{x}\,\Pr(\mathbf{x})\,\Pr(\mathbf{y}|\mathbf{x})\,\left\|\mathbf{x}-\mathbf{x}'(\mathbf{y})\right\|^2 \tag{1.12}$$

$$D_2 \equiv \frac{2(n-1)}{n}\int d\mathbf{x}\,\Pr(\mathbf{x})\left\|\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \Pr(\mathbf{y}|\mathbf{x})(\mathbf{x}-\mathbf{x}'(\mathbf{y}))\right\|^2$$

D_1 and D_2 are both non-negative. $D_1 \to 0$ as $n \to \infty$, and $D_2 = 0$ when $n = 0$, so the D_1 term is the sole contribution to the upper bound when $n = 0$, and the D_2 term provides the dominant contribution as $n \to \infty$. The difference between the D_1 and the D_2 terms is the location of the $\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \Pr(\mathbf{y}|\mathbf{x}) (\cdots)$ average: in the D_2 term it averages a vector quantity, whereas in the D_1 term it averages a Euclidean distance. The D_2 term will therefore exhibit interference effects, whereas the D_1 term will not.

E. Probability Leakage

The model may be further extended to the case where the probability that a node fires is a weighted average of the underlying probabilities that the nodes in its vicinity fire. Thus $\Pr(\mathbf{y}|\mathbf{x})$ becomes

$$\Pr(\mathbf{y}|\mathbf{x}) \to \sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} \Pr(\mathbf{y}|\mathbf{y}') \Pr(\mathbf{y}'|\mathbf{x}) \tag{1.13}$$

where $\Pr(\mathbf{y}|\mathbf{y}')$ is the conditional probability that node $\mathbf{y}$ fires given that node $\mathbf{y}'$ would have liked to fire. In a sense, $\Pr(\mathbf{y}|\mathbf{y}')$ describes a "leakage" of probability from node $\mathbf{y}'$ that onto node $\mathbf{y}$. $\Pr(\mathbf{y}|\mathbf{y}')$ then plays the role of a soft "neighbourhood function" for node $\mathbf{y}'$. This expression for $\Pr(\mathbf{y}|\mathbf{x})$ can be used wherever a plain $\Pr(\mathbf{y}|\mathbf{x})$ has been used before. The main purpose of introducing leakage is to encourage neighbouring nodes to perform a similar function. This occurs because the effect of leakage is to soften the posterior probability $\Pr(\mathbf{y}|\mathbf{x})$, and thus reduce the ability to reconstruct $\mathbf{x}$ accurately from knowledge of $\mathbf{y}$, which thus increases the average Euclidean distortion D. To reduce the damage that leakage causes, the optimisation must ensure that nodes that leak probability onto each other have similar properties, so that it does not matter much that they leak.

F. The Model

The focus of this paper is on minimisation of the upper bound $D_1 + D_2$ (see equation 1.12) to D in the multiple firing model, using a scalable posterior probability $\Pr(\mathbf{y}|\mathbf{x})$ (see equation 1.8), with the effect of activity leakage $\Pr(\mathbf{y}|\mathbf{y}')$ taken into account (see equation 1.13). Gathering all of these pieces together yields

$$\begin{aligned} D_1 &= \frac{2}{n\,M} \int d\mathbf{x} \Pr(\mathbf{x}) \sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} \Pr(\mathbf{y}|\mathbf{y}') \sum_{\mathbf{y}'' \in \tilde{N}(\mathbf{y}')} \Pr(\mathbf{y}'|\mathbf{x};\mathbf{y}'') \left\|\mathbf{x} - \mathbf{x}'(\mathbf{y})\right\|^2 \\ D_2 &= \frac{2(n-1)}{n\,M^2} \int d\mathbf{x} \Pr(\mathbf{x}) \left\| \sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} \Pr(\mathbf{y}|\mathbf{y}') \sum_{\mathbf{y}'' \in \tilde{N}(\mathbf{y}')} \Pr(\mathbf{y}'|\mathbf{x};\mathbf{y}'') \left(\mathbf{x} - \mathbf{x}'(\mathbf{y})\right) \right\|^2 \end{aligned} \tag{1.14}$$

where $\Pr(\mathbf{y}|\mathbf{x};\mathbf{y}') \equiv \frac{Q(\mathbf{x}|\mathbf{y})}{\sum_{\mathbf{y}'' \in N(\mathbf{y}')} Q(\mathbf{x}|\mathbf{y}'')}$.

In order to ensure that the model is truly scalable, it is necessary to restrict the dimensionality of the reference vectors. In equation 1.14 $\dim \mathbf{x}'(\mathbf{y}) = \dim \mathbf{x}$, which is not acceptable in a scalable network. In practice, it will be assumed any properties of node $\mathbf{y}$ that are vectors in input space will be limited to occupy an "input window" of restricted size that is centred on node $\mathbf{y}$. This restriction applies to the node reference vector $\mathbf{x}'(\mathbf{y})$, which prevents $D_1 + D_2$ from being fully minimised, because $\mathbf{x}'(\mathbf{y})$ is allowed to move only in a subspace of the full-dimensional input space. However, useful results can nevertheless be obtained, so this restriction is acceptable.

G. Optimisation

Optimisation is achieved by minimising $D_1 + D_2$ with respect to its free parameters. Thus the derivatives with respect to $\mathbf{x}'(\mathbf{y})$ are given by

$$\begin{aligned} \frac{\partial D_1}{\partial \mathbf{x}'(\mathbf{y})} &= -\frac{4}{n\,M} \int d\mathbf{x} \Pr(\mathbf{x})\, \mathbf{f}_1(\mathbf{x},\mathbf{y}) \\ \frac{\partial D_2}{\partial \mathbf{x}'(\mathbf{y})} &= -\frac{4(n-1)}{n\,M^2} \int d\mathbf{x} \Pr(\mathbf{x})\, \mathbf{f}_2(\mathbf{x},\mathbf{y}) \end{aligned} \tag{1.15}$$

and the variations with respect to $Q(\mathbf{x}|\mathbf{y})$ are given by

$$\begin{aligned} \delta D_1 &= \frac{2}{n\,M} \sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \int d\mathbf{x} \Pr(\mathbf{x})\, g_1(\mathbf{x},\mathbf{y})\, \delta \log Q(\mathbf{x}|\mathbf{y}) \\ \delta D_2 &= \frac{4(n-1)}{n\,M^2} \sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \int d\mathbf{x} \Pr(\mathbf{x})\, g_2(\mathbf{x},\mathbf{y})\, \delta \log Q(\mathbf{x}|\mathbf{y}) \end{aligned} \tag{1.16}$$

The functions $\mathbf{f}_1(\mathbf{x},\mathbf{y})$, $\mathbf{f}_2(\mathbf{x},\mathbf{y})$, $g_1(\mathbf{x},\mathbf{y})$, and $g_2(\mathbf{x},\mathbf{y})$ are derived in appendix C. Inserting a sigmoidal function $Q(\mathbf{x}|\mathbf{y}) = \frac{1}{1+\exp(-\mathbf{w}(\mathbf{y})\cdot\mathbf{x}-b(\mathbf{y}))}$ then yields the derivatives with respect to $\mathbf{w}(\mathbf{y})$ and $b(\mathbf{y})$ as

$$\begin{aligned}\frac{\partial D_1}{\partial\begin{pmatrix} b(\mathbf{y}) \\ \mathbf{w}(\mathbf{y})\end{pmatrix}} &= \frac{2}{n\,M}\int d\mathbf{x}\,\Pr(\mathbf{x})\;g_1(\mathbf{x},\mathbf{y})\;(1-Q(\mathbf{x}|\mathbf{y}))\begin{pmatrix}1\\ \mathbf{x}\end{pmatrix} \\ \frac{\partial D_2}{\partial\begin{pmatrix} b(\mathbf{y}) \\ \mathbf{w}(\mathbf{y})\end{pmatrix}} &= \frac{4(n-1)}{n\,M^2}\int d\mathbf{x}\,\Pr(\mathbf{x})\;g_2(\mathbf{x},\mathbf{y})\;(1-Q(\mathbf{x}|\mathbf{y}))\begin{pmatrix}1\\ \mathbf{x}\end{pmatrix}\end{aligned} \tag{1.17}$$

Because all of the properties of node $\mathbf{y}$ that are vectors in input space (i.e. $\mathbf{x}'(\mathbf{y})$ and $\mathbf{w}(\mathbf{y})$) are assumed to be restricted to an input window centred on node $\mathbf{y}$, the eventual result of evaluating the right hand sides of the above equations must be similarly restricted to the same input window.

H. The Effect of the Euclidean Norm on Minimising $D_1 + D_2$

The expressions for D_1 and D_2, and especially their derivatives, are fairly complicated, so an intuitive interpretation will now be presented. When $D_1 + D_2$ is stationary with respect to variations of $\mathbf{x}'(\mathbf{y})$ it may be written as (see appendix D).

$$D_1 + D_2 = -\frac{2}{n}\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\Pr(\mathbf{y}|\mathbf{x})\;\|\mathbf{x}'(\mathbf{y})\|^2 - \frac{2(n-1)}{n}\int d\mathbf{x}\,\Pr(\mathbf{x})\left\|\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\Pr(\mathbf{y}|\mathbf{x})\;\mathbf{x}'(\mathbf{y})\right\|^2 + \text{ constant} \tag{1.18}$$

The M and M^2 factors do not appear in this expression because $\Pr(\mathbf{y}|\mathbf{x})$ is normalised to sum to unity. The first term (which derives from D_1) is an incoherent sum (i.e. a sum of Euclidean distances), whereas the second term (which derives from D_2) is a coherent sum (i.e. a sum of vectors). The first term contributes for all values of n, whereas the second term contributes only for $n \geq 2$, and dominates for $n \gg 1$. In order to minimise the first term the $\|\mathbf{x}'(\mathbf{y})\|^2$ like to be as large as possible for those nodes that have a large $\Pr(\mathbf{y}|\mathbf{x})$. Since $\mathbf{x}'(\mathbf{y})$ is the centroid of the probability density $\Pr(\mathbf{x}|\mathbf{y})$, this implies that node $\mathbf{y}$ prefers to encode a region of input space that is as far as possible from the origin. This is a consequence of using a Euclidean distortion measure $\|\mathbf{x}-\mathbf{x}'\|^2$, which has the dimensions of $\|\mathbf{x}\|^2$, in the original definition of the distortion in equation 1.2. In order to minimise the second term the superposition of $\mathbf{x}'(\mathbf{y})$ weighted by the $\Pr(\mathbf{y}|\mathbf{x})$ likes to have as large a Euclidean norm as possible. Thus the nodes co-operate amongst themselves to ensure that the nodes that have a large $\Pr(\mathbf{y}|\mathbf{x})$ also have a large $\left\|\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\Pr(\mathbf{y}|\mathbf{x})\;\mathbf{x}'(\mathbf{y})\right\|^2$.

II. SOLVABLE ANALYTIC MODEL

The purpose of this section is to work through a case study in order to demonstrate the various properties that emerge when $D_1 + D_2$ is minimised.

A. The Model

It convenient to begin by ignoring the effects of leakage $\Pr(\mathbf{y}|\mathbf{y}')$, and to concentrate on a simple (non-scaling) version of the posterior probability model (as in equation 1.6) $\Pr(\mathbf{y}|\mathbf{x}) = \frac{Q(\mathbf{x}|\mathbf{y})}{\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} Q(\mathbf{x}|\mathbf{y}')}$, where the $Q(\mathbf{x}|\mathbf{y})$ are threshold functions of $\mathbf{x}$

$$Q(\mathbf{x}|\mathbf{y}) = \begin{cases} 0 & \text{below threshold} \\ 1 & \text{above threshold}\end{cases} \tag{2.1}$$

It is also convenient to imagine that a hypothetical infinite-sized training set is available, so it may be described by a probability density $\Pr(\mathbf{x})$. This is a "frequentist", rather than a "Bayesian", use of the $\Pr(\mathbf{x})$ notation, but the distinction is not important in the context of this paper. Assume that $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ is drawn from a training set, that has 2 statistically independent subspaces, so that

$$\Pr(\mathbf{x}_1, \mathbf{x}_2) = \Pr(\mathbf{x}_1)\Pr(\mathbf{x}_2) \tag{2.2}$$

Furthermore, assume that $\Pr(\mathbf{x}_1)$ and $\Pr(\mathbf{x}_2)$ each have the form

$$\Pr(\mathbf{x}_i) = \frac{1}{2\pi}\int_0^{2\pi} d\theta_i\,\delta(\mathbf{x}_i - \mathbf{x}_i(\theta_i)) \tag{2.3}$$

i.e. $\Pr(\mathbf{x}_i)$ is a loop (parameterised by a phase angle θ_i) of probability density that sits in $\mathbf{x}_i$-space. In order to make it easy to deduce the optimum reference vectors, choose $\mathbf{x}_i(\theta_i)$ so that the following 2 conditions are satisfied for $i = 1, 2$

$$\begin{aligned} \|\mathbf{x}_i(\theta_i)\|^2 &= \text{constant} \\ \left\|\frac{\partial \mathbf{x}_i(\theta_i)}{\partial \theta_i}\right\|^2 &= \text{constant} \end{aligned} \tag{2.4}$$

This type of training set can be visualised topologically. Each training vector $(\mathbf{x}_1, \mathbf{x}_2)$ consists of 2 subvectors, each of which is parameterised by a phase angle, and which therefore lives in a subspace that has the topology of a circle, which is denoted as S^1. Because of the independence assumption in equation 2.2, the pair $(\mathbf{x}_1, \mathbf{x}_2)$ lives on the surface of a 2-torus, which is denoted as $S^1 \times S^1$. The minimisation of $D_1 + D_2$ thus reduces to finding the optimum way of designing an encoder/decoder for input vectors that live on a 2-torus, with the proviso that their probability density is uniform (this follows from equation 2.3 and equation 2.4). In order to derive the reference vectors $\mathbf{x}'(\mathbf{y})$, the solution(s) of the stationarity condition $\frac{\partial(D_1+D_2)}{\partial \mathbf{x}'(\mathbf{y})} = 0$ must be computed. The stationarity condition reduces to (see appendix D)

$$n\int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1, \mathbf{x}_2|\mathbf{y})\,(\mathbf{x}_1, \mathbf{x}_2) = (n-1)\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}\left(\int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1, \mathbf{x}_2|\mathbf{y})\Pr(\mathbf{y}'|\mathbf{x}_1, \mathbf{x}_2)\right)(\mathbf{x}_1'(y'), \mathbf{x}_2'(y')) + (\mathbf{x}_1'(y), \mathbf{x}_2'(y)) \tag{2.5}$$

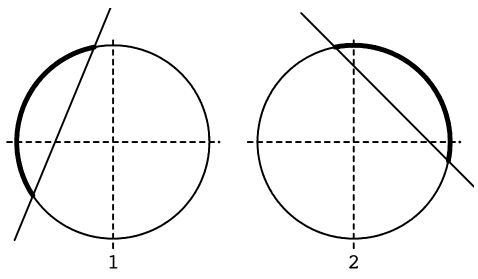


Figure 1: Representation of $S^1 \times S^1$ topology with a threshold $Q(x|y)$ superimposed.

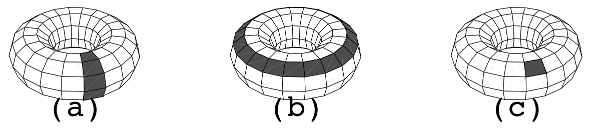


Figure 2: Explicit representation of $S^1 \times S^1$ topology as a torus with the effect of 3 different types of threshold $Q(x|y)$ shown.

It is useful to use the simple diagrammatic notation shown in figure 1. Each circle in figure 1 represents one of the S^1 subspaces, so the two circles together represent the product $S^1 \times S^1$. The constraints in equation 2.4 are represented by each circle being centred on the origin of its subspace ($\|\mathbf{x}_i(\theta_i)\|^2$ is constant), and the probability density around each circle being constant ($\left\|\frac{\partial \mathbf{x}_i(\theta_i)}{\partial \theta_i}\right\|^2$ is constant). A single threshold function $Q(\mathbf{x}|\mathbf{y})$ is represented by a chord cutting through each circle (with 0 and 1 indicating on which side of the chord the threshold is triggered). The $\mathbf{x}_i$ that lie above threshold in each subspace are highlighted. Both $\mathbf{x}_1$ and $\mathbf{x}_2$ must lie above threshold in order to ensure $Q(\mathbf{x}|\mathbf{y}) = 1$, i.e. they must both lie within regions that are highlighted in figure 1. In this case node $\mathbf{y}$ will be said to be "attached" to both subspace 1 and subspace 2. A special case arises when the chord in one of the subspaces (say it is $\mathbf{x}_2$) does not intersect the circle at all, and the circle lies on the side of the chord where the threshold is triggered. In this case $Q(\mathbf{x}|\mathbf{y})$ does not depend on $\mathbf{x}_2$, so that $\Pr(\mathbf{y}|\mathbf{x}_1, \mathbf{x}_2) = \Pr(\mathbf{y}|\mathbf{x}_1)$, in which case node $\mathbf{y}$ will be said to be "attached" to subspace 1 but "detached" from subspace 2. The typical ways in which a node becomes attached to the 2-torus are shown in figure 2. In figure 2(a) the node is attached to one of the S^1 sub-

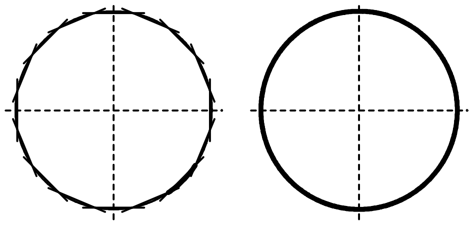


Figure 3: 16 nodes are shown, which are all attached to subspace 1, and all detached from subspace 2.

spaces and detached from the other. In figure 2(b) the attached and detached subspaces are interchanged with respect to figure 2(a). In figure 2(c) the node is attached to both subspaces.

B. All Nodes Attached to One Subspace

Consider the configuration of threshold functions shown in figure 3. This is equivalent to all of the nodes being attached to loops to cover the 2-torus, with a typical node being as shown in figure 2(a) (or, equivalently, figure 2(b)). When $D_1 + D_2$ is minimised, it is assumed that the 4 nodes are symmetrically disposed in subspace 1, as shown. Each is triggered if and only if $\mathbf{x}_1$ lies within its quadrant, and one such quadrant is highlighted in figure 3. This implies that only 1 node is triggered at a time. The assumed form of the threshold functions implies $\Pr(y|\mathbf{x}_1,\mathbf{x}_2) = \Pr(y|\mathbf{x}_1)$, so equation 2.5 reduces to

$$n \int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1|y)\, \Pr(\mathbf{x}_2)\, (\mathbf{x}_1,\mathbf{x}_2) = \int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1|y)\, \Pr(\mathbf{x}_2) \left((n-1) \sum_{y'=1}^{M} \Pr(y'|\mathbf{x}_1) \left(\mathbf{x}_1'(y'), \mathbf{x}_2'(y')\right) + \left(\mathbf{x}_1'(y), \mathbf{x}_2'(y)\right) \right) \quad (2.6)$$

whence

$$\begin{aligned} \mathbf{x}_1'(y) &= \int d\mathbf{x}_1 \Pr(\mathbf{x}_1|y)\, \mathbf{x}_1 \\ \mathbf{x}_2'(y) &= 0 \end{aligned} \quad (2.7)$$

C. All Nodes Attached to Both Subspaces

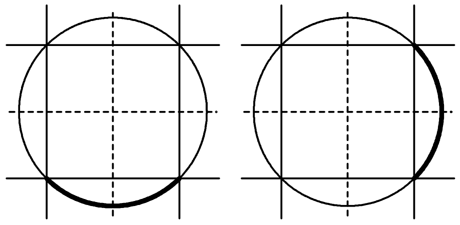


Figure 4: 16 nodes are shown, which are all attached to both subspace 1 and subspace 2.

Consider the configuration of threshold functions shown in figure 4. This is equivalent to all of the nodes being attached to patches to cover the 2-torus, with a typical node being as shown in figure 2(c). In this case, when $D_1 + D_2$ is minimised, it is assumed that each subspace is split into 2 halves. This requires a total of 4 nodes, each of which is triggered if, and only if, both $\mathbf{x}_1$ and $\mathbf{x}_2$ lie on the corresponding half-circles. This implies that only 1 node is triggered at a time. The assumed form of the threshold functions implies that the stationarity condition becomes

$$n \Pr(y) \int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1,\mathbf{x}_2|y)\, (\mathbf{x}_1,\mathbf{x}_2) = \Pr(y) \int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1,\mathbf{x}_2|y) \left((n-1) \sum_{y'=1}^{M} \Pr(y'|\mathbf{x}_1,\mathbf{x}_2) \left(\mathbf{x}_1'(y'), \mathbf{x}_2'(y')\right) + \left(\mathbf{x}_1'(y), \mathbf{x}_2'(y)\right) \right) \quad (2.8)$$

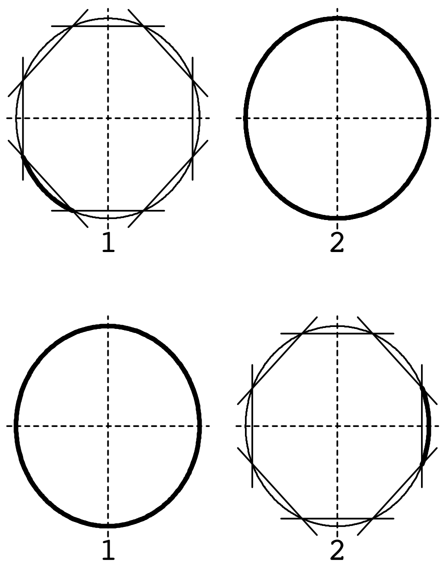


Figure 5: 16 nodes are shown, 8 of which are attached to subspace 1 and detached from subspace 2 (top row), and 8 of which are attached to subspace 2 and detached from subspace 1 (bottom row).

whence

$$\begin{aligned}\mathbf{x}_1'(y) &= \int d\mathbf{x}_1 \Pr(\mathbf{x}_1|y)\, \mathbf{x}_1 \\ \mathbf{x}_2'(y) &= \int d\mathbf{x}_2 \Pr(\mathbf{x}_2|y)\, \mathbf{x}_2 \end{aligned} \quad (2.9)$$

D. Half the Nodes Attached to One Subspace, and Half to the Other Subspace

Consider the configuration of threshold functions shown in figure 5. This is equivalent to half of the nodes being attached to loops to cover the 2-torus, with a typical node being as shown in figure 2(a). The other half of the nodes would then be attached in an analogous way, but as shown in figure 2(b). Thus the 2-torus is covered twice over. In this case, when $D_1 + D_2$ is minimised, it is assumed that each subspace is split into 2 halves. This requires a total of 4 nodes, each of which is triggered if $\mathbf{x}_1$ (or $\mathbf{x}_2$) lies on the half-circle in the subspace to which the node is attached. Thus exactly 2 nodes $y_1(\mathbf{x}_1)$ and $y_2(\mathbf{x}_2)$ are triggered at a time, so that

$$\begin{aligned}\Pr(y|\mathbf{x}_1,\mathbf{x}_2) &= \frac{1}{2}\left(\delta_{y,y_1(\mathbf{x}_1)} + \delta_{y,y_2(\mathbf{x}_2)}\right) \\ &= \frac{1}{2}\left(\Pr(y|\mathbf{x}_1) + \Pr(y|\mathbf{x}_2)\right) \end{aligned} \quad (2.10)$$

For simplicity, assume that node y is attached to subspace 1, then $\Pr(\mathbf{x}_1,\mathbf{x}_2|y) = \Pr(\mathbf{x}_1|y)\Pr(\mathbf{x}_2)$ and the stationarity condition becomes

$$n \Pr(y) \int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1|y)\, \Pr(\mathbf{x}_2)\, (\mathbf{x}_1,\mathbf{x}_2) =$$

$$\Pr(y) \int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1|y)], \Pr(\mathbf{x}_2) \left(\frac{n-1}{2} \sum_{y'=1}^{M} \left(\Pr(y'|\mathbf{x}_1) + \Pr(y'|\mathbf{x}_2)\right) \left(\mathbf{x}_1'(y'), \mathbf{x}_2'(y')\right) + \left(\mathbf{x}_1'(y), \mathbf{x}_2'(y)\right) \right) \quad (2.11)$$

This may be simplified to yield

$$\begin{aligned} n \int d\mathbf{x}_1\, \Pr(\mathbf{x}_1|y)\, (\mathbf{x}_1, 0) &= \frac{n+1}{2} \left(\mathbf{x}_1'(y), \mathbf{x}_2'(y)\right) + \frac{n-1}{2} \int d\mathbf{x}_2\, \Pr(\mathbf{x}_2) \sum_{y'=1}^{M} \Pr(y'|\mathbf{x}_2) \left(\mathbf{x}_1'(y'), \mathbf{x}_2'(y')\right) \\ &= \frac{n+1}{2} \left(\mathbf{x}_1'(y), \mathbf{x}_2'(y)\right) + \frac{n-1}{2} \left\langle \mathbf{x}_1'(y), \mathbf{x}_2'(y) \right\rangle_2 \end{aligned} \quad (2.12)$$

Write the 2 subspaces separately (remember that node y is assumed to be attached to subspace 1)

$$\begin{aligned} \mathbf{x}_1'(y) &= \frac{2n}{n+1} \int d\mathbf{x}_1\, \Pr(\mathbf{x}_1|y)\, \mathbf{x}_1 - \frac{n-1}{n+1} \left\langle \mathbf{x}_1'(y) \right\rangle_2 \\ \mathbf{x}_2'(y) &= -\frac{n-1}{n+1} \left\langle \mathbf{x}_2'(y) \right\rangle_2 \end{aligned} \quad (2.13)$$

If this result is simultaneously solved with the analogous result for node y attached to subspace 2, then the $\langle\cdots\rangle$ terms vanish to yield

$$\begin{aligned}\mathbf{x}_1'(y) &= \begin{cases}\frac{2n}{n+1}\int d\mathbf{x}_1\ \Pr(\mathbf{x}_1|y)\ \mathbf{x}_1 & y \text{ attached to subspace 1}\\ 0 & y \text{ attached to subspace 2}\end{cases}\\ \mathbf{x}_2'(y) &= \begin{cases}0 & y \text{ attached to subspace 1}\\ \frac{2n}{n+1}\int d\mathbf{x}_2\ \Pr(\mathbf{x}_2|y)\ \mathbf{x}_2 & y \text{ attached to subspace 2}\end{cases}\end{aligned} \tag{2.14}$$

E. Compare $D_1 + D_2$ for the 3 Different Types of Solution

Consider the left hand side of figure 3 for the case of M nodes, when the M threshold functions form a regular M-ogon. $\Pr(\mathbf{x}|y)$ then denotes the part of the circle that is associated with node y, whose radius of gyration squared is given by (assuming that the circle has unit radius)

$$\begin{aligned}R_M &\equiv \left\|\int d\mathbf{x}\ \Pr(\mathbf{x}|y)\ \mathbf{x}\right\|^2\\ &= \left(\frac{M}{2\pi}\int_0^{\frac{2\pi}{M}} d\theta\ \cos\theta\right)^2\\ &= \left(\frac{M}{2\pi}\sin\left(\frac{2\pi}{M}\right)\right)^2\end{aligned} \tag{2.15}$$

Gather the results for $(\mathbf{x}_1'(y), \mathbf{x}_2'(y))$ in equations 2.7 (referred to as type 1), 2.9 (referred to as type 2), and 2.14 (referred to as type 3) together and insert them into $D_1 + D_2$ in equation 1.18 to obtain (see appendix E)

$$D_1 + D_2 = \begin{cases}\text{constant} - 2R_M & \text{type 1}\\ \text{constant} - 4R_{\sqrt{M}} & \text{type 2}\\ \text{constant} - \frac{4n}{n+1}R_{\frac{M}{2}} & \text{type 3}\end{cases} \tag{2.16}$$

In figure 6 the 3 solutions are plotted for the case $n = 1$. For $n = 1$ the type 3 solution is never optimal, the type 1 solution is optimal for $M \leq 19$, and the type 2 solution is optimal for $M \geq 20$. This behaviour is intuitively sensible, because a larger number of nodes is required to cover a 2-torus as shown in figure 2(c) than as shown in figure 2(a) (or figure 2(b)).

In figure 7 the 3 solutions are plotted for the case $n = 2$. For $n = 2$ the type 1 solution is optimal for $M \leq 12$, and the type 2 solution is optimal for large $M \geq 30$, but there is now an intermediate region $12 \leq M \leq 29$ (type 1 and type 3 have an equal $D_1 + D_2$ at $M = 12$) where the n-dependence of the type 3 solution has now made it optimal. Again, this behaviour is intuitively reasonable, because the type 3 solution requires at least 2 observations in order to be able to yield a small Euclidean resonstruction error in each of the 2 subspaces, i.e. for $n = 2$ the 2 nodes that fire must be attached to different subspaces. Note that in the type 3 solution the nodes that fire are not guaranteed to be attached to different subspaces. In the type 3 solution there is a probability $\frac{1}{2^n}\frac{n!}{n_1!\,n_2!}$ that n_i (where $n = n_1 + n_2$) nodes are attached to subspace i, so the trend is for the type 3 solution to become more favoured as n is increased.

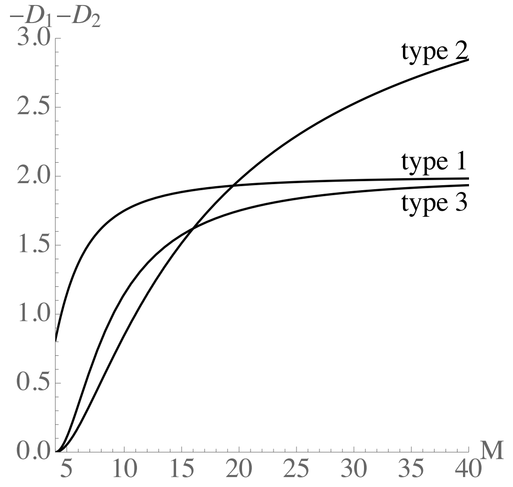


Figure 6: Plots of $-D_1 - D_2$ for $n = 1$ for each of the 3 types of optimum.

In figure 8 the 3 solutions are plotted for the case $n \to \infty$. For $n \to \infty$ the type 2 solution is never optimal , the type 1 solution is optimal for $M \leq 8$, and the type 3 solution is optimal for $M \geq 8$. The type 2 solution approaches the type 3 solution from below asymptotically as $M \to \infty$. In figure 9 a phase diagram is given which shows how the relative stability of the 3 types of solution for different M and n, where the type 3 solution is seen to be optimal over a large part of the (M, n) plane. Thus the most interesting, and commonly occurring, solution is the one in which half the nodes are attached to one subspace and half to the other subspace (i.e. solution type 3). Although this result has been derived using the non-scaling version of the posterior probability model $\Pr(\mathbf{y}|\mathbf{x})$ (as in equation 1.6), it may also be used for scaling posterior probabilities (as given in equation 1.8) in certain limiting cases, and also for cases where the effect of leakage $\Pr(\mathbf{y}|\mathbf{y}')$ is small.

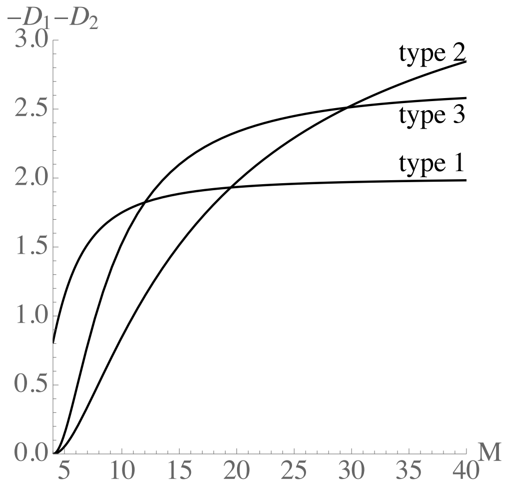


Figure 7: Plots of $-D_1 - D_2$ for $n = 2$ for each of the 3 types of optimum.

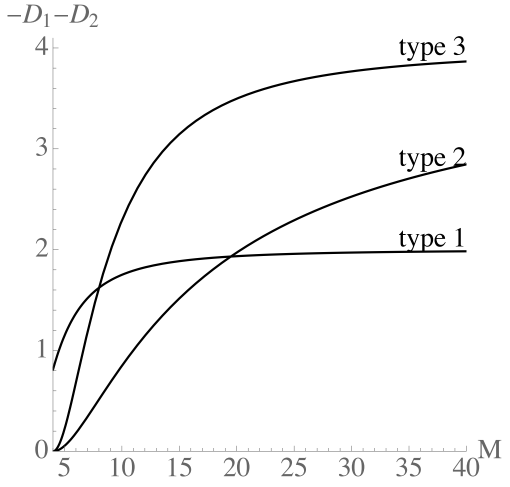


Figure 8: Plots of $-D_1 - D_2$ for $n \to \infty$ for each of the 3 types of optimum.

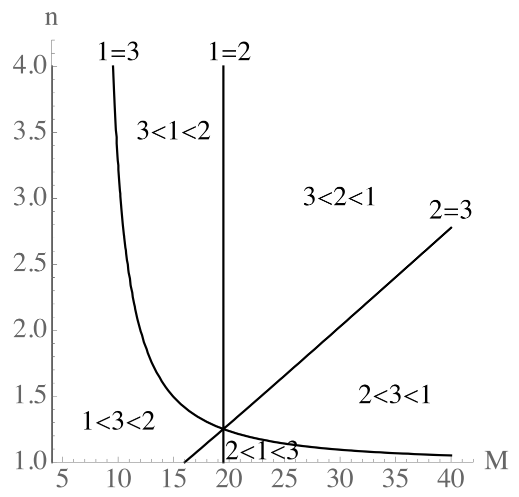


Figure 9: Phase diagram of the regions in which the 3 types of solution are optimal.

F. Various Extensions

1. The Effect of Leakage

The effect of leakage will not be analysed in detail here. However, its effect may readily be discussed phenomenologically, because the optimisation acts to minimise the damaging effect of leakage on the posterior probability by ensuring that the properties of nodes that are connected by leakage are similar. This has the most dramatic effect on the type 3 solution, where the way in which the nodes are partitioned into 2 halves must be very carefully chosen in order to minimise the damage due to leakage. If the leakage is presumed to be a local function, so that $\Pr(y|y') = \pi(y - y')$, which is a localised "blob"-shaped function, then the properties of adjacent node are similar (after optimisation). Since nodes that are attached to 2 different subspaces necessarily have very different properties, whereas nodes that are attached to the same subspace can have similar properties, it follows that the nodes must split into 2 continguous halves, where nodes $1, 2, \cdots, \frac{M}{2}$ are attached to subspace 1 and nodes $\frac{M}{2}+1, \frac{M}{2}+2, \cdots, M$ are attached to subspace 2, or vice versa. The effect of leakage is thereby minimised, with the worst effect occurring at the boundary between the 2 halves of nodes.

2. Modifying the Posterior Probability to Become Scalable

The above analysis has focussed on the non-scaling version of the posterior probability, in which all M nodes act together as a unit. The more general scaling case where the M nodes are split up by the effect of the neighbourhood function $N(y)$ will not be analysed in detail, because many of its properties are essentially the same as in the non-scaling case. For simplicity assume that the neighbourhood function $N(y)$ is a "top-hat" with width w (an odd integer) centred on y. Impose periodic boundary conditions so that the inverse neighbourhood function $\tilde{N}(y)$ is also a top-hat, $\tilde{N}(y) = N(y)$. In this case an optimum solution in the non-scaling case (with $M = w$) can be directly related to a corresponding optimum solution in the scaling case by simply repeating the node properties periodically every w nodes. Strictly speaking, higher order periodicities can also occur in the scaling case (and can be favoured under certain conditions), where the period is $\frac{w}{k}$ (k is an integer), but these will not be discussed here.

The effect of the periodic replication of node properties is interesting. The type 3 solution (with leakage and with $M = w$) splits the nodes into 2 halves, where nodes $1, 2, \cdots, \frac{w}{2}$ are attached to subspace 1 and nodes $\frac{w}{2}+1, \frac{w}{2}+2, \cdots, w$ are attached to subspace 2, or vice versa. When this is replicated periodically every w nodes it produces an alternating structure of node properties, where $\frac{w}{2}$ nodes are attached to subspace 1, then the next $\frac{w}{2}$ nodes are attached to subspace 2, and thenthe next $\frac{w}{2}$ nodes are attached to subspace 1, and so on. This behaviour is reminiscent of the so-called "dominance stripes" that are observed in the mammalian visual cortex.

III. EXPERIMENT

The purpose of this section is to demonstrate the emergence of the dominance stripes in numerical simulations. The main body of the software is concerned with evaluating the derivatives of $D_1 + D_2$, and the main difficulty is choosing an appropriate form for the leakage (this has not yet been automated).

A. The Parameters

The parameters that are required for a simulation are as follows:

1. (m_1, m_2): size of 2D rectangular array of nodes. $M = m_1 m_2$.

2. (i_1, i_2): size of 2D rectangular input window for each node (odd integers). Ensure that the input window is not too many input data "correlation areas" in size, otherwise dominance stripes may not emerge. Dominance stripes require that the correlation *within* an input window are substantially stronger than the correlations *between* input windows that are attached to different subspaces.

3. (w_1, w_2): size of 2D rectangular neighbourhood window for each node (odd integers). The neighbourhood function $N(y_1, y_2)$ is a rectangular top-hat centred on (y_1, y_2). The size of the neighbourhood window has to lie within a limited range to ensure that dominance stripes are produced. This corresponds to ensuring that M lies in the type 3 region of the phase diagram in figure 9. It is also preferable for the size of the neighbourhood window to be substantially smaller than the input window, otherwise different parts of a neighbourhood window will see different parts of the input data, which will make the network behaviour more difficult to interpret.

4. (l_1, l_2): size of 2D rectangular leakage window for each node (odd integers). For simplicity the leakage $\Pr\left(\mathbf{y}|\mathbf{y}'\right)$ is assumed to be given by $\Pr\left(\mathbf{y}|\mathbf{y}'\right) = \pi(\mathbf{y} - \mathbf{y}')$, where $\pi(\mathbf{y} - \mathbf{y}')$ is a "top-hat" function of $\mathbf{y} - \mathbf{y}'$ which covers a rectangular region of size (l_1, l_2) centred on $\mathbf{y} - \mathbf{y}' = 0$. The size of the leakage window must be large enough to correlate the parameters of adjacent nodes, but not so large that it enforces such strong correlations between the node parameters that it destroys dominance stripes.

5. ν: additive noise level used to corrupt each member of the training set.

6. κ: wavenumber of sinusoids used in the training set. In describing the training sets the index y will be used to denote position in input space, thus position y in input space lies directly "under" node y of the network. In 1D simulations each training vector is a sinusoid of the form $\sin(\kappa y + \phi) + r$, where ϕ is a random phase angle, and r is a random number sampled uniformly from the interval $\left[-\frac{\nu}{2}, \frac{\nu}{2}\right]$ – this generates an S^1 topology training set (i.e. parameterised by 1 random angle). In 2D simulations each training vector is a sinusoid of the form $\sin(\kappa\,(y_1 \cos\theta + y_2 \sin\theta) + \phi) + r$, where the additional angle θ is a random azimuthal orientation for the sine wave – this generates an $S^1 \times S^1$ topology training set (i.e. parameterised by 2 independent random angles). Note that κi_1 and κi_2 must be an integer multiple of 2π in order to ensure that the probability density around the S^1 subspace generated by ϕ has uniform density (in effect, the S^1 then becomes a circular Lissajous figure, which therefore has uniform probability density, unlike non-circular Lissajous figures), and thus to ensure that there are no artefacts induced by the periodicity of the training data that might mimic the effect of dominance stripes. If $\kappa\, i_1$ and $\kappa\, i_2$ are much greater than 2π then it is not necessary to fix them to be integer multiples of 2π – because the fluctuations in the probability density are then negligible. Note that this restriction on the value of κ would not have been necessary had complex exponentials been used rather than sinusoids.

7. s: number of subspaces. This fixes the number of statistically independent subspaces in the training set. When $s = 1$ the training set is generated exactly as above. When $s = 2$ the training set is split up as follows. The 1D case has even y in one subspace, and odd y in the other subspace, thus successive components of each training vector alternate between the 2 subspaces. The 2D case has even $y_1 + y_2$ in one subspace, and odd $y_1 + y_2$ in the other subspace, thus each training vector is split up into a chessboard pattern of interlocking subspaces. This strategy readily generalises for $s \geq 3$, although this is not used here. Within each subspace the training vector is generated as above, and the subspaces are generated so that they are statistically independent.

8. ε: update parameter used in gradient descent. This is used to update parameters thus

$$\text{parameter} \to \text{parameter} - \varepsilon \, \frac{\partial \left(D_1 + D_2 \right)}{\partial \text{parameter}} \tag{3.1}$$

There are 3 internally generated update parameter, which control the update of the 3 different types of parameter, i.e. the biases, the weights, and the reference vectors. This is necessary because these parameters all have different dimensionalities, and by inspection of equation 3.1 the dimensionality of an update parameter is the dimensionality of the parameter it updates (squared) divided by the dimensionality of the Euclidean distortion. These 3 internal parameters are automatically adjusted to ensure that the average change in absolute value of each of the 3 types of parameter is equal to ε times the typical diameter of the region of parameter space populated by the parameters. This adjustment is made anew as each training vector is presented. The size of ε determines the "memory time" of the node parameters. This memory time determines the effective number of training vectors that the nodes are being optimised against, and thus must be sufficiently long (i.e. ε sufficiently small) that if $s \geq 2$ it is possible to discern that the subspaces are indeed statistically independent. This is crucially important, for dominance stripes cannot be obtained if the subspaces are not sufficiently statistically independent. So ε must be small, which unfortunately leads to correspondingly long training times.

B. Initialisation

The training set is globally translated and scaled so that the components of all of its training vectors lie in the interval $[-1, +1]$. There are 3 parameter types to initialise. The weights were all initialised to random numbers sampled from a uniform distribution in the interval $[-0.1, +0.1]$, whereas the biasses and the reference vector components were all initialised to 0. Because the 2D simulations took a very long time to run, they were periodically interrupted and the state of all the variables written to an output file. The simulation could then be continued by reading this output file in again and simply continuing where the simulation left off. Alternatively, some of the variables might have their values changed before continuing. In particular, the random number generator could thus be manipulated to simulate the effect of a finite sized training set (i.e. use the *same* random number seed at the start of each part of the simulation), or an infinite-sized training set (i.e. use a *different* random number seed at the start of each part of the simulation). The size of the ε parameter could also thus be manipulated should a large value be required initially, and reduced to a small value later on, as required in order to guarantee that when $n \geq 2$ the input subspaces are seen to be statistically independent, and dominance stripes may emerge.

C. Boundary Conditions

There are many ways to choose the boundary conditions. In the numerical simulations periodic boundary conditions will be avoided, because they can lead to artefacts in which the node parameters become topologically trapped. For instance, in a 2D simulation, periodic boundary conditions imply that the nodes sit on a 2-torus. Leakage implies that the node parameter values are similar for adjacent nodes, which limits the freedom for the parameters to adjust their values on the surface of the 2-torus. For instance, any acceptable set of parameters that sits on the 2-torus can be converted into another acceptable set by mapping the 2-torus to itself, so that each of its S^1 "coils up" an integer number of times onto itself. Such a multiply wrapped parameter configuration is topologically trapped, and cannot be perturbed to its original form. This problem does not arise with non-periodic boundary conditions.

There are several different problems that arise at the boundaries of the array of nodes:

1. The neighbourhood function $N(y_1, y_2)$ cannot be assumed to be a rectangular top-hat centred on (y_1, y_2). Instead, it will simply be truncated so that it does not fall off the edge of array of nodes, i.e. $N(y_1, y_2) = 0$ for those (y_1, y_2) that lie outside the array.

2. The leakage function $\pi(y_1 - y_1', y_2 - y_2')$ will be similarly truncated. However, in this case $\pi(y_1 - y_1', y_2 - y_2')$ must normalise to unity when summed over (y_1, y_2), so the effect of the truncation must be compensated by scaling the remaining elements of $\pi(y_1 - y_1', y_2 - y_2')$.

3. The input window for each node implies that the input array must be larger than the node array in order that the input windows never fall off the edge of the input array.

D. Presentation of Results

The most important result is the emergence of dominance stripes. For $n = 2$ there are thus 2 numbers that need to be displayed for each node: the "degree of attachment" to subspace 1, and similarly for subspace 2. There are many ways to measure degree of attachment, for instance the probability density $\Pr(\mathbf{x}|\mathbf{y})$ gives a direct measurement of how strongly node $\mathbf{y}$ depends on the input vector $\mathbf{x}$, so its "width" or "volume" in each

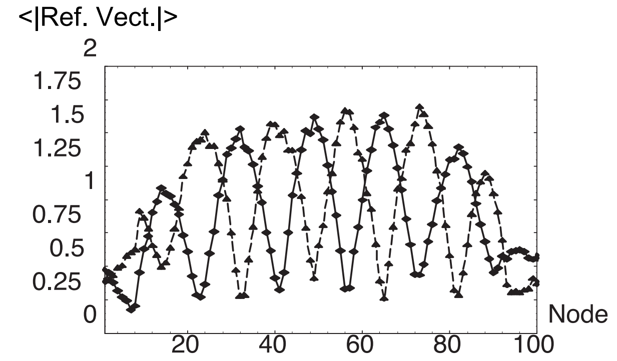


Figure 10: Dominance plots for a 1D simulation with 2 statistically independent training set subspaces.

of the subspaces could be used to measure degree of attachment. However, in the simulations presented here (i.e. sinusoidal training vectors) the degree of attachment is measured as the average of the absolute values of the components of the reference vector in the subspace concerned. This measure tends to zero for complete detachment. For 1D simulations 2 dominance plots can be overlaid to show the dominance of subspaces 1 and 2 for each node. For 2D simulations it is simplest to present only 1 of these plots as a 2D array of grey-scale pixels, where the grey level indicates the dominance of subspace 1 (or, alternatively, subspace 2).[1]

[1] The results of the 2D simulations do not appear in this draft paper.

E. 1D Simulation

The parameter values used were: $(m_1, m_2) = (1, 100)$, $(i_1, i_2) = (1, 41)$, $(w_1, w_2) = (1, 21)$, $(l_1, l_2) = (1, 15)$, $\kappa = 0.3$, $\nu = 0.1$, $s = 2$, $n = 400$, $\varepsilon = 0.002$. This value of κ implies $\frac{\kappa i_2}{2\pi} \simeq 1.96$, so κ is approximately an integer multiple of 2π, as required for an artefact-free simulation. In figure 10 a plot of the 2 dominance curves obtained after 3200 training updates is shown. This dominance plot clearly shows alternating regions where subspace 1 dominates and subspace 2 dominates. The width of the neighbourhood function is 21, which is the same the period of the variations in the dominance plots, i.e. within each set of adjacent 21 nodes half the nodes are attached to subspace 1 and half to subspace 2. There are boundary effects, but these are unimportant.

Appendix A: Normalisation of $\Pr(\mathbf{y}|\mathbf{x})$

The normalisation of the expression for $\Pr(\mathbf{y}|\mathbf{x})$ in equation 1.8 may be demonstrated as follows:

$$\begin{aligned}
\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \Pr(\mathbf{y}|\mathbf{x}) &= \frac{1}{M}\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} Q(\mathbf{x}|\mathbf{y}) \sum_{\mathbf{y}'\in\tilde{N}(\mathbf{y})} \frac{1}{\sum_{\mathbf{y}''\in N(\mathbf{y}')} Q(\mathbf{x}|\mathbf{y}'')} \\
&= \frac{1}{M}\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} \sum_{\mathbf{y}\in N(\mathbf{y}')} Q(\mathbf{x}|\mathbf{y}) \frac{1}{\sum_{\mathbf{y}''\in N(\mathbf{y}')} Q(\mathbf{x}|\mathbf{y}'')} \\
&= \frac{1}{M}\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} 1 \\
&= \frac{m_1 m_2 \cdots m_d}{M} \\
&= 1 \qquad \text{(A1)}
\end{aligned}$$

In the first step the order of the $\mathbf{y}$ and the $\mathbf{y}'$ summations is interchanged using $\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \sum_{\mathbf{y}'\in\tilde{N}(\mathbf{y})}(\cdots) = \sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} \sum_{\mathbf{y}\in N(\mathbf{y}')}(\cdots)$, in the second step the numerator and denominator of the summand cancel out.

Appendix B: Upper Bound for Multiple Firing Model

It is possible to simplify equation 1.10 by using the following identity

$$\mathbf{x} - \mathbf{x}'(\mathbf{y}_1, \mathbf{y}_2, \cdots \mathbf{y}_n) \equiv \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x} - \mathbf{x}'(\mathbf{y}_i)) + \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}'(\mathbf{y}_i) - \mathbf{x}'(\mathbf{y}_1, \mathbf{y}_2, \cdots \mathbf{y}_n)) \qquad \text{(B1)}$$

Note that this holds for all choices of $\mathbf{x}'(\mathbf{y}_i)$. This allows the Euclidean distance to be expanded thus

$$\begin{aligned}\left\|\mathbf{x}-\mathbf{x}'(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)\right\|^2 &= \frac{1}{n^2}\sum_{i=1}^{n}\left\|\mathbf{x}-\mathbf{x}'(\mathbf{y}_i)\right\|^2+\frac{1}{n^2}\sum_{\substack{i,j=1\\ i\neq j}}^{n}(\mathbf{x}-\mathbf{x}'(\mathbf{y}_i))\cdot(\mathbf{x}-\mathbf{x}'(\mathbf{y}_j))\\ &= -\frac{2}{n^2}\sum_{i,j=1}^{n}(\mathbf{x}-\mathbf{x}'(\mathbf{y}_i))(\mathbf{x}'(\mathbf{y}_j)-\mathbf{x}'(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n))\\ &\quad+\frac{1}{n^2}\left\|\sum_{i=1}^{n}(\mathbf{x}'(\mathbf{y}_i)-\mathbf{x}'(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n))\right\|^2\end{aligned} \quad \text{(B2)}$$

Each term of this expansion can be inserted into equation 1.10 to yield

$$\begin{aligned}\text{term 1} &= \frac{1}{n}\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\int d\mathbf{x}\,\Pr(\mathbf{x})\,\Pr(\mathbf{y}|\mathbf{x})\,\left\|\mathbf{x}-\mathbf{x}'(\mathbf{y})\right\|^2\\ \text{term 2} &= \frac{n-1}{n}\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{\mathbf{y}_1,\mathbf{y}_2=\mathbf{1}}^{\mathbf{m}}\Pr(\mathbf{y}_1,\mathbf{y}_2|\mathbf{x})\,(\mathbf{x}-\mathbf{x}'(\mathbf{y}_1)).(\mathbf{x}-\mathbf{x}'(\mathbf{y}_2))\\ \text{term 3} &= -2\times\text{term 4}\\ \text{term 4} &= \sum_{\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n=\mathbf{1}}^{\mathbf{m}}\Pr(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)\left\|\mathbf{x}'(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)-\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}'(\mathbf{y}_i)\right\|^2\end{aligned} \quad \text{(B3)}$$

$\Pr(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n|\mathbf{x})$ has been assumed to be a symmetric function of $(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)$ in the first two results, and the definition of $\mathbf{x}'(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)$ in equation 1.11 has been used to obtain the third result. These results allow D in equation 1.10 to be expanded as $D=D_1+D_2-D_3$, where

$$\begin{aligned}D_1 &\equiv \frac{2}{n}\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\int d\mathbf{x}\,\Pr(\mathbf{x})\,\Pr(\mathbf{y}|\mathbf{x})\,\left\|\mathbf{x}-\mathbf{x}'(\mathbf{y})\right\|^2\\ D_2 &\equiv \frac{2(n-1)}{n}\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{\mathbf{y}_1,\mathbf{y}_2=\mathbf{1}}^{\mathbf{m}}\Pr(\mathbf{y}_1,\mathbf{y}_2|\mathbf{x})\,(\mathbf{x}-\mathbf{x}'(\mathbf{y}_1)).(\mathbf{x}-\mathbf{x}'(\mathbf{y}_2))\\ D_3 &\equiv 2\sum_{\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n=\mathbf{1}}^{\mathbf{m}}\Pr(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)\left\|\mathbf{x}'(\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_n)-\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}'(\mathbf{y}_i)\right\|^2\end{aligned} \quad \text{(B4)}$$

By noting that $D_3\geq 0$, an upper bound for D in the form $D\leq D_1+D_2$ follows immediately from these results. Note that $D_1\geq 0$ whereas D_2 can have either sign. In the special case where $\Pr(\mathbf{y}_1,\mathbf{y}_2|\mathbf{x})=\Pr(\mathbf{y}_1|\mathbf{x})\,\Pr(\mathbf{y}_1|\mathbf{x})$ (i.e. $\mathbf{y}_1$ and $\mathbf{y}_2$ are independent of each other given that x is known) D_2 reduces to

$$D_2\equiv\frac{2(n-1)}{n}\int d\mathbf{x}\,\Pr(\mathbf{x})\left\|\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\Pr(\mathbf{y}|\mathbf{x})\,(\mathbf{x}-\mathbf{x}'(\mathbf{y}))\right\|^2 \quad \text{(B5)}$$

which is manifestly positive. This is the form of D_2 that is used throughout this paper.

Appendix C: Derivatives of the Objective Function

D_1 and D_2 are as given in equation 1.12, i.e. it is assumed that $\Pr(\mathbf{y}_1,\mathbf{y}_2|\mathbf{x})=\Pr(\mathbf{y}_2|\mathbf{x})\Pr(\mathbf{y}_2|\mathbf{x})$ and $\Pr(\mathbf{y}|\mathbf{x})$ has the scalable form given in equation 1.8. Define a compact matrix notation as follows

$$\begin{array}{ll} L_{\mathbf{y},\mathbf{y}'}\equiv\Pr(\mathbf{y}'|\mathbf{y}) & P_{\mathbf{y},\mathbf{y}'}\equiv\Pr(\mathbf{y}'|\mathbf{x};\mathbf{y})\\ p_{\mathbf{y}}\equiv\sum_{\mathbf{y}'\in\tilde{N}(\mathbf{y})}P_{\mathbf{y}',\mathbf{y}} & \left(L^T p\right)_{\mathbf{y}}\equiv\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}L_{\mathbf{y}',\mathbf{y}}\,p_{\mathbf{y}'}\\ \mathbf{d}_{\mathbf{y}}\equiv\mathbf{x}-\mathbf{x}'(\mathbf{y}) & (L\,\mathbf{d})_{\mathbf{y}}\equiv\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}L_{\mathbf{y},\mathbf{y}'}\,\mathbf{d}_{\mathbf{y}'}\\ (P\,L\,\mathbf{d})_{\mathbf{y}}\equiv\sum_{\mathbf{y}'\in N(\mathbf{y})}P_{\mathbf{y},\mathbf{y}'}\,(L\,\mathbf{d})_{\mathbf{y}'} & \left(P^T\,P\,L\,\mathbf{d}\right)_{\mathbf{y}}\equiv\sum_{\mathbf{y}'\in\tilde{N}(\mathbf{y})}P_{\mathbf{y}',\mathbf{y}}\,(P\,L\,\mathbf{d})_{\mathbf{y}'}\\ e_{\mathbf{y}}\equiv\left\|\mathbf{x}-\mathbf{x}'(\mathbf{y})\right\|^2 & (L\,e)_{\mathbf{y}}\equiv\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}L_{\mathbf{y},\mathbf{y}'}\,e_{\mathbf{y}'}\\ (P\,L\,e)_{\mathbf{y}}\equiv\sum_{\mathbf{y}'\in N(\mathbf{y})}P_{\mathbf{y},\mathbf{y}'}\,(L\,e)_{\mathbf{y}'} & \left(P^T\,P\,L\,e\right)_{\mathbf{y}}\equiv\sum_{\mathbf{y}'\in\tilde{N}(\mathbf{y})}P_{\mathbf{y}',\mathbf{y}}\,(P\,L\,e)_{\mathbf{y}'}\\ \bar{\mathbf{d}}\equiv\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\left(L^T p\right)_{\mathbf{y}}\,\mathbf{d}_{\mathbf{y}} & \text{or }\bar{\mathbf{d}}\equiv\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}(P\,L\,\mathbf{d})_{\mathbf{y}}\end{array} \quad \text{(C1)}$$

Using this matrix notation, the functions $\mathbf{f}_1(\mathbf{x},\mathbf{y})$, $\mathbf{f}_2(\mathbf{x},\mathbf{y})$, $g_1(\mathbf{x},\mathbf{y})$, and $g_2(\mathbf{x},\mathbf{y})$ may be defined as

$$\begin{aligned}
\mathbf{f}_1(\mathbf{x},\mathbf{y}) &\equiv \left(L^T p\right)_{\mathbf{y}} \mathbf{d}_{\mathbf{y}} && \text{(C2)}\\
\mathbf{f}_2(\mathbf{x},\mathbf{y}) &\equiv \left(L^T p\right)_{\mathbf{y}} \bar{\mathbf{d}} && \text{(C3)}\\
g_1(\mathbf{x},\mathbf{y}) &\equiv p_{\mathbf{y}}\left(L\,e\right)_{\mathbf{y}} - \left(P^T P\,L\,e\right)_{\mathbf{y}} && \text{(C4)}\\
g_2(\mathbf{x},\mathbf{y}) &\equiv \left(p_{\mathbf{y}}\left(L\,\mathbf{d}\right)_{\mathbf{y}} - \left(P^T P\,L\,\mathbf{d}\right)_{\mathbf{y}}\right)\cdot\bar{\mathbf{d}} && \text{(C5)}
\end{aligned}$$

The variation of $\Pr(\mathbf{y}|\mathbf{x};\mathbf{y}')$ is then given by

$$\begin{aligned}
\delta\Pr(\mathbf{y}|\mathbf{x};\mathbf{y}') &= \Pr(\mathbf{y}|\mathbf{x};\mathbf{y}')\left(\delta\log Q(\mathbf{x}|\mathbf{y}) - \sum_{\mathbf{y}''\in N(\mathbf{y}')}\Pr(\mathbf{y}''|\mathbf{x};\mathbf{y}')\,\delta\log Q(\mathbf{x}|\mathbf{y}'')\right)\\
&= \Pr(\mathbf{y}|\mathbf{x};\mathbf{y}')\sum_{\mathbf{y}''\in N(\mathbf{y}')}\delta\log Q(\mathbf{x}|\mathbf{y}'')\left(\delta_{\mathbf{y}'',\mathbf{y}} - \Pr(\mathbf{y}''|\mathbf{x};\mathbf{y}')\right)\\
&= P_{\mathbf{y}',\mathbf{y}}\sum_{\mathbf{y}''\in N(\mathbf{y}')}\delta\log Q(\mathbf{x}|\mathbf{y}'')\left(\delta_{\mathbf{y}'',\mathbf{y}} - P_{\mathbf{y}',\mathbf{y}''}\right) && \text{(C6)}
\end{aligned}$$

In order to rearrange the expressions to ensure that only a single dummy index is required at every stage of evaluation of the sums it will be necessary to use the result

$$\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\sum_{\mathbf{y}'\in N(\mathbf{y})} = \sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}\sum_{\mathbf{y}\in\tilde{N}(\mathbf{y}')} \tag{C7}$$

1. Calculate $\frac{\partial D_1}{\partial \mathbf{x}'(\mathbf{y})}$.

The derivative is given by

$$\frac{\partial D_1}{\partial \mathbf{x}'(\mathbf{y})} = -\frac{4}{n\,M}\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}\Pr(\mathbf{y}|\mathbf{y}')\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')}\Pr(\mathbf{y}'|\mathbf{x};\mathbf{y}'')\left(\mathbf{x}-\mathbf{x}'(\mathbf{y})\right) \tag{C8}$$

Use matrix notation to write this as

$$\frac{\partial D_1}{\partial \mathbf{x}'(\mathbf{y})} = -\frac{4}{n\,M}\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}L_{\mathbf{y}',\mathbf{y}}\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')}P_{\mathbf{y}'',\mathbf{y}'}\,\mathbf{d}_{\mathbf{y}} \tag{C9}$$

Finally remove the explicit summations to obtain the required result

$$\begin{aligned}
\frac{\partial D_1}{\partial \mathbf{x}'(\mathbf{y})} &= -\frac{4}{n\,M}\int d\mathbf{x}\,\Pr(\mathbf{x})\left(L^T p\right)_{\mathbf{y}}\mathbf{d}_{\mathbf{y}}\\
&= -\frac{4}{n\,M}\int d\mathbf{x}\,\Pr(\mathbf{x})\,\mathbf{f}_1(\mathbf{x},\mathbf{y}) && \text{(C10)}
\end{aligned}$$

2. Calculate $\frac{\partial D_2}{\partial \mathbf{x}'(\mathbf{y})}$.

The derivative is given by

$$\begin{aligned}
\frac{\partial D_2}{\partial \mathbf{x}'(\mathbf{y})} &= -\frac{4(n-1)}{n\,M^2}\int d\mathbf{x}\,\Pr(\mathbf{x})\left(\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}\Pr(\mathbf{y}|\mathbf{y}')\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')}\Pr(\mathbf{y}'|\mathbf{x};\mathbf{y}'')\right)\\
&\quad\times\left(\sum_{\bar{\mathbf{y}}=\mathbf{1}}^{\mathbf{m}}\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}\Pr(\bar{\mathbf{y}}|\mathbf{y}')\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')}\Pr(\mathbf{y}'|\mathbf{x};\mathbf{y}'')\left(\mathbf{x}-\mathbf{x}'(\bar{\mathbf{y}})\right)\right) && \text{(C11)}
\end{aligned}$$

Use matrix notation to write this as

$$\frac{\partial D_2}{\partial \mathbf{x}'(\mathbf{y})} = -\frac{4(n-1)}{n\,M^2}\int d\mathbf{x}\,\Pr(\mathbf{x})\left(\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} L_{\mathbf{y}',\mathbf{y}} \sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')} P_{\mathbf{y}'',\mathbf{y}'}\right)\left(\sum_{\bar{\mathbf{y}}=\mathbf{1}}^{\mathbf{m}}\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} L_{\mathbf{y}',\bar{\mathbf{y}}} \sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')} P_{\mathbf{y}'',\mathbf{y}'}\,\mathbf{d}_{\bar{\mathbf{y}}}\right) \tag{C12}$$

Finally remove the explicit summations to obtain the required result

$$\begin{aligned}\frac{\partial D_2}{\partial \mathbf{x}'(\mathbf{y})} &= -\frac{4(n-1)}{n\,M^2}\int d\mathbf{x}\,\Pr(\mathbf{x})\,\left(L^T p\right)_{\mathbf{y}}\,\bar{\mathbf{d}} \\ &= -\frac{4(n-1)}{n\,M^2}\int d\mathbf{x}\,\Pr(\mathbf{x})\,\mathbf{f}_2(\mathbf{x},\mathbf{y})\end{aligned} \tag{C13}$$

3. Calculate $\frac{\delta D_1}{\delta \log Q(\mathbf{x}|\mathbf{y})}$

The differential is given by

$$\delta D_1 = \frac{2}{n\,M}\sum_{\mathbf{y}=1}^{\mathbf{m}}\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}\Pr(\mathbf{y}|\mathbf{y}')\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')}\delta\Pr(\mathbf{y}'|\mathbf{x};\mathbf{y}'')\,\left\|\mathbf{x}-\mathbf{x}'(\mathbf{y})\right\|^2 \tag{C14}$$

Use matrix notation to write this as

$$\delta D_1 = \frac{2}{n\,M}\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} L_{\mathbf{y}',\mathbf{y}}\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')} P_{\mathbf{y}'',\mathbf{y}'}\sum_{\mathbf{y}'''\in N(\mathbf{y}'')}\delta\log Q(\mathbf{x}|\mathbf{y}''')\,\left(\delta_{\mathbf{y}''',\mathbf{y}'} - P_{\mathbf{y}'',\mathbf{y}'''}\right)\,e_{\mathbf{y}} \tag{C15}$$

Reorder the summations to obtain

$$\delta D_1 = \frac{2}{n\,M}\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{\mathbf{y}'''=\mathbf{1}}^{\mathbf{m}}\delta\log Q(\mathbf{x}|\mathbf{y}''')\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}''')}\left(\sum_{\mathbf{y}'\in N(\mathbf{y}'')}\delta_{\mathbf{y}''',\mathbf{y}'}\,P_{\mathbf{y}'',\mathbf{y}'} - P_{\mathbf{y}'',\mathbf{y}'''}\sum_{\mathbf{y}'\in N(\mathbf{y}'')} P_{\mathbf{y}'',\mathbf{y}'}\right)\sum_{\mathbf{y}=1}^{\mathbf{m}} L_{\mathbf{y}',\mathbf{y}}\,e_{\mathbf{y}} \tag{C16}$$

Relabel the indices and evaluate the sum over the Kronecker delta to obtain

$$\begin{aligned}\delta D_1 &= \frac{2}{n\,M}\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\int d\mathbf{x}\,\Pr(\mathbf{x})\,\delta\log Q(\mathbf{x}|\mathbf{y}) \\ &\quad\times\left(\left(\sum_{\mathbf{y}'\in\tilde{N}(\mathbf{y})} P_{\mathbf{y}',\mathbf{y}}\right)\left(\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} L_{\mathbf{y},\mathbf{y}'}\,e_{\mathbf{y}'}\right) - \sum_{\mathbf{y}'''\in\tilde{N}(\mathbf{y})} P_{\mathbf{y}''',\mathbf{y}}\sum_{\mathbf{y}''\in N(\mathbf{y}''')} P_{\mathbf{y}''',\mathbf{y}''}\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} L_{\mathbf{y}'',\mathbf{y}'}\,e_{\mathbf{y}'}\right)\end{aligned} \tag{C17}$$

Finally remove the explicit summations to obtain the required result

$$\begin{aligned}\delta D_1 &= \frac{2}{n\,M}\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\int d\mathbf{x}\,\Pr(\mathbf{x})\,\delta\log Q(\mathbf{x}|\mathbf{y})\,\left(p_{\mathbf{y}}\,(L\,e)_{\mathbf{y}} - (P^T\,P\,L\,e)_{\mathbf{y}}\right) \\ &= \frac{2}{n\,M}\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\int d\mathbf{x}\,\Pr(\mathbf{x})\,g_1(\mathbf{x},\mathbf{y})\,\delta\log Q(\mathbf{x}|\mathbf{y})\end{aligned} \tag{C18}$$

4. Calculate $\frac{\delta D_2}{\delta \log Q(\mathbf{x}|\mathbf{y})}$

The differential is given by

$$\begin{aligned}\delta D_2 &= \frac{4\,(n-1)}{n\,M^2}\int d\mathbf{x}\;\Pr(\mathbf{x})\\ &\times\left(\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}\Pr(\mathbf{y}|\mathbf{y}')\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')}\Pr(\mathbf{y}'|\mathbf{x};\mathbf{y}'')\,(\mathbf{x}-\mathbf{x}'(\mathbf{y}))\right)\\ &\cdot\left(\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}\Pr(\mathbf{y}|\mathbf{y}')\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')}\delta\Pr(\mathbf{y}'|\mathbf{x};\mathbf{y}'')\,(\mathbf{x}-\mathbf{x}'(\mathbf{y}))\right)\end{aligned}\tag{C19}$$

Use matrix notation to write this as

$$\begin{aligned}\delta D_2 &= \frac{4\,(n-1)}{n\,M^2}\int d\mathbf{x}\;\Pr(\mathbf{x})\left(\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}L_{\mathbf{y}',\mathbf{y}}\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')}P_{\mathbf{y}'',\mathbf{y}'}\,\mathbf{d}_{\mathbf{y}}\right)\\ &\cdot\left(\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}L_{\mathbf{y}',\mathbf{y}}\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')}P_{\mathbf{y}'',\mathbf{y}'}\sum_{\mathbf{y}'''\in N(\mathbf{y}'')}\delta\log Q(\mathbf{x}|\mathbf{y}''')\,\left(\delta_{\mathbf{y}''',\mathbf{y}'}-P_{\mathbf{y}'',\mathbf{y}'''}\right)\mathbf{d}_{\mathbf{y}}\right)\end{aligned}\tag{C20}$$

Reorder the summations to obtain

$$\begin{aligned}\delta D_2 &= \frac{4\,(n-1)}{n\,M^2}\int d\mathbf{x}\;\Pr(\mathbf{x})\left(\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}L_{\mathbf{y}',\mathbf{y}}\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')}P_{\mathbf{y}'',\mathbf{y}'}\,\mathbf{d}_{\mathbf{y}}\right)\\ &\cdot\left(\sum_{\mathbf{y}'''=\mathbf{1}}^{\mathbf{m}}\delta\log Q(\mathbf{x}|\mathbf{y}''')\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}''')}\left(\sum_{\mathbf{y}'\in N(\mathbf{y}'')}\delta_{\mathbf{y}''',\mathbf{y}'}\,P_{\mathbf{y}'',\mathbf{y}'}-P_{\mathbf{y}'',\mathbf{y}'''}\sum_{\mathbf{y}'\in N(\mathbf{y}'')}P_{\mathbf{y}'',\mathbf{y}'}\right)\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}L_{\mathbf{y}',\mathbf{y}}\,\mathbf{d}_{\mathbf{y}}\right)\end{aligned}\tag{C21}$$

Relabel the indices and evaluate the sum over the Kronecker delta to obtain

$$\begin{aligned}\delta D_2 &= \frac{4\,(n-1)}{n\,M^2}\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\int d\mathbf{x}\;\Pr(\mathbf{x})\;\delta\log Q(\mathbf{x}|\mathbf{y})\\ &\times\left(\left(\sum_{\mathbf{y}'\in\tilde{N}(\mathbf{y})}P_{\mathbf{y}',\mathbf{y}}\right)\left(\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}L_{\mathbf{y},\mathbf{y}'}\,\mathbf{d}_{\mathbf{y}'}\right)-\left(\sum_{\mathbf{y}'''\in\tilde{N}(\mathbf{y})}P_{\mathbf{y}''',\mathbf{y}}\sum_{\mathbf{y}''\in N(\mathbf{y}''')}P_{\mathbf{y}''',\mathbf{y}''}\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}L_{\mathbf{y}'',\mathbf{y}'}\,\mathbf{d}_{\mathbf{y}'}\right)\right)\\ &\cdot\left(\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}}L_{\mathbf{y}',\mathbf{y}}\sum_{\mathbf{y}''\in\tilde{N}(\mathbf{y}')}P_{\mathbf{y}'',\mathbf{y}'}\,\mathbf{d}_{\mathbf{y}}\right)\end{aligned}\tag{C22}$$

Finally remove the explicit summations to obtain the required result

$$\begin{aligned}\delta D_2 &= \frac{4\,(n-1)}{n\,M^2}\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\int d\mathbf{x}\;\Pr(\mathbf{x})\;\delta\log Q(\mathbf{x}|\mathbf{y})\left(p_{\mathbf{y}}\,(L\,\mathbf{d})_{\mathbf{y}}-\left(P^T\,P\,L\,\mathbf{d}\right)_y\right)\cdot\bar{\mathbf{d}}\\ &= \frac{4\,(n-1)}{n\,M^2}\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}}\int d\mathbf{x}\;\Pr(\mathbf{x})\;g_2(\mathbf{x},\mathbf{y})\;\delta\log Q(\mathbf{x}|\mathbf{y})\end{aligned}\tag{C23}$$

Appendix D: Expression for $D_1 + D_2$ in Terms of $\mathbf{x}'(c)$

From equation 1.12 $D_1 + D_2$ can be written as

$$\begin{aligned} D_1 + D_2 &= \frac{2}{n} \int d\mathbf{x} \, \Pr(\mathbf{x}) \sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \Pr(\mathbf{y}|\mathbf{x}) \left(\|\mathbf{x}'(\mathbf{y})\|^2 - 2\mathbf{x} \cdot \mathbf{x}'(\mathbf{y}) \right) \\ &+ \frac{2(n-1)}{n} \int d\mathbf{x} \, \Pr(\mathbf{x}) \left(\left\| \sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \Pr(\mathbf{y}|\mathbf{x}) \, \mathbf{x}'(\mathbf{y}) \right\|^2 - 2 \left(\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \Pr(\mathbf{y}|\mathbf{x}) \, \mathbf{x}'(\mathbf{y}) \right) \cdot \left(\sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \Pr(\mathbf{y}|\mathbf{x}) \, \mathbf{x} \right) \right) \\ &+ \text{constant} \end{aligned} \tag{D1}$$

where the constant terms do not depend on $\mathbf{x}'(\mathbf{y})$. However, from equation 1.12 the derivative $\frac{\partial(D_1+D_2)}{\partial \mathbf{x}'(\mathbf{y})}$ can be written as

$$\frac{\partial(D_1 + D_2)}{\partial \mathbf{x}'(\mathbf{y})} = -\frac{4}{n} \int d\mathbf{x} \, \Pr(\mathbf{x}) \, \Pr(\mathbf{y}|\mathbf{x}) \left(\mathbf{x} - \mathbf{x}'(\mathbf{y}) + (n-1) \sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} \Pr(\mathbf{y}'|\mathbf{x}) \, (\mathbf{x} - \mathbf{x}'(\mathbf{y}')) \right) \tag{D2}$$

Using Bayes' theorem the stationarity condition $\frac{\partial(D_1+D_2)}{\partial \mathbf{x}'(\mathbf{y})} = 0$ yields a matrix equation for the $\mathbf{x}'(\mathbf{y})$

$$n \int d\mathbf{x} \, \Pr(\mathbf{x}|\mathbf{y}) \, \mathbf{x} = (n-1) \sum_{\mathbf{y}'=\mathbf{1}}^{\mathbf{m}} \left(\int d\mathbf{x} \, \Pr(\mathbf{x}|\mathbf{y}) \, \Pr(\mathbf{y}'|\mathbf{x}) \right) \mathbf{x}'(\mathbf{y}') + \mathbf{x}'(\mathbf{y}) \tag{D3}$$

which may then be used to replace all instances of $\mathbf{x}$ in equation D1. This yields the result

$$D_1 + D_2 = -\frac{2}{n} \int d\mathbf{x} \, \Pr(\mathbf{x}) \sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \Pr(\mathbf{y}|\mathbf{x}) \, \|\mathbf{x}'(\mathbf{y})\|^2 - \frac{2(n-1)}{n} \int d\mathbf{x} \, \Pr(\mathbf{x}) \left\| \sum_{\mathbf{y}=\mathbf{1}}^{\mathbf{m}} \Pr(\mathbf{y}|\mathbf{x}) \, \mathbf{x}'(\mathbf{y}) \right\|^2 + \text{constant} \tag{D4}$$

Appendix E: Comparison of $D_1 + D_2$ for Different Types of Optima

In order to compare the value of D_1+D_2 that is obtained when different types of supposedly optimum configurations of the threshold functions $Q(\mathbf{x}|\mathbf{y})$ are tried, the $\mathbf{x}'(\mathbf{y})$ that solves $\frac{\partial(D_1+D_2)}{\partial \mathbf{x}'(\mathbf{y})} = 0$ (see appendix D) must be inserted into the expression for $D_1 + D_2$. In the following derivations the constant term is omitted, and the definition $R_M \equiv \left\| \int d\mathbf{x} \Pr(\mathbf{x}|y) \, \mathbf{x} \right\|^2$ (see equation 2.15) has been used.

1. Type 1 Optimum: all the nodes are attached one subspace

In equation 1.18 $D_1 + D_2$ becomes

$$\begin{aligned} D_1 + D_2 &= -\frac{2}{n} \int d\mathbf{x} \, \Pr(\mathbf{x}) \sum_{y=1}^{M} \Pr(y|\mathbf{x}) \left\| \left(\int d\mathbf{x}_1 \, \Pr(\mathbf{x}_1|y) \, \mathbf{x}_1, 0 \right) \right\|^2 \\ &\quad - \frac{2(n-1)}{n} \int d\mathbf{x} \, \Pr(\mathbf{x}) \left\| \sum_{y=1}^{M} \Pr(y|\mathbf{x}) \left(\int d\mathbf{x}_1 \, \Pr(\mathbf{x}_1|y) \, \mathbf{x}_1, 0 \right) \right\|^2 \\ &= -\frac{2}{n} R_M - \frac{2(n-1)}{n} R_M \\ &= -2R_M \end{aligned} \tag{E1}$$

2. Type 2 Optimum: all the nodes are attached both subspaces

In equation 1.18 $D_1 + D_2$ becomes

$$\begin{aligned}
D_1 + D_2 &= -\frac{2}{n} \int d\mathbf{x} \, \Pr(\mathbf{x}) \sum_{y=1}^{M} \Pr(y|\mathbf{x}) \left\| \left(\int d\mathbf{x}_1 \, \Pr(\mathbf{x}_1|y) \, \mathbf{x}_1, \int d\mathbf{x}_2 \, \Pr(\mathbf{x}_2|y) \, \mathbf{x}_2 \right) \right\|^2 \\
&\quad -\frac{2(n-1)}{n} \int d\mathbf{x} \, \Pr(\mathbf{x}) \left\| \sum_{y=1}^{M} \Pr(y|\mathbf{x}) \left(\int d\mathbf{x}_1 \, \Pr(\mathbf{x}_1|y) \, \mathbf{x}_1, \int d\mathbf{x}_2 \, \Pr(\mathbf{x}_2|y) \, \mathbf{x}_2 \right) \right\|^2 \\
&= -\left(\frac{2}{n} + \frac{2(n-1)}{n} \right) 2R_{\sqrt{M}} \\
&= -4R_{\sqrt{M}} \qquad \text{(E2)}
\end{aligned}$$

3. Type 3 Optimum: half the nodes are attached one subspace and half are attached to the other

In equation 1.18 $D_1 + D_2$ becomes

$$\begin{aligned}
D_1 + D_2 &= -\frac{2}{n} \left(\frac{2n}{n+1} \right)^2 \int d\mathbf{x} \, \Pr(\mathbf{x}) \\
&\quad \times \left(\sum_{y=1}^{\frac{M}{2}} \Pr(y|\mathbf{x}) \left\| \left(\int d\mathbf{x}_1 \, \Pr(\mathbf{x}_1|y) \, \mathbf{x}_1, 0 \right) \right\|^2 + \sum_{y=\frac{M}{2}+1}^{M} \Pr(y|\mathbf{x}) \left\| \left(0, \int d\mathbf{x}_2 \, \Pr(\mathbf{x}_2|y) \, \mathbf{x}_2 \right) \right\|^2 \right) \\
&\quad -\frac{2(n-1)}{n} \left(\frac{2n}{n+1} \right)^2 \int d\mathbf{x} \, \Pr(\mathbf{x}) \\
&\quad \times \left\| \left(\sum_{y=1}^{\frac{M}{2}} \Pr(y|\mathbf{x}) \int d\mathbf{x}_1 \, \Pr(\mathbf{x}_1|y) \, \mathbf{x}_1, \sum_{y=\frac{M}{2}+1}^{M} \Pr(y|\mathbf{x}) \int d\mathbf{x}_2 \, \Pr(\mathbf{x}_2|y) \, \mathbf{x}_2 \right) \right\|^2 \\
&= -\left(\frac{2n}{n+1} \right)^2 \left(2 \, \frac{1}{2} \, \frac{2}{n} + 2 \left(\frac{1}{2} \right)^2 \frac{2(n-1)}{n} \right) R_{\frac{M}{2}} \\
&= -\frac{4n}{n+1} R_{\frac{M}{2}} \qquad \text{(E3)}
\end{aligned}$$

[1] S P Luttrell, *A Bayesian analysis of self-organising maps*, Neural Computation **6** (1994), no. 5, 767–794.

[2] ———, *Partitioned mixture distribution: an adaptive Bayesian network for low-level image processing*, IEE Proceedings on Vision, Image, and Signal Processing **141** (1994), no. 4, 251–260.

The Development of Dominance Stripes and Orientation Maps in a Self-Organising Visual Cortex Network (VICON) [*]

S P Luttrell[†]
Defence Research Agency, St Andrews Road, Malvern, Worcs, WR14 3PS, United Kingdom, tel: +44 (0) 1684 894046, fax: +44 (0) 1684 894384

A self-organising neural network is presented that is based on a rigorous Bayesian analysis of the information contained in individual neural firing events. This leads to a visual cortex network (VICON) that has many of the properties emerge when a mammalian visual cortex is exposed to data arriving from two imaging sensors (i.e. the two retinae), such as dominance stripes and orientation maps.

I. INTRODUCTION

The overall goal of this work is to automate as far as is possible the processing of data from multiple sensors (data fusion), which includes the automatic design of the architecture and functionality of the network(s) that do the processing. In [10] a novel approach to this automation problem was introduced, and the purpose of this paper is to refine and extend the previously reported results.

The problem of automating the design of a data fusion network has many interesting special case solutions. In particular, the type of self-organising neural network (in the mammalian visual cortex) that processes the images arriving from a pair of retinae is one such special case, where the number of sensors involved is just two. For a review of visual cortex neural network models see [2, 13].

The basic idea is to use a soft encoder (i.e. its output is a distributed code in which more than one, and possibly all, of the output neurons is active) to transform the input vector (i.e. the input image) into a posterior probability over various possible classes (i.e. alternative possible interpretations of the input vector), and to optimise the encoder so that this posterior probability is able to retain as much information as possible about the input vector, as measured in the minimum mean square reconstruction error (i.e. L_2 error) sense [8, 12].

In the special case where the optimisation is performed over the space of all possible soft encoders, the optimum solution is a hard encoder (i.e. it is a "winner-take-all" network in which only one of the output neurons is active) which is an optimal vector quantiser (VQ), of the type described in [4], for encoding the input vector with minimum L_2 error. In the slightly less special case where the space of possible soft encoders is restricted to include only those whose output is deliberately damaged by the effects of a noise process, this produces a different type of hard encoder which is an optimal self-organising map (SOM) for encoding the input vector with minimum L_2 error; this is very closely related to the well-known Kohonen map [3], as was demonstrated in [5].

This paper will examine yet another special case, where the optimisation is performed over a very special subspace of soft encoders, rather than over all possible soft encoders. The behaviour of each soft encoder is modelled by a set of posterior probabilities over various possible classes. When a particular parametric form for these posterior probabilities is chosen, a corresponding subspace of possible soft encoders is thus automatically selected, which may be explored by varying the parameters. The parametric form of the posterior probability that is used in this paper is based on the so-called partitioned mixture distribution (PMD) [7, 9], which is a natural generalisation of the standard mixture distribution to a high-dimensional input space.

This use of a PMD leads to a 2-layer visual cortex network (VICON), where the components of the input vector are the output activities of the input neurons, and the components of the PMD posterior probability are the output activities of the output neurons. Various physically realistic constraints are placed on the PMD optimisation (both on the internal PMD structure, and on the type of training data that is used), and these will be described in the text as they arise.

The layout of this paper is as follows. In section II all of the necessary theoretical machinery is developed, including folded Markov chains, posterior probability models, derivatives of the objective function, and receptive fields. In section III the concepts of dominance stripes and orientation maps are explained, both in the context of the elastic net model, and in the context of theory presented in this paper. In section IV the results of computer simulations are presented, including both 1 and 2-dimensional retinae, single and pairs of retinae, both for synthetic and natural training data. In appendix B some explicit optimal solutions that minimise the objective function are derived, including the periodicity property of some types of optimal solution.

[*]Typeset in LaTeX on May 9, 2019.
This is archived at http://arxiv.org/abs/1012.3724, 16 Dec 2010.
This paper was submitted to Network on 6 November 1996. Paper reference number NET/79294/PAP. It was not accepted for publication, but it underpins several subsequently published papers.

[†]Electronic address: luttrell@signal.dra.hmg.gb

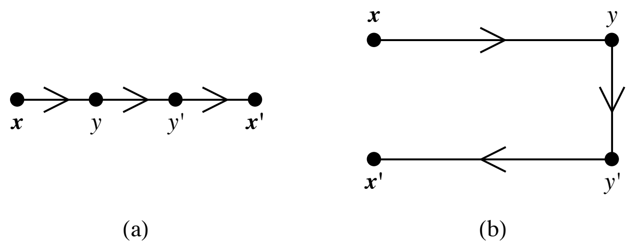


Figure 1: (a) A Markov chain of transitions $\mathbf{x} \to y \to y' \to \mathbf{x}'$. (b) The same diagram as (a), but folded.

II. THEORY

This section covers all of the basic theoretical machinery that is required to design and train a 2-layer VICON. In section II A the theory of folded Markov chains (FMC) is summarised. In section II B the basic idea of a posterior probability model is introduced, and in section II C this is developed into a full partitioned posterior probability model. In section II D the derivatives of the FMC objective function are derived assuming a partitioned posterior probability model, and in section II E the influence of finite-sized receptive fields on these derivatives is derived.

A. Folded Markov Chain

The basis of the entire theoretical treatment is a communication channel model [6] in which an input vector $\mathbf{x}$ is encoded to produce a conditional probability $\Pr(y|\mathbf{x})$ over code indices y, which is then transmitted along a distorting communication channel to produce a conditional probability $\Pr(y'|y)$ over distorted code indices y', which is finally decoded to produce a conditional PDF $\Pr(\mathbf{x}'|y')$ over reconstructions $\mathbf{x}'$ of the original input vector $\mathbf{x}$. The three steps in the sequence $\mathbf{x} \to y \to y' \to \mathbf{x}'$ are modelled by the conditional probabilities $\Pr(y|\mathbf{x})$, $\Pr(y'|y)$, and $\Pr(\mathbf{x}'|y')$, which describe a Markov chain of transitions, which is shown diagrammatically in figure 1(a). $\Pr(\mathbf{x}'|y)$ is completely determined from other defined quantities by using Bayes' theorem in the form $\Pr(\mathbf{x}|y) = \frac{\Pr(\mathbf{x})\Pr(y|\mathbf{x})}{\int d\mathbf{x}' \Pr(\mathbf{x}')\Pr(y|\mathbf{x}')}$.

Because $\mathbf{x}$ and $\mathbf{x}'$ live in the same vector space it is convenient to fold this diagram to produce figure 1(b); this is called a folded Markov chain (FMC) [6]. Figure 1(b) is directly related to a 2-layer unsupervised neural network, where $\mathbf{x}$ and $\mathbf{x}'$ represent the activity pattern of the whole set of neurons in the input layer, and y and y' represent the location(s) of winning neuron(s) in the ouput layer. The overall conditional PDF generated by an FMC is $\Pr(\mathbf{x}'|\mathbf{x})$, which is obtained by marginalising y and y' in the joint probability $\Pr(\mathbf{x}', y', y|\mathbf{x}) = \Pr(\mathbf{x}'|y')\Pr(y'|y)\Pr(y|\mathbf{x})$.

Define a network objective function D as [6]

$$D = \int d\mathbf{x}\, d\mathbf{x}' \Pr(\mathbf{x}) \Pr(\mathbf{x}'|\mathbf{x}) \left\|\mathbf{x} - \mathbf{x}'\right\|^2 \tag{2.1}$$

which measures the expected Euclidean (or L_2) reconstruction error caused by feeding input vectors sampled from $\Pr(\mathbf{x})$ into the FMC, where each $\mathbf{x}$ is returned as a PDF $\Pr(\mathbf{x}'|\mathbf{x})$ of alternative reconstructions $\mathbf{x}'$ of $\mathbf{x}$. For simplicity, assume that the communication channel has been assumed to be distortionless so that $\Pr(y'|y) = \delta_{yy'}$, and that $y = 1, 2, \cdots, M$, then

$$D = \sum_{y=1}^{M} \int d\mathbf{x}\, d\mathbf{x}' \Pr(\mathbf{x}) \Pr(y|\mathbf{x}) \Pr(\mathbf{x}'|y) \left\|\mathbf{x} - \mathbf{x}'\right\|^2 \tag{2.2}$$

An FMC is completely described by the form of its encoder $\Pr(y|\mathbf{x})$ and the form of its reconstruction error $\left\|\mathbf{x} - \mathbf{x}'\right\|^2$. The functional form of the encoder may be chosen arbitrarily, and independently of the assumed Euclidean form of the reconstruction error, so the FMC does *not* correspond to a Gaussian mixture distribution model in input space. This is a general result for FMCs in which the functional forms of the encoder and the reconstruction error may be independently chosen. It is only when these functional forms are carefully chosen that a density model interpretation of an FMC is possible (for instance a Euclidean reconstruction error $\left\|\mathbf{x} - \mathbf{x}'\right\|^2$ must be paired with an encoder $\Pr(y|\mathbf{x})$ that describes the posterior probability over class labels that would arise in a Gaussian mixture distribution model).

The expression for D given in equation 2.2 may be simplified to yield [6] (this readily generalises to the case where $\Pr(y'|y) \neq \delta_{yy'}$ (i.e. the communication channel causes distortion))

$$D = 2 \int d\mathbf{x} \Pr(\mathbf{x}) \sum_{y=1}^{M} \Pr(y|\mathbf{x}) \left\|\mathbf{x} - \mathbf{x}'(y)\right\|^2 \tag{2.3}$$

where $\mathbf{x}'(y)$ is a reference vector defined as

$$\mathbf{x}'(y) \equiv \int d\mathbf{x} \Pr(\mathbf{x}|y)\, \mathbf{x} \tag{2.4}$$

If this definition of $\mathbf{x}'(y)$ is not used, and instead D in equation 2.3 is minimised with respect to $\mathbf{x}'(y)$, then the stationary solution is $\mathbf{x}'(y) = \int d\mathbf{x} \Pr(\mathbf{x}|y)\, \mathbf{x}$, which is consistent with the definition in equation 2.4. In practice, it is better to determine the stationary $\mathbf{x}'(y)$ by following the gradient $\frac{\partial D}{\partial \mathbf{x}'(y)}$ than to use the explicit expression $\int d\mathbf{x} \Pr(\mathbf{x}|y)\, \mathbf{x}$ for the stationary point, because $\frac{\partial D}{\partial \mathbf{x}'(y)}$ is cheap to evaluate whereas $\int d\mathbf{x} \Pr(\mathbf{x}|y)\, \mathbf{x}$ is expensive to evaluate. In effect, the $\frac{\partial D}{\partial \mathbf{x}'(y)}$ approach is an example of on-line training, whereas the $\int d\mathbf{x} \Pr(\mathbf{x}|y)\, \mathbf{x}$ approach is the corresponding example of batch training, and the on-line and batch approaches each have their own areas where they are best used.

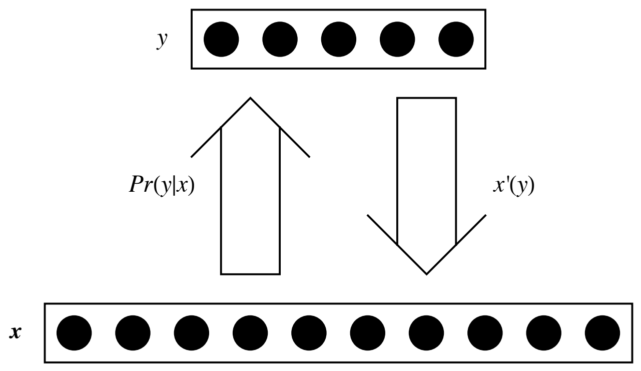


Figure 2: A neural network representation of a folded Markov chain.

In equation 2.3 $\Pr(y|\mathbf{x})$ is a "recognition model" (i.e. it takes an input vector and recognises by assigning to it a posterior probability over class labels) and $\mathbf{x}'(y)$ is the corresponding "generative model" (i.e. it takes a class label and generates a corresponding vector in input space). This is a simpler type of generative model than appeared in the original expression for D in equation 2.2, where the generative model is $\Pr(\mathbf{x}'|y)$, which generates a whole distribution of possible vectors in input space, rather than just a single vector which is the centroid of $\Pr(\mathbf{x}'|y)$. The transformation of the FMC from one that uses the PDF $\Pr(\mathbf{x}'|y)$ into one that uses the reference vector $\mathbf{x}'(y)$ is not possible in general; it was made possible here by choosing to use a Euclidean reconstruction error in D. In general, an FMC reconstruction is a distribution over alternative inputs, rather than a single representative input, as might be used in decision theory, for instance.

The operation of the various terms in the expression for D in equation 2.3 is shown in figure 2, which is rotated through 90° anticlockwise with respect to the corresponding diagram in figure 1, and also for simplicity $y' = y$ because $\Pr(y'|y) = \delta_{yy'}$ was assumed above. When D is minimised with respect to the choice of encoder $\Pr(y|\mathbf{x})$ and reconstruction vector $\mathbf{x}'(y)$ it yields a standard minimum mean square error vector quantiser (VQ) with M code indices [4], and if $\Pr(y'|y) \neq \delta_{yy'}$ then the VQ produces code indices that carry information in such a way that it is maximally robust with respect to the damaging effects of communication channel distortion modelled by $\Pr(y'|y)$ [5]. This latter type of VQ can be shown to be approximately equivalent to a self-organising map (SOM) of the type introduced by Kohonen [3].

B. Basic Posterior Probability (Single Recognition Model)

The minimisation procedure that leads to a VQ-like optimum assumed that the entire space of posterior probability functions $\Pr(y|\mathbf{x})$ was available to be searched. In the neural network interpretation, $\Pr(y|\mathbf{x})$ models the probability that neuron y fires first (this encompasses both the case of a soft encoder where more than one neuron can potentially fire first, and the case of a hard encoder where only one neuron can potentially fire first; this is the winner-take-all case), which depends on the detailed underlying dynamics of how all of the neurons interact with each other. Because these neural dynamics are not arbitrary (e.g. they are constrained to be a physically realisable process), it constrains the space of possible posterior probabilities $\Pr(y|\mathbf{x})$ that is available to the neural network. $\Pr(y|\mathbf{x})$ may then be modelled by the functional form

$$\Pr(y|\mathbf{x}) \equiv \frac{Q(\mathbf{x}|y)}{\sum_{y'=1}^{M} Q(\mathbf{x}|y')} \tag{2.5}$$

where $Q(\mathbf{x}|y)$ is the raw "response function" of neuron y.

$Q(\mathbf{x}|y)$ may be interpreted as the raw firing rate of neuron y, and $\Pr(y|\mathbf{x})$ is then the probability that neuron y fires first out of all of the M competing neurons. This functional form makes it clear that there is a type of lateral inhibition occurring between $\Pr(y_1|\mathbf{x})$ and $\Pr(y_2|\mathbf{x})$ (for $y_1 \neq y_2$), because if the raw firing rate $Q(\mathbf{x}|y_1)$ is *increased* so that $\Pr(y_1|\mathbf{x})$ increases, nevertheless the denominator $\sum_{y'=1}^{M} Q(\mathbf{x}|y')$ ensures that $\Pr(y_2|\mathbf{x})$ *decreases* (for $y_1 \neq y_2$); i.e. the $Q(\mathbf{x}|y)$ do not exhibit lateral inhibition, but the $\Pr(y_1|\mathbf{x})$ do exhibit lateral inhibition.

The raw receptive field of a neuron depends on the form of $Q(\mathbf{x}|y)$. Thus if the functional form of $Q(\mathbf{x}|y)$ depends only on a subset $\tilde{\mathbf{x}}(y)$ of components of $\mathbf{x}$, then $\tilde{\mathbf{x}}(y)$ is the raw receptive field of neuron y. However, this is *not* the same as the the receptive field that is effective in producing the first firing event, because $\Pr(y|\mathbf{x})$ depends on all of the $\tilde{\mathbf{x}}(y')$ (for $y' = 1, \cdots, M$) as shown in equation 2.5.

The effect of the distortion $\Pr(y|y')$ process, as modelled by $\Pr(y|y')$, is to alter at the last minute, as it were, the probability that each neuron fires first. Thus the posterior probability is modified as follows

$$\Pr(y|\mathbf{x}) \to \sum_{y'=1}^{M} \Pr(y|y') \Pr(y'|\mathbf{x}) \tag{2.6}$$

where the matrix element $\Pr(y|y')$ leaks posterior probability from neuron y' onto neuron y. Such cross-talk amongst the neurons exists independently of the lateral inhibition effect produced by the denominator term $\sum_{y'=1}^{M} Q(\mathbf{x}|y')$ in equation 2.5.

The VQ and SOM results (see [3, 4]) may be obtained as special cases of raw neuron firing rates $Q(\mathbf{x}|y)$, where one neuron's firing rate is much larger than the other $M-1$ neurons' firing rates (i.e. there is effectively only one neuron that can fire, so it is the winner-take-all).

C. Partitioned Posterior Probability (Multiple Recognition Models)

The form of the posterior probability $\Pr(y|\mathbf{x})$ introduced in equation 2.5 is unsuitable for networks with a large number of neurons M, because the lateral inhibition is *global* rather than *local*. This can readily be inferred because the denominator term $\sum_{y'=1}^{M} Q(\mathbf{x}|y')$ in equation 2.5 computes a quantity that is the sum over *all* of the raw neuron firing rates.

This problem can be amended by defining a *localised* posterior probability $\Pr(y|\mathbf{x};y')$ as

$$\Pr(y|\mathbf{x};y') \equiv \frac{Q(\mathbf{x}|y)\ \delta_{y\in\mathcal{N}(y')}}{\sum_{y''\in\mathcal{N}(y')} Q(\mathbf{x}|y'')} \tag{2.7}$$

where $\mathcal{N}(y')$ is the *local* neighbourhood of neuron y', which is assumed to contain at least neuron y', and $\delta_{y\in\mathcal{N}(y')}$ is a Kronecker delta that constrains y to lie in the neighbourhood $\mathcal{N}(y')$. If $\mathcal{N}(y')$ contains all M neurons then $\Pr(y|\mathbf{x};y')$ reduces to $\Pr(y|\mathbf{x})$ as previously defined in equation 2.5. $\Pr(y|\mathbf{x};y')$ has the required normalisation property that $\sum_{y=1}^{M}\Pr(y|\mathbf{x};y') = 1$ for all y'. Because y' can take M possible values, there are M complete localised posterior probability functions $\Pr(y|\mathbf{x};y')$. In effect, the neural network is split up into M overlapping subnetworks (these subnetworks overlap where $\mathcal{N}(y_1)\cap\mathcal{N}(y_2)\neq\emptyset$ for $y_1\neq y_2$), each of which computes its own posterior probability function; note that any overlap between a pair of subnetworks causes the corresponding $\Pr(y|\mathbf{x};y')$ to be mutually dependent.

It is not always convenient to use a neural network model in which there are M separate posterior probability models $\Pr(y|\mathbf{x};y')$. However, these M localised posterior probability functions $\Pr(y|\mathbf{x};y')$ (for the M different choices of y') may be averaged together to produce a *single* posterior probability function. Thus define $\Pr(y|\mathbf{x})$ as

$$\begin{aligned}\Pr(y|\mathbf{x}) &\equiv \frac{1}{M}\sum_{y'\in\mathcal{N}^{-1}(y)}\Pr(y|\mathbf{x};y') \\ &= \frac{1}{M}Q(\mathbf{x}|y)\sum_{y'\in\mathcal{N}^{-1}(y)}\frac{1}{\sum_{y''\in\mathcal{N}(y')}Q(\mathbf{x}|y'')}\end{aligned} \tag{2.8}$$

where $\mathcal{N}^{-1}(y)$ is the inverse neighbourhood of neuron y defined as $\mathcal{N}^{-1}(y)\equiv\{y'|y\in\mathcal{N}(y')\}$. This definition has all of the properties of a posterior probability function, including the normalisation property $\sum_{y=1}^{M}\Pr(y|\mathbf{x}) = 1$ (which may be derived by swapping the order of summations using the result $\sum_{y=1}^{M}\sum_{y'\in\mathcal{N}^{-1}(y)}(\cdots) = \sum_{y'=1}^{M}\sum_{y\in\mathcal{N}(y')}(\cdots)$). The form of the posterior probability given in equation 2.8, in which M individual posterior probabilities $\Pr(y|\mathbf{x};y')$ are averaged together, can be rigorously justified from a Bayesian point of view (see appendix A). The averaging process produces the posterior probability that should be used when there are M contributing models (as specified by the $\Pr(y|\mathbf{x};y')$ for $y' = 1,2,\cdots,M$) that have equal prior weight. The average over the M models then simply marginalises over an unobserved degree of freedom (the model index y').

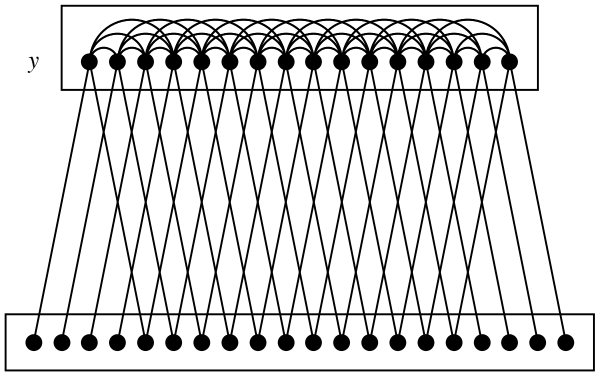


Figure 3: A partitioned mixture distribution (PMD) neural network.

If this localised definition of $\Pr(y|\mathbf{x})$ given in equation 2.8 is compared with the global definition given in equation 2.5 it is seen that the normalisation factor has been modified thus

$$\frac{1}{\sum_{y'=1}^{M}Q(\mathbf{x}|y')} \to \frac{1}{M}\sum_{y'\in\mathcal{N}^{-1}(y)}\frac{1}{\sum_{y''\in\mathcal{N}(y')}Q(\mathbf{x}|y'')} \tag{2.9}$$

which alters its lateral inhibition properties. $\frac{1}{\sum_{y''\in\mathcal{N}(y')}Q(\mathbf{x}|y'')}$ is the lateral inhibition factor that derives from the neighbourhood of neuron y', which gives rise to a contribution to the lateral inhibition factor for all neurons y in the neighbourhood of y' via the average $\frac{1}{M}\sum_{y'\in\mathcal{N}^{-1}(y)}(\cdots)$. Thus the overall lateral inhibition factor acting on neuron y is derived *locally* from those neurons y'' that lie in the set $\mathcal{N}\left(\mathcal{N}^{-1}(y)\right)$. The posterior probability model defined in equation 2.8 has been used before in the context of partitioned mixture distributions (PMDs), where multiple mixture distribution models are simultaneously optimised [7, 9].

Figure 3 shows the structure of the neural network corresponding to the PMD posterior probability in equation 2.8. Each output neuron has a raw receptive field of input neurons (which contains 5 input neurons in the example shown), and is also laterally inhibited by its neighbouring output neurons (the size of a neuron neighbourhood is 3 neurons to either side in the example shown). Note that the input-output links in figure 3 do not imply that the raw neuron firing rates $Q(\mathbf{x}|y)$ can be computed by using simple weighted connections; they are drawn merely to indicate the set of input neurons that influences the raw firing rate of each output neuron. Similarly, the output-output links in figure 3 are drawn to indicate the sizes of the output neuron neighbourhoods; the details of how lateral inhibition modifies the raw firing rates $Q(\mathbf{x}|y)$ of the output neurons to produce the probability $\Pr(y|\mathbf{x})$ that neuron y fires first is given in equation 2.8.

For completeness, the PMD objective function in equation 2.3 may now be written out in full using the expression for the PMD posterior probability in equation 2.8 to yield (where the effects of leakage have been included, as defined in equation 2.6)

$$D = \frac{2}{M}\int d\mathbf{x}\, \Pr(\mathbf{x}) \sum_{y=1}^{M}\sum_{y'=1}^{M} \Pr(y|y')\, Q(\mathbf{x}|y') \sum_{y''\in\mathcal{N}^{-1}(y')} \frac{1}{\sum_{y'''\in\mathcal{N}(y'')} Q(\mathbf{x}|y''')} \left\|\mathbf{x}-\mathbf{x}'(y)\right\|^2 \tag{2.10}$$

This is the objective function that will be used to characterise to performance of the neural networks in all of the computer simulations.

D. Derivatives of the Objective Function

In order to minimise the PMD objective function in equation 2.10 its derivatives must be calculated. First of all, define some convenient notation [12]

$$\begin{array}{ll}
L_{y,y'} \equiv \Pr(y'|y) & P_{y,y'} \equiv \Pr(y'|\mathbf{x};y) \equiv \frac{Q(\mathbf{x}|y')\,\delta_{y'\in\mathcal{N}(y)}}{\sum_{y''\in\mathcal{N}(y)} Q(\mathbf{x}|y'')} \\
p_y \equiv \sum_{y'\in\mathcal{N}^{-1}(y)} P_{y',y} & \left(L^T p\right)_y \equiv \sum_{y'\in\mathcal{L}^{-1}(y)} L_{y',y}\, p_{y'} \\
e_y \equiv \left\|\mathbf{x}-\mathbf{x}'(y)\right\|^2 & (L\,e)_y \equiv \sum_{y'\in\mathcal{L}(y)} L_{y,y'}\, e_{y'} \\
(PLe)_y \equiv \sum_{y'\in\mathcal{N}(y)} P_{y,y'}\, (L\,e)_{y'} & \left(P^T\,P\,L\,e\right)_y \equiv \sum_{y'\in\mathcal{N}^{-1}(y)} P_{y',y}\, (P\,L\,e)_{y'}
\end{array} \tag{2.11}$$

where $\mathcal{L}(y)$ denotes the leakage neighbourhood of neuron y, which is the set of neurons that have posterior probability leaked onto them by neuron y, and the inverse leakage neighbourhood $\mathcal{L}^{-1}(y)$ is defined as $\mathcal{L}^{-1}(y) \equiv \{y'|y\in\mathcal{L}(y')\}$. Assume that the raw neuron firing rates may be modelled using a sigmoid function

$$Q(\mathbf{x}|y) = \frac{1}{1+\exp(-\mathbf{w}(y)\cdot\mathbf{x}-b(y))} \tag{2.12}$$

whence the derivatives may be obtained in the form [12]

$$\begin{aligned}
\frac{\partial D}{\partial \mathbf{x}'(y)} &= -\frac{4}{M}\int d\mathbf{x}\, \Pr(\mathbf{x})\, \left(L^T p\right)_y (\mathbf{x}-\mathbf{x}'(y)) \\
\frac{\partial D}{\partial \begin{pmatrix} b(y) \\ \mathbf{w}(y) \end{pmatrix}} &= \frac{2}{M}\int d\mathbf{x}\, \Pr(\mathbf{x})\, \left(p_y\,(Le)_y - (P^T PLe)_y\right) (1-Q(\mathbf{x}|y)) \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}
\end{aligned} \tag{2.13}$$

where the two derivatives $\frac{\partial D}{\partial b(y)}$ and $\frac{\partial D}{\partial \mathbf{w}(y)}$ have been written together for compactness.

E. Receptive Fields

The raw firing rate $Q(\mathbf{x}|y)$ of neuron y depends only on a subset $\tilde{\mathbf{x}}(y)$ of components of $\mathbf{x}$; $\tilde{\mathbf{x}}(y)$ is thus the *raw* receptive field of neuron y. However, the posterior probability $\Pr(y|\mathbf{x})$ that neuron y fires first is derived from $Q(\mathbf{x}|y)$ by weighting it with a lateral inhibition factor that depends on the raw firing rates of all neurons in $\mathcal{N}\left(\mathcal{N}^{-1}(y)\right)$, as seen in equation 2.8, so the *overall* receptive field of a neuron is rather broader than its raw receptive field. The effect of leakage, as defined in equation 2.6, is to broaden the overall receptive field further still. The optimal reference vector $\mathbf{x}'(y)$ has non-trivial structure only within this overall receptive field, so inside the overall receptive field the components of $\mathbf{x}'(y)$ must be subjected to an optimisation procedure to discover their optimal form, whereas outside the overall receptive field the components of $\mathbf{x}'(y)$ may be set to be the average values of the corresponding components of the training vectors $\mathbf{x}$ (see the definition of $\mathbf{x}'(y)$ in equation 2.4, which reduces to $\mathbf{x}'(y) = \int d\mathbf{x}\, \Pr(\mathbf{x})\, \mathbf{x}$ for those components of $\mathbf{x}'(y)$ that lie outside the overall receptive field of neuron y).

In the simulations that will be presented here a suboptimal approach is used, where only those components of $\mathbf{x}'(y)$ that lie inside the *raw* receptive field are optimised; this produces a least upper bound on the value of the objective function that would have been obtained if a full optimisation had been used. Also, it is assumed that the input data has been prepared in such a way that each component is zero mean. This is not actually a restriction, because the objective function is invariant with respect to adding a different constant to each component of $\mathbf{x}$, because it is a function of the difference $\mathbf{x} - \mathbf{x}'$. In this suboptimal approach, and with the zero mean assumption, the components of $\mathbf{x}'(y)$ that lie outside the *raw* receptive field of neuron y will be set to zero.

The fact that the components of $\mathbf{x}'(y)$ that lie outside the *raw* receptive field of neuron y are zero may be used to simplify the evaluation of the various terms $\frac{\partial D}{\partial b(y)}$ and $\frac{\partial D}{\partial \mathbf{w}(y)}$ in equation 2.13. Thus evaluate $p_y (Le)_y - (P^T P L e)_y$ by expanding e_y as

$$\begin{aligned} e_y &= \|\mathbf{x}\|^2 - 2\mathbf{x}\cdot\mathbf{x}'(y) + \|\mathbf{x}'(y)\|^2 \\ &= \|\mathbf{x}\|^2 + \mathbf{x}'(y)\cdot(\mathbf{x}'(y) - 2\mathbf{x}) \qquad (2.14) \end{aligned}$$

which is a sum of a constant (i.e. does not depend on y) term $\|\mathbf{x}\|^2$ and a term $\mathbf{x}'(y)\cdot(\mathbf{x}'(y)-2\mathbf{x})$ that does depend on y. What happens to the constant term when it is substituted into $p_y\,(L\,e)_y - (P^T\,P\,L\,e)_y$?

$$\begin{aligned} p_y\,(L\,e)_y - (P^T\,P\,L\,e)_y &\to p_y\,(L\cdot\mathbf{1})_y - (P^T\,P\,L\cdot\mathbf{1})_y \\ &= p_y\,\mathbf{1}_y - (P^T\,P\cdot\mathbf{1})_y \\ &= p_y - p_y \\ &= 0 \qquad (2.15) \end{aligned}$$

It cancels out, so e_y might as well be replaced as follows in $p_y\,(L\,e)_y - (P^T\,P\,L\,e)_y$

$$e_y \to \mathbf{x}'(y)\cdot(\mathbf{x}'(y)-2\mathbf{x}) \qquad (2.16)$$

Because the components of $\mathbf{x}'(y)$ that lie outside the *raw* receptive field of neuron y are set to zero, the $\mathbf{x}'(y)\cdot(\cdots)$ operation effectively projects out any components of $(\cdots)$ that happen to lie outside this raw receptive field. This means that the only components of $\mathbf{x}$ in equation 2.16 that survive are those that lie inside the raw receptive field, so effectively e_y depends only on quantities that lie inside the raw receptive field of neuron y. Note that a full optimisation of $\mathbf{x}'(y)$, in which all components that lie inside the *overall* receptive field of neuron y are optimised, would produce a different result.

III. DOMINANCE STRIPES AND ORIENTATION MAPS

The purpose of this section is to discuss the two phenomena of dominance stripes and orientation maps. In section III A a brief review of the popular elastic net model of dominance stripes is presented, and in section III B an informal derivation of the origin of both dominance stripes and orientation maps is given.

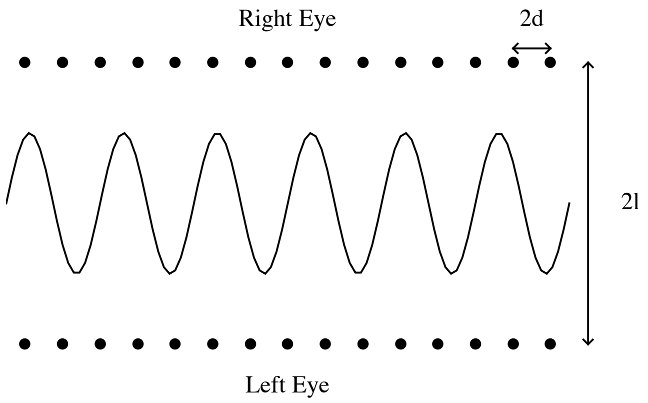


Figure 4: An elastic net oscillating back and forth in ocularity between a pair of retinae.

A. Review of Dominance Stripes Using the Elastic Net Model

The results that will be presented here are, broadly speaking, equivalent to the way in which ocular dominance stripes are obtained in the elastic net model (as reviewed in [2, 13]) as applied to a pair of retinae. The essential features of this type of model of ocular dominance are shown in figure 4 (which is copied from [2]). The left and right retinae are represented as 1-dimensional lines of units at the top and bottom of the diagram. The horizontal dimension represents distance across a retina, and the vertical dimension represents the ocularity degree of freedom. The distance between any two retinal units, either within or between retinae, represents the correlation between those two units [2]. Thus the ratio $\frac{l}{d}$ determines the relative strength of the inter-retinal and intra-retinal correlations. The elastic net is represented by the line oscillating back and forth between the retinae. The net effect of the elastic net algorithm is to encourage the elastic net to pass as close as possible (in a well-defined sense) to all of the retinal units, and also to minimise its total length. These are conflicting requirements, and the oscillatory solution shown in figure 4 is typical of an optimal elastic net configuration, which thus predicts an oscillatory pattern of ocular dominance (i.e. which corresponds to dominance stripes in the case of 2-dimensional retinae).

This type of model inevitably leads to dominance stripe formation, because the elastic net model separates the input components into two clusters (see figure 4) according to whether they belong to the left or right retina. In effect the output layer of the network is explicitly told which retina an input component belongs to, and this fact is expressed by the position of the component along the ocularity dimension. The goal in this paper is to construct a more natural model of dominance stripe formation, in which the ocularity dimension is revealed by a

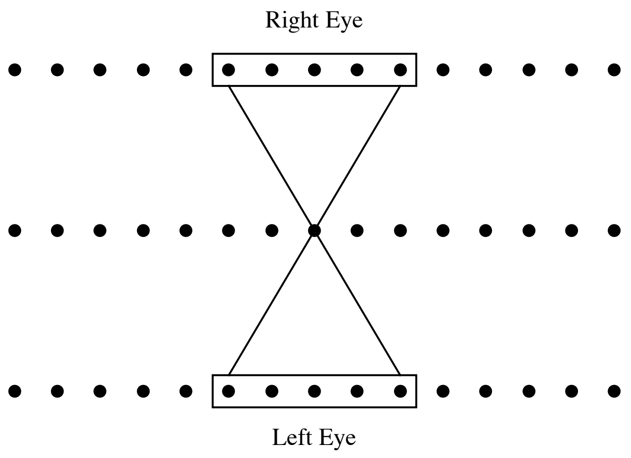


Figure 5: Neural network model with a limited receptive field.

process of self-organisation, rather than being hard-wired into the model. Thus, the visual cortex model that is presented in this paper will *not* explicitly label the input pixels as belonging to the right or left retina (as they are in figure 4), but will have to deduce their left/right retina membership from the properties of the training set instead.

B. Informal Derivation of Dominance Stripes and Orientation Maps

The purpose of this section is to present a simple picture that makes it clear what types of behaviour should be expected from neural network that minimises the objective function in equation 2.10.

1. *Neural Network Model*

It is assumed that each of the output neurons has only a limited receptive field of input neurons within each of the two retinae. In effect, this is a hand-crafted version of a "wire length" constraint, which ensures that the total length of the input-to-output connections is limited. In the context of the elastic net model this corresponds to the limited range of interaction between retinal units (the input) and elastic net units (the output). Also, it is assumed that sigmoidal neurons with local probability leakage are used, which generates an effect that is analogous to the elastic tension in the elastic net model, because it encourages neighbouring neurons to adopt similar parameter values.

This model is drawn in figure 5 in an analogous way to the elastic net model in figure 4. In this model the ocularity dimension is not explicitly present, and the elasticity (of the elastic net) is replaced by the probability leakage mechanism that enables neighbouring output neurons to communicate with each other. The separation of input neurons into left and right retinae in figure 5 is made only for comparison between figure 5 and the elastic net model in figure 4. When the left and right receptive fields are presented to the output neuron, all information about which retina the various input neurons belong to has been discarded; all input neurons within the left and right receptive fields are treated on an equal basis. The ocularity dimension will emerge by a process of self-organisation driven by the statistical properties of the images received by the left and right retinae.

2. *Very Low Resolution Input Images*

The simplest situation is when there are two retinae (as in the above elastic net model), each of which senses independently a featureless scene, i.e. all the units in a retina sense the same brightness value, but the two brightnesses that the left and right retinae sense are independent of each other. This situation would arise if the images projected onto the two retinae were very low resolution, so all spatial detail is lost. This limits the input data to lying in a 2-dimensional space R^2. If these two featureless input images (i.e. left and right retinae) are then normalised so that the sum of left and right retina brightness is constrained to be constant, then the input data is projected down onto a 1-dimensional space R^1, which effectively becomes the ocularity dimension. If each of the M output neurons had an infinite-sized receptive field, then the optimal network would be the one in which the M neurons cooperate to give the best soft encoding of R^1.

However, because of the limited receptive field size and output neuron neighbourhood size, the neurons can at best co-operate together a few at a time (this also depends on the size of the leakage neighbourhood). If the network properties are translation invariant this leads to an optimal network whose properties fluctuate periodically across the network (see appendix B), where each period typically contains a complete repertoire of the computing machinery that is needed to process the contents of a receptive field; this effect is called completeness, and it is a characteristic emergent property of this type of neural network.

The only unexplained step in this argument is the use of a normalisation procedure on the input. However, if the input to this network is the PMD posterior probability computed by the output layer of another such network, then there is already such a normalisation effect induced by the lateral inhibition within the PMD posterior probability. For featureless input images, this lateral inhibition effect causes precisely the type of normalisation that is used above (i.e. left plus right retina brightness is constant) to occur naturally.

These results are summarised in figure 6 where the ocularity dimension runs from $(0, 1)$ to $(1, 0)$, and a typical

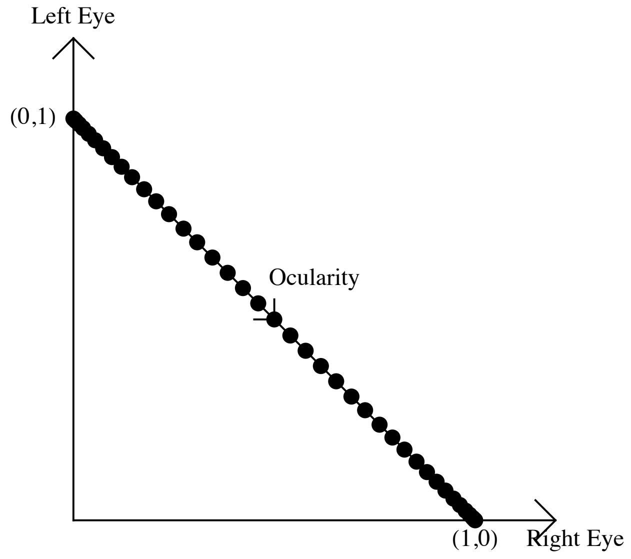


Figure 6: Typical neural reference vectors for very low resolution input images.

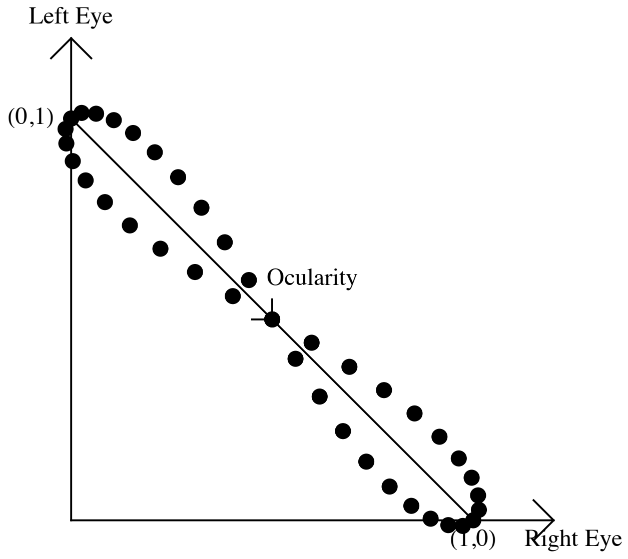


Figure 7: Typical neural reference vectors for low resolution input images.

set of neural reference vectors is shown. The oscillation of these reference vectors back and forth along the ocularity dimension corresponds to the oscillations of the elastic net that are represented in figure 4.

3. Low Resolution Input Images

A natural generalisation of the above is to the case of not-quite-featureless input images. This could be brought about by gradually increasing the resolution of the input images until it is sufficient to reveal spatial detail on a size scale equal to the receptive field size. Instead of seeing a featureless input, each neuron would then see a brightness gradient within its receptive field. This could be interpreted by considering the low order terms of a Taylor expansion of the input image about a point at the centre of the neuron's receptive field: the zeroth term is local average brightness (which lives on a 1-dimensional line R^1), and the two first order terms are the local brightness gradient (which lives in a 2-dimensional space R^2). When normalisation is applied this reduces the space in which the two images live to $R^1 \times R^2 \times R^2$ (R^1 from the zeroth order Taylor term with normalisation taken into account, R^2 from the first order Taylor terms, counted twice to deal with each retina).

The R^1 from the zeroth order Taylor term gives rise to ocular dominance stripes, which thus causes the left and right retinae to map to different stripe-shaped regions of the output layer. The remaining $R^2 \times R^2$ then naturally splits into two contributions (left retina and right retina), each of which maps to the appropriate stripe. If the stripes did not separate the left and right retinae, then the $R^2 \times R^2$ could not be split apart in this simple manner. Finally, since each ocular dominance stripe occupies a 2-dimensional region of the output layer, a direct mapping of the corresponding R^2 (which carries local brightness gradient information) to output space can be made. As in the case of dominance stripes alone, the limited receptive field size and output neuron neighbourhood size causes the neurons to co-operate together only a few at a time, so that each local patch of neurons contains a complete mapping from R^2 to the 2-dimensional output layer.

These results are summarised in figure 7 where the pure oscillation back and forth along the ocularity dimension that occurred in figure 6 develops to reveal some additional degrees of freedom, only one of which is represented in figure 7 (it is perpendicular to the ocularity axis).

If the leakage is reduced then the oscillation back and forth along the dominance axis tends to be more like a square wave than a sine wave, in which case figure 7 becomes as shown in figure 8 where the neural reference vectors are bunched near to the points $(0, 1)$ and $(1, 0)$, and explore the additional degree(s) of freedom at each end of the ocularity axis. In the extreme case, where the ocularity switches back and forth as a square wave, the neurons separate into two clusters, one of which responds only to the left retina's image and the other to the right retina's image. Furthermore, within each of these clusters, the neurons explore the additonal degree(s) of freedom that occur within the corresponding retina's image. Note only one such degree of freedom is represented in figure 8; it is perpendicular to the ocularity axis.

The above arguments can be generalised to the case of input images with fine spatial structure (i.e. lots of high order terms in the Taylor expansion are required). However, more and more neurons (per receptive field) are

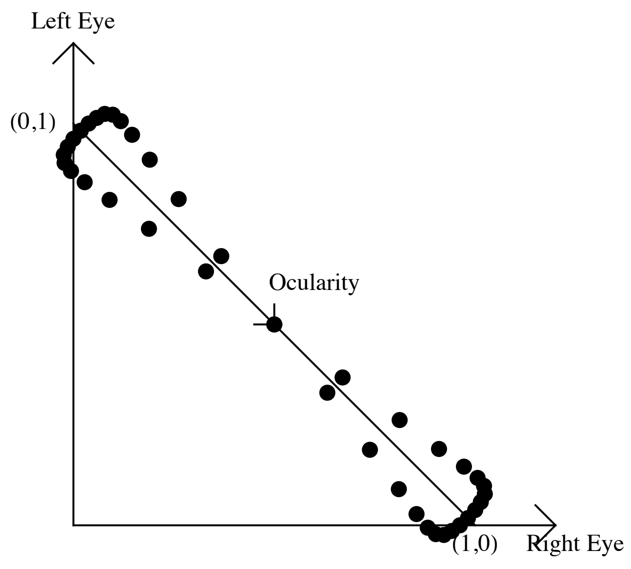


Figure 8: Typical neural reference vectors for low resolution input images, where reduced leakage causes the ocularity to switch abruptly back and forth.

required in order to build a faithful mapping from input space to a 2-dimensional representation in output space. For a given number of neurons (per receptive field) a saturation point will quickly be reached, where the least important detail (from the point of view of the objective function) is discarded, keeping only those properties of the input images that best preserve the ability of the neural network to reconstruct its own input with minimum Euclidean error (on average).

IV. SIMULATIONS

Two types of training data will be used: synthetic, and natural. Synthetic data is used in order to demonstrate simple properties of the neural network, without introducing extraneous detail to complicate the interpretation of the results. Natural data is used to remove any doubt that the neural network is capable of producing interesting and useful results when it encounters data that is more representative of what it might encounter in the real world.

In section IV A dominance stripes are produced from a 1-dimensional retina, and in section IV B these results are generalised to a 2-dimensional retina. In both cases both synthetic and natural image results are shown. In section IV C orientation maps are produced for the case of two retinae trained with natural images.

A. Dominance Stripes: The 1-Dimensional Case

The purpose of the simulations that are presented in this section is to demonstrate the emergence of ocular dominance stripes in the simplest possible realistic case. The results will correspond to the situation outlined in figure 6.

1. Synthetic Training Data

The purpose of this simulation is to demonstrate the emergence of ocular dominance stripes, of the type that were shown in figure 6, by presenting a model of the type shown in figure 5 with very low-resolution input images. In fact, the resolution is so low that each image is entirely featureless, so that all the neurons in a retina have the same input brightness, but the two retinae have independent input brightnesses. These input images are normalised by processing them so that they look like the PMD posterior probability computed by the output layer of another such network; the neighbourhood size used for this normalisation process was chosen to be the same as the network's own output layer neighbourhood size.

In the first simulation the parameters used were: network size $= 30$, receptive field size $= 9$, output layer neighbourhood size $= 5$ (centred on the source neuron), leakage neighbourhood size $= 5$ (centred on the source neuron), number of training updates $= 2000$, update step size $= 0.01$. For each neuron the leakage probability had a Gaussian profile centred on the neuron, and the standard deviation was chosen as 1, to make the profile fall from 1 on the source neuron to $\exp\left(\frac{1}{2}\right)$ on each of its two closest neighbours.

The update scheme used was a crude gradient following algorithm parameterised by three numbers which controlled the rate at which the weight vectors, biasses and reference vectors were updated. These three numbers were continuously adjusted to ensure that the maximum rate of change (as measured over all the neurons in the network) of the length of each weight vector, and also the maximum rate of change of the absolute value of each bias, was always equal to the requested update step size; this prescription will adjust the parameter values until they jitter around in the neighbourhood of their optimum values. The optimum reference vectors could in principle be completely determined using equation 2.4 for each choice of weights and biasses, but it is not necessary for the reference vectors to keep in precise synchrony with the weights and biasses. Rather, the reference vectors were controlled in a similar way to the weight vectors, except that they used three times the update step size, which made them more agile than the weights and biasses they were trying to follow.

The ocular dominance stripes that emerge from this simulation are shown in figure 9. The ocularity for a given neuron was estimated by computing the average of the absolute deviations (as measured with respect to the

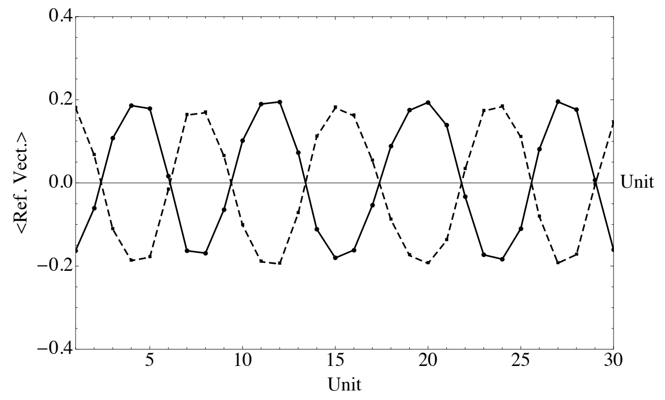


Figure 9: 1-dimensional dominance stripes after training on synthetic data.

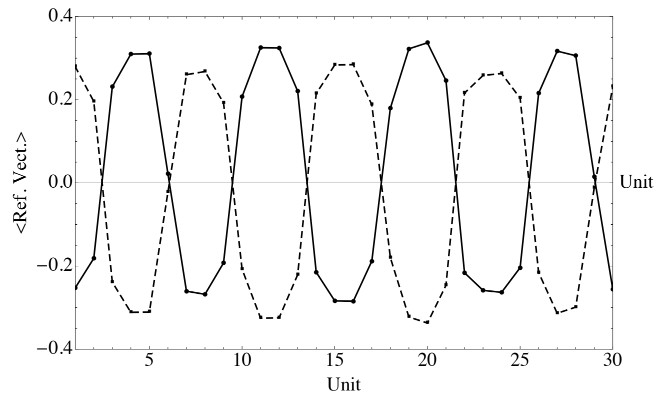


Figure 10: 1-dimensional square wave dominance stripes after further training with reduced probability leakage on synthetic data.

overall mean reference vector component value, which is zero for the zero mean training data that is used here) of its reference vector components within its receptive field, both for the left retina and the right retina. This allows two plots to be drawn: average value of absolute deviations from the mean in left retina's receptive field as a function of position across the network, and similarly the right retina's receptive field. As can be seen in figure 9, these two curves are approximately periodic, and are in antiphase with each other; this corresponds to the situation shown in figure 6. The amplitude of the ocularity curves is less than the 0.5 that would be required for the end points of the ocularity dimension to be reached, because one of the effects of leakage is to introduce a type of elastic tension between the reference vectors that causes them to contract towards zero ocularity. Note how the ocular dominance curves have a period of approximately 7, which is slightly greater than the output layer neighbourhood size (which is 5). In the limit of zero leakage and infinite receptive field size the period would be equal to the output layer neighbourhood size, in order to guarantee that a complete set of processing machinery is contained within each output layer neighbourhood size; this effect is called completeness.

If the above simulation is continued for a further 2000 updates with a reduced leakage, by reducing the standard deviation of the Gaussian leakage profile from 1 to 0.5, then the ocular dominance curves become more like square waves than sine waves, as shown in figure 10; this is similar to the type of situation that was shown in figure 8, except that the input images are featureless in this case.

2. Natural Training Data

Figure 11 shows the Brodatz texture image [1] that was used to generate a more realistic training set than was used in the synthetic simulations described above. Figure 12 shows an enlarged portion of figure 11, where it is clear that the characteristic length scale of the texture structure is in the range $5-10$ pixels. This is large enough compared to the receptive field size (9) and the output layer neighbourhood size (5) that a simulation using 1-dimensional training vectors extracted from this 2-dimensional Brodatz image will effectively see very low resolution training data, and should repond approximately as described in figure 6.

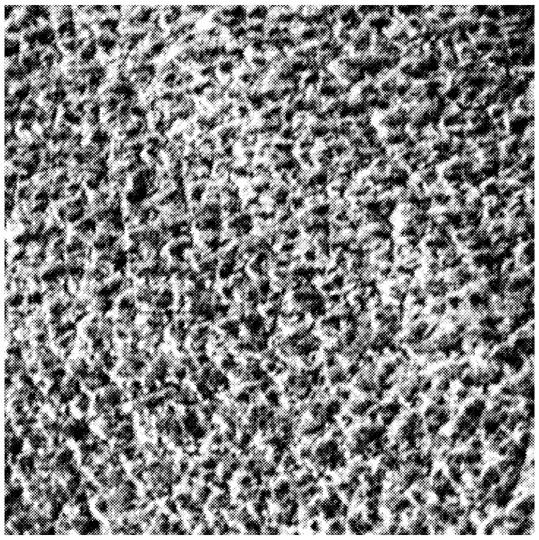


Figure 11: Brodatz texture image used as a natural training image.

The results corresponding to figure 9 and figure 10 are shown in figure 13 and figure 14, respectively.

The general behaviour is much the same in the synthetic and Brodatz cases, except that the depth of the ocularity fluctuations is somewhat less in the real case, because in the Brodatz case the training data is not actually featureless within each receptive field.

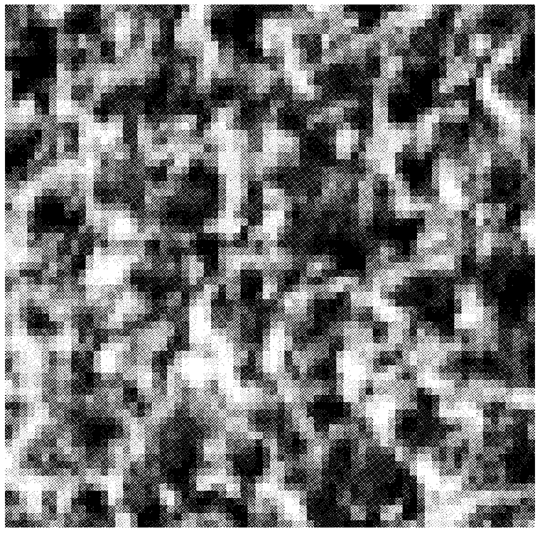


Figure 12: Magnified portion of the Brodatz texture training image.

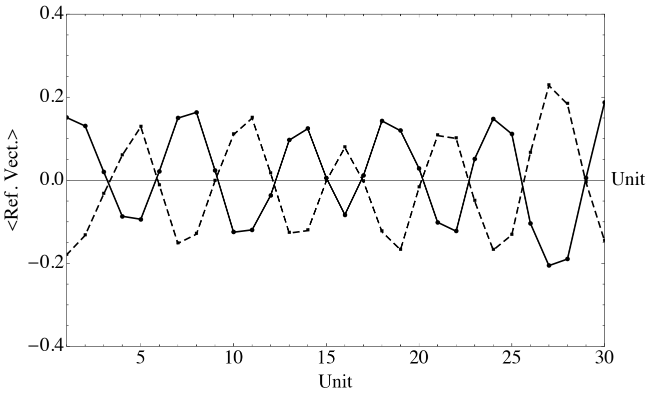


Figure 13: 1-dimensional dominance stripes after training on natural data.

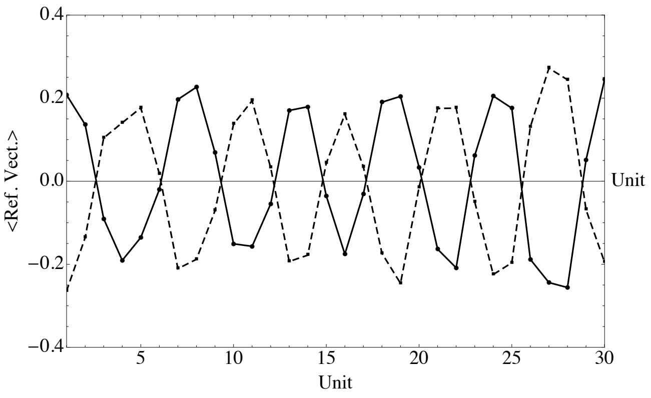


Figure 14: 1-dimensional square wave dominance stripes after further training with reduced probability leakage on natural data.

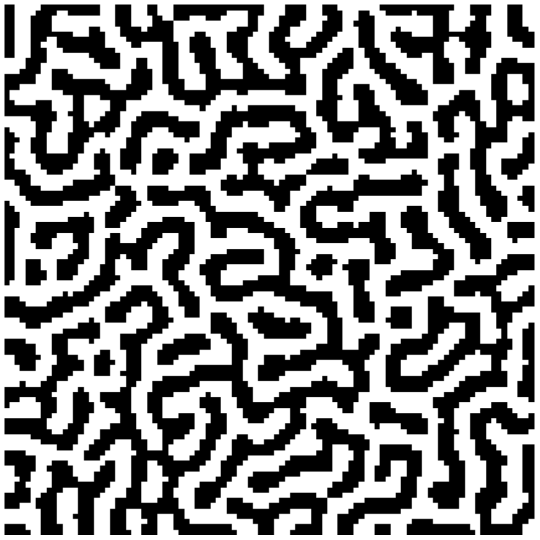


Figure 15: 2-dimensional dominance stripes after training on synthetic data.

B. Dominance Stripes: The 2-Dimensional Case

This section extends the results of the previous section to the case of 2-dimensional neural networks. The training schedule(s) used in the simulations have not been optimised. Usually the update rate is chosen conservatively (i.e. smaller than it needs to be) to avoid possible numerical instabilities, and the number of training updates is chosen to be larger than it needs to be to ensure that convergence has occurred. It is highly likely that much more efficient training schedules could be found.

1. *Synthetic Training Data*

The results that were presented in figure 9 may readily be extended to the case of a 2-dimensional network. The parameters used were: network size = 100×100, receptive field size = 3×3 (which is artificially small to allow the simulation to run faster), output layer neighbourhood size = 5×5 (centred on the source neuron), leakage neighbourhood size = 3×3 (centred on the source neuron), number of training updates = 24000 (dominance stripes develop quickly, so far fewer than 24000 training updates could be used), update step size = 0.001. For each neuron the leakage probability had a Gaussian profile centred on the neuron, and the standard deviations were chosen as 1×1, to make the profile fall from 1 on the source neuron to $\exp\left(\frac{1}{2}\right)$ on each of its four closest neighbours.

Apart from the different parameter values, the simulation was conducted in precisely the same way as in the 1-dimensional case, and the results for ocular dominance are shown in figure 15, where ocularity has been quantised as a binary-valued quantity. These results show the characteristic striped structure that is familiar from

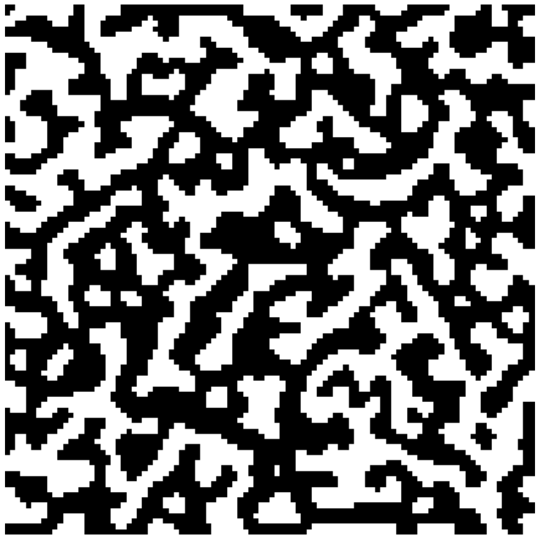


Figure 16: 2-dimensional dominance stripes after training on natural data.

experiments on the mammalian visual cortex. The behaviour near to the boundary depends critically on the interplay between the receptive field size(s) and the output layer neighbourhood size(s).

2. Natural Training Data

The simulation, whose results were shown in figure 15, may be repeated using the Brodatz image training set shown in figure 11, to yield the results shown in figure 16. These results are not quite as stripe-like as the results in figure 15, because in the Brodatz case the training data is not actually featureless within each receptive field.

C. Orientation Maps

The purpose of the simulations that are presented in this section is to demonstrate the emergence of orientation maps in the simplest possible realistic case. In the case of two retinae, the results will correspond to the situation outlined in figure 7 (or, at least, a higher dimensional version of that figure).

1. Orientation Map (One Retina)

In this simulation the parameters used were: network size $= 30 \times 30$, receptive field size $= 17 \times 17$, output layer neighbourhood size $= 9 \times 9$ (centred on the source neuron), leakage neighbourhood size $= 3 \times 3$ (centred on the source neuron), number of training updates $= 24000$, update step size $= 0.01$. For each neuron the leakage probability had a Gaussian profile centred on the neuron, and the standard deviations were chosen as 1×1, to make

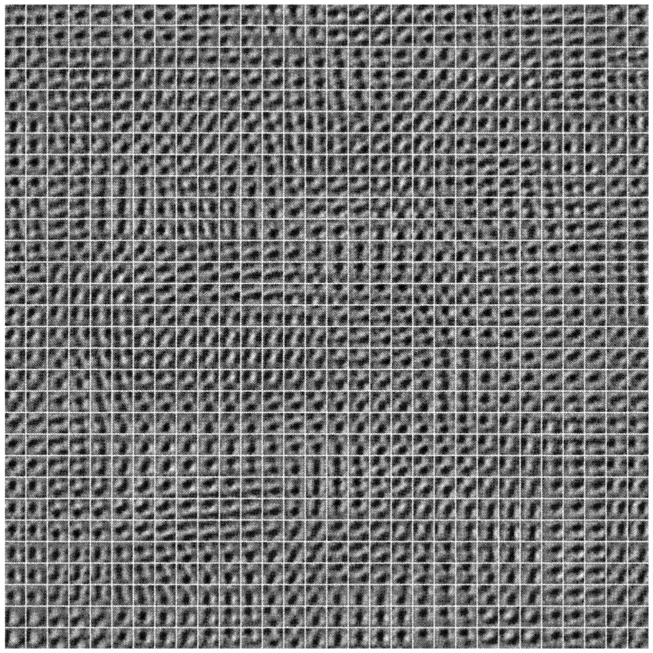


Figure 17: Orientation map after training on natural data.

the profile fall from 1 on the source neuron to $\exp\left(\frac{1}{2}\right)$ on each of its four closest neighbours.

Note that both the receptive field size and the output layer neighbourhood size are substantially larger than in the 2-dimensional dominance stripe simulations, because many more neurons are required in order to allow orientation maps to develop than to allow dominance stripes to develop; in fact it would be preferable to use even larger sizes than were used here. To limit the computer run time this meant that the overall size of the neural network had to be reduced from 100×100 to 30×30. The training set was the Brodatz texture image in figure 11.

The results are shown in figure 17 where the receptive fields have been gathered together in a montage. There is a clear swirl-like pattern that is characteristic of orientation maps. Each local clockwise or anticlockwise swirl typically circulates around an unoriented region.

2. Using the Orientation Map

In figure 18 the orientation map network shown in figure 17 is used to encode and decode a typical input image. On the left of figure 18 the input image (i.e. $\mathbf{x}$) is shown, in the centre of figure 18 the corresponding output (i.e. its PMD posterior probability $\Pr(y|\mathbf{x})$) produced by the orientation map is shown, and on the right of figure 18 the corresponding reconstruction (i.e. $\sum_{y=1}^{M} \Pr(y|\mathbf{x})\, \mathbf{x}'(y)$) is shown.

The output consists of a number of isolated "activity bubbles" of posterior probability, and the reconstruction is a low resolution version of the original input. The form

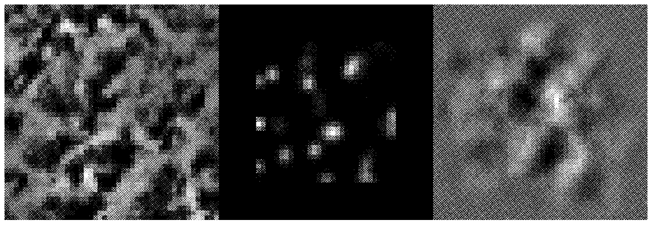


Figure 18: Typical input, output and reconstruction produced by the orientation map.

of output is familiar as a type of "sparse coding" of the input, where only a small fraction of the neurons participate in encoding a given input (this type of transformation of the input is central to the work that was reported in [14]). This type of encoding is very convenient because it has effectively transformed the input into a small number of constituents each of which corresponds to an activity bubble, rather than transforming the input into a representation where the output activity is spread over all of the neurons, which is thus not easily interpretable as arising from a small number of constituents.

The reconstruction has a lower resolution than the input because there are insufficient neurons to faithfully record all the information that is required to reconstruct the input exactly (e.g. probability leakage causes neighbouring neurons to have a correlated response, thus reducing the effective number of neurons that are available). The featureless region around the edge of the reconstruction is an artefact, which occurs because fewer neurons (per unit area) contribute to the reconstruction near the edge of the input array.

3. *Orientation Map (Two Retinae)*

The above orientation map results may be generalised to the case of two retinae. The parameter values used were the same, apart from the standard deviation of the leakage Gaussian which was reduced to 0.5×0.5 in order to allow more detailed structure to develop in the adaptive parameter values of the output neurons. This is necessary because the presence of two retinae causes dominance stripes to develop, which allows only half of the neurons to be allocated to each retina, so a complete repertoire of computing machinery must be forced into half the number of neurons that were used in the case of one retina.

The results are shown in figure 19 where the receptive fields for the left and right retinae have been used to create a colour separation in which one retina is coded as blue and the other as yellow.[1] Within each retina there is a long-scale periodic fluctuation in overall brightness which corresponds to the dominance stripes. Within each dominance stripe there is the characteristic swirl-like pattern of the orientation map. Note that the unoriented regions typically occur at the centre of dominance stripes, as observed in the visual cortex; this can be understood intuitively by referring to figure 8.

A larger simulation would be required in order to accurately estimate the detailed orientation map as a vector flow field. Such simulations could be used to verify whether the iso-orientation contours typically lie perpendicular to the dominance stripe boundaries, as observed in the visual cortex. The dominance stripe structure that appears in this simulation is not as distinct as the stripes in figure 16. This is not a fundamental problem, but rather it is a result of the limited size of computer simulation that could be run in a reasonable length of time. It should also be noted that the dominance stripes that are observed in the visual cortex are sometimes more blob-like than stripe-like [13], so it is pleasing that different choices of parameter value should yield a variety of degrees of stripiness in our simulations.

V. CONCLUSIONS

This paper has shown how folded Markov chains (FMCs) [6] can be combined with partitioned mixture distributions (PMDs) [7] to yield a class of self-organising neural networks that has many of the properties that are observed in the mammalian visual cortex [2, 13], which are thus called visual cortex networks (VICON). These neural networks differ from previous models of the visual cortex, insofar as they model the neuron behaviour in terms of their individual firing events, and operate in the real space of input images rather than a hand-crafted abstract space, and the use of Bayesian methods makes the nature of the network's computations clearer than in the case where the network behaviour is simply postulated. When the neural network structure (e.g. receptive field size) parameters are appropriately chosen, dominance stripes and orientation maps emerge naturally when the network is trained on a natural image (e.g. a Brodatz texture image).

These results show how this type of network is capable of self-organising its internal parameters in familiar ways when trained on data from multiple sources (actually, only two sources in the case of the visual cortex-like network). The same network objective function could be used when an arbitrary number of data sources is presented, and it is anticipated that it would lead to analogous results.

An extension of the network objective function to the case where sets of multiple neural firing events are considered has been published [8, 12], and an extension to the case of a multilayer network has been published [11]. When combined, these extensions could be applied to the problem of the processing of data from multiple sensors

[1] In this black and white version of the paper the blue/yellow channel is displayed on the left/right respectively.

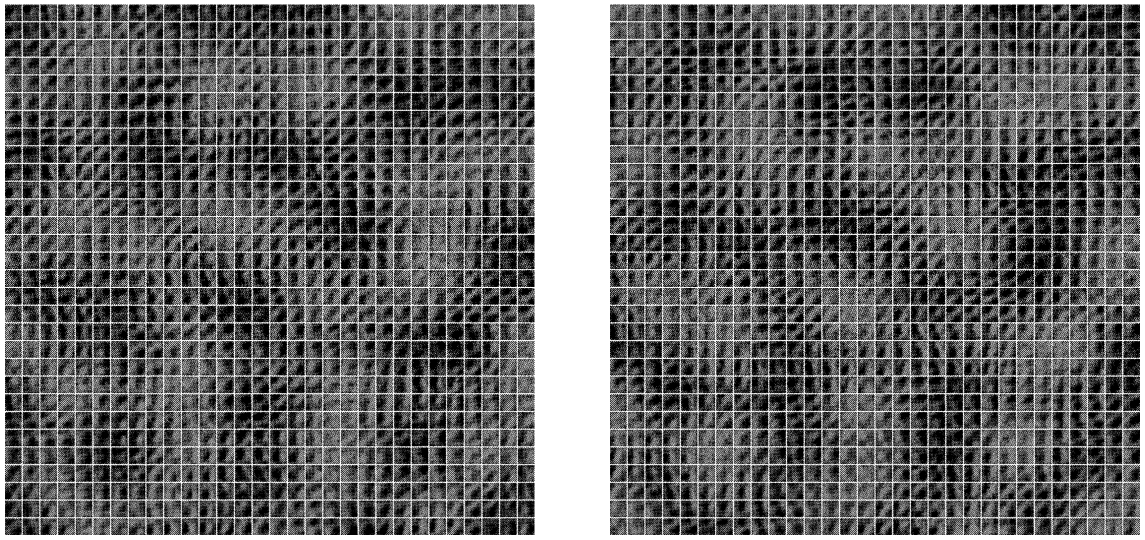


Figure 19: Orientation map and dominance stripes after training on natural data.

(i.e. data fusion).

Appendix A: Bayesian PMD

In this section a fully Bayesian interpretation of a partitioned mixture distribution (PMD) will be presented.

Consider the general problem of computing a posterior probability $\Pr(y|\mathbf{x})$ over classes y given an input vector $\mathbf{x}$. If there is more than one model k then $\Pr(y|\mathbf{x})$ is given by a marginal PDF

$$\Pr(y|\mathbf{x}) = \sum_k \Pr(y,k|\mathbf{x}) \tag{A1}$$

where $\Pr(y,k|\mathbf{x})$ is the joint PDF of class y and model k given an input vector $\mathbf{x}$. Bayes' theorem may be used to rewrite this as follows

$$\begin{aligned} \Pr(y,k|\mathbf{x}) &= \frac{\Pr(y,k,\mathbf{x})}{\Pr(\mathbf{x})} \\ &= \frac{\Pr(y|k,\mathbf{x})\ \Pr(k,\mathbf{x})}{\Pr(\mathbf{x})} \\ &= \Pr(y|k,\mathbf{x})\ \Pr(k) \end{aligned} \tag{A2}$$

where $\Pr(k,\mathbf{x}) = \Pr(k)\Pr(\mathbf{x})$ (i.e. independence of model k and data vector $\mathbf{x}$) has been assumed in the last step. Thus the posterior probability $\Pr(y|\mathbf{x})$ may be written as

$$\Pr(y|\mathbf{x}) = \sum_k \Pr(y|k,\mathbf{x})\ \Pr(k) \tag{A3}$$

Assume that there are M models, and that the prior probabilities $\Pr(k)$ of the various models are equal, so that $\Pr(k) = \frac{1}{M}$, in which case the posterior probability reduces to

$$\Pr(y|\mathbf{x}) = \frac{1}{M}\sum_{k=1}^{M} \Pr(y|k,\mathbf{x}) \tag{A4}$$

which is an average of M contributing posterior probabilities (one from each of the contributing models). The PMD posterior probability in equation 2.8 is a special case of this result.

More generally, the prior probabilities $\Pr(k)$ are k-dependent, and might be chosen in some optimal fashion to best handle the training set. The simplest way of determining an optimal $\Pr(k)$ is to minimise D with respect to $\Pr(k)$; this merely extends the space in which D is optimised to include more of the parameters inside $\Pr(y|\mathbf{x})$.

Appendix B: Optimal Solutions

In this section the the objective function D will be minimised in the case where the input space consists of one or more subspaces, within each of which all of the input vector components have the same value. In the language of imaging sensors, these special cases correspond to each sensor viewing a featureless scene (i.e. all pixels having the same brightness value), which is effectively the lowest order term in a Taylor expansion of the spatial variation of pixel brightness values. This might not appear to be an interesting scenario to consider, but it leads to a highly non-trivial optimal network behaviour when D is minimised. More complicated input statistics leads to even more complicated optimal network behaviour, so only the simplest case described above will be considered at first.

1. One Input Subspace

This may be used to optimise the network for a single sensor viewing a featureless scene. For a d-dimensional input space $\Pr(\mathbf{x})$ is thus given by

$$\Pr(\mathbf{x}) = \Pr(x_1) \prod_{i=2}^{d} \delta(x_i - x_1) \tag{B1}$$

whence the objective function D in equation 2.3 reduces to

$$D = 2d \int dx_1 \; \Pr(x_1) \sum_{y=1}^{M} \Pr(y|x_1) \; (x_1 - x_1'(y))^2 \tag{B2}$$

This is d times the objective function for a 1-dimensional soft scalar quantiser which encodes inputs in x_1-space whose PDF is $\Pr(x_1)$.

2. Two Input Subspaces

This may be used to optimise the network for a pair of sensors each of which views a featureless scene, and which are possibly correlated with each other. The one input subspace case above can readily be generalised to more input subspaces. Let the d-dimensional input space be split into two $\frac{d}{2}$-dimensional subspaces, where $\Pr(\mathbf{x})$ is given by

$$\Pr(\mathbf{x}) = \Pr(x_1, x_2) \prod_{i=2}^{\frac{d}{2}} \delta(x_{2\bar{\imath}-1} - x_1) \; \delta(x_{2\bar{\imath}} - x_2) \tag{B3}$$

where one of the subspaces consists of the odd-numbered components, and the other the even-numbered components of the input vector (this particular ordering of the components is not important). Whence the objective function D in equation 2.3 reduces to

$$D = d \int dx_1 \, dx_2 \; \Pr(x_1, x_2) \sum_{y=1}^{M} \Pr(y|x_1, x_2) \; \left((x_1 - x_1'(y))^2 + (x_2 - x_2'(y))^2 \right) \tag{B4}$$

This is $\frac{d}{2}$ times the objective function for a 2-dimensional soft vector quantiser which encodes inputs in (x_1, x_2)-space whose PDF is $\Pr(x_1, x_2)$. This result generalises in the obvious way to a larger number of input subspaces.

3. PMD Posterior Probability

In the above special cases each neuron potentially responds to all of the components of the input vector. If this were to be built in hardware, then each neuron would have a number of inputs equal to the dimensionality of the input space, which becomes unwieldy if the input space had a high dimensionality (e.g. an image). For high-dimensional inputs it is sensible to limit the number of inputs to each neuron, which can readily be implemented by imposing a finite-sized receptive field on the input of each neuron, such that it can respond only to a limited subset of all of the input vector components. This constraint will prevent the ideal vector quantiser solutions from being obtained, so the purpose of this section is to derive the constrained optimal solution. Note that this type of input is a special case of the type of solution that would be obtained by adding a "wire-length" penalty term to the objective function in order to penalise the connection of a neuron to too many input components.

Even if receptive fields are used to restrict the length of the input connections, the posterior probability $\Pr(y|\mathbf{x})$ effectively needs long-range lateral connections between the output neurons in order to implement the normalisation condition $\sum_{y=1}^{M} \Pr(y|\mathbf{x}) = 1$. The simplest example of this is the standard vector quantiser, whose winner-take-all property requires that all neurons are laterally connected to all other neurons *even if* each of them has only a finite-sized receptive field. A partitioned mixture distribution (PMD) posterior probability, in which the posterior probability $\Pr(y|\mathbf{x})$ is only locally connected, can be used to ensure that all the connections in the network are local (see section II C).

a. *Receptive Fields*

Write the input vector as $\mathbf{x} = (\tilde{\mathbf{x}}(y), \bar{\mathbf{x}}(y))$ where $\tilde{\mathbf{x}}(y)$ is the part of $\mathbf{x}$ that lies within the receptive field of neuron y, and, for simplicity, assume that the receptive field used for $\mathbf{x}'(y)$ is chosen to be the same as that for $\tilde{\mathbf{x}}(y)$, and that all receptive fields see the same number w of input components. Because the input vector is split into two subspaces as $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$, its decompositon as $(\tilde{\mathbf{x}}(y), \bar{\mathbf{x}}(y))$ may similarly be split into two subspaces as $\tilde{\mathbf{x}}(y) = (\tilde{\mathbf{x}}_1(y), \tilde{\mathbf{x}}_2(y))$ and $\bar{\mathbf{x}}(y) = (\bar{\mathbf{x}}_1(y), \bar{\mathbf{x}}_2(y))$. Use the orthogonality of $\tilde{\mathbf{x}}(y)$ and $\bar{\mathbf{x}}(y)$ to write (for $i=1,2$)

$$\left\|\bar{\mathbf{x}}_i(y)+\tilde{\mathbf{x}}_i(y)-\mathbf{x}'_i(y)\right\|^2=\left\|\bar{\mathbf{x}}_i(y)\right\|^2+\left\|\tilde{\mathbf{x}}_i(y)-\mathbf{x}'_i(y)\right\|^2$$

and simplify D in equation 2.3 thus

$$D=2\int d\mathbf{x}_1\,d\mathbf{x}_2\,\Pr(\mathbf{x}_1,\mathbf{x}_2)\sum_y\Pr(y|\mathbf{x}_1,\mathbf{x}_2)\left(\left\|\bar{\mathbf{x}}_1(y)\right\|^2+\left\|\bar{\mathbf{x}}_2(y)\right\|^2+\left\|\tilde{\mathbf{x}}_1(y)-\mathbf{x}'_1(y)\right\|^2+\left\|\tilde{\mathbf{x}}_2(y)-\mathbf{x}'_2(y)\right\|^2\right) \quad \text{(B5)}$$

There are two terms to consider.

1. $\left\|\bar{\mathbf{x}}_1(y)\right\|^2+\left\|\bar{\mathbf{x}}_2(y)\right\|^2$. This is the contribution from *outside* the y^{th} receptive field, which is the L_2 norm of those components of the input vector that lie outside the y^{th} receptive field.

2. $\left\|\tilde{\mathbf{x}}_1(y)-\mathbf{x}'_1(y)\right\|^2+\left\|\tilde{\mathbf{x}}_2(y)-\mathbf{x}'_2(y)\right\|^2$: This is the contribution from *inside* the y^{th} receptive field, which is the L_2 norm of those components of the error vector (i.e. input minus reconstruction) that lie inside the y^{th} receptive field.

b. *Simplify the* $\left\|\bar{\mathbf{x}}_1(y)\right\|^2+\left\|\bar{\mathbf{x}}_2(y)\right\|^2$ *Term*

$\left\|\bar{\mathbf{x}}_1(y)\right\|^2+\left\|\bar{\mathbf{x}}_2(y)\right\|^2$ is the L_2 norm of those components of the input vector that lie outside the y^{th} receptive field, which is known once the input vector is specified. Furthermore, because of the assumed input PDF (i.e. all input components in each subspace have the same value), together with the assumed receptive field prescription (i.e. all receptive fields are the same size w), this L_2 norm is independent of y given that $\mathbf{x}$ is known, so this term has the following contribution to D

$$D=(d-w)\left(\int dx_1\,\Pr(x_1)\,x_1^2+\int dx_2\,\Pr(x_2)\,x_2^2\right) \quad \text{(B6)}$$

where $d-w$ is the number of input components that lie *outside* each receptive field.

c. *Simplify the* $\left\|\tilde{\mathbf{x}}_1(y)-\mathbf{x}'_1(y)\right\|^2+\left\|\tilde{\mathbf{x}}_2(y)-\mathbf{x}'_2(y)\right\|^2$ *Term*

Assume that $\Pr(y|\mathbf{x}_1,\mathbf{x}_2)$ has the PMD form of a sum over mixture distribution posterior probabilities (as described in section II C), so that

$$\begin{aligned}\Pr(y|\mathbf{x}_1,\mathbf{x}_2) &= \frac{1}{M}\sum_{y'\in\mathcal{N}^{-1}(y)}\Pr(y|\mathbf{x}_1,\mathbf{x}_2;y') \\ &= \frac{1}{M}Q(\tilde{\mathbf{x}}_1,\tilde{\mathbf{x}}_2|y)\sum_{y'\in\mathcal{N}^{-1}(y)}\frac{1}{\sum_{y''\in\mathcal{N}(y')}Q(\tilde{\mathbf{x}}_1,\tilde{\mathbf{x}}_2|y'')}\end{aligned} \quad \text{(B7)}$$

The overall receptive field that effects the value of $\Pr(y|\mathbf{x}_1,\mathbf{x}_2)$ (for a given y) may be read off this expression. Thus $\tilde{\mathbf{x}}(y'')$ comprises those components of $\mathbf{x}$ that lie within the receptive field of neuron y'', and the $\sum_{y'\in\mathcal{N}^{-1}(y)}\frac{1}{\sum_{y''\in\mathcal{N}(y')}(\cdots)}$ operation compounds these $\tilde{\mathbf{x}}(y'')$ so that the overall set of components of $\mathbf{x}$ that are needed for the purposes of calculating $\Pr(y|\mathbf{x}_1,\mathbf{x}_2)$ is given by (using a somewhat cavalier notation)

$$\tilde{\mathbf{X}}(y)\equiv\bigcup_{y'\in\mathcal{N}^{-1}(y)}\bigcup_{y''\in\mathcal{N}(y')}\tilde{\mathbf{x}}(y'') \quad \text{(B8)}$$

The individual $\Pr(y|\mathbf{x}_1,\mathbf{x}_2;y')$ that contribute to $\Pr(y|\mathbf{x}_1,\mathbf{x}_2)$ each depend on a smaller set of components of $\mathbf{x}$ than the full $\Pr(y|\mathbf{x}_1,\mathbf{x}_2)$, because there is one less summation over a y variable. However, it is convenient, and imposes no constraint, to use the full set of components thus

$$\Pr(y|\mathbf{x}_1,\mathbf{x}_2;y')=\Pr\left(y|\tilde{\mathbf{X}}_1(y),\tilde{\mathbf{X}}_2(y);y'\right) \quad \text{(B9)}$$

The y and y' summations can be interchanged using $\sum_{y=1}^{M}\sum_{y'\in\mathcal{N}^{-1}(y)}(\cdots)=\sum_{y'=1}^{M}\sum_{y\in\mathcal{N}(y')}(\cdots)$, whence the contribution to D is

$$D=\frac{2}{M}\int d\mathbf{x}_1\,d\mathbf{x}_2\,\Pr(\mathbf{x}_1,\mathbf{x}_2)\sum_{y'=1}^{M}\sum_{y\in\mathcal{N}(y')}\Pr\left(y|\tilde{\mathbf{X}}_1(y),\tilde{\mathbf{X}}_2(y);y'\right)\left(\|\tilde{\mathbf{x}}_1(y)-\mathbf{x}_1'(y)\|^2+\|\tilde{\mathbf{x}}_2(y)-\mathbf{x}_2'(y)\|^2\right)\quad\text{(B10)}$$

Because the components of $\tilde{\mathbf{x}}_i(y)$ are a subset of the components of $\tilde{\mathbf{X}}_i(y)$ (for $i=1,2$), $\Pr(\mathbf{x}_1,\mathbf{x}_2)$ can be marginalised to yield

$$\begin{aligned}D&=\frac{2}{M}\sum_{y'=1}^{M}\sum_{y\in\mathcal{N}(y')}\int d\tilde{\mathbf{X}}_1(y)\,d\tilde{\mathbf{X}}_2(y)\,\Pr\left(\tilde{\mathbf{X}}_1(y),\tilde{\mathbf{X}}_2(y)\right)\Pr\left(y|\tilde{\mathbf{X}}_1(y),\tilde{\mathbf{X}}_2(y);y'\right)\\&\quad\times\left(\|\tilde{\mathbf{x}}_1(y)-\mathbf{x}_1'(y)\|^2+\|\tilde{\mathbf{x}}_2(y)-\mathbf{x}_2'(y)\|^2\right)\end{aligned}\quad\text{(B11)}$$

Because $\Pr(\mathbf{x}_1,\mathbf{x}_2)$ specifies that all of the components in each subspace are the same, this contribution to D may be simplified to

$$D=\frac{w}{M}\sum_{y'=1}^{M}\sum_{y\in\mathcal{N}(y')}\int dx_1\,dx_2\,\Pr(x_1,x_2)\Pr(y|x_1,x_2;y')\left((x_1-x_1'(y))^2+(x_2-x_2'(y))^2\right)\quad\text{(B12)}$$

d. Periodic Optimal Solutions

Combining the results from outside (equation B6) and inside (equation B12) the receptive fields yields finally

$$\begin{aligned}D&=(d-w)\left(\int dx_1\,\Pr(x_1)\,x_1^2+\int dx_2\,\Pr(x_2)\,x_2^2\right)\\&\quad+\frac{w}{M}\sum_{y'=1}^{M}\sum_{y\in\mathcal{N}(y')}\int dx_1\,dx_2\,\Pr(x_1,x_2)\Pr(y|x_1,x_2;y')\,16\left((x_1-x_1'(y))^2+(x_2-x_2'(y))^2\right)\end{aligned}\quad\text{(B13)}$$

The first of these terms is constant, so it may be ignored insofar as network optimisation is concerned. The second term is much more interesting. It is the sum of the objective functions of a large number of 2-dimensional soft vector quantisers. However, these objective functions cannot be optimised independently of each other, because the posterior probabilities $\Pr(y|x_1,x_2;y')$ force the neurons to share parameters with each other.

Drop the constant term, and interchange the order of summation to obtain

$$D=w\sum_{y=1}^{M}\int dx_1\,dx_2\,\Pr(x_1,x_2)\Pr(y|x_1,x_2)\left((x_1-x_1'(y))^2+(x_2-x_2'(y))^2\right)\quad\text{(B14)}$$

where $\Pr(y|x_1,x_2)$ is the PMD posterior probability given by

$$\Pr(y|x_1,x_2)=\frac{1}{M}\sum_{y'\in\mathcal{N}^{-1}(y)}\Pr(y|x_1,x_2;y')\quad\text{(B15)}$$

Now suppose that $\Pr(y|x_1,x_2)$ and $x_i'(y)$ have the periodicity property

$$\begin{aligned}\Pr(y+m|x_1,x_2)&=\Pr(y|x_1,x_2)\\x_i'(y+m)&=x_i'(y)\end{aligned}\quad\text{(B16)}$$

where the fact that y is restricted to $1\leq y\leq M$ has been ignored for simplicity, then D can be simplified thus (again, ignoring the fact that y is restricted to $1\leq y\leq M$)

$$\begin{aligned}D&=w\sum_{y_0=0}^{\frac{M}{m}-1}\sum_{y=my_0+1}^{m(y_0+1)}\int dx_1\,dx_2\,\Pr(x_1,x_2)\Pr(y|x_1,x_2)\left((x_1-x_1'(y))^2+(x_2-x_2'(y))^2\right)\\&=w\sum_{y=1}^{m}\int dx_1\,dx_2\,\Pr(x_1,x_2)\,\frac{M}{m}\Pr(y|x_1,x_2)\left((x_1-x_1'(y))^2+(x_2-x_2'(y))^2\right)\end{aligned}\quad\text{(B17)}$$

where $\frac{M}{m}\sum_{y=1}^{m}\Pr(y|x_1,x_2)=1$ follows from $\sum_{y=1}^{M}\Pr(y|x_1,x_2)=1$ and the periodicity property, so $\frac{M}{m}\Pr(y|x_1,x_2)$ serves as a posterior probability for $1\leq y\leq m$.

This demonstrates that *if* the optimal solution is periodic, with period m, then the objective function is proportional to the objective function for a 2-dimensional soft vector quantiser with m neurons. Note that thus far nothing has been said about the actual value of m; its optimal value depends on the interplay between the receptive field size(s), the output layer neighbourhood size(s), and the leakage neighbourhood size(s). Because this type of periodic solution is essentially a set of overlapping m neuron soft vector quantisers, each set of m neurons will typically exhibit the properties of such quantisers. In particular this means that each set of m neurons will have the means to encode and (approximately) reconstruct those components of the input vector that it sees via its receptive fields.

This type of solution is the archetype for orientation maps, where the neurons arrange their properties so that each local patch (corresponding to the m neurons in the periodic solution derived above) has the means to encode whatever orientation of object it sees via its receptive fields. The full derivation of an orientation map would require a more sophisticated analysis than the simple 1-dimensional case derived above.

[1] P Brodatz, *Textures - a photographic album for artists and designers*, Dover, New York, 1966.

[2] G Goodhill, *Correlations, competition and optimality: modelling the development of topography and ocular dominance*, Technical Report CSRP 226, Sussex University, 1992.

[3] T Kohonen, *Self-organisation and associative memory*, Springer-Verlag, Berlin, 1984.

[4] Y Linde, A Buzo, and R M Gray, *An algorithm for vector quantiser design*, IEEE Transactions on Communications **28** (1980), no. 1, 84–95.

[5] S P Luttrell, *Derivation of a class of training algorithms*, IEEE Transactions on Neural Networks **1** (1990), no. 2, 229–232.

[6] ———, *A Bayesian analysis of self-organising maps*, Neural Computation **6** (1994), no. 5, 767–794.

[7] ———, *Partitioned mixture distribution: an adaptive Bayesian network for low-level image processing*, IEE Proceedings on Vision, Image, and Signal Processing **141** (1994), no. 4, 251–260.

[8] ———, *Handbook of neural computation*, ch. Designing analysable networks, pp. B5.3:1–B5.3:8, IOP and OUP, Oxford, 1995.

[9] ———, *Maximum entropy and Bayesian methods*, ch. The partitioned mixture distribution: multiple overlapping density models, pp. 279–286, Kluwer, Dordrecht, 1995.

[10] ———, *A self-organising network for processing data from multiple sensors*, Technical Report DRA/CIS(SE1)/651/11/RP/1, Defence Research Agency, Malvern, 1995.

[11] ———, *A discrete firing event analysis of the adaptive cluster expansion network*, Network: Computation in Neural Systems **7** (1996), no. 2, 285–290.

[12] ———, *Mathematics of neural networks: models, algorithms and applications*, ch. A theory of self-organising neural networks, pp. 240–244, Kluwer, Boston, 1997.

[13] N V Swindale, *The development of topography in the visual cortex: a review of models*, Network: Computation in Neural Systems **7** (1996), no. 2, 161–247.

[14] C J S Webber, *Self-organisation of transformation-invariant detectors for constituents of perceptual patterns*, Network: Computation in Neural Systems **5** (1994), no. 4, 471–496.

Some Theoretical Properties of a Network of Discretely Firing Neurons [*]

Stephen P Luttrell[†]
Defence Evaluation and Research Agency, St Andrews Rd, Malvern, Worcs, WR14 3PS, United Kingdom, tel: +44 (0) 1684-894046, fax: +44 (0) 1684-894384

The problem of optimising a network of discretely firing neurons is addressed. An objective function is introduced which measures the average number of bits that are needed for the network to encode its state. When this is minimised, it is shown that this leads to a number of results, such as topographic mappings, piecewise linear dependence on the input of the probability of a neuron firing, and factorial encoder networks.

I. INTRODUCTION

In this paper the problem of optimising the firing characteristics of a network of discretely firing neurons will be considered. The approach adopted will not be based on any particular model of how real neurons operate, but will focus on theoretically analysing some of the information processing capabilities of a layered network of units (which happen to be called neurons). Ideal network behaviour is derived by choosing the ideal neural properties that minimise an information theoretic objective function which specifies the number of bits required by the network to encode the state of its layers. This is done in preference to assuming a highly specific neural behaviour at the outset, followed by optimisation of a few remaining parameters such as weight and bias values.

Why use an objective function in the first place? An objective function is a very convenient starting point (a set of "axioms", as it were), from which everything else can, in principle, be derived (as "theorems", as it were). An objective function has the same status as a model, which may be falsified should some counterevidence be discovered. The objective function used in this paper is the simplest that is consistent with predicting a number of non-trivial results, such as topographic mappings, and factorial encoders (which are discussed in this paper). However, it does not include any temporal information, nor any biological plausibility constraints (other than the fact that the network is assumed to be layered). More complicated objective functions will be the subject of future publications.

In section II an objective function is introduced, and its connection with discretely firing neural networks is derived. In section III some examples are presented which show how this theory of discretely firing neural networks leads to some non-trivial results.

[*]Typeset in LaTeX on May 3, 2019.
This is archived at http://arxiv.org/abs/1505.00444, 3 May 2015.
This paper was submitted to Special Issue of Neurocomputing on Theoretical Analysis of Real-Valued Function Classes on 19 January 1998. It was not accepted for publication, but it underpins several subsequently published papers. ©1998 British Crown Copyright/DERA

[†]Electronic address: luttrell@signal.dera.gov.uk

II. THEORY

In this section a theory of discretely firing neural networks is developed. Section II A introduces the objective function for optimising an encoder, and section II B shows how this can be applied to the problem of optimising a discretely firing neural network.

A. Objective Function for Optimal Coding

The inspiration for the approach that is used here is the minimum description length (MDL) method [5]. In this paper, a training set vector (which is unlabelled) will be denoted as $\mathbf{x}$, a vector of statistics which are stochastically derived from $\mathbf{x}$ will be denoted as $\mathbf{y}$, and their joint probability density function (PDF) will be denoted as $\Pr(\mathbf{x},\mathbf{y})$. The problem is to learn the functional form of $\Pr(\mathbf{x},\mathbf{y})$, so that vectors $(\mathbf{x},\mathbf{y})$ sampled from $\Pr(\mathbf{x},\mathbf{y})$ can be encoded using the minimum number of bits on average. It is unconventional to consider the problem of encoding $(\mathbf{x},\mathbf{y})$, rather than $\mathbf{x}$ alone, but it turns out that this leads to many useful results.

Thus $\Pr(\mathbf{x},\mathbf{y})$ is approximated by a learnt model $Q(\mathbf{x},\mathbf{y})$, in which case the average number of bits required to encode an $(\mathbf{x},\mathbf{y})$ sampled from the PDF $\Pr(\mathbf{x},\mathbf{y})$ is given by the objective function D, which is defined as

$$D \equiv -\int d\mathbf{x} \sum_{\mathbf{y}} \Pr(\mathbf{x},\mathbf{y}) \log Q(\mathbf{x},\mathbf{y}) \tag{1}$$

Now split D into two contributions by using $\Pr(\mathbf{x},\mathbf{y}) = \Pr(\mathbf{x}) \Pr(\mathbf{y}|\mathbf{x})$ and $Q(\mathbf{x},\mathbf{y}) = Q(\mathbf{x})\, Q(\mathbf{y}|\mathbf{x})$.

$$D = -\int d\mathbf{x}\, \Pr(\mathbf{x}) \sum_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x}) \log Q(\mathbf{x}|\mathbf{y}) - \sum_{\mathbf{y}} \Pr(\mathbf{y}) \log Q(\mathbf{y}) \tag{2}$$

The first term is the cost (i.e. the average number of bits), averaged over all possible values of $\mathbf{y}$, of encoding an $\mathbf{x}$ sampled from $\Pr(\mathbf{x}|\mathbf{y})$ using the model $Q(\mathbf{x}|\mathbf{y})$. This interpretation uses that $\Pr(\mathbf{x})\ \Pr(\mathbf{y}|\mathbf{x}) = \Pr(\mathbf{y})\ \Pr(\mathbf{x}|\mathbf{y})$. The second term is the cost of encoding a $\mathbf{y}$ sampled from $\Pr(\mathbf{y})$ using the model $Q(\mathbf{y})$. Together these two terms correspond to encoding $\mathbf{y}$ (the second term), then encoding $\mathbf{x}$ given that $\mathbf{y}$ is known.

The model $Q(\mathbf{x}, \mathbf{y})$ may be optimised so that it minimises D, and thus leads to the minimum cost of encoding $(\mathbf{x}, \mathbf{y})$ sampled from $\Pr(\mathbf{x}, \mathbf{y})$. Ideally $Q(\mathbf{x}, \mathbf{y}) = \Pr(\mathbf{x}, \mathbf{y})$, but in practice this is not possible because insufficient information is available to determine $\Pr(\mathbf{x}, \mathbf{y})$ exactly (i.e. the training set does not contain an infinite number of $(\mathbf{x}, \mathbf{y})$ vectors). It is therefore necessary to introduce a parametric model $Q(\mathbf{x}, \mathbf{y})$, and to choose the values of the parameters so that D is minimised. If the number of parameters is small enough, and the training set is large enough, then the parameter values can be accurately determined.

A further simplification may be made if $\mathbf{y}$ can occupy much fewer states than $\mathbf{x}$ (given $\mathbf{y}$) can, because then the cost of encoding $\mathbf{y}$ is much less than the cost of encoding $\mathbf{x}$ (given $\mathbf{y}$) (i.e. the second and first terms in equation 2, respectively). In this case, it is a good approximation to retain only the first term in equation 2. This approximation becomes exact if $Q(\mathbf{y})$ assigns equal probability to all states $\mathbf{y}$, because then the third term is a constant. The reason for defining the objective function D as in equation 1, rather than defining it to be the first term of equation 2, is because equation 1 may be readily generalised to more complex systems, such as $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ in which $\Pr(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \Pr(\mathbf{x}) \Pr(\mathbf{y}|\mathbf{x}) \Pr(\mathbf{z}|\mathbf{y})$, and so on. An example of this is given in section III A.

It is possible to relate the minimisation of D to the maximisation of the mutual information I between $\mathbf{x}$ and $\mathbf{y}$. If the cost of encoding an $\mathbf{x}$ sampled from $\Pr(\mathbf{x})$ using the model $Q(\mathbf{x})$ (i.e. $-\int d\mathbf{x}\ \Pr(\mathbf{x})\ \log Q(\mathbf{x})$) and the cost of encoding a $\mathbf{y}$ sampled from $\Pr(\mathbf{y})$ using the model $Q(\mathbf{y})$ (i.e. $-\sum_{\mathbf{y}} \Pr(\mathbf{y})\ \log Q(\mathbf{y})$) are both subtracted from D, then the result is $-\int d\mathbf{x} \sum_{\mathbf{y}} \Pr(\mathbf{x}, \mathbf{y})\ \log\left(\frac{Q(\mathbf{x},\mathbf{y})}{Q(\mathbf{x})\, Q(\mathbf{y})}\right)$. When $Q(\mathbf{x}, \mathbf{y}) \longrightarrow \Pr(\mathbf{x}, \mathbf{y})$ this reduces to (minus) the mutual information I between $\mathbf{x}$ and $\mathbf{y}$. Thus, if the cost of encoding the correlations between $\mathbf{x}$ and $\mathbf{y}$ is much greater than the cost of separately encoding $\mathbf{x}$ and $\mathbf{y}$ (i.e. the $\log(Q(\mathbf{x})\, Q(\mathbf{y}))$ term can be ignored in I), then D-minimisation approximates I-maximisation, which is another commonly used objective function.

B. Application to Neural Networks

In order to apply the above coding theory results to a 2-layer discretely firing neural network, it is necessary to interpret $\mathbf{x}$ as a pattern of activity in the input layer, and $\mathbf{y}$ as the vector of locations in the output layer of a finite number of firing events. The objective function D is then the cost of using the model $Q(\mathbf{x}, \mathbf{y})$ of the network behaviour to encode the state $(\mathbf{x}, \mathbf{y})$ of the neural network (i.e. the input pattern and the location of the firing events), which is sampled from the $\Pr(\mathbf{x}, \mathbf{y})$ that describes the true network behaviour. For instance, a second neural network can be used solely for computing the model $Q(\mathbf{x}, \mathbf{y})$, which is then used to encode the state $(\mathbf{x}, \mathbf{y})$ of the above first neural network. Note that no temporal information is included in this analysis, so the input and output of the network is a static $(\mathbf{x}, \mathbf{y})$ vector containing no time variables.

These two neural networks can be combined into a single hybrid network, in which the machinery for computing the model $Q(\mathbf{x}, \mathbf{y})$ is interleaved with the neural network, whose true behavior is described by $\Pr(\mathbf{x}, \mathbf{y})$. The notation of equation 2 can now be expressed in more neural terms, where $\Pr(\mathbf{y}|\mathbf{x})$ is then a recognition model (i.e. bottom-up) and $Q(\mathbf{x}|\mathbf{y})$ is then a generative model (i.e. top-down), both of which live inside the same neural network. This is an unsupervised neural network, because it is trained with examples of only $\mathbf{x}$-vectors, and the network uses its $\Pr(\mathbf{y}|\mathbf{x})$ to stochastically generate a $\mathbf{y}$ from each $\mathbf{x}$.

Now introduce a Gaussian parametric model $Q(\mathbf{x}|\mathbf{y})$

$$Q(\mathbf{x}|\mathbf{y}) = \frac{1}{\left(\sqrt{2\pi}\sigma\right)^{\dim \mathbf{x}}} \exp\left(-\frac{\left\|\mathbf{x} - \mathbf{x}'(\mathbf{y})\right\|^2}{2\sigma^2}\right) \tag{3}$$

where $\mathbf{x}'(\mathbf{y})$ is the centroid of the Gaussian (given $\mathbf{y}$), σ is the standard deviation of the Gaussian. Also define a soft vector quantiser (VQ) objective function D_{VQ} as

$$D_{\mathrm{VQ}} \equiv 2 \int d\mathbf{x}\ \Pr(\mathbf{x}) \sum_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x})\ \left\|\mathbf{x} - \mathbf{x}'(\mathbf{y})\right\|^2 \tag{4}$$

which is (twice) the average Euclidean reconstruction error that results when $\mathbf{x}$ is probabilistically encoded as $\mathbf{y}$ and then deterministically reconstructed as $\mathbf{x}'(\mathbf{y})$. These definitions of $Q(\mathbf{x}|\mathbf{y})$ and D_{VQ} allow D to be written as

$$D = \frac{1}{4\sigma^2} D_{\mathrm{VQ}} - \log\left(\sqrt{2\pi}\sigma\right) \dim \mathbf{x} - \sum_{\mathbf{y}} \Pr(\mathbf{y})\ \log Q(\mathbf{y}) \tag{5}$$

where the second term is constant, and the third term may be ignored if $\mathbf{y}$ can occupy much fewer states than $\mathbf{x}$ (given $\mathbf{y}$) can. The conditions under which the third term can be ignored are satisfed in a neural network, because $\mathbf{x}$ is an activity pattern, and $\mathbf{y}$ as the vector of locations of a finite number of firing events.

The first term of D is proportional to D_{VQ}, whose properties may be investigated using the techniques in [3]. Assume that there are n firing events, so that $\mathbf{y} = (y_1, y_2, \cdots, y_n)$, then the marginal probabilities of the symmetric part $S\left[\Pr(\mathbf{y}|\mathbf{x})\right]$ of $\Pr(\mathbf{y}|\mathbf{x})$ under interchange of its $(y_1, y_2, \cdots, y_n)$ arguments are given by

$$
\begin{aligned}
\Pr(y_1|\mathbf{x}) &\equiv \sum_{y_2,\cdots,y_n=1}^{M} S\left[\Pr(y_1,y_2,\cdots,y_n|\mathbf{x})\right] \\
\Pr(y_1,y_2|\mathbf{x}) &\equiv \sum_{y_3,\cdots,y_n=1}^{M} S\left[\Pr(y_1,y_2,\cdots,y_n|\mathbf{x})\right]
\end{aligned}
\tag{6}
$$

where $\Pr(y_1|\mathbf{x})$ may be interpreted as the probability that the next firing event occurs on neuron y (given $\mathbf{x}$), Also define 2 useful integrals, D_1 and D_2, as

$$
\begin{aligned}
D_1 &\equiv \frac{2}{n}\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{y=1}^{M}\Pr(y|\mathbf{x})\,\left\|\mathbf{x}-\mathbf{x}'(y)\right\|^2 \\
D_2 &\equiv \frac{2(n-1)}{n}\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{y_1,y_2=1}^{M}\Pr(y_1,y_2|\mathbf{x})\,(\mathbf{x}-\mathbf{x}'(y_1))\cdot(\mathbf{x}-\mathbf{x}'(y_2))
\end{aligned}
\tag{7}
$$

where $\mathbf{x}'(y)$ is any vector function of y (i.e. not necessarily related to $\mathbf{x}'(\mathbf{y})$), to yield the following upper bound on D_{VQ}

$$D_{\mathrm{VQ}} \leq D_1 + D_2 \tag{8}$$

where D_1 is non-negative but D_2 can have either sign, and the inequality reduces to an equality in the case $n = 1$. Thus far nothing specific has been assumed about $\Pr(\mathbf{y}|\mathbf{x})$, other than the fact that it contains no temporal information, so the upper bound on D_{VQ} applies whatever the form of $\Pr(\mathbf{y}|\mathbf{x})$.

If the firing events occur independently of each other (given $\mathbf{x}$), then $\Pr(y_1,y_2|\mathbf{x}) = \Pr(y_1|\mathbf{x})\,\Pr(y_2|\mathbf{x})$, which allows D_2 to be redefined as

$$D_2 \equiv \frac{2(n-1)}{n}\int d\mathbf{x}\,\Pr(\mathbf{x})\left\|\mathbf{x}-\sum_{y=1}^{M}\Pr(y|\mathbf{x})\,\mathbf{x}'(y)\right\|^2 \tag{9}$$

where D_2 is non-negative.

In summary, the assumptions which have been made in order to obtain the upper bound on D_{VQ} in equation 8 with the definition of D_1 as given in equation 7 and D_2 as given in equation 9 are: no temporal information is included in the network state vector $(\mathbf{x},\mathbf{y})$, $\mathbf{y}$ can occupy much fewer states than $\mathbf{x}$ (given $\mathbf{y}$) can, and firing events occur independently of each other (given $\mathbf{x}$). In reality, there is always temporal information available, and the firing events are correlated with each other, so a more realistic objective function could be constructed. However, it is worthwhile to consider the consequences of equation 8, because it turns out that it leads to many non-trivial results.

The upper bound on D_{VQ} may be minimised with respect to all free parameters in order to obtain a least upper bound. In the case of independent firing events, the free parameters are the $\mathbf{x}'(y)$ and the $\Pr(y|\mathbf{x})$. These two types of parameters cannot be independently optimised, because they correspond to the generative and recognition models implicit in the neural network, respectively.

A gradient descent algorithm for optimising the parameter values may readily be obtained by differentiating D_1 and D_2 with respect to $\mathbf{x}'(y)$ and $\Pr(y|\mathbf{x})$. Given the freedom to explore the entire space of functions $\Pr(y|\mathbf{x})$, the optimum neural firing behaviour (given $\mathbf{x}$) can in principle be determined, and in certain simple cases this can be determined by inspection. If this option is not available, such as would be the case if biological contraints restricted the allowed functional form of $\Pr(y|\mathbf{x})$, then a limited search of the entire space of functions $\Pr(y|\mathbf{x})$ can be made by invoking parametric model of the neural firing behaviour (given $\mathbf{x}$).

III. EXAMPLES

In this section several examples are presented which illustrate the use of $D_1 + D_2$ in the optimisation of discretely firing neural networks. In section III A a topographic mapping network is derived from D_1 alone, in section III B $\Pr(y|\mathbf{x})$ that minimises $D_1 + D_2$ is shown to be piecewise linear, and a solved example is presented. Finally, in section III C a more detailed worked example is presented, which demonstrates how a factorial encoder emerges when $D_1 + D_2$ is minimised.

A. Topographic Mapping Neural Network

When an appropriate from of V_{VQ} is considered, it can be seen that it leads to a network that is closely related to Kohonen's topographic mapping network [1].

The derivation of a topographic mapping network that was given in [2] will now be recast in the framework of section II B. Thus, consider the objective function for a

3-layer network $(\mathbf{x}, y, z)$, in which (compare equation 1)

$$D = -\int \sum_{z} d\mathbf{x}\ \Pr(\mathbf{x}, z)\ \log Q(\mathbf{x}, z) \tag{10}$$

where the cost of encoding y has been ignored, so that effectively only a 2-layer network $(\mathbf{x}, z)$ is visible, and D_{VQ} is given by

$$D_{\mathrm{VQ}} = 2\int d\mathbf{x}\ \Pr(\mathbf{x}) \sum_{z=1}^{M_z} \Pr(z|\mathbf{x})\ \left\| \mathbf{x} - \mathbf{x}'(z) \right\|^2 \tag{11}$$

This expression for D_{VQ} explicitly involves $(\mathbf{x}, z)$, but it may be manipulated into a form that explicitly involves $(\mathbf{x}, y)$. In order to make simplify this calculation, D_{VQ} will be replaced by the equivalent objective function

$$D_{\mathrm{VQ}} = \int d\mathbf{x}\ \Pr(\mathbf{x}) \sum_{z=1}^{M_z} \Pr(z|\mathbf{x}) \int d\mathbf{x}'\ \Pr(\mathbf{x}'|z)\ \left\| \mathbf{x} - \mathbf{x}' \right\|^2 \tag{12}$$

Now introduce dummy integrations over y to obtain

$$D_{\mathrm{VQ}} = \int d\mathbf{x}\ \Pr(\mathbf{x}) \sum_{y=1}^{M_y} \Pr(y|\mathbf{x}) \sum_{z=1}^{M_z} \Pr(z|y) \sum_{y'=1}^{M_y} \Pr(y'|z) \int d\mathbf{x}'\ \Pr(\mathbf{x}'|y')\ \left\| \mathbf{x} - \mathbf{x}' \right\|^2 \tag{13}$$

and rearrange to obtain

$$D_{\mathrm{VQ}} = \int d\mathbf{x}\ \Pr(\mathbf{x}) \sum_{y'=1}^{M_y} \Pr(y'|\mathbf{x}) \int d\mathbf{x}'\ \Pr(\mathbf{x}'|y')\ \left\| \mathbf{x} - \mathbf{x}' \right\|^2 \tag{14}$$

where

$$\begin{aligned} \Pr(y'|y) &= \sum_{z=1}^{M_z} \Pr(y'|z)\,\Pr(z|y) \\ \Pr(y'|\mathbf{x}) &= \sum_{y=1}^{M_y} \Pr(y'|y)\,\Pr(y|\mathbf{x}) \end{aligned} \tag{15}$$

which may be replaced by the equivalent objective function

$$D_{\mathrm{VQ}} = 2\int d\mathbf{x}\ \Pr(\mathbf{x}) \sum_{y'=1}^{M_y} \Pr(y'|\mathbf{x})\ \left\| \mathbf{x} - \mathbf{x}'(y') \right\|^2 \tag{16}$$

By manipulating D_{VQ} from the form it has in equation 11 to the form it has in equation 16, it becomes clear that optimisation of the $(\mathbf{x}, z)$ network involves optimisation of the $(\mathbf{x}, y')$ subnetwork, for which an objective function can be written that uses a $\Pr(y'|\mathbf{x})$ as defined in equation 15. When optimising the $(\mathbf{x}, y')$ subnetwork, $\Pr(y'|y)$ takes account of the effect that z has on y.

If $n = 1$, so that only 1 firing event is observed, then $D_{\mathrm{VQ}} = D_1$, and the optimum $\Pr(y|\mathbf{x})$ must ensure that y depends deterministically on $\mathbf{x}$, so that $\Pr(y|\mathbf{x}) = \delta_{y,y(\mathbf{x})}$ where $y(\mathbf{x})$ is an encoding function that converts $\mathbf{x}$ into the index of the neuron that fires in response to $\mathbf{x}$. This allows D_{VQ} to be simplified to

$$D_{\mathrm{VQ}} = 2\int d\mathbf{x}\ \Pr(\mathbf{x}) \sum_{y'=1}^{M_y} \Pr(y'|y(\mathbf{x}))\ \left\| \mathbf{x} - \mathbf{x}'(y') \right\|^2 \tag{17}$$

where $\Pr(y'|y(\mathbf{x}))$ is $\Pr(y'|y)$ with y replaced by $y(\mathbf{x})$. Note that if $\Pr(y'|y) = \delta_{y,y'}$ then D_{VQ} reduces to the objective function $2\int d\mathbf{x}\ \Pr(\mathbf{x})\ \left\| \mathbf{x} - \mathbf{x}'(y(\mathbf{x})) \right\|^2$ for a standard vector quantiser (VQ).

The optimum $y(\mathbf{x})$ is given by $y(\mathbf{x}) = \underset{y}{\arg\min} \sum_{y'=1}^{M_y} \Pr(y'|y)\ \left\| \mathbf{x} - \mathbf{x}'(y') \right\|^2$ (which is not quite the same as the $y(\mathbf{x}) = \underset{y}{\arg\min}\ \left\| \mathbf{x} - \mathbf{x}'(y) \right\|^2$ used by Kohonen in his topographic mapping neural network [1]), and a gradient descent algorithm for updating $\mathbf{x}'(y')$ is $\mathbf{x}'(y') \longrightarrow \mathbf{x}'(y') + \varepsilon\ \Pr(y'|y(\mathbf{x}))$ (which is identical to Kohonen's prescription [1]). The $\Pr(y'|y)$ may thus be interpreted as the neighbourhood function, and the $\mathbf{x}'(y')$ may be interpreted as the weight vectors, of a topographic mapping. Because all states y that can give rise to the same state z (as specified by $\Pr(z|y)$) become neighbours (as specified by $\Pr(y'|y)$ in equation 15), $\Pr(y'|y)$ includes a much larger class of neighbourhood functions than has hitherto been used in topographic mapping neural networks.

Because of the principled way in which the topographic mapping objective function has been derived here, it is the preferred way to optimise topographic mapping networks. It also allows the objective function to be generalised to the case $n > 1$, where more than one firing event is observed.

B. Piecewise Linear Probability of Firing

The optimal $\Pr(y|\mathbf{x})$ has some interesting properties that can be obtained by inspecting its stationarity condition. For instance, the $\Pr(y|\mathbf{x})$ that minimise $D_1 + D_2$ will be shown to be piecewise linear functions of $\mathbf{x}$.

Thus, functionally differentiate $D_1 + D_2$ with respect to $\log \Pr(y|\mathbf{x})$, where logarithmic differentation implicitly imposes the constraint $\Pr(y|\mathbf{x}) \geq 0$, and use a Lagrange multiplier term $L \equiv \int d\mathbf{x}'\, \lambda(\mathbf{x}') \sum_{y'=1}^{M} \Pr(y'|\mathbf{x}')$ to impose the normalisation constraint $\sum_{y=1}^{M} \Pr(y|\mathbf{x}) = 1$ for each $\mathbf{x}$, to obtain

$$\begin{aligned}\frac{\delta\left(D_1+D_2-L\right)}{\delta \log \Pr\left(y|\mathbf{x}\right)} &= \frac{2}{n}\Pr\left(\mathbf{x}\right)\,\Pr(y|\mathbf{x})\left\|\mathbf{x}-\mathbf{x}'(y)\right\|^2 \\ &\quad -\frac{4(n-1)}{n}\Pr(\mathbf{x})\,\Pr(y|\mathbf{x})\,\mathbf{x}'(y)\cdot\left(\mathbf{x}-\sum_{y=1}^{M}\Pr(y|\mathbf{x})\,\mathbf{x}'(y)\right) \\ &\quad -\lambda(\mathbf{x})\,\Pr(y|\mathbf{x}) \end{aligned} \tag{18}$$

The stationarity condition implies that $\sum_{y=1}^{M} \Pr(y|\mathbf{x}) \frac{\delta(D_1+D_2-L)}{\delta \Pr(y|\mathbf{x})} = 0$, which may be used to determine the Lagrange multiplier function $\lambda(\mathbf{x})$. When $\lambda(\mathbf{x})$ is substituted back into the stationarity condition itself, it yields

$$0 = \Pr(\mathbf{x})\,\Pr(y|\mathbf{x})\sum_{y'=1}^{M}\left(\Pr(y'|\mathbf{x})-\delta_{y,y'}\right)\mathbf{x}'(y')\cdot\left(\frac{\mathbf{x}'(y')}{2}-n\,\mathbf{x}+(n-1)\sum_{y''=1}^{M}\Pr(y''|\mathbf{x})\,\mathbf{x}'(y'')\right) \tag{19}$$

There are several classes of solution to this stationarity condition, corresponding to one (or more) of the three factors in equation 19 being zero.

1. $\Pr(\mathbf{x}) = 0$ (the first factor is zero). If the input PDF is zero at $\mathbf{x}$, then nothing can be deduced about $\Pr(y|\mathbf{x})$, because there is no training data to explore the network's behaviour at this point.

2. $\Pr(y|\mathbf{x}) = 0$ (the second factor is zero). This factor arises from the differentiation with respect to $\log \Pr(y|\mathbf{x})$, and it ensures that $\Pr(y|\mathbf{x}) < 0$ cannot be attained. The singularity in $\log \Pr(y|\mathbf{x})$ when $\Pr(y|\mathbf{x}) = 0$ is what causes this solution to emerge.

3. $\sum_{y'=1}^{M} \left(\Pr(y'|\mathbf{x}) - \delta_{y,y'}\right) \mathbf{x}'(y')\cdot(\cdots) = 0$ (the third factor is zero). The solution to this equation is a $\Pr(y|\mathbf{x})$ that has a piecewise linear dependence on $\mathbf{x}$. This result can be seen to be intuitively reasonable because $D_1 + D_2$ is of the form $\int d\mathbf{x}\, \Pr(\mathbf{x})\, f(\mathbf{x})$, where $f(\mathbf{x})$ is a linear combination of terms of the form $\mathbf{x}^i\, \Pr(y|\mathbf{x})^j$ (for $i = 0, 1, 2$ and $j = 0, 1, 2$), which is a quadratic form in $\mathbf{x}$ (ignoring the $\mathbf{x}$-dependence of $\Pr(y|\mathbf{x})$). However, the terms that appear in this linear combination are such that a $\Pr(y|\mathbf{x})$ that is a piecewise linear function of $\mathbf{x}$ guarantees that $f(\mathbf{x})$ is a piecewise linear combination of terms of the form $\mathbf{x}^i$ (for $i = 0, 1, 2$), which is a quadratic form in $\mathbf{x}$ (the normalisation constraint $\sum_{y=1}^{M} \Pr(y|\mathbf{x}) = 1$ is used to remove a contribution to that is potentially quartic in $\mathbf{x}$). Thus a piecewise linear dependence of $\Pr(y|\mathbf{x})$ on $\mathbf{x}$ does not lead to any dependencies on $\mathbf{x}$ that are not already explicitly present in $D_1 + D_2$. The stationarity condition on $\Pr(y|\mathbf{x})$ (see equation 19) then imposes conditions on the allowed piecewise linearities that $\Pr(y|\mathbf{x})$ can have.

For the purpose of doing analytic calculations, it is much easier to obtain analytic results with the ideal piecewise linear $\Pr(y|\mathbf{x})$ than with some other functional form. If the optimisation of $\Pr(y|\mathbf{x})$ is constrained, by introducing a parametric form which has some biological plausibility, for instance, then analytic optimum solutions are not in general possible to calculate, and it becomes necessary to resort to numerical simulations. Piecewise linear $\Pr(y|\mathbf{x})$ should therefore be regarded as a convenient theoretical laboratory for investigating the properties of idealised neural networks.

1. Solved Example

A simple example illustrates how the piecewise linearity property of $\Pr(y|\mathbf{x})$ may be used to find optimal solutions. Thus consider a 1-dimensional input coordinate $x \in [-\infty, +\infty]$, with $\Pr(x) = P_0$. Assume that the number of neurons M tends to infinity in such a way that there is 1 neuron per unit length of x, so that $\Pr(y|x) = p(y - x)$, where the piecewise linear property gives $p(x)$ as

$$p(x) = \begin{cases} 1 & |x| \leq \frac{1}{2} - s \\ \frac{2s-2|x|+1}{4s} & \frac{1}{2} - s \leq |x| \leq \frac{1}{2} + s \\ 0 & |x| \geq \frac{1}{2} + s \end{cases} \tag{20}$$

and by symmetry $x'(y) = y$.

This $\Pr(y|x)$ and $x'(y)$ allow D_1 to be derived as

$$\begin{aligned} D_1 \text{ (per neuron)} &= \frac{2P_0}{n} \left(\int_{-\frac{1}{2}+s}^{\frac{1}{2}-s} dx\, x^2 + 2 \int_{\frac{1}{2}-s}^{\frac{1}{2}+s} dx\, \frac{2s-2x+1}{4s}\, x^2 \right) \\ &= \frac{P_0}{6n} \left(1 + 4s^2\right) \end{aligned} \tag{21}$$

and D_2 to be derived as

$$\begin{aligned} D_2 \text{ (per unit length)} &= \frac{2(n-1)P_0}{n} \left(\int_{-\frac{1}{2}+s}^{\frac{1}{2}-s} x^2\, dx + 2 \int_{\frac{1}{2}-s}^{\frac{1}{2}} \left(x - \frac{2s+2(x-1)+1}{4s} \right)^2 dx \right) \\ &= \frac{(n-1)P_0}{6n} (2s-1)^2 \end{aligned} \tag{22}$$

Because there is one neuron per unit length, the contribution per unit length to D_1+D_2 is the sum of the above two results

$$D_1 + D_2 \text{ (per unit length)} = \frac{P_0}{6n} \left(n(2s-1)^2 + 4s \right) \tag{23}$$

If D_1+D_2 is differentiated with respect to s, then stationarity condition $\frac{d(D_1+D_2)}{ds} = 0$ yields the optimum value of s as

$$s = \frac{n-1}{2n} \tag{24}$$

and the stationary value of $D_1 + D_2$ as

$$D_1 + D_2 \text{ (per unit length)} = \frac{(2n-1)\, P_0}{6n^2} \tag{25}$$

When $n = 1$ the stationary solution reduces to $s = 0$ and $D_1 + D_2$ (per unit length) $= \frac{P_0}{6}$, which is a standard vector quantiser with nonoverlapping neural response regions which partition the input space into unit width quantisation cells, so that for all x there is exactly one neuron that responds. Although the neurons have been manually arranged in topographic order by imposing $\Pr(y|x) = p(y-x)$, any permutation of the neuron indices in this stationary solution will also be stationary solution. This derivation could be generalised to the type of 3-layer network that was considered in section III A , in which case a neighbourhod function $\Pr(y'|y)$ would emerge automatically.

As $n \longrightarrow \infty$ the stationary solution behaves as $s \longrightarrow \frac{1}{2}$ and $D_1 + D_2$ (per unit length) $\longrightarrow \frac{P_0}{3n}$, with overlapping linear neural response regions which cover the input space, so that for all x there are exactly two neurons that respond with equal and opposite linear dependence on x. As $n \longrightarrow \infty$ the ratio of the number of firing events that occur on these two neurons is sufficient to determine x to $\mathcal{O}\left(\frac{1}{n}\right)$. When $n = \infty$ this stationary solution is $s = \frac{1}{2}$ and $D_1 + D_2$ (per unit length) $= 0$. However, when $n = \infty$ there are infinitely many other ways in which the neurons could be used to yield $D_1 + D_2$ (per unit length) $= 0$, because only the D_2 term contributes, and it is 0 when $x = \sum_{y=1}^{M} \Pr(y|x)\, x'(y)$. This is possible for any set of basis elements $x'(y)$ that span the input space, provided that the expansion coefficients $\Pr(y|x)$ satisfy $\Pr(y|x) \geq 0$. In this 1-dimensional example only two basis elements are required (i.e. $M = 2$), which are $x'(1) = -\infty$ and $x'(2) = +\infty$. More generally, for this type of stationary solution, $M = \dim \mathbf{x} + 1$ is required to span the input space in such a way that $\Pr(y|x) \geq 0$, and if $M < \dim \mathbf{x} + 1$ then the stationary solution will span the input subspace (of dimension $M - 1$) that has the largest variance.

The $n = 1$ and $n \longrightarrow \infty$ limiting cases are very different. When $n = 1$ the optimum network splits up the input space into non-overlapping quantisation cells, and as $n \longrightarrow \infty$ the optimum network does a linear decomposition of the input space using non-negative expansion coefficients. This behaviour occurs because for $n > 1$ the neurons can cooperate when encoding the input x, so that by allowing more than one neuron to fire in response to x, the encoded version of x is distributed over more than one neuron. In the above 1-dimensional example, the code is spread over one or two neurons depending on the value of x. This cooperation amongst neurons is a property of the coherent part D_2 of the upper bound on D_{VQ} (see equation 8).

C. Factorial Encoder Network

For certain types of distribution of data in input space the optimal network consists of a number of subnetworks, each of which responds to only a subspace of the input space. This is called factorial encoding, where the encoded input is distributed over more than one neuron, and this distributed code typically has a much richer structure than was encountered in section III B.

The simplest problem that demonstrates factorial encoding will now be investigated (this example was presented in [4], but the derivation given here is more direct). Thus, assume that the data in input space uniformly populates the surface of a 2-torus $S^1 \times S^1$. Each of the S^1 is a plane unit circle embedded in $R^1 \times R^1$ and centred on the origin, and $S^1 \times S^1$ is the Cartesian product of a pair of such circles. Overall, the 2-torus lives in a 4-dimensional input space whose elements are denoted as $\mathbf{x} = (x_1, x_2, x_3, x_4)$, where one of the circles lives in (x_1, x_2) and the other lives in (x_3, x_4). These circles may be parameterised by angular degrees of freedom θ_{12} and θ_{34}, respectively.

The optimal $\Pr(y|\mathbf{x})$ (i.e. a piecewise linear stationary solution of the type that was encountered in section III B could be derived from this input data PDF $\Pr(\mathbf{x})$. However, the properties of the sought-after optimal $\Pr(y|\mathbf{x})$ are preserved if one restricts the solution space to the following types of $\Pr(y|\mathbf{x})$

$$\Pr(y|\mathbf{x}) = \begin{cases} \delta_{y,y(\theta_{12})} \text{ or } \delta_{y,y(\theta_{34})} & \text{type 1} \\ \delta_{y,y(\theta_{12},\theta_{34})} & \text{type 2} \\ \frac{1}{2}\left(\delta_{y,y_{12}(\theta_{12})} + \delta_{y,y_{34}(\theta_{34})}\right) & \text{type 3} \end{cases} \tag{26}$$

where $y(\theta_{12})$ and $y_{12}(\theta_{12})$ encode θ_{12}, $y(\theta_{34})$ and $y_{34}(\theta_{34})$ encode θ_{34}, and $y(\theta_{12}, \theta_{34})$ encodes $(\theta_{12}, \theta_{34})$. The allowed ranges of the code indices are $1 \leq y(\theta_{12}) \leq M$ (and similarly $y(\theta_{34})$), $1 \leq y_{12}(\theta_{12}) \leq \frac{M}{2}$, $\frac{M}{2} + 1 \leq y_{34}(\theta_{34}) \leq M$, and $1 \leq y(\theta_{12}, \theta_{34}) \leq M$. The type 1 solution assumes that all M neurons respond only to θ_{12} (or, alternatively, all respond only to θ_{34}), the type 2 solution assumes that all M neurons respond to $(\theta_{12}, \theta_{34})$, and the type 3 solution (which is very simple type of factorial encoder) assumes that $\frac{M}{2}$ neurons respond only to θ_{12}, and the other $\frac{M}{2}$ neurons respond only to θ_{34}.

In order to derive explicit results for the stationary value of $D_1 + D_2$, it is necessary to optimise the $\mathbf{x}'(y)$. The stationary condition on $\mathbf{x}'(y)$ may readily be deduced from the stationarity condition $\frac{\partial(D_1+D_2)}{\partial \mathbf{x}'(y)} = 0$ as

$$n \int d\mathbf{x} \, \Pr(\mathbf{x}|y) \, \mathbf{x} = \mathbf{x}'(y) + (n-1) \int d\mathbf{x} \, \Pr(\mathbf{x}|y) \sum_{y'=1}^{M} \Pr(y'|\mathbf{x}) \, \mathbf{x}'(y') \tag{27}$$

If $\Pr(y|\mathbf{x})$ (and hence $\Pr(\mathbf{x}|y)$) are inserted into this stationarity condition, then it may be solved for the corresponding $\mathbf{x}'(y)$.

Assume that the encoding functions partition up the 2-torus symmetrically, the three types of solution may be optimised as described in the following three sections.

1. *Type 1 Solution*

Assume that $\Pr(y|\mathbf{x}) = \delta_{y,y(x_1,x_2)}$, and that the $y = 1$ quantisation cell is the Cartesian product of the arcs $|\theta_{12}| \leq \frac{\pi}{M}$ and $\theta_{34} \leq 2\pi$ of the 2 unit circles that form the 2-torus, then the stationarity condition for $\mathbf{x}'(1)$ becomes

$$n \frac{M}{2\pi} \int_{-\frac{\pi}{M}}^{\frac{\pi}{M}} d\theta_{12} \frac{1}{2\pi} \int_0^{2\pi} d\theta_{34} \, (\cos\theta_{12}, \sin\theta_{12}, \cos\theta_{34}, \sin\theta_{34}) = \mathbf{x}'(1) + (n-1) \frac{M}{2\pi} \int_{-\frac{\pi}{M}}^{\frac{\pi}{M}} d\theta_{12} \frac{1}{2\pi} \int_0^{2\pi} d\theta_{34} \, \mathbf{x}'(1) \tag{28}$$

which yields the solution $\mathbf{x}'(1) = \left(\frac{M}{\pi} \sin\left(\frac{\pi}{M}\right), 0, 0, 0\right)$. The first two components are the centroid of the arc $|\vartheta| \leq \frac{\pi}{M}$ of a unit circle centred on the origin. All of the $\mathbf{x}'(y)$ can be obtained by rotating $\mathbf{x}'(1)$ about the origin by multiples of $\frac{2\pi}{M}$. Using the assumed symmetry of the solution, the expression for $D_1 + D_2$ becomes

$$D_1 + D_2 = \frac{M}{\pi} \int_{-\frac{\pi}{M}}^{\frac{\pi}{M}} d\theta_{12} \left\| (\cos\theta_{12}, \sin\theta_{12}) - \left(\frac{M}{\pi} \sin\left(\frac{\pi}{M}\right), 0\right) \right\|^2 + \frac{1}{\pi} \int_0^{2\pi} d\theta_{34} \, \left\| (\cos\theta_{34}, \sin\theta_{34}) - (0,0) \right\|^2 \tag{29}$$

where the first (or second) term corresponds to the subspace to which the neurons respond (or not respond). This gives the stationary value of $D_1 + D_2$ as $D_1 + D_2 = 4 - \frac{2M^2}{\pi^2} \sin^2\left(\frac{\pi}{M}\right)$. Only one neuron can fire (given $\mathbf{x}$), because $\Pr(y|\mathbf{x}) = \delta_{y,y(\theta_{12})}$ or $\delta_{y,y(\theta_{34})}$, no further information about $\mathbf{x}$ can be obtained after the first firing event has occurred, so this result for $D_1 + D_2$ is independent of n, as expected.

2. *Type 2 Solution*

Assume that the $y = 1$ quantisation cell is the Cartesian product of the arcs $|\vartheta_{12}| \leq \frac{\pi}{\sqrt{M}}$ and $|\vartheta_{34}| \leq \frac{\pi}{\sqrt{M}}$ of the two unit circles that form the 2-torus. The stationarity condition for $\mathbf{x}'(1)$ can be deduced from the type 1 case with the replacement $M \longrightarrow \sqrt{M}$, which gives $\mathbf{x}'(1) = \left(\frac{\sqrt{M}}{\pi} \sin\left(\frac{\pi}{\sqrt{M}}\right), 0, \frac{\sqrt{M}}{\pi} \sin\left(\frac{\pi}{\sqrt{M}}\right), 0\right)$. The expression for $D_1 + D_2$ may similarly be deduced from the type 1 case as twice the first term in equation 29 with the replacement $M \longrightarrow \sqrt{M}$, to yield the stationary value of $D_1 + D_2$ as $D_1 + D_2 = 4 - \frac{4M}{\pi^2} \sin^2\left(\frac{\pi}{\sqrt{M}}\right)$. As in the type 1 case, this result for $D_1 + D_2$ is independent of n.

3. *Type 3 Solution*

The stationarity condition for $\mathbf{x}'(1)$ can be written by analogy with the type 1 case, with the replacement $M \longrightarrow \frac{M}{2}$, and modifying the last term to take account of the more complicated form of $\Pr(y|\mathbf{x})$, to yield

$$n \frac{M}{4\pi} \int_{-\frac{2\pi}{M}}^{\frac{2\pi}{M}} d\theta_{12} \frac{1}{2\pi} \int_0^{2\pi} d\theta_{34} \left(\cos\theta_{12}, \sin\theta_{12}, \cos\theta_{34}, \sin\theta_{34}\right) = \mathbf{x}'(1) + \frac{1}{2}(n-1) \frac{M}{4\pi} \int_{-\frac{2\pi}{M}}^{\frac{2\pi}{M}} d\theta_{12} \frac{1}{2\pi} \int_0^{2\pi} d\theta_{34}\, \mathbf{x}'(1) \quad (30)$$

where $\frac{1}{2\pi} \int_0^{2\pi} d\theta_{34} \sum_{y'=\frac{M}{2}+1}^{M} \delta_{y',y_{34}(\theta_{34})}\, \mathbf{x}'(y') = 0$ has been used (this follows from the assumed symmetry of the solution). This yields the solution $\mathbf{x}'(1) = \frac{2n}{n+1} \left(\frac{M}{2\pi} \sin\left(\frac{2\pi}{M}\right), 0, 0, 0\right)$. Using the assumed symmetry of the solution, the expression for D_1 becomes

$$D_1 = \frac{2}{n} \left(\frac{M}{4\pi} \int_{-\frac{2\pi}{M}}^{\frac{2\pi}{M}} d\theta_{12} \left\| (\cos\theta_{12}, \sin\theta_{12}) - \frac{2n}{n+1} \left(\frac{M}{2\pi} \sin\left(\frac{2\pi}{M}\right), 0\right) \right\|^2 + \frac{1}{2\pi} \int_0^{2\pi} d\theta_{34} \left\| (\cos\theta_{34}, \sin\theta_{34}) \right\|^2 \right) \quad (31)$$

and the expression for D_2 becomes

$$D_2 = \frac{2(n-1)}{n} \left(\frac{M}{2\pi} \int_{-\frac{2\pi}{M}}^{\frac{2\pi}{M}} d\theta_{12} \left\| (\cos\theta_{12}, \sin\theta_{12}) - \frac{n}{n+1} \left(\frac{M}{2\pi} \sin\left(\frac{2\pi}{M}\right), 0\right) \right\|^2 \right) \quad (32)$$

This gives the stationary value of $D_1 + D_2$ as $D_1 + D_2 = 4 - \frac{nM^2}{(n+1)\pi^2} \sin^2\left(\frac{2\pi}{M}\right)$. Because $\Pr(y|\mathbf{x}) = \frac{1}{2}\left(\delta_{y,y_{12}(\theta_{12})} + \delta_{y,y_{34}(\theta_{34})}\right)$, one firing event has to occur in each of the intervals $1 \leq y \leq \frac{M}{2}$ and $\frac{M}{2} + 1 \leq y \leq M$ for all of the information to be collected about $\mathbf{x}$. However, the random nature of the firing events means that the probability with which this condition is satisfied increases with n, so this result for $D_1 + D_2$ decreases as n increases.

4. *Relative Stability of Solutions*

Collect the above results together for comparison.

$$D_1 + D_2 = \begin{cases} 4 - \frac{2M^2}{\pi^2} \sin^2\left(\frac{\pi}{M}\right) & \text{type 1} \\ 4 - \frac{4M}{\pi^2} \sin^2\left(\frac{\pi}{\sqrt{M}}\right) & \text{type 2} \\ 4 - \frac{nM^2}{(n+1)\pi^2} \sin^2\left(\frac{2\pi}{M}\right) & \text{type 3} \end{cases} \quad (33)$$

For constant M and letting $n \longrightarrow \infty$, the value of $D_1 + D_2$ for the type 3 solution asymptotically behaves as $D_1 + D_2$ (type 3) $\longrightarrow 4 - \frac{M^2}{\pi^2} \sin^2\left(\frac{2\pi}{M}\right)$, in which case the relative stability of the three types of solution is: type 3 (most stable), type 2 (intermediate), type 1 (least stable). Similarly, for constant n and letting $M \longrightarrow \infty$, the relative stability of the three types of solution is: type 2 (most stable), type 3 (intermediate), type 1 (least stable).

In both of these limiting cases the type 1 solution is least stable. If there is a fixed number of firing events n, and there is no upper limit on the number of neurons M, then the type 2 solution is most stable, because it can partition the 2-torus into lots of small quantisation cells. If there is a fixed number of neurons M (which is the usual case), and there is no upper limit on the number of firing events n, then the type 3 solution is most stable, because the limited size of M renders the type 2 solution inefficient (the quantisation cells would be too large), so the 2-torus $S^1 \times S^1$ is split into two S^1 subspaces each of which is assigned a subset of $\frac{M}{2}$ neurons. If n is large enough, then each of these two subsets of neurons has a high probability of occurrence of a firing event, which ensures that both of the S^1 subspaces are encoded.

More generally, when there is a limited number of neurons they will tend to split into subsets, each of which encodes a separate subspace of the input. The assumed form of $\Pr(y|\mathbf{x})$ in equation 26 does not allow an unrestricted search of all possible $\Pr(y|\mathbf{x})$. If the global optimum solution (which has piecewise linear $\Pr(y|\mathbf{x})$, as proved in section III B) cuts up the input space into partially overlapping pieces, then it is well approximated by a solution such as one of those listed in equation 26. Typically, curved input spaces lead to such solutions, because a piecewise linear $\Pr(y|\mathbf{x})$ can readily quantise such spaces by slicing off the curved "corners"' that occur in such spaces.

IV. CONCLUSIONS

In this paper an objective function for optimising a layered network of discretely firing neurons has been presented, and three non-trivial examples of how it is applied have been shown: topographic mapping networks, piecewise linear dependence on the input of the probability of a neuron firing, and factorial encoder networks. Many other examples could be given, such as combining the first and third of the above results to obtain factorial topographic networks, or extending the theory to multilayer networks, or introducing temporal information.

V. ACKNOWLEDGEMENTS

I thank Chris Webber for many useful conversations that we had during the course of this research.

[1] T Kohonen, *Self-organisation and associative memory*, Springer-Verlag, Berlin, 1989.
[2] S P Luttrell, *Hierarchical self-organising networks*, Proceedings of IEE Conference on Artificial Neural Networks (London), IEE, 1989, pp. 2–6.
[3] ———, *Mathematics of neural networks: models, algorithms and applications*, ch. A theory of self-organising neural networks, pp. 240–244, Kluwer, Boston, 1997.
[4] ———, *Self-organisation of multiple winner-take-all neural networks*, Connection Science **9** (1997), no. 1, 11–30.
[5] J Rissanen, *Modelling by shortest data description*, Automatica **14** (1978), no. 5, 465–471.

Modelling the Probability Density of Markov Source [*]

Stephen Luttrell
Room EX21, QinetiQ, Malvern Technology Centre

This paper introduces an objective function that seeks to minimise the average total number of bits required to encode the joint state of all of the layers of a Markov source. This type of encoder may be applied to the problem of optimising the bottom-up (recognition model) and top-down (generative model) connections in a multilayer neural network, and it unifies several previous results on the optimisation of multilayer neural networks.

I. INTRODUCTION

There is currently a great deal of interest in modelling probability density functions (PDF). This research is motivated by the fact that the joint PDF of a set of variables can be used to deduce any conditional PDF which involves these variables alone, which thus allows all inference problems in the space of these variables to be addressed quantitatively. The only limitation of this approach to solving inference problems is that a model of the PDF is used, rather than the actual PDF itself, which can lead to inaccurate inferences. The objective function for optimising a PDF model is usually to maximise the log-likelihood that it could generate the training set: i.e. maximise $\langle \log(\text{model probability}) \rangle_{\text{training set}}$.

In this paper the problem of modelling the PDF of a Markov source will be studied. In the language of neural networks, this type of source can be viewed as a layered network, in which the state of each layer directly influences the states of only the layers immediately above and below it. The optimal PDF model then approximates the joint PDF of the states of all of the layers of the network, or at least some subset of the layers of the network, which is a generalisation of what is conventionally done in neural network PDF models.

Markov source density modelling is interesting because it unifies a number of existing neural network techniques into a single framework, and may also be viewed as a density modelling perspective on the results reported in [10]. For instance, an approximation to the standard Kohonen topographic mapping neural network [3] emerges from density modelling the joint PDF of the input and output layers of a 3-layer network, and generalisations of the Kohonen network also emerge naturally from this framework.

In section II the relevant parts of the Shannon theory of information are summarised [12, 13], and the application to coding various types of source is derived [11]; in particular, Markov sources are discussed, because they are the key to the approach that is presented in this paper. In section III the application of Markov source coding to unsupervised neural networks is discussed in detail, including the Kohonen network [3]. In section IV (and appendix A) hierarchical encoding using an adaptive cluster expansion (ACE) is discussed [5], and in section V (and appendix B) factorial encoding using a partitioned mixture distribution (PMD) are discussed [9]. Finally in appendix III density modelling of Markov sources is compared with standard density modelling using a Helmholtz machine [2].

II. CODING THEORY

In section II A the basic ideas of information theory are outlined (this discussion is inspired by the reasoning presented in [12, 13]), and in section II B the process of using a model to code a source described in detail. In section II C this is extended to the case of a Markov source. In section II D the relationship between conventional density models and Markov density models is discussed.

See [12, 13] for a lucid introduction to information theory, and see [11] for a discussion of the number of bits required to encode a source using a model.

A. Information Theory

A source of symbols (drawn from an alphabet of M distinct symbols) is modelled as a vector of probabilities denoted as $\mathbf{P}$

$$\mathbf{P} \equiv (P_1, P_2, \cdots, P_M) \tag{2.1}$$

which describes the relative frequency with which each symbol is drawn independently from the source $\mathbf{P}$. A trivial example is an unbiassed die, which has $M = 6$ and $P_i = \frac{1}{6}$ for $i = 1, 2, \cdots, 6$.

The ordered sequence of symbols drawn independently from a source may be partitioned into subsequences of N symbols, and each such subsequence will be called a message. If N is very large, then a message is called "likely" if the relative frequency of occurrence of its symbols approximates $\mathbf{P}$, otherwise it is called "unlikely". As $N \to \infty$ the set of likely messages is very sharply defined, in the sense that the proportion of all messages that lie

[*]Typeset in LaTeX on May 3, 2019.
This is archived at http://arxiv.org/abs/cs/0607019, 6 Jul 2006.
This paper was submitted to Neural Computation on 2 February 1998. Manuscript no. 1729. It was not accepted for publication, but it underpins several subsequently published papers. © Crown Copyright 1998

in the transition region between being likely and being unlikely becomes vanishingly small. Thus there is a set of likely messages all with equal probability of occurring (because each likely message has the same relative frequency of occurrence of each of the M possible symbols), and a set of unlikely messages (i.e. all the messages that are not likely messages) that have essentially zero probability of occurring. It is this separation of messages into a likely set (all with equal probability) and an unlikely set (all with zero probability) that underlies information theory, as discussed in [12, 13].

A likely message from $\mathbf{P}$ will be called a likely $\mathbf{P}$-message. As $N \longrightarrow \infty$ the number of times n_i that each symbol i occurs in a $\mathbf{P}$-message of length N is $n_i = NP_i$, where $\sum_{i=1}^{M} P_i = 1$ guarantees that the normalisation condition $\sum_{i=1}^{M} n_i = N$ is satisfied. The logarithm of the number of different likely $\mathbf{P}$-messages is given by (using Stirling's approximation $\log x! \approx x \log x - x$ when x is large)

$$\log\left(\frac{N!}{n_1!\,n_2!\,\cdots\,n_M!}\right) \approx -N\sum_{i=1}^{M} P_i \log P_i \tag{2.2}$$

Now define the entropy $H(\mathbf{P})$ of source $\mathbf{P}$ as the logarithm of the number of different likely $\mathbf{P}$-messages (measured per message symbol):

$$H(\mathbf{P}) \equiv -\sum_{i=1}^{M} P_i \log P_i \geq 0 \tag{2.3}$$

Thus $H(\mathbf{P})$ is the number of bits per symbol (on average) required to encode the source (assuming a perfect encoder), because the only messages that the source has a finite probability of producing are the likely $\mathbf{P}$-messages that are enumerated in equation 2.2.

It is usually very difficult to encode the source $\mathbf{P}$ using $H(\mathbf{P})$ bits per symbol on average. This is because although the boundary between the set of likely $\mathbf{P}$-messages and the set of unlikely $\mathbf{P}$-messages is sharply defined in principle, in practice it is very hard to model mathematically. If this boundary is not precisely defined, then it is impossible to compute the value of $H(\mathbf{P})$ accurately. In order to ensure that all of the likely $\mathbf{P}$-messages are accounted for, it is necessary for the mathematical model of the boundary to lie outside the true boundary, which thus overestimates the value of $H(\mathbf{P})$. This demonstrates that $H(\mathbf{P})$ is in fact a lower bound on the true number of bits per symbol that must be used to encode the source $\mathbf{P}$.

B. Source Coding

The mathematical model (or, simply, the model) of the boundary between the set of likely $\mathbf{P}$-messages and the set of unlikely $\mathbf{P}$-messages may be derived from a another vector of probabilities, denoted as $\mathbf{Q}$, whose M elements model the probability of each symbol drawn from an alphabet of M distinct symbols. If $\mathbf{Q} = \mathbf{P}$ then the boundary is modelled perfectly, and hence in principle the lower bound $H(\mathbf{P})$ on the number of bits per symbol may be attained, although even this is usually difficult to realise constructively in practice. In practical situations $\mathbf{Q} \neq \mathbf{P}$ is invariably the case, so the problem of coding a source with an inaccurate model cannot be avoided.

Since the only $\mathbf{P}$-messages that can occur are the likely $\mathbf{P}$-messages (which all occur with equal probability), the number of bits required when using $\mathbf{Q}$ to encode $\mathbf{P}$ is (minus) the logarithm of the probability $\Pi_N(\mathbf{P},\mathbf{Q})$ that a $\mathbf{Q}$-message is one of the likely $\mathbf{P}$-messages. $\Pi_N(\mathbf{P},\mathbf{Q})$ is given by

$$\begin{aligned}\Pi_N(\mathbf{P},\mathbf{Q}) &= \log\left(\frac{N!}{n_1!\,n_2!\,\cdots\,n_M!}\,Q_1^{n_1}\,Q_2^{n_2}\,\cdots\,Q_M^{n_M}\right) \\ &\approx -N\sum_{i=1}^{M} P_i \log\left(\frac{P_i}{Q_i}\right) \leq 0 \end{aligned}\tag{2.4}$$

which is negative because the model $\mathbf{Q}$ generates likely $\mathbf{P}$-messages with less than unit probability. The model $\mathbf{Q}$ must be used to generate enough $\mathbf{Q}$-messages to ensure that all of the likely $\mathbf{P}$-messages are reproduced, which requires the basic $H(\mathbf{P})$ bits per symbol (that would be required if $\mathbf{Q} = \mathbf{P}$), plus some extra bits to compensate for the less than 100% efficiency with which $\mathbf{Q}$ generates likely $\mathbf{P}$-messages (because $\mathbf{Q} \neq \mathbf{P}$). The number of extra bits per symbol is the relative entropy $G(\mathbf{P},\mathbf{Q})$

$$G(\mathbf{P},\mathbf{Q}) \equiv \sum_{i=1}^{M} P_i \log\left(\frac{P_i}{Q_i}\right) \geq 0 \tag{2.5}$$

which is $-\frac{\Pi_N(\mathbf{P},\mathbf{Q})}{N}$, or minus the logarithm of the probability per symbol that a $\mathbf{Q}$-message is a likely $\mathbf{P}$-message. Thus $\mathbf{Q}$ is used to generate exactly the number of extra $\mathbf{Q}$-messages required to compensate for the fact that the probability that each $\mathbf{Q}$-message is a likely $\mathbf{P}$-message is less than unity (i.e. $\Pi_N(\mathbf{P},\mathbf{Q}) \leq 0$). $G(\mathbf{P},\mathbf{Q})$ (i.e. relative entropy) is the amount by which the number of bits per symbol exceeds the lower bound $H(\mathbf{P})$ (i.e. source entropy). For completeness, also define the total number of bits per symbol $H(\mathbf{P}) + G(\mathbf{P},\mathbf{Q})$ as $L(\mathbf{P},\mathbf{Q})$, which is given by

$$\begin{aligned}L(\mathbf{P},\mathbf{Q}) &\equiv H(\mathbf{P}) + G(\mathbf{P},\mathbf{Q}) \\ &= -\sum_{i=1}^{M} P_i \log Q_i \geq 0\end{aligned}\tag{2.6}$$

The expression for $G(\mathbf{P},\mathbf{Q})$ provides a means of optimising the model $\mathbf{Q}$. If the optimisation criterion is that the average number of bits per symbol required when using $\mathbf{Q}$ to encode $\mathbf{P}$ should be minimised, then the optimum model $\mathbf{Q}_{opt}$ should minimise the objective function $G(\mathbf{P},\mathbf{Q})$ with respect to $\mathbf{Q}$, thus

$$\mathbf{Q}_{opt} = \underset{\mathbf{Q}}{\arg\min}\; G(\mathbf{P},\mathbf{Q}) \tag{2.7}$$

This criterion for optimising a model does not include the number of bits required to specify the model itself, such as is used in the minimum description length approach [11], although the objective function could be extended to include such additional contributions.

$G(\mathbf{P},\mathbf{Q})$ is frequently used as an objective function in density modelling, where the source $\mathbf{P}$ is the vector of observed symbol frequencies. Since $\mathbf{Q}_{opt}$ must, in some sense, be close to $\mathbf{P}$, this affords a practical way of ensuring that the optimum model probabilities $\mathbf{Q}_{opt}$ are similar to the source symbol frequencies $\mathbf{P}$, which is the goal of density modelling.

C. Markov Source Coding

The above scheme for using a model $\mathbf{Q}$ to encode symbols derived from a source $\mathbf{P}$ may be extended to the case where the source and the model are L-layer first order Markov chains. The word "layer" is used in anticipation of the connection with multilayer neural networks that will be discussed in section III. Thus split up both of $\mathbf{P}$ and $\mathbf{Q}$ into their constituent transition probabilities

$$\begin{aligned}\mathbf{P} &= \left(\mathbf{P}^0, \mathbf{P}^{1|0}, \cdots, \mathbf{P}^{L-1|L-2}, \mathbf{P}^{L|L-1}\right)\\ &= \left(\mathbf{P}^{0|1}, \mathbf{P}^{1|2}, \cdots, \mathbf{P}^{L-1|L}, \mathbf{P}^{L}\right)\\ \mathbf{Q} &= \left(\mathbf{Q}^0, \mathbf{Q}^{1|0}, \cdots, \mathbf{Q}^{L-1|L-2}, \mathbf{Q}^{L|L-1}\right)\\ &= \left(\mathbf{Q}^{0|1}, \mathbf{Q}^{1|2}, \cdots, \mathbf{Q}^{L-1|L}, \mathbf{Q}^{L}\right)\end{aligned} \tag{2.8}$$

These two ways of decomposing $\mathbf{P}$ (and $\mathbf{Q}$) are equivalent, because a forward pass through a Markov chain may be converted into a backward pass through a different Markov chain, whose transition probabilities are uniquely determined by applying Bayes' theorem to the original Markov chain. $\mathbf{P}^{k|l}$ (and $\mathbf{Q}^{k|l}$) is the matrix of transition probabilities from layer l to layer k of the Markov chain of the source (and model), $\mathbf{P}^0$ ($\mathbf{Q}^0$) is the vector of marginal probabilities in layer 0, $\mathbf{P}^L$ (and $\mathbf{Q}^L$) is the vector of marginal probabilities in layer L. This may be written out in detail as

$$\begin{aligned}P^0_{i_0} &= \text{true probability that layer 0 has state } i_0\\ P^L_{i_L} &= \text{true probability that layer } L \text{ has state } i_L\\ P^{l+1|l}_{i_{l+1},i_l} &= \text{true probability that layer } l+1 \text{ has state } i_{l+1}\\ &\quad\ \text{given that layer } l \text{ has state } i_l\\ P^{l|l+1}_{i_l,i_{l+1}} &= \text{true probability that layer } l \text{ has state } i_l\\ &\quad\ \text{given that layer } l+1 \text{ has state } i_{l+1}\end{aligned} \tag{2.9}$$

$$\begin{aligned}Q^0_{i_0} &= \text{model probability that layer 0 has state } i_0\\ Q^L_{i_L} &= \text{model probability that layer } L \text{ has state } i_L\\ Q^{l+1|l}_{i_{l+1},i_l} &= \text{model probability that layer } l+1 \text{ has state } i_{l+1}\\ &\quad\ \text{given that layer } l \text{ has state } i_l\\ Q^{l|l+1}_{i_l,i_{l+1}} &= \text{model probability that layer } l \text{ has state } i_l\\ &\quad\ \text{given that layer } l+1 \text{ has state } i_{l+1}\end{aligned} \tag{2.10}$$

The number of extra bits per symbol $G(\mathbf{P},\mathbf{Q})$ (see equation 2.5) required to encode each symbol from the source $\mathbf{P}$ using the model $\mathbf{Q}$ may then be written as

$$\begin{aligned}G(\mathbf{P},\mathbf{Q}) &= \sum_{i_0=1}^{M_0}\cdots\sum_{i_L=1}^{M_L} P^{0|1}_{i_0,i_1} P^{1|2}_{i_1,i_2}\cdots P^{L-1|L}_{i_{L-1},i_L} P^L_{i_L} \log\left(\frac{P^{0|1}_{i_0,i_1} P^{1|2}_{i_1,i_2}\cdots P^{L-1|L}_{i_{L-1},i_L} P^L_{i_L}}{Q^{0|1}_{i_0,i_1} Q^{1|2}_{i_1,i_2}\cdots Q^{L-1|L}_{i_{L-1},i_L} Q^L_{i_L}}\right)\\ &= \sum_{l=0}^{L-1}\sum_{i_{l+1}=1}^{M_{l+1}} P^{l+1}_{i_{l+1}} G_{i_{l+1}}\left(\mathbf{P}^{l|l+1},\mathbf{Q}^{l|l+1}\right) + G\left(\mathbf{P}^L,\mathbf{Q}^L\right)\end{aligned} \tag{2.11}$$

where the flow of influence in both $\mathbf{P}$ and $\mathbf{Q}$ is from layer 0 to layer L. The suffix i_{l+1} that appears on the $G_{i_{l+1}}\left(\mathbf{P}^{l|l+1},\mathbf{Q}^{l|l+1}\right)$ indicates that the state of layer $l+1$ is fixed during the evaluation of $G_{i_{l+1}}\left(\mathbf{P}^{l|l+1},\mathbf{Q}^{l|l+1}\right)$ (i.e. it is the relative entropy of layer l, given that the state of layer $l+1$ is known). Similarly, the total number of bits per symbol required to encode each symbol from the source $\mathbf{P}$ using the model $\mathbf{Q}$ is $L(\mathbf{P},\mathbf{Q})$ (i.e. $H(\mathbf{P})+G(\mathbf{P},\mathbf{Q})$), which is given by

$$L(\mathbf{P},\mathbf{Q}) = \sum_{l=0}^{L-1}\sum_{i_{l+1}=1}^{M_{l+1}} P^{l+1}_{i_{l+1}} L_{i_{l+1}}\left(\mathbf{P}^{l|l+1},\mathbf{Q}^{l|l+1}\right) + L\left(\mathbf{P}^L,\mathbf{Q}^L\right) \tag{2.12}$$

This result has a very natural interpretation. Both the source $\mathbf{P}$ and the model $\mathbf{Q}$ are Markov chains, and corresponding parts of the model are matched up with corresponding parts of the source. First of all, the number of bits required to encode the L^{th} layer of the source is $L\left(\mathbf{P}^L, \mathbf{Q}^L\right)$. Having done that, the number of bits required to encode the $L-1^{th}$ layer of the source, given that the state of the L^{th} layer is already known, is $L\left(\mathbf{P}^{L-1|L}, \mathbf{Q}^{L-1|L}\right)$, which must then be averaged over the alternative possible states of the L^{th} layer to yield $\sum_{i_L=1}^{M_L} P_{i_L}^L \, L\left(\mathbf{P}^{L-1|L}, \mathbf{Q}^{L-1|L}\right)$. This process is then repeated to encode the $L-2^{th}$ layer of the source, given that the state of the $L-1^{th}$ layer is already known, and so on back to layer 0. This yields precisely the expression for $L\left(\mathbf{P}, \mathbf{Q}\right)$ given above.

Bayes' theorem (in the form $P_{i_{l+1}}^{l+1} \, P_{i_l, i_{l+1}}^{l|l+1} = P_{i_l}^l \, P_{i_{l+1}, i_l}^{l+1|l}$) may be used to rewrite the expression for $L\left(\mathbf{P}, \mathbf{Q}\right)$ so that the flow of influence in $\mathbf{P}$ and $\mathbf{Q}$ runs in opposite directions. Thus

$$L\left(\mathbf{P}, \mathbf{Q}\right) = \sum_{l=0}^{L-1} \sum_{i_l=1}^{M_l} P_{i_l}^l \, K_{i_l}\left(\mathbf{P}^{l+1|l}, \mathbf{Q}^{l|l+1}\right) + L\left(\mathbf{P}^L, \mathbf{Q}^L\right) \tag{2.13}$$

where $K_{i_l}\left(\mathbf{P}^{l+1|l}, \mathbf{Q}^{l|l+1}\right)$ is defined as

$$K_{i_l}\left(\mathbf{P}^{l+1|l}, \mathbf{Q}^{l|l+1}\right) \equiv -\sum_{i_{l+1}=1}^{M_{l+1}} P_{i_{l+1}, i_l}^{l+1|l} \, \log Q_{i_l, i_{l+1}}^{l|l+1} \tag{2.14}$$

The expression for $L\left(\mathbf{P}, \mathbf{Q}\right)$ in equation 2.13 has an analogous interpretation to that in equation 2.12.

Other types of Markov chain may also be considered, such as ones in which some of the layers are not included in the calculation of the number of bits required to encode the source. One such example is discussed in section III D.

D. Alternative Viewpoints

The relationship between conventional density models and Markov density models can be stated from the point of view of a conventional density modeller. The goal is to build a density model $\mathbf{Q}^0$ of the source $\mathbf{P}^0$, such that the number of bits per symbol $L\left(\mathbf{P}^0, \mathbf{Q}^0\right)$ required to encode $\mathbf{P}^0$ is minimised. However, if the source $\mathbf{P}^0$ is transformed through L layers of a network to produce a transformed source $\mathbf{P}^L$, then $L\left(\mathbf{P}^0, \mathbf{Q}^0\right) \leq L\left(\mathbf{P}, \mathbf{Q}\right)$ where $L\left(\mathbf{P}, \mathbf{Q}\right)$ is given in equation 2.12, which is the sum of the number of bits per symbol $L\left(\mathbf{P}^L, \mathbf{Q}^L\right)$ required to encode $\mathbf{P}^L$, plus (for $l = 0, 1, \cdots, L-1$) the number of bits per symbol $\sum_{i_{l+1}=1}^{M_{l+1}} P_{i_{l+1}}^{l+1} \, L_{i_{l+1}}\left(\mathbf{P}^{l|l+1}, \mathbf{Q}^{l|l+1}\right)$ required to encode $\mathbf{P}^{l|l+1}$.

Thus the problem of encoding the source $\mathbf{P}^0$ can be split into three steps: transform the source from $\mathbf{P}^0$ to $\mathbf{P}^L$, encode the transformed source $\mathbf{P}^L$, and encode all of the transformations $\mathbf{P}^{l|l+1}$ (for $l = 0, 1, \cdots, L-1$) to allow the original source to be reconstructed from the transformed source. The total number of bits $L\left(\mathbf{P}, \mathbf{Q}\right)$ required to encode $\mathbf{P}^L$ and $\mathbf{P}^{l|l+1}$ (for $l = 0, 1, \cdots, L-1$) is then an upper bound on the total number of bits $L\left(\mathbf{P}^0, \mathbf{Q}^0\right)$ required to encode $\mathbf{P}^0$. In this picture, a Markov chain is used to connect the original source $\mathbf{P}^0$ to the transformed source $\mathbf{P}^L$, so the Markov chain relates one conventional density modelling problem (i.e. optimising $\mathbf{Q}^0$) to another (i.e. optimising $\mathbf{Q}^L$).

The above description of the relationship between conventional density models and Markov density models was presented from the point of view of a conventional density modeller, who asserts that the goal is to build an optimum (i.e. minimum number of bits per symbol) density model $\mathbf{Q}^0$ of the source $\mathbf{P}^0$. From this point of view, the Markov chain is merely a means of transforming the problem from modelling the source $\mathbf{P}^0$ to modelling the transformed source $\mathbf{P}^L$. That this transformation process is imperfect is reflected in the fact that more bits per symbol are required to encode the transformed source $\mathbf{P}^L$ (plus the state of the Markov chain that generates it) than the original source $\mathbf{P}^0$. A conventional density modeller might reasonably ask what is the point of using Markov density models, if they give only an upper bound on the number of bits per symbol for encoding the original source $\mathbf{P}^0$?

However, it is not at all clear that the conventional density modeller is using the correct objective function in the first place. Why should the number of bits per symbol for encoding the original source $\mathbf{P}^0$ be especially important? It is as if the world has been separated into an external world (i.e. $\mathbf{P}^0$) and an internal world (i.e. the $\mathbf{P}^{l+1|l}$ for $l = 0, 1, \cdots, L-1$), and a special status is accorded to the external world, which deems that it is important to model its density $\mathbf{P}^0$ accurately, at the expense of modelling the $\mathbf{P}^{l+1|l}$ accurately. In the Markov density modelling approach, this artificial boundary between external and internal worlds is removed, because the Markov chain models the joint density $\left(\mathbf{P}^0, \mathbf{P}^{1|0}, \cdots, \mathbf{P}^{L|L-1}\right)$, where $\mathbf{P}^0$ and the $\mathbf{P}^{l+1|l}$ are all accorded equal status. This even-handed approach is much more natural than one in which a particular part of the source (i.e. the external source) is accorded a special status.

In the language of multilayer neural networks, the vector $\left(\mathbf{P}^0, \mathbf{P}^{0|1}, \cdots, \mathbf{P}^{L|L-1}\right)$ is the source which comprises the bottom-up transformations (or recognition models) which generate the states of the internal layers of the network, and the vector $\left(\mathbf{Q}^{0|1}, \mathbf{Q}^{1|2}, \cdots, \mathbf{Q}^L\right)$ is the model of the source which comprises the top-down transformations (or generative models). Thus the network is self-referential, because it forms a model of a source that includes its own internal states. This self-referential behaviour is present in both the conventional density modelling and in the Markov density modelling approaches, but whereas in the former case it is not optimised in an even-handed fashion, in the latter case it is optimised in an even-handed fashion.

III. APPLICATION TO UNSUPERVISED NEURAL NETWORKS

In section III A the theory of Markov source coding (that was presented in section II C) is applied to a multilayer neural network. In section III B this approach is applied to a 2-layer neural network to obtain a soft vector quantiser (VQ), which is generalised to a multilayer neural network in section III C to obtain a network of coupled soft VQs. In section III D it is shown how an approximation to Kohonen's topographic mapping network can be derived from the theory of Markov source coding. Finally, some additional results are briefly dicussed in section III E.

A. Source Model of a Layered Network

In this section the optimisation of the joint PDF of the states of all of the layers of an $(L+1)$-layer encoder of the type that was discussed in section II C will be considered. It turns out that this leads to new insights into the optimisation of a multilayer unsupervised neural networks.

The Markov chain source $\mathbf{P} = \left(\mathbf{P}^0, \mathbf{P}^{1|0}, \cdots, \mathbf{P}^{L-1|L-2}, \mathbf{P}^{L|L-1}\right)$ (or, equivalently, $\mathbf{P} = \left(\mathbf{P}^{0|1}, \mathbf{P}^{1|2}, \cdots, \mathbf{P}^{L-1|L}, \mathbf{P}^{L}\right)$) may be used to describe the true behaviour (i.e. not merely a model) of a layered neural network as follows. $\mathbf{P}^0$ is an external source, and $\left(\mathbf{P}^{1|0}, \cdots, \mathbf{P}^{L-1|L-2}, \mathbf{P}^{L|L-1}\right)$ is an internal source, where external/internal describes whether the source is outside/inside the layered network, respectively. $\mathbf{P}^{l+1|l}$ is not part of the source itself (i.e. the external source), rather it is a transition matrix that describes the way in which the state of layer l of the neural network influences the state of layer $l+1$. There is an analogous interpretation of $\mathbf{P}^L$ and the $\mathbf{P}^{l|l+1}$.

The Markov chain model $\mathbf{Q} = \left(\mathbf{Q}^0, \mathbf{Q}^{1|0}, \cdots, \mathbf{Q}^{L-1|L-2}, \mathbf{Q}^{L|L-1}\right)$ (or, equivalently, $\mathbf{Q} = \left(\mathbf{Q}^{0|1}, \mathbf{Q}^{1|2}, \cdots, \mathbf{Q}^{L-1|L}, \mathbf{Q}^{L}\right)$) may then be used as a model (i.e. not actually the true behaviour) of a layered neural network. $\mathbf{Q}$ has an analogous interpretation to $\mathbf{P}$, except that it is a model of the source, rather than the true behaviour of the source.

It turns out to be useful for the true Markov behaviour (i.e. $\mathbf{P}$) and the model Markov behaviour (i.e. $\mathbf{Q}$) to run in opposite directions through the Markov chain. Thus $\mathbf{P} = \left(\mathbf{P}^0, \mathbf{P}^{1|0}, \cdots, \mathbf{P}^{L-1|L-2}, \mathbf{P}^{L|L-1}\right)$ (flow of influence from layer 0 to layer L of the Markov chain) and $\mathbf{Q} = \left(\mathbf{Q}^{0|1}, \mathbf{Q}^{1|2}, \cdots, \mathbf{Q}^{L-1|L}, \mathbf{Q}^{L}\right)$ (flow of influence from layer L to layer 0 of the Markov chain). In the conventional language of neural networks, $\mathbf{P}$ is a "recognition model" and $\mathbf{Q}$ is a "generative model". Note that the use of the word "model" in the terminology "recognition model" is strictly speaking not accurate in this context, because $\mathbf{P}$ is a source, not a model. However, terminology depends on one's viewpoint. In Markov chain density modelling $\mathbf{P}$ is a source when viewed from the point of view of the model $\mathbf{Q}$. Whereas, in conventional density modelling $\mathbf{P}^0$ is a source when viewed from the point of model $\mathbf{Q}^0$, in which case $\left(\mathbf{P}^{1|0}, \cdots, \mathbf{P}^{L-1|L-2}, \mathbf{P}^{L|L-1}\right)$ is a recognition model and $\left(\mathbf{Q}^{0|1}, \mathbf{Q}^{1|2}, \cdots, \mathbf{Q}^{L-1|L}, \mathbf{Q}^{L}\right)$ is a generative model.

B. 2-Layer Soft Vector Quantiser (VQ) Network

The expression for $L(\mathbf{P}, \mathbf{Q})$ in equation 2.13 has a simple internal structure which allows it to be systematically analysed. Thus apply equation 2.13 to a 2-layer network where

$$\begin{aligned} \mathbf{P} &= \left(\mathbf{P}^0, \mathbf{P}^{1|0}\right) \\ \mathbf{Q} &= \left(\mathbf{Q}^{0|1}, \mathbf{Q}^{1}\right) \end{aligned} \tag{3.1}$$

to obtain the objective function

$$L(\mathbf{P}, \mathbf{Q}) = -\sum_{i_0=1}^{M_0} P^0_{i_0} \sum_{i_1=1}^{M_1} P^{1|0}_{i_1,i_0} \log Q^{0|1}_{i_0,i_1} + L\left(\mathbf{P}^1, \mathbf{Q}^1\right) \tag{3.2}$$

Now change notation in order to make contact with previous results on vector quantisers (VQ)

$$\begin{array}{lll} i_0 \to \mathbf{x} \quad \sum_{i_0=1}^{M_0} \to \int d\mathbf{x} & \text{input vector} & \\ i_1 \to y \quad \sum_{i_1=1}^{M_1} \to \sum_{y=1}^{M} & \text{output code index} & \\ P^0_{i_0} \to \Pr(\mathbf{x}) & \text{input PDF} & \\ P^{1|0}_{i_1,i_0} \to \Pr(y|\mathbf{x}) & \text{recognition model} & (3.3) \\ Q^{0|1}_{i_0,i_1} \to V \frac{1}{\left(\sqrt{2\pi}\,\sigma\right)^{\dim \mathbf{x}}} \exp\left(-\frac{\left\|\mathbf{x}-\mathbf{x}'(y)\right\|^2}{2\sigma^2}\right) & \text{Gaussian generative model} & \\ Q^1_{i_1} \to Q(y) & \text{output prior} & \end{array}$$

where $\mathbf{x}$ is a continuous-valued input vector (e.g. the activity pattern in layer 0), σ is the (isotropic) variance of the Gaussian generative model, V is an infinitesimal volume element in input space which may be used to convert the Gaussian probability density into a probability, and y is a discrete-valued output index (e.g. the location of the next neuron to fire in layer 1). Note that the parameter V must be introduced in order to regularise the number of bits required to specify each source state. In effect, V specifies a resolution scale, such that details on smaller scales are ignored.

The notation defined in equation 3.3 allows $L(\mathbf{P},\mathbf{Q})$ to be written as

$$L(\mathbf{P},\mathbf{Q}) = \frac{D_{VQ}}{4\sigma^2} + L\left(\mathbf{P}^1,\mathbf{Q}^1\right) - \log\left(\frac{V}{\left(\sqrt{2\pi}\,\sigma\right)^{\dim \mathbf{x}}}\right) \tag{3.4}$$

where D_{VQ} is defined as

$$D_{VQ} \equiv 2\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{y=1}^{M}\Pr(y|\mathbf{x})\,\left\|\mathbf{x}-\mathbf{x}'(y)\right\|^2 \tag{3.5}$$

The first term in equation 3.4 is proportional to the objective function D_{VQ} for a soft vector quantiser (VQ), where $\Pr(y|\mathbf{x})$ is a soft encoder, and $\mathbf{x}'(y)$ is the corresponding reconstruction vector attached to code index y, and $\left\|\mathbf{x}-\mathbf{x}'(y)\right\|^2$ is the L^2 norm of the reconstruction error. A standard VQ [4] (i.e. winner-take-all encoder) has $\Pr(y|\mathbf{x}) = \delta_{y,y(\mathbf{x})}$, which emerges as the optimal form when this VQ objective function is minimised w.r.t. $\Pr(y|\mathbf{x})$ (see [10] for a detailed discussion of these issues). The second term in equation 3.4 (i.e. $L\left(\mathbf{P}^1,\mathbf{Q}^1\right)$) is the cost of coding the output layer, and the third term is constant.

The effect of the $L\left(\mathbf{P}^1,\mathbf{Q}^1\right)$ term in $L(\mathbf{P},\mathbf{Q})$ (see equation 3.4) is to encourage $P_i^1 \to \delta_{i,i_0}$ (only one state in layer 1 is used) and $\mathbf{Q}^1 \to \mathbf{P}^1$ (perfect model in layer 1). The behaviour $P_i^1 \to \delta_{i,i_0}$ is in conflict with the requirements of the first term (i.e. the soft VQ) in $L(\mathbf{P},\mathbf{Q})$, which requires that more than one state in layer 1 is used, in order to minimise the reconstruction distortion. There is a tradeoff between increasing the number of active states in layer 1 in order to enable the Gaussian generative model ($\mathbf{Q}^0$ is a Gaussian mixture distribution) to make a good approximation to the external source $\mathbf{P}^0$, and decreasing the number of active states in layer 1 in order to make the average total number of bits $L\left(\mathbf{P}^1,\mathbf{Q}^1\right)$ required to specify an output state as small as possible.

C. Coupled Soft VQ Networks

The results of section III B will now be generalised to an $(L+1)$-layer network. The objective function for coding a Markov source (equation 2.13) can be written, using a notation which is analogous to that given in equation 3.3 as

$$L(\mathbf{P},\mathbf{Q}) = \sum_{l=0}^{L-1}\frac{D_{VQ}^l}{4(\sigma_l)^2} + L\left(\mathbf{P}^L,\mathbf{Q}^L\right) - \sum_{l=0}^{L-1}\log\left(\frac{V_l}{\left(\sqrt{2\pi}\,\sigma_l\right)^{\dim \mathbf{x}_l}}\right) \tag{3.6}$$

where D_{VQ}^l is defined as ($D_{VQ}^0 = D_{VQ}$ as defined in equation 3.5)

$$D_{VQ}^l \equiv 2\int d\mathbf{x}_l\,\Pr(\mathbf{x}_l)\sum_{y_{l+1}=1}^{M_{l+1}}\Pr(y_{l+1}|\mathbf{x}_l)\,\left\|\mathbf{x}_l-\mathbf{x}_l'(y_l)\right\|^2 \tag{3.7}$$

where $\mathbf{x}_l$ and y_l are both used to denote the state of layer l. The notation $\mathbf{x}_l$ is used to denote the input to the encoder that connects layers l and $l+1$, whereas the notation y_l denotes the output of the encoder that connects layers $l-1$ and l. This redundancy of notation is not actually necessary, but is used here to preserve the distinction between input vectors and output codes.

The first term in equation 3.6 is a weighted sum (where each term is weighted by $(\sigma_l)^{-2}$) of objective functions for a set of soft VQs connecting each of the L neighbouring pairs of layers in the network. This type of network structure will be called a VQ-ladder. The second term in equation 3.6 (i.e. $L\left(\mathbf{P}^L,\mathbf{Q}^L\right)$) is the cost of coding the output layer, and the third term is constant.

If the cost $L\left(\mathbf{P}^L,\mathbf{Q}^L\right)$ of coding the output layer is ignored, then the multilayer Markov source coding objective function $L(\mathbf{P},\mathbf{Q})$ is minimised by minimising the the objective function $\sum_{l=0}^{L-1}\frac{D_{VQ}^l}{(\sigma_l)^2}$ for a VQ-ladder (see [10] discussion of this point in the context of folded Markov chains (FMC)). As the number L of network layers is increased, the effect of the $L\left(\mathbf{P}^L,\mathbf{Q}^L\right)$ term has less and less effect on the overall optimisation, because its effect is swamped by the VQ-ladder term.

D. Topographic Mapping Network

The results obtained in section III B for a soft VQ may be generalised to obtain a topographic mapping network whose properties closely resemble those of a Kohonen network [3]. This derivation is based on the approach to topographic mappings that was presented in [6]. Thus apply equation 2.13 to a 3-layer network

$$\begin{aligned}\mathbf{P} &= \left(\mathbf{P}^{0}, \mathbf{P}^{1|0}, \mathbf{P}^{2|1}\right)\\ \mathbf{Q} &= \left(\mathbf{Q}^{0|1}, \mathbf{Q}^{1|2}, \mathbf{Q}^{2}\right)\end{aligned} \tag{3.8}$$

$$\begin{aligned}\mathbf{P} &= \left(\mathbf{P}^{0}, \mathbf{P}^{1|0}, \mathbf{P}^{2|1}\right)\\ \mathbf{Q} &= \left(\mathbf{Q}^{0|1}, \mathbf{Q}^{1|2}, \mathbf{Q}^{2}\right)\end{aligned} \tag{3.9}$$

where only layers 0 and 2 are included in the objective function, to obtain

$$L(\mathbf{P},\mathbf{Q}) = -\sum_{i_0=1}^{M_0} P^{0}_{i_0} \sum_{i_2=1}^{M_2} P^{2|0}_{i_2,i_0} \log Q^{0|2}_{i_0,i_2} + L\left(\mathbf{P}^2,\mathbf{Q}^2\right) \tag{3.10}$$

which should be compared with equation 3.2. An analogous change of notation to that defined in equation 3.3 can be made

$$\begin{array}{ll}
i_0 \to \mathbf{x} \quad \sum_{i_0=1}^{M_0} \to \int d\mathbf{x} & \text{input vector}\\
i_1 \to y & \text{hidden code index}\\
i_2 \to z & \text{output code index}\\
P^{0}_{i_0} \to \Pr(\mathbf{x}) & \text{input PDF}\\
P^{1|0}_{i_1,i_0} \to \Pr(y|\mathbf{x}) & \text{recognition model (first stage)}\\
P^{2|1}_{i_2,i_1} \to \Pr(z|y) & \text{recognition model (second stage)}\\
Q^{0|2}_{i_0,i_2} \to V \frac{1}{\left(\sqrt{2\pi}\,\sigma\right)^{\dim \mathbf{x}}} \exp\left(-\frac{\left\|\mathbf{x}-\mathbf{x}'(z)\right\|^2}{2\sigma^2}\right) & \text{Gaussian generative model}\\
Q^{2}_{i_2} \to Q(z) & \text{output prior}
\end{array} \tag{3.11}$$

to obtain

$$L(\mathbf{P},\mathbf{Q}) = \frac{D_{VQ}}{4\sigma^2} + L\left(\mathbf{P}^2,\mathbf{Q}^2\right) - \log\left(\frac{V}{\left(\sqrt{2\pi}\,\sigma\right)^{\dim \mathbf{x}}}\right) \tag{3.12}$$

where D_{VQ} is defined as

$$D_{VQ} \equiv 2\int d\mathbf{x}\,\Pr(\mathbf{x}) \sum_{z=1}^{M_2} \Pr(z|\mathbf{x})\, \left\|\mathbf{x}-\mathbf{x}'(z)\right\|^2 \tag{3.13}$$

which should be compared with the objective function in equation 3.5.

This expression for D_{VQ} explicitly involves the states of layers 0 and 2 of a 3-layer network, and it will now be manipulated into a form that explicitly involves the states of layers 0 and 1. In order to simplify this calculation, D_{VQ} will be replaced by the equivalent objective function [10]

$$D_{VQ} \equiv \int d\mathbf{x}\,\Pr(\mathbf{x}) \sum_{z=1}^{M_2} \Pr(z|\mathbf{x}) \int d\mathbf{x}'\,\Pr(\mathbf{x}'|z)\, \left\|\mathbf{x}-\mathbf{x}'\right\|^2 \tag{3.14}$$

Now introduce dummy integrations over the state of layer 1 to obtain

$$D_{VQ} \equiv \int d\mathbf{x}\,\Pr(\mathbf{x}) \sum_{y=1}^{M_1} \Pr(y|\mathbf{x}) \sum_{z=1}^{M_2} \Pr(z|y) \sum_{y'=1}^{M_1} \Pr(y'|z) \int d\mathbf{x}'\,\Pr(\mathbf{x}'|y')\, \left\|\mathbf{x}-\mathbf{x}'\right\|^2 \tag{3.15}$$

and rearrange to obtain

$$D_{VQ} \equiv \int d\mathbf{x}\,\Pr(\mathbf{x}) \sum_{y'=1}^{M_1} \Pr(y'|\mathbf{x}) \int d\mathbf{x}'\,\Pr(\mathbf{x}'|y')\, \left\|\mathbf{x}-\mathbf{x}'\right\|^2 \tag{3.16}$$

where

$$\begin{aligned}\Pr(y'|y) &= \sum_{z=1}^{M_2} \Pr(y'|z)\,\Pr(z|y)\\ \Pr(y'|\mathbf{x}) &= \sum_{y=1}^{M_1} \Pr(y'|y)\,\Pr(y|\mathbf{x}) \qquad (3.17)\end{aligned}$$

which may be replaced by the equivalent objective function

$$D_{VQ} \equiv 2\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{y'=1}^{M_1}\Pr(y'|\mathbf{x})\,\|\mathbf{x}-\mathbf{x}'(y')\|^2 \qquad (3.18)$$

which should be compared with the objective function in equation 3.13.

The overall effect of manipulating equation 3.13 into the form given in equation 3.18 is to convert the objective function from one that explicitly involves the states of layers 0 and 2, to one that explicitly involves the states of layers 0 and 1. This change is reflected in the replacement of $\Pr(z|\mathbf{x})$ by $\Pr(y'|\mathbf{x})$. This new form for the objective function (see equation 3.18) is exactly the same as for a standard VQ (see equation 3.5), except that the posterior probability $\Pr(y|\mathbf{x})$ is now processed through a transition matrix $\Pr(y'|y)$ to produce $\Pr(y'|\mathbf{x})$. Because $\Pr(y'|y) = \sum_{z=1}^{M_2}\Pr(y'|z)\Pr(z|y)$, it takes account of the effect of the state z of layer 2 on the training of layer 1, which is a type of self-supervision [7] in which higher layers of a network coordinate the training of lower layers. However, viewed from the point of view of layer 1, the effect of the transition matrix $\Pr(y'|y)$ is to do damage to the posterior probability by redistributing probability amongst the states of layer 1. This process is thus called probability leakage, and $\Pr(y'|y)$ is called a probability leakage matrix.

The objective function in equation 3.18 gives rise to a neural network that closely resembles a Kohonen topographic mapping neural network [3], where $\Pr(y'|y)$ may be identified as the topographic neighbourhood function, as was shown in [6]. Note that in order for the topographic neighbourhood to be localised (i.e. $\Pr(y'|y) > 0$ only for y' in some local neighbourhood of y), the transition matrix $\Pr(z|y)$ that generates the state of layer 2 from the state of layer 1 must generate each z state from y states that are all close to each other. This connection with Kohonen topographic mapping neural networks is only approximate, because the training algorithm proposed by Kohonen does not correspond to the minimisation of any objective function. A generalised version of the Kohonen network which allows a factorial code to emerge may be derived using the results in section V [9].

E. Additional Results

The objective function $\sum_{l=0}^{L-1}\frac{D_{VQ}^{l}}{4(\sigma_l)^2}$ for a VQ-ladder couples the optimisation of the individual 2-layer VQs together. Because the output of the l^{th} VQ is the input to the $(l+1)^{th}$ VQ (for $l = 0, 1, 2\cdots, L-1$), the optimisation of the k^{th} VQ has side effects on the optimisation of the l^{th} VQs (for $l = k+1, k+2, \cdots, L-1$). This leads to the effect called self-supervision, in which top-down connections from higher to lower network layers are automatically generated, to allow the lower layers to process their input more effectively in the light of what the higher layers discover in the data [7]. This is the multilayer extension of the self-supervision effect that led to topographic mappings in section III D.

The general expression for $L(\mathbf{P},\mathbf{Q})$ in equation 2.13 is the sum of two terms: the objective function $\sum_{l=0}^{L-1}\sum_{i_l=1}^{M_l} P_{i_l}^{l}\, K_{i_l}\left(\mathbf{P}^{l+1|l}, \mathbf{Q}^{l|l+1}\right)$ for a ladder (because $\mathbf{Q}$ is not necessarily Gaussian, the ladder is not necessarily a VQ-ladder), plus the cost $L\left(\mathbf{P}^{L},\mathbf{Q}^{L}\right)$ of encoding layer L. The $L\left(\mathbf{P}^{L},\mathbf{Q}^{L}\right)$ term has precisely the form that is commonly used in density modelling, so any convenient density model could be used to parameterise $\mathbf{Q}^{L}$ in layer L. A typical implementation of the type of network that minimises $L(\mathbf{P},\mathbf{Q})$ thus splits into two pieces corresponding to the two different types of term in the objective function. In the special case where $L = 0$ (i.e. no ladder is used) this approach reduces to standard input density modelling.

IV. HIERARACHICAL ENCODING USING AN ADAPTIVE CLUSTER EXPANSION (ACE)

In this section the adaptive cluster expansion (ACE) network is discussed [5]. ACE is a tree-structured network, whose purpose is to decompose high-dimensional input vectors into a number of lower dimensional pieces. In section IV A the case of a deterministic source and a perfect model is considered, and in section IV B the case of a Gaussian model is discussed.

A. ACE: Tree-Structured Density Network

Consider the objective function $L(\mathbf{P},\mathbf{Q})$ for encoding an $L+1$ layer Markov source (see equation 2.13), and assume that the $Q_{i_l,i_{l+1}}^{l|l+1}$ part of the model is perfect so that $Q_{i_l,i_{l+1}}^{l|l+1} = P_{i_l,i_{l+1}}^{l|l+1}$ (for $l = 0, 1, \cdots, L-1$), and that the $P_{i_{l+1},i_l}^{l+1|l}$ part of the source is deterministic so that $P_{i_{l+1},i_l}^{l+1|l} = \delta_{i_{l+1},i_{l+1}(i_l)}$ (for $l = 0, 1, \cdots, L-1$), in which case $L(\mathbf{P},\mathbf{Q})$ simplifies as follows (see appendix A 1)

$$L(\mathbf{P},\mathbf{Q}) = H\left(\mathbf{P}^{0}\right) - H\left(\mathbf{P}^{L}\right) + L\left(\mathbf{P}^{L},\mathbf{Q}^{L}\right) \qquad (4.1)$$

where $H\left(\mathbf{P}^{0}\right) - H\left(\mathbf{P}^{L}\right)$ is the number of bits per symbol required to convert a $\mathbf{P}^{L}$-message into a $\mathbf{P}^{0}$-message, assuming that the $P_{i_{l+1},i_l}^{l+1|l}$ part of the source is deterministic, and that the model is perfect. This result is not very interesting in itself.

However, if the $P_{i_{l+1},i_l}^{l+1|l}$ part of the source is not only deterministic, but is also tree-structured, and the model is similarly tree-structured, then the notation must be modified thus

$$\begin{aligned}
i_l \to \mathbf{i}_l &= \left(\mathbf{i}_l^1, \mathbf{i}_l^2, \cdots\right) \\
i_{l+1} \to \mathbf{i}_{l+1} &= \left(i_{l+1}^1, i_{l+1}^2, \cdots\right) \\
P_{i_{l+1},i_l}^{l+1|l} \to P_{\mathbf{i}_{l+1},\mathbf{i}_l}^{l+1|l} &= P_{i_{l+1}^1,\mathbf{i}_l^1}^{l+1|l} P_{i_{l+1}^2,\mathbf{i}_l^2}^{l+1|l} \cdots = \delta_{i_{l+1}^1,i_{l+1}^1\left(\mathbf{i}_l^1\right)} \delta_{i_{l+1}^2,i_{l+1}^2\left(\mathbf{i}_l^2\right)} \cdots \\
Q_{i_l,i_{l+1}}^{l|l+1} \to Q_{\mathbf{i}_l,\mathbf{i}_{l+1}}^{l|l+1} &= P_{\mathbf{i}_l^1,i_{l+1}^1}^{l|l+1} P_{\mathbf{i}_l^2,i_{l+1}^2}^{l|l+1} \cdots
\end{aligned} \tag{4.2}$$

where the state i_l of layer l of the tree-structured Markov source is more naturally written as a vector state $\mathbf{i}_l$ that specifies the joint state of each branch of layer l of the tree (the i_l style of notation is more suitable for a non-tree-structured Markov source). Furthermore, the components of the vector $\mathbf{i}_l$ are partitioned as $\left(\mathbf{i}_l^1, \mathbf{i}_l^2, \cdots\right)$, where each $\mathbf{i}_l^c$ is the joint state of a subset c of nodes in layer l, where all the nodes in each subset are all siblings as seen from the point of view of layer $l+1$. Such a set of siblings is called a cluster. The components of the vector $\mathbf{i}_{l+1}$ are partitioned as $\left(i_{l+1}^1, i_{l+1}^2, \cdots\right)$, where i_{l+1}^c is the state of the parent (in layer $l+1$) of the siblings in cluster c in layer l.

This notation may be used to rearrange $L(\mathbf{P},\mathbf{Q})$ as follows (see appendix A 2)

$$L(\mathbf{P},\mathbf{Q}) = \sum_{l=0}^{L-1} \sum_{\text{cluster } c} H\left(\mathbf{P}_c^l\right) - \sum_{l=1}^{L} \sum_{\text{component } c} H\left(\mathbf{P}_c^l\right) + L\left(\mathbf{P}^L, \mathbf{Q}^L\right) \tag{4.3}$$

This expression for $L(\mathbf{P},\mathbf{Q})$ can be rewritten in terms of the mutual information $I\left(\mathbf{P}_c^l\right)$ between the components of cluster $\mathbf{i}_{l+1}^c$ as (see appendix A 2)

$$L(\mathbf{P},\mathbf{Q}) = -\sum_{l=1}^{L} \sum_{\text{cluster } c} I\left(\mathbf{P}_c^l\right) + \sum_{\text{cluster } c} H\left(\mathbf{P}_c^0\right) - \sum_{\text{cluster } c} H\left(\mathbf{P}_c^L\right) + L\left(\mathbf{P}^L, \mathbf{Q}^L\right) \tag{4.4}$$

Now assume that the model is perfect in the output layer, so that $\mathbf{Q}^L$ is given by $Q_{\mathbf{i}_L}^L = P_{\mathbf{i}_l^1}^L P_{\mathbf{i}_l^2}^L \cdots$. This allows $L\left(\mathbf{P}^L, \mathbf{Q}^L\right)$ to be simplified as $L\left(\mathbf{P}^L, \mathbf{Q}^L\right) = \sum_{\text{cluster } c} H\left(\mathbf{P}_c^L\right)$, so that $L(\mathbf{P},\mathbf{Q})$ may finally be expressed as

$$L(\mathbf{P},\mathbf{Q}) = -\sum_{l=1}^{L} \sum_{\text{cluster } c} I\left(\mathbf{P}_c^l\right) + \sum_{\text{cluster } c} H\left(\mathbf{P}_c^0\right) \tag{4.5}$$

The $-\sum_{l=1}^{L} \sum_{\text{cluster } c} I\left(\mathbf{P}_c^l\right)$ term is (minus) the sum of the mutual informations within all of the clusters in the $L+1$ layer network, and the $\sum_{\text{cluster } c} H\left(\mathbf{P}_c^0\right)$ term is constant for a given external source $\mathbf{P}^0$. This means that minimising $L(\mathbf{P},\mathbf{Q})$ is equivalent to maximising $\sum_{l=1}^{L} \sum_{\text{cluster } c} I\left(\mathbf{P}_c^l\right)$. This is the maximum mutual information result for ACE networks [8], which includes the mutual information maximisation principle in [1] as a special case.

Note that if the source is deterministic and the model is perfect (as they are here), then $L\left(\mathbf{P}^0, \mathbf{Q}^0\right) = L(\mathbf{P},\mathbf{Q})$, which implies that input density optimisation is equivalent to joint density optimisation. This equivalence was used in [8], where the sum-of-mutual-informations objective function was derived by minimising $L\left(\mathbf{P}^0, \mathbf{Q}^0\right)$.

B. ACE: Hierarchical Vector Quantiser

If the above ACE network is modified slightly, so that the model $\mathbf{Q}$ has exactly the same structure as before, but is Gaussian rather than perfect, then $Q_{i_l,i_{l+1}}^{l|l+1}$ becomes

$$Q_{i_l,i_{l+1}}^{l|l+1} \to Q_{\mathbf{i}_l,\mathbf{i}_{l+1}}^{l|l+1} = Q_{\mathbf{i}_l^1,i_{l+1}^1}^{l|l+1} Q_{\mathbf{i}_l^2,i_{l+1}^2}^{l|l+1} \cdots \tag{4.6}$$

where the individual $Q_{\mathbf{i}_l^c,i_{l+1}^c}^{l|l+1}$ are Gaussian. The expression for $L(\mathbf{P},\mathbf{Q})$ may then be written down by analogy with equation 3.6

$$L\left(\mathbf{P},\mathbf{Q}\right)=\sum_{l=0}^{L-1}\sum_{\text{cluster } c}\frac{D_{VQ}^{l,c}}{4\left(\sigma_{l,c}\right)^{2}}+L\left(\mathbf{P}^{L},\mathbf{Q}^{L}\right)-\sum_{l=0}^{L-1}\sum_{\text{cluster } c}\log\left(\frac{V_{l,c}}{\left(\sqrt{2\pi}\,\sigma_{l,c}\right)^{\dim\mathbf{i}_{l}^{c}}}\right) \tag{4.7}$$

Thus the ACE network, with a Gaussian model $\mathbf{Q}$, is a hierarchical VQ-ladder (or VQ-tree), in which each layer encodes the clusters in the previous layer [6].

V. FACTORIAL ENCODING USING A PARTITIONED MIXTURE DISTRIBUTION (PMD)

In this section a useful parameterisation of the conditional probability $\mathbf{P}^{l+1|l}$ for building the Markov source is introduced in order to encourage $\mathbf{P}^{l+1|l}$ to form factorial codes of the state of layer l. It turns out that there is a simple way of allowing such codes to develop, which is called the partitioned mixture distribution (PMD) [9]. A PMD achieves this by encoding its input simultaneously with a number of different recognition models, each of which potentially can encode a different part of the input.

In section V A two ways in which multiple recognition models can be used for factorial encoding are discussed, and a hybrid approach (which is a PMD) is discussed in section V B.

A. Multiple Recognition Models

In the expression for the $L\left(\mathbf{P},\mathbf{Q}\right)$ (see equation 2.13) the generative models $\mathbf{Q}^{l|l+1}$ may be parameterised as Gaussian probability densities, whereas the recognition models $\mathbf{P}^{l+1|l}$ may be parameterised in a more general way as

$$P_{i_{l+1},i_{l}}^{l+1|l}=\frac{P_{i_{l},i_{l+1}}^{l|l+1}\,P_{i_{l+1}}^{l+1}}{\sum_{i_{l+1}'=1}^{M_{l+1}}P_{i_{l},i_{l+1}'}^{l|l+1}\,P_{i_{l+1}'}^{l+1}} \tag{5.1}$$

which guarantees the normalisation condition $\sum_{i_{l+1}=1}^{M_{l+1}}P_{i_{l+1},i_{l}}^{l+1|l}=1$. A limitation of this type of recognition model is that it allows only a single explanation i_{l+1} of the data i_{l} (in the case of a hard $P_{i_{l+1},i_{l}}^{l+1|l}$), or a probability distribution over single explanations (in the case of a soft $P_{i_{l+1},i_{l}}^{l+1|l}$), so it cannot lead to a factorial encoding of the data.

The simplest way of allowing a factorial encoding to develop is to make simultaneous use more than one recognition model. Each recognition model uses its own $\mathbf{P}^{l+1}$ vector and $\mathbf{P}^{l|l+1}$ matrix to compute a posterior probability of the type shown in equation 5.1, so that if each recognition model is sensitised to a different part of the input, then a factorial code can develop. This approach can be formalised by making the replacement $i_{l+1}\to\mathbf{i}_{l+1}$ in equation 5.1 (i.e. replace the scalar code index by a vector code index, where the number of vector components is equal to the number of recognition models). If the components of $\mathbf{i}_{l+1}$ are determined independently of each other, then their joint posterior probability $P_{\mathbf{i}_{l+1},i_{l}}^{l+1|l}$ is a product of independent posterior probabilities, where each posterior probability corresponds to one of the recognition models, and thus has its own $\mathbf{P}^{l+1}$ vector and $\mathbf{P}^{l|l+1}$ matrix.

If this type of posterior probability, which is a product of n independent factors if there are n independent recognition models, is then inserted into equation 3.5 it yields (see appendix B)

$$D_{VQ}\leq 2\int d\mathbf{x}\,\Pr\left(\mathbf{x}\right)\sum_{y_{1}=1}^{M_{1}}\sum_{y_{2}=1}^{M_{2}}\cdots\sum_{y_{n}=1}^{M_{n}}\Pr\left(y_{1}|\mathbf{x},1\right)\,\Pr\left(y_{2}|\mathbf{x},2\right)\,\cdots\,\Pr\left(y_{n}|\mathbf{x},n\right)\,\left\|\mathbf{x}-\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}_{k}'\left(y_{k}\right)\right\|^{2} \tag{5.2}$$

If a single recognition model is independently used n times, rather than n independent recognition models each independently being used once, then the above result becomes

$$D_{VQ}\leq\frac{2}{n}\int d\mathbf{x}\,\Pr\left(\mathbf{x}\right)\sum_{y=1}^{M}\Pr\left(y|\mathbf{x}\right)\,\left\|\mathbf{x}-\mathbf{x}'\left(y\right)\right\|^{2}+\frac{2\left(n-1\right)}{n}\int d\mathbf{x}\,\Pr\left(\mathbf{x}\right)\,\left\|\mathbf{x}-\sum_{y=1}^{M}\Pr\left(y|\mathbf{x}\right)\,\mathbf{x}'\left(y\right)\right\|^{2} \tag{5.3}$$

In the case $n = 1$ this correctly reduces to equation 3.5 (the inequality reduces to an equality in this case). When $n > 1$ the second term offers the possibility of factorial encoding, because it contains a weighted linear combination $\sum_{y=1}^{M} \Pr(y|\mathbf{x})\, \mathbf{x}'(y)$ of vectors.

B. Average Over Recognition Models

Now combine the above two approaches to factorial encoding, so that a single recognition model is used (as in equation 5.3), which is parameterised in such a way that it can emulate multiple recognition models (as in equation 5.2). The simplest possibility is to firstly make the replacement $P_{i_{l+1}}^{l+1} \to A_{k,i_{l+1}}^{l+1} P_{i_{l+1}}^{l+1}$ (where $A_{k,i_{l+1}}^{l+1} \geq 0$) in equation 5.1, where k is a recognition model index which ranges over $k = 1, 2, \cdots, K$ (note that K is not constrained to be the same as n), and then secondly to average over k, to produce

$$P_{i_{l+1},i_l}^{l+1|l} \to \frac{1}{K} \sum_{k=1}^{K} \frac{P_{i_l,i_{l+1}}^{l|l+1} A_{k,i_{l+1}}^{l+1} P_{i_{l+1}}^{l+1}}{\sum_{i'_{l+1}=1}^{M_{l+1}} P_{i_l,i'_{l+1}}^{l|l+1} A_{k,i'_{l+1}}^{l+1} P_{i'_{l+1}}^{l+1}} \tag{5.4}$$

In effect, K recognition models are embedded between layer l and layer $l+1$, and the $\mathbf{A}^{l+1}$ matrix specifies which indices i_{l+1} in layer $l+1$ are associated with recognition model k.

The result in equation 5.4 is not the same as the result that would have been obtained using a Bayesian analysis, in which the posterior probabilities generated by different models are combined to yield a single posterior probability. In appendix B there is a discussion of the relationship between the above proposed PMD recognition model and a full Bayesian average over alternative recognition models.

A partitioned mixture distribution (PMD) is precisely this type of multiple embedded recognition model. In the simplest type of PMD the $\mathbf{A}^{l+1}$ matrix is chosen to contain only 0's and 1's, which are arranged so that the K recognition models partition layer $l+1$ into K overlapping patches [9]. A wide range of types of PMD can be constructed by choosing $\mathbf{A}^{l+1}$ appropriately.

In section III D it was shown how a Kohonen topographic mapping emerged when a 3-layer Markov source network was optimised. If the PMD posterior probability (see equation 5.4) had been used in section III D, then a more general form of topographic mapping (i.e. a factorial topographic mapping) would have emerged (this is briefly discussed in [9]).

VI. CONCLUSIONS

The objective function for optimising the density model of a Markov source may be applied to the problem of optimising the joint density of all the layers of a neural network. This is possible because the joint state of all of the network layers may be viewed as a Markov chain of states (each layer is connected only to adjacent layers). This representation makes contact with the results that were reported in [10], and allows many results to be unified into a single approach (i.e. a single objective function).

The most significant aspect of this unification is the fact that all layers of a neural network are treated on an equal footing, unlike in the conventional approach to density modelling where the input layer is accorded a special status. For instance, this leads to a modular approach to building neural networks, where all of the modules have the same structure.

VII. ACKNOWLEDGEMENTS

I thank Chris Webber for many useful conversations that we had during the course of this research. I also thank Peter Dayan and Geoffrey Hinton for conversations that we had about the relationship between Markov source modelling and Helmholtz machines during the 1997 “Neural Networks and Machine Learning” programme at the Newton Institute in Cambridge.

Appendix A: ACE

In this appendix some of the more technical details relevant to section IV are given.

1. Perfect Model, Deterministic Source

The derivation of the result in equation 4.1 for a perfect model (i.e. $\mathbf{Q} = \mathbf{P}$) and a deterministic source (i.e. $P_{i_{l+1},i_l}^{l+1|l} = \delta_{i_{l+1},i_{l+1}(i_l)}$ using scalar notation i_l rather than vector notation $\mathbf{i}_l$ for the state of layer l, because here the Markov source is not assumed to be tree-structured) is as follows. The basic definition of $L(\mathbf{P}, \mathbf{Q})$ in equation 2.13

may be written as

$$L(\mathbf{P},\mathbf{Q}) - L\left(\mathbf{P}^L,\mathbf{Q}^L\right) = -\sum_{l=0}^{L-1}\sum_{i_l=1}^{M_l} P_{i_l}^l \sum_{i_{l+1}=1}^{M_{l+1}} P_{i_{l+1},i_l}^{l+1|l} \log P_{i_l,i_{l+1}}^{l|l+1} \tag{A1}$$

This may be simplified by noting that $P_{i_{l+1},i_l}^{l+1|l} = \delta_{i_{l+1},i_{l+1}(i_l)}$, and that Bayes' theorem gives $P_{i_l,i_{l+1}}^{l|l+1} = \frac{P_{i_{l+1},i_l}^{l+1|l} P_{i_l}^l}{P_{i_{l+1}}^{l+1}}$, which yields

$$L(\mathbf{P},\mathbf{Q}) - L\left(\mathbf{P}^L,\mathbf{Q}^L\right) = -\sum_{l=0}^{L-1}\sum_{i_l=1}^{M_l} P_{i_l}^l \sum_{i_{l+1}=1}^{M_{l+1}} \delta_{i_{l+1},i_{l+1}(i_l)} \log \frac{\delta_{i_{l+1},i_{l+1}(i_l)} P_{i_l}^l}{P_{i_{l+1}}^{l+1}} \tag{A2}$$

Now use that $\sum_{i_{l+1}=1}^{M_{l+1}} \delta_{i_{l+1},i_{l+1}(i_l)} = 1$, $\sum_{i_{l+1}=1}^{M_{l+1}} \delta_{i_{l+1},i_{l+1}(i_l)} \log \delta_{i_{l+1},i_{l+1}(i_l)} = 0$ and $\sum_{i_l=1}^{M_l} P_{i_l}^l \, \delta_{i_{l+1},i_{l+1}(i_l)} = P_{i_{l+1}}^{l+1}$ to reduce this to

$$L(\mathbf{P},\mathbf{Q}) - L\left(\mathbf{P}^L,\mathbf{Q}^L\right) = -\sum_{l=0}^{L-1}\sum_{i_l=1}^{M_l} P_{i_l}^l \log P_{i_l}^l + \sum_{l=0}^{L-1}\sum_{i_{l+1}=1}^{M_{l+1}} P_{i_{l+1}}^{l+1} \log P_{i_{l+1}}^{l+1} \tag{A3}$$

The terms in these two series mostly cancel each other to yield

$$L(\mathbf{P},\mathbf{Q}) - L\left(\mathbf{P}^L,\mathbf{Q}^L\right) = -\sum_{i_0=1}^{M_0} P_{i_0}^0 \log P_{i_0}^0 + \sum_{i_L=1}^{M_L} P_{i_L}^L \log P_{i_L}^L \tag{A4}$$

and using the definition of entropy (see equation 2.3) this may finally be written as

$$L(\mathbf{P},\mathbf{Q}) - L\left(\mathbf{P}^L,\mathbf{Q}^L\right) = H\left(\mathbf{P}^0\right) - H\left(\mathbf{P}^L\right) \tag{A5}$$

2. Perfect Model, Deterministic Source: Tree-Structured Case

The derivation of the result in equation 4.3 for a perfect tree-structured model and a deterministic tree-structured source is may be obtained by altering the notation in appendix A 1 to reflect the fact that both the Markov source and model are now tree-structured. Thus use the notation defined in equation 4.2 to write $L(\mathbf{P},\mathbf{Q})$ (see equation 2.13) as

$$L(\mathbf{P},\mathbf{Q}) - L\left(\mathbf{P}^L,\mathbf{Q}^L\right) = -\sum_{l=0}^{L-1}\sum_{\mathbf{i}_l} P_{\mathbf{i}_l}^l \sum_{\mathbf{i}_{l+1}} P_{\mathbf{i}_{l+1},\mathbf{i}_l}^{l+1|l} \sum_{\text{cluster } c} \log P_{\mathbf{i}_l^c,i_{l+1}^c}^{l|l+1} \tag{A6}$$

Now use Bayes' theorem in the form $P_{\mathbf{i}_l^c,i_{l+1}^c}^{l|l+1} = \frac{P_{i_{l+1}^c,\mathbf{i}_l^c}^{l+1|l} P_{\mathbf{i}_l^c}^l}{P_{i_{l+1}^c}^{l+1}}$ to write this as

$$L(\mathbf{P},\mathbf{Q}) - L\left(\mathbf{P}^L,\mathbf{Q}^L\right) = -\sum_{l=0}^{L-1}\sum_{\mathbf{i}_l} P_{\mathbf{i}_l}^l \sum_{\mathbf{i}_{l+1}} P_{\mathbf{i}_{l+1},\mathbf{i}_l}^{l+1|l} \sum_{\text{cluster } c} \log\left(\frac{P_{i_{l+1}^c,\mathbf{i}_l^c}^{l+1|l} P_{\mathbf{i}_l^c}^l}{P_{i_{l+1}^c}^{l+1}}\right) \tag{A7}$$

This may be simplified by using that $\sum_{\mathbf{i}_{l+1}} P_{\mathbf{i}_{l+1},\mathbf{i}_l}^{l+1|l} = 1$ (for the $\log P_{\mathbf{i}_l^c}^l$ term), $\sum_{\mathbf{i}_l} P_{\mathbf{i}_l}^l \sum_{\mathbf{i}_{l+1}} P_{\mathbf{i}_{l+1},\mathbf{i}_l}^{l+1|l} (\cdots) = \sum_{\mathbf{i}_{l+1}} P_{\mathbf{i}_{l+1}}^{l+1} (\cdots)$ (for the $\log P_{i_{l+1}^c}^{l+1}$ term), and $\sum_{i_{l+1}=1}^{M_{l+1}} \delta_{i_{l+1}^c,i_{l+1}^c(\mathbf{i}_l^c)} \log \delta_{i_{l+1}^c,i_{l+1}^c(\mathbf{i}_l^c)} = 0$ (for the $\log P_{i_{l+1}^c,\mathbf{i}_l^c}^{l+1|l}$ term), to yield

$$L(\mathbf{P},\mathbf{Q}) - L\left(\mathbf{P}^L,\mathbf{Q}^L\right) = -\sum_{l=0}^{L-1}\sum_{\mathbf{i}_l} P_{\mathbf{i}_l}^l \sum_{\text{cluster } c} \log P_{\mathbf{i}_l^c}^l + \sum_{l=0}^{L-1}\sum_{\mathbf{i}_{l+1}} P_{\mathbf{i}_{l+1}}^{l+1} \sum_{\text{cluster } c} \log P_{i_{l+1}^c}^{l+1} \tag{A8}$$

The first term may be simplified by interchanging the order of summation $\sum_{\mathbf{i}_l}\sum_c (\cdots) = \sum_c \sum_{\mathbf{i}_l} (\cdots)$, and then marginalising the probabilities using that $\sum_{\text{cluster } c}\sum_{\mathbf{i}_l} P_{\mathbf{i}_l}^l \log P_{\mathbf{i}_l^c}^l = \sum_{\text{cluster } c}\sum_{\mathbf{i}_l^c} P_{\mathbf{i}_l^c}^l \log P_{\mathbf{i}_l^c}^l$. The second term may be simplified by interchanging the order of summation $\sum_{\mathbf{i}_{l+1}} \sum_{\text{cluster } c} (\cdots) = \sum_{\text{cluster } c} \sum_{\mathbf{i}_{l+1}} (\cdots)$, then marginalising the probabilities using that $\sum_{\text{cluster } c} \sum_{\mathbf{i}_{l+1}} P^{l+1}_{\mathbf{i}_{l+1}} \log P^{l+1}_{i^c_{l+1}} = \sum_{\text{cluster } c} \sum_{i^c_{l+1}} P^{l+1}_{i^c_{l+1}} \log P^{l+1}_{i^c_{l+1}}$, and then using that component c in layer $l+1$ is the parent of cluster c in layer l, to obtain

$$L(\mathbf{P},\mathbf{Q}) - L\left(\mathbf{P}^L,\mathbf{Q}^L\right) = -\sum_{l=0}^{L-1} \sum_{\mathbf{i}^c_l} P^l_{\mathbf{i}^c_l} \sum_{\text{cluster } c} \log P^l_{\mathbf{i}^c_l} + \sum_{l=1}^{L} \sum_{i^c_l} P^l_{i^c_l} \sum_{\text{component } c} \log P^l_{i^c_l} \tag{A9}$$

and using the definition of entropy (see equation 2.3) this may finally be written as

$$L(\mathbf{P},\mathbf{Q}) - L\left(\mathbf{P}^L,\mathbf{Q}^L\right) = \sum_{l=0}^{L-1} \sum_{\text{cluster } c} H\left(\mathbf{P}^l_c\right) - \sum_{l=1}^{L} \sum_{\text{component } c} H\left(P^l_c\right) \tag{A10}$$

where $H\left(\mathbf{P}^l_c\right)$ is the entropy of cluster c and $H\left(P^l_c\right)$ is the entropy of component c (both in layer l). The mutual information $I\left(\mathbf{P}^l_c\right)$ between the components c' of cluster c is defined as

$$I\left(\mathbf{P}^l_c\right) \equiv \sum_{\substack{\text{component } c' \\ \text{in cluster } c}} H\left(P^l_{c'}\right) - H\left(\mathbf{P}^l_c\right) \tag{A11}$$

and using that $\sum_{\text{cluster } c} \sum_{\substack{\text{component } c' \\ \text{in cluster } c}} (\cdots) = \sum_{\text{component } c'}$ this yields

$$\sum_{\text{cluster } c} I\left(\mathbf{P}^l_c\right) \equiv \sum_{\text{component } c} H\left(P^l_c\right) - \sum_{\text{cluster } c} H\left(\mathbf{P}^l_c\right) \tag{A12}$$

which allows $L(\mathbf{P},\mathbf{Q}) - L\left(\mathbf{P}^L,\mathbf{Q}^L\right)$ to be simplified to

$$L(\mathbf{P},\mathbf{Q}) - L\left(\mathbf{P}^L,\mathbf{Q}^L\right) = -\sum_{l=1}^{L} \sum_{\text{cluster } c} I\left(\mathbf{P}^l_c\right) + \sum_{\text{cluster } c} H\left(\mathbf{P}^0_c\right) - \sum_{\text{cluster } c} H\left(\mathbf{P}^L_c\right) \tag{A13}$$

Appendix B: PMD

In this appendix some of the more technical details relevant to section V are given.

1. PMD Recognition Model

If the type of posterior probability introduced in section V A, which is a product of n independent factors if there are n independent recognition models, is then inserted into equation 3.5 it yields a D_{VQ} of the form

$$D_{VQ} = 2 \int d\mathbf{x} \, \Pr(\mathbf{x}) \sum_{y_1=1}^{M_1} \sum_{y_2=1}^{M_2} \cdots \sum_{y_n=1}^{M_n} \Pr(y_1|\mathbf{x},1) \Pr(y_2|\mathbf{x},2) \cdots \Pr(y_n|\mathbf{x},n) \left\|\mathbf{x} - \mathbf{x}'(y_1,y_2,\cdots,y_n)\right\|^2 \tag{B1}$$

where $\Pr(y_k|\mathbf{x},k)$ denotes the posterior probability that (given input $\mathbf{x}$) code index y_k occurs in recognition model k. If $\mathbf{x}'(y_1,y_2,\cdots,y_n)$ is optimised (i.e. takes the value that minimises D_{VQ}) then it becomes

$$\mathbf{x}'(y_1,y_2,\cdots,y_n) = \int d\mathbf{x} \, \Pr(y_1|\mathbf{x}) \Pr(y_2|\mathbf{x}) \cdots \Pr(y_n|\mathbf{x}) \, \mathbf{x} \tag{B2}$$

The $\left\|\mathbf{x} - \mathbf{x}'(y_1,y_2,\cdots,y_n)\right\|^2$ term may be expanded thus (by adding and subtracting $\frac{1}{n}\sum_{k=1}^{n} \mathbf{x}'_k(y_k)$)

$$\left\|\mathbf{x} - \mathbf{x}'(y_1,y_2,\cdots,y_n)\right\|^2 \equiv \left\| \left(\mathbf{x} - \frac{1}{n}\sum_{k=1}^{n} \mathbf{x}'_k(y_k)\right) + \left(\frac{1}{n}\sum_{k=1}^{n} \mathbf{x}'_k(y_k) - \mathbf{x}'(y_1,y_2,\cdots,y_n)\right) \right\|^2 \tag{B3}$$

Using these two results, together with Bayes' theorem, allows an upper bound on D_{VQ} to be derived as

$$D_{VQ} \le 2\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{y_1=1}^{M_1}\sum_{y_2=1}^{M_2}\cdots\sum_{y_n=1}^{M_n}\Pr(y_1|\mathbf{x},1)\,\Pr(y_2|\mathbf{x},2)\,\cdots\,\Pr(y_n|\mathbf{x},n)\left\|\mathbf{x}-\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}'_k(y_k)\right\|^2 \tag{B4}$$

In this upper bound, the $\Pr(y_k|\mathbf{x},k)$ are used to produce soft encodings in each of the recognition models ($k = 1,2,\cdots,n$), then a sum $\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}'_k(y_k)$ of the vectors $\mathbf{x}'_k(y_k)$ is used as the reconstruction of the input $\mathbf{x}$. In the special case where hard encodings are used, so that $\Pr(y_k|\mathbf{x},k) = \delta_{y_k,y_k(\mathbf{x})}$, then the upper bound on D_{VQ} reduces to $D_{VQ} \le 2\int d\mathbf{x}\,\Pr(\mathbf{x})\left\|\mathbf{x}-\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}'_k(y_k(\mathbf{x}))\right\|^2$. Note that the code vectors used for the encoding operation $y_k(\mathbf{x})$ are not necessarily the same as the $\mathbf{x}'_k(y_k)$, except in the special case $n = 1$.

Suppose that a single recognition model is independently used n times, rather than n independent recognition models each independently being used once. This corresponds to constraining the $\mathbf{P}^{l+1}$ vectors and $\mathbf{P}^{l|l+1}$ matrices to be the same for each of the n recognition models. The upper bound on D_{VQ} can be manipulated into the form

$$D_{VQ} \le \frac{2}{n}\int d\mathbf{x}\,\Pr(\mathbf{x})\sum_{y=1}^{M}\Pr(y|\mathbf{x})\,\|\mathbf{x}-\mathbf{x}'(y)\|^2 + \frac{2(n-1)}{n}\int d\mathbf{x}\,\Pr(\mathbf{x})\left\|\mathbf{x}-\sum_{y=1}^{M}\Pr(y|\mathbf{x})\,\mathbf{x}'(y)\right\|^2 \tag{B5}$$

where the k index is no longer needed.

B. Full Bayesian Average Over Recognition Models

One possible criticism of the recognition model given in equation 5.4 is that it is a mixture of K recognition models, where each contributing model is assigned the same weight $\frac{1}{K}$. Normally, a posterior probability $\Pr(y|\mathbf{x})$ is decomposed as a sum over posterior probabilities $\Pr(y|\mathbf{x},k)$ derived from each contributing model, as follows

$$\Pr(y|\mathbf{x}) = \sum_{k=1}^{K}\Pr(y|\mathbf{x},k)\,\Pr(k|\mathbf{x}) \tag{2.6}$$

where each of the K recognition models is assigned a different data-dependent weight $\Pr(k|\mathbf{x})$. The conditional probabilities $\Pr(k|\mathbf{x})$ and $\Pr(y|\mathbf{x})$ can be evaluated to yield

$$\begin{aligned}\Pr(k|\mathbf{x}) &= \frac{\sum_{y=1}^{M}\Pr(\mathbf{x}|y,k)\,\Pr(y|k)\,\Pr(k)}{\sum_{k'=1}^{K}\sum_{y'=1}^{M}\Pr(\mathbf{x}|y',k')\,\Pr(y'|k')\,\Pr(k')}\\ \Pr(y|\mathbf{x},k) &= \frac{\Pr(\mathbf{x}|y,k)\,\Pr(y|k)\,\Pr(k)}{\sum_{y'=1}^{M}\Pr(\mathbf{x}|y',k)\,\Pr(y'|k)\,\Pr(k)}\end{aligned} \tag{2.7}$$

so that

$$\Pr(y|\mathbf{x}) = \sum_{k=1}^{K}\frac{\Pr(\mathbf{x}|y,k)\,\Pr(y|k)\,\Pr(k)}{\sum_{k'=1}^{K}\sum_{y'=1}^{M}\Pr(\mathbf{x}|y',k')\,\Pr(y'|k)\,\Pr(k')} \tag{2.8}$$

If the replacements $\Pr(k)\to 1$, $\Pr(y|k)\to A^{l+1}_{k,i_{l+1}}\,P^{l+1}_{i_{l+1}}$, $\Pr(\mathbf{x}|y,k)\to P^{l|l+1}_{i_l,i_{l+1}}$, and $\Pr(y|\mathbf{x})\to P^{l+1|l}_{i_{l+1},i_l}$ are made, then $\Pr(y|\mathbf{x})$ reduces to

$$P^{l+1|l}_{i_{l+1},i_l} = \sum_{k=1}^{K}\frac{P^{l|l+1}_{i_l,i_{l+1}}\,A^{l+1}_{k,i_{l+1}}\,P^{l+1}_{i_{l+1}}}{\sum_{k'=1}^{K}\sum_{i'_{l+1}=1}^{M_{l+1}}P^{l|l+1}_{i_l,i'_{l+1}}\,A^{l+1}_{k',i'_{l+1}}\,P^{l+1}_{i'_{l+1}}} \tag{2.9}$$

which is not the same as the PMD recognition model in equation 5.4. The difference between equation 2.9 and equation 5.4 arises because the full Bayesian approach in equation 2.9 ensures that the model index k and the input $\mathbf{x}$ are mutually dependent (via the factor $\Pr(k|\mathbf{x})$), whereas the PMD approach in equation 5.4 ignores such dependencies.

In the full Bayesian approach (see equation 2.9) the normalisation term in the denominator has a double summation $\sum_{k=1}^{K}\sum_{i_{l+1}=1}^{M_{l+1}}P^{l|l+1}_{i_l,i_{l+1}}\,A^{l+1}_{k,i_{l+1}}\,P^{l+1}_{i_{l+1}}$, which involves all pairs of indices k and i_{l+1} with $A^{l+1}_{k,i_{l+1}} > 0$, which thus corresponds to long-range lateral interactions in layer $l+1$. On the other hand, in the PMD approach (see equation 5.4) the normalisation term in the denominator has only a single summation $\sum_{i_{l+1}=1}^{M_{l+1}}P^{l|l+1}_{i_l,i_{l+1}}\,A^{l+1}_{k,i_{l+1}}\,P^{l+1}_{i'_{l+1}}$, so the lateral interactions in layer $l+1$ are determined by the structure of the matrix $A^{l+1}_{k,i_{l+1}}$, which defines only short-range lateral connections (i.e. for a given recognition model k, only a limited number of index values i_{l+1} satisfy $A^{l+1}_{k,i_{l+1}} > 0$.

III. COMPARISON WITH THE HELMHOLTZ MACHINE

In this appendix the relationship between two types of density model is discussed. The first type is a conventional density model that approximates the input probability density (i.e. the objective function is $L(\mathbf{P}^0,\mathbf{Q}^0)$), and the second type is the one introduced here that approximates the joint probability density of a Markov source (i.e. the objective function is $L(\mathbf{P},\mathbf{Q})$). In order to relate $L(\mathbf{P}^0,\mathbf{Q}^0)$ to $L(\mathbf{P},\mathbf{Q})$ it is necessary to introduce additional layers (i.e. layers $1,2,\cdots,L$) into $L(\mathbf{P}^0,\mathbf{Q}^0)$ in an appropriate fashion.

The Helmholtz machine (HM) [2] does this by replacing $L\left(\mathbf{P}^0,\mathbf{Q}^0\right)$ by a different objective function (which has these additional layers present as hidden variables), and which is an upper bound on the original objective function $L\left(\mathbf{P}^0,\mathbf{Q}^0\right)$. It turns out that Helmholtz machine (HM) objective function D_{HM} and the Markov source objective function $L\left(\mathbf{P},\mathbf{Q}\right)$ are closely related. The essential difference between the two is that D_{HM} does not include the cost of specifying the state of layers $1,2,\cdots,L$ given that the state of layer 0 is known, which thus allows it to develop distributed codes (which are expensive to specify) more easily.

In the conventional density modelling approach to neural networks, there are two basic classes of model. In the case of both unsupervised and supervised neural networks the source is $\mathbf{P}^0$, which is the network input (unsupervised case) or the network output (supervised case). Additionally, in the case of supervised neural networks $\mathbf{P}^0$ is conditioned on an additional network input as $\mathbf{P}^{0|\text{input}}$. Thus in both cases there is only an external source (i.e. source layers $1,2,\cdots,L$ are not present), which is modelled by $\mathbf{Q}^0$ (unsupervised case) or $\mathbf{Q}^{0|\text{input}}$ (supervised case). $\mathbf{Q}^0$ or $\mathbf{Q}^{0|\text{input}}$ can be modelled in any way that is convenient. Frequently a multilayer generative model of the form

$$Q^0_{i_0} = \sum_{i_1,i_2,\cdots,i_L} Q^{0|1}_{i_0,i_1}\cdots Q^{l|l+1}_{i_l,i_{l+1}}\cdots Q^L_{i_L} \tag{3.1}$$

is used, where the i_l (for $1\leq l\leq L$) are hidden variables, which need to be summed over in order to calculate the required marginal probability $Q^0_{i_0}$, and the notation is deliberately chosen to be the same as is used in the Markov chain model

$$Q_{i_0,i_1,\cdots,i_L} = Q^{0|1}_{i_0,i_1}\cdots Q^{l|l+1}_{i_l,i_{l+1}}\cdots Q^L_{i_L} \tag{3.2}$$

Helmholtz machines and Markov sources are related to each other. Thus the $L\left(\mathbf{P}^0,\mathbf{Q}^0\right)$ that is minimised in conventional density modelling can be manipulated in order to derive D_{HM}

$$\begin{aligned}
L\left(\mathbf{P}^0,\mathbf{Q}^0\right) &\leq L\left(\mathbf{P}^0,\mathbf{Q}^0\right) + \sum_{i_0=1}^{M_0} P^0_{i_0} G_{i_0}\left(\mathbf{P}^{1|0},\mathbf{Q}^{1|0}\right)\\
&= -\sum_{i_0=1}^{M_0} P^0_{i_0}\sum_{i_1=1}^{M_1} P^{1|0}_{i_1,i_0}\log\left(Q^0_{i_0}Q^{1|0}_{i_1,i_0}\right) + \sum_{i_0=1}^{M_0} P^0_{i_0}\sum_{i_1=1}^{M_1} P^{1|0}_{i_1,i_0}\log P^{1|0}_{i_1,i_0}\\
&= L\left(\left(\mathbf{P}^0,\mathbf{P}^{1|0}\right),\left(\mathbf{Q}^0,\mathbf{Q}^{1|0}\right)\right) - \sum_{i_0=1}^{M_0} P^0_{i_0} H_{i_0}\left(\mathbf{P}^{1|0}\right)\\
&= L\left(\mathbf{P},\mathbf{Q}\right) - \sum_{i_0=1}^{M_0} P^0_{i_0} H_{i_0}\left(\mathbf{P}^{1|0}\right)\\
&\equiv D_{HM}
\end{aligned} \tag{3.3}$$

The inequality $L\left(\mathbf{P}^0,\mathbf{Q}^0\right)\leq D_{HM}$ follows from $G_{i_0}\left(\mathbf{P}^{1|0},\mathbf{Q}^{1|0}\right)\geq 0$ (i.e. the model $\mathbf{Q}^{1|0}$ is imperfect, so that $\mathbf{Q}^{1|0}\neq\mathbf{P}^{1|0}$). The inequality $D_{HM}\leq L\left(\mathbf{P},\mathbf{Q}\right)$ follows from $H_{i_0}\left(\mathbf{P}^{1|0}\right)\geq 0$ (i.e. the source $\mathbf{P}^{1|0}$ is stochastic). If the model is perfect ($\mathbf{Q}^{1|0}=\mathbf{P}^{1|0}$) and the source is deterministic ($\mathbf{P}^{1|0}$ is such that the state of layer 1 is known once the state of layer 0 is given), then these two inequalities reduce to $L\left(\mathbf{P}^0,\mathbf{Q}^0\right)=L\left(\mathbf{P},\mathbf{Q}\right)$.

The properties of the optimal codes that are used by a Helmholtz machine when D_{HM} is minimised may be investigated by writing the expression for D_{HM} as a sum of two terms

$$D_{HM} = \sum_{i_0=1}^{M_0} P^0_{i_0} K_{i_0}\left(\mathbf{P}^{1|0},\mathbf{Q}^{0|1}\right) + \sum_{i_0=1}^{M_0} P^0_{i_0} G_{i_0}\left(\mathbf{P}^{1|0},\mathbf{Q}^{1}\right) \tag{3.4}$$

The $\sum_{i_0=1}^{M_0} P^0_{i_0} K\left(\mathbf{P}^{1|0},\mathbf{Q}^{0|1}\right)$ part and the $\sum_{i_0=1}^{M_0} P^0_{i_0} G\left(\mathbf{P}^{1|0},\mathbf{Q}^{1}\right)$ part compete with each other when D_{HM} is minimised. Assuming that $P^0_{i_0}>0$, the $\sum_{i_0=1}^{M_0} P^0_{i_0} G\left(\mathbf{P}^{1|0},\mathbf{Q}^{1}\right)$ part likes to make $\mathbf{Q}^1$ approximate $\mathbf{P}^{1|0}$, which tends to make $\mathbf{P}^{1|0}$ behave like a distributed encoder. On the other hand, assuming that $P^1_{i_1}>0$, the $\sum_{i_0=1}^{M_0} P^0_{i_0} K\left(\mathbf{P}^{1|0},\mathbf{Q}^{0|1}\right)$ part likes to make $\mathbf{Q}^{0|1}$ approximate $\mathbf{P}^{0|1}$, which tends to make $\mathbf{P}^{1|0}$ behave like a sparse encoder. The tension between these two terms is optimally balanced when D_{HM} is minimised.

The properties of the optimal codes that are used in the Markov source approach when $L\left(\mathbf{P},\mathbf{Q}\right)$ (see equation 2.13) is minimised are different. The 2-layer expression for $L(\mathbf{P},\mathbf{Q})$ is

$$L(\mathbf{P},\mathbf{Q}) = \sum_{i_0=1}^{M_0} P_{i_0}^0 K_{i_0}\left(\mathbf{P}^{1|0},\mathbf{Q}^{0|1}\right) + L\left(\mathbf{P}^L,\mathbf{Q}^L\right) \quad (3.5)$$

which contains the same sparse encoder term $\sum_{i_0=1}^{M_0} P_{i_0}^0 K_{i_0}\left(\mathbf{P}^{1|0},\mathbf{Q}^{0|1}\right)$ as D_{HM}. However, the distributed encoder term $\sum_{i_0=1}^{M_0} P_{i_0}^0 G_{i_0}\left(\mathbf{P}^{1|0},\mathbf{Q}^{1}\right)$ is missing, and is replaced by $L\left(\mathbf{P}^L,\mathbf{Q}^L\right)$ which does not have the effect of encouraging any particular type of code (other than one in which $\mathbf{P}^L$ approximates $\mathbf{Q}^L$).

These differences between D_{HM} and $L(\mathbf{P},\mathbf{Q})$ show how the Markov source approach encourages sparse codes to develop, whereas the Helmholtz machine does not. It is not clear whether using D_{HM} is the best approach to forming distributed codes, because there are other ways of encouraging distributed codes to develop, such as the factorial encoder discussed in section V, which is based on $L(\mathbf{P},\mathbf{Q})$ rather than D_{HM}.

[1] S Becker and G E Hinton, *Self-organising neural network that discovers surfaces in random-dot stereograms*, Nature **355** (1992), no. 6356, 161–163.

[2] P Dayan, G E Hinton, R M Neal, and R S Zemel, *The Helmholtz machine*, Neural Computation **7** (1995), no. 5, 889–904.

[3] T Kohonen, *Self-organisation and associative memory*, Springer-Verlag, Berlin, 1989.

[4] Y Linde, A Buzo, and R M Gray, *An algorithm for vector quantiser design*, IEEE Transactions on Communications **28** (1980), no. 1, 84–95.

[5] S P Luttrell, *Image compression using a multilayer neural network*, Pattern Recognition Letters **10** (1989), no. 1, 1–7.

[6] ———, *Self-organisation: a derivation from first principles of a class of learning algorithms*, Proceedings of IJCNN international joint conference on neural networks (Washington DC), IEEE, 1989, pp. 495–498.

[7] ———, *A hierarchical network for clutter and texture modelling*, Proceedings of SPIE conference on adaptive signal processing (San Diego), SPIE, 1991, pp. 518–528.

[8] ———, *Self-supervised training of hierarchical vector quantisers*, Proceedings of IEE conference on artificial neural networks (Bournemouth), IEE, 1991, pp. 5–9.

[9] ———, *An adaptive Bayesian network for low-level image processing*, Proceedings of IEE conference on artificial neural networks (Brighton), IEE, 1993, pp. 61–65.

[10] ———, *Maximum entropy and Bayesian methods*, ch. The cluster expansion: a hierarchical density model, pp. 269–278, Kluwer, Dordrecht, 1995.

[11] J Rissanen, *Modelling by shortest data description*, Automatica **14** (1978), no. 5, 465–471.

[12] C E Shannon, *The mathematical theory of communication*, Bell System Technical Journal **27** (1948), no. 3, 379–423.

[13] ———, *The mathematical theory of communication*, Bell System Technical Journal **27** (1948), no. 6, 623–656.

Self-Organised Factorial Encoding of a Toroidal Manifold [*]

Stephen Luttrell
Room EX21, QinetiQ, Malvern Technology Centre

It is shown analytically how a neural network can be used optimally to encode input data that is derived from a toroidal manifold. The case of a 2-layer network is considered, where the output is assumed to be a set of discrete neural firing events. The network objective function measures the average Euclidean error that occurs when the network attempts to reconstruct its input from its output. This optimisation problem is solved analytically for a toroidal input manifold, and two types of solution are obtained: a joint encoder in which the network acts as a soft vector quantiser, and a factorial encoder in which the network acts as a pair of soft vector quantisers (one for each of the circular subspaces of the torus). The factorial encoder is favoured for small network sizes when the number of observed firing events is large. Such self-organised factorial encoding may be used to restrict the size of network that is required to perform a given encoding task, and will decompose an input manifold into its constituent submanifolds.

I. INTRODUCTION

The purpose of this paper is to show analytically how a neural network can be used to optimally encode input data that is derived from a toroidal manifold. For simplicity, only the case of a 2-layer network is considered, and an objective function is defined [12] that measures the average ability of the network to reconstruct the state of its input layer from the state of its output layer. The optimum network parameter values must then minimise this objective function. In this paper the output state is chosen to be the vector of locations of a finite number of the neural firing events that arise when an input vector is presented to the network, and, in the limit of a single firing event, this reduces to a winner-take-all encoder network.

If the input vector is obtained from an arbitrary input probability density function (PDF), then the network would have to be optimised numerically, and a simple interpretation of its optimal parameters would not then be guaranteed. On the other hand, if the input PDF is constrained to have a simple enough form, then an analytic optimisation guarantees that the results can be interpreted. Because the purpose of this paper is mainly to interpret the nature of the optimal solution(s) that arise from the interplay between the input PDF and the network objective function, an analytic rather than a numerical approach will be used.

The detailed form of the optimum network parameters depends on the chosen input PDF, and, for simplicity, the input PDF will be chosen to define a curved manifold which is uniformly populated by all of the allowed input vectors. The shape of this manifold then determines the type of optimum solution that the network adopts. For instance, a 1-dimensional linear manifold with a uniform distribution of input vectors leads to an optimum solution in which each neuron fires only if the input lies within a small range of values, so the network behaves as a soft scalar quantiser. This result generalises to higher dimensional linear manifolds, where the network behaves as a soft vector quantiser. A more interesting type of optimum solution can occur when the manifold is curved. For instance, a circular manifold (which is a 1-dimensional manifold embedded in a 2-dimensional space) leads to an optimum solution that is analogous to the soft scalar quantiser obtained with a 1-dimensional linear manifold, but a toroidal manifold (which is a 2-dimensional manifold embedded in a 4-dimensional space) does not necessarily lead to an optimum solution that is analogous to the soft vector quantiser obtained with a 2-dimensional linear manifold.

For a 2-dimensional toroidal manifold, it is possible for the optimum solution to be constructed out of a pair of soft scalar quantisers, each of which encodes only one of the two circular manifolds that form the toroidal manifold. This is called a factorial encoder (because it breaks the input into its constituent factors, which it then encodes), as opposed to a joint encoder (which directly encodes the input, without first breaking it into its constituent factors). Because a factorial encoder splits up the overall encoding problem into a number of smaller encoding problems, which it then tackles in parallel, it requires fewer neurons than a joint encoder would have needed for the same encoding problem.

For the type of network objective function that is discussed in this paper, factorial encoding does not occur with linear manifolds. This is because the random nature of the neural firing events does not guarantee that at least one such event occurs in each of the soft scalar quantisers in a factorial encoder, and, for a linear manifold, this leads to a much larger average reconstruction error if a factorial encoder is used than if a joint encoder is used. This effect is summarised in figure 1 for a linear manifold, and in figure 2 for a toroidal manifold. Henceforth, only the toroidal case will be discussed, because

[*]Typeset in LaTeX on May 8, 2019.
This is archived at http://arxiv.org/abs/cs/0410036, 15 Oct 2004.
This paper was submitted to Neural Computation on 18 May 1998. Manuscript no. 1810. It was not accepted for publication, but it underpins several subsequently published papers.

it is a curved manifold which thus has interesting factorial encoding properties, whereas a linear manifold would not.

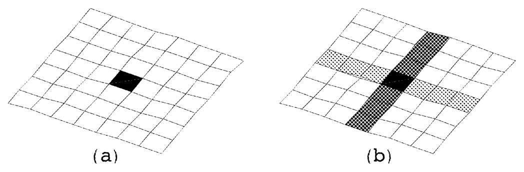


Figure 1: *Diagram (a) shows the encoding cells for joint encoding of a 2-dimensional linear manifold; a typical encoding cell is shaded. Diagram (b) shows the corresponding encoding cells for a factorial encoder; typical encoding cells for each of the two factors and their intersection are shaded. The distortion that would result from only one of the two factors is large, because the encoding cell is a long thin rectangular region.*

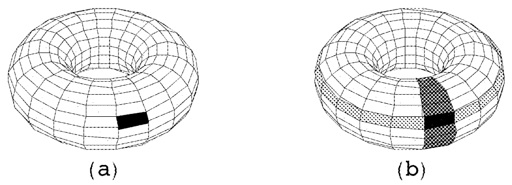


Figure 2: *Diagram (a) shows the encoding cells for joint encoding of a 2-dimensional toroidal manifold; a typical encoding cell is shaded. Diagram (b) shows the corresponding encoding cells for a factorial encoder; typical encoding cells for each of the two factors and their intersection are shaded. The distortion that would result from only one of the two factors is not as large as in the case of the corresponding linear manifold, because the long thin rectangular encoding cells are now wrapped round into loops, thus reducing the average separation (in the Euclidean sense) of points within each encoding cell.*

In figure 2(a) the torus is overlaid with a 20×20 toroidal lattice, and a typical joint encoding cell is highlighted (this would use a total of $400 = 20 \times 20$ neurons). Figure 2(a) makes clear why such encoding is described as "joint", because the response of each neuron depends on the values of both dimensions of the input. The neural network implementation of this type of joint encoder would have connections from each output neuron to all of the input neurons.

In figure 2(b) the torus is overlaid with a 20×20 toroidal lattice, and a typical pair of intersecting factorial encoding cells is highlighted (this would use a total of $40 = 20 + 20$ neurons). Figure 2(b) makes clear why such encoding is described as "factorial", because the response of each neuron depends on only one of the dimensions of the input, or, in other words, on only one factor that parameterises the input space. The neural network implementation of this type of factorial encoder would have connections from each output neuron to only half of the input neurons. In figure 2(b) an accurate encoding is obtained by a process that is akin to triangulation, in which the intersection between the 2 orthogonal encoding cells defines a region of the 2-torus that is equivalent to the corresponding joint encoding cell in figure 2(a).

For a toroidal input manifold it turns out that there is an upper limit to the number of neurons that can be used if a factorial encoder is to have a smaller average reconstruction error than the corresponding joint encoder. This limit is smaller than the number of neurons that are used in figure 2(b), so that diagram should not be interpreted too literally.

A. Vector Quantisers

The existing literature on the simplest type of encoder (i.e. the vector quantiser (VQ)) includes the following examples:

1. A standard VQ, in which the input space is partitioned into a number of non-overlapping encoding cells, which is also known as an LBG vector quantiser (after the initials of the authors of [7]). In operation, all of the input vectors that lie closest (in the Euclidean sense) to a given code vector are assigned the same code index (which thus defines an encoding cell), and the approximate reconstruction of these inputs is then the centroid of the encoding cell. This type of VQ can be viewed as a single-layer winner-take-all (WTA) neural network.

2. A topographic VQ (TVQ), in which the code indices and encoding cells are arranged so that code indices that differ by a small amount are assigned to encoding cells that are close to each other (in the Euclidean sense). This topographic property automatically emerges if a VQ is optimised for encoding input vectors to be transmitted along a noisy communication channel [1, 3, 6, 8]. The Kohonen topographic mapping network [5] is an approximation to this type of encoder, as was explained in [8]. The TVQ may be generalised to a soft TVQ (STVQ) in which each code index is chosen probabilistically in response to the corresponding input vector [4, 9].

3. Simultaneously use more than one standard VQ, with each VQ encoding only a subspace of the input (see for example [13]); in effect, more than one code index is used to encode the input vector. By this means, a high-dimensional space can be split up into a number of lower dimensional pieces. This type of VQ is equivalent to multiple single-layer WTA neural network modules, each of which operates on a subspace of the input. This is an example of a factorial encoder, in which the input is split into a number of separate parts, or factors.

4. The simultaneous use of multiple VQs can be extended to a tree-like network of VQs [11]. This

type of VQ is equivalent to multiple single layer WTA neural network modules which are connected together in a tree-like network of modules.

For simplicity, only the case of a 2-layer network (i.e. an input and an output layer) will be considered, but otherwise the network will be obliged to learn how to make use of all of its neurons. The simplest encoder which has all of the required behaviour, and which includes the above 2-layer examples as special cases, is one in which the neurons fire discretely in response to the input, and, after a finite number of firing events has occurred, the input is then reconstructed as accurately as possible (in the Euclidean sense). In the special case where only a single firing event is observed, this reduces to a standard LBG vector quantiser that was discussed in case 1 above. In the more general case, where a finite number of firing events is observed, this can lead to factorial encoder networks of the type that was discussed in case 3 above.

B. Curved Manifolds

The purpose of this paper is to derive optimal ways of encoding data using neural networks in which multiple firing events are observed, and to show that factorial encoder networks can be optimal when the input data lies on a curved manifold. In order to get a feel for how curved manifolds arise in image data, consider the examples shown in figure 3 and figure 4, which show the manifold generated by a single target (figure 3) and by a pair of targets (figure 4), when projected onto three neighbouring pixels (i.e. the locus of the 3-vector formed from these pixel values is plotted as the target(s) move around).

Clearly, these image manifolds are curved, and the curvature gets greater the narrower the Gaussian profiles used to generate the target images become.

It is not at all obvious how best to encode vectors that lie on such manifolds. For instance, one might try to tile the manifold with a large number of small encoding cells obtained from some variant of a VQ, or one might try to project the manifold onto a basis obtained from some variant of principal components analysis (PCA). In fact, these two examples are both special cases of the approach that is advocated in this paper; a VQ corresponds to a single firing event, whereas PCA corresponds to an infinite number of firing events.

The problem of optimally encoding data that is derived from a general curved manifold requires a numerical solution. However, in order to develop our understanding, it is best to start with an analytically tractable example based on a simple curved manifold, which is carefully selected to preserve the essential features of more general curved manifolds. With this in mind, the most important feature to preserve in the analytic example is curvature. A circle is the simplest 1-dimensional curved manifold, which may then be used to construct higher dimensional toroidal manifolds. For instance, a pair of circles may

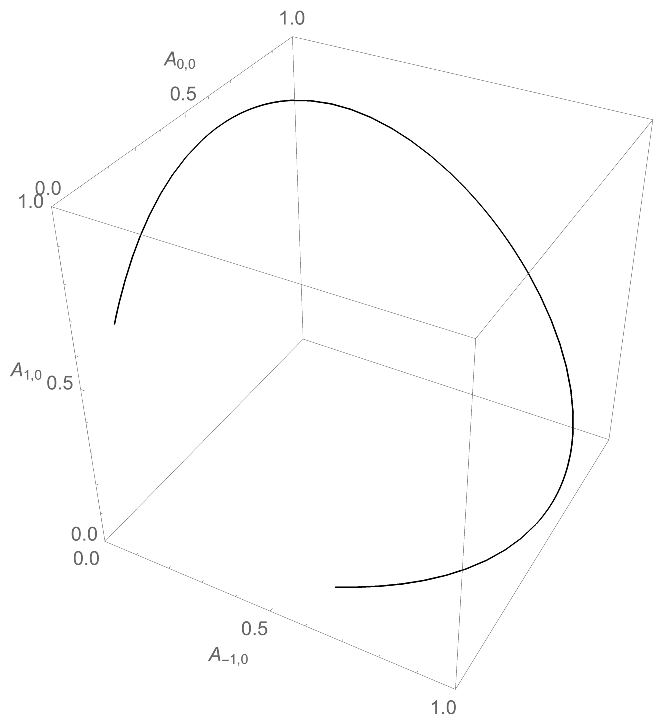


Figure 3: *Manifold formed when the 1-dimensional image of a target (a Gaussian profile with a half-width of one pixel) is moved around. Only the projection $A_{i,j}$ onto the pixels at $(i,j) = (-1,0)$, $(0,0)$ and $(1,0)$ is shown.*

be used to construct the 2-dimensional toroidal manifold shown in figure 2. It turns out that, if a toroidal manifold is used, then the network objective function can be analytically minimised to yield results that exhibit interesting joint encoder and factorial encoder properties.

C. Structure of this Paper

In section II the basic theoretical framework is introduced, from which some expressions are derived for optimising a network which is trained on data from a toroidal input manifold. In section III the detailed results for encoding a circular input manifold are given (which are trivially related to the corresponding results for the case of joint encoding of a 2-torus), and in section IV these results are extended to the case of factorial encoding of a 2-torus. The results for joint encoding and factorial encoding are compared in section V. Some useful asymptotic approximations are discussed in section VI, and a useful approximation to the optimal network is discussed in section VII.

The main steps in the derivations are reported in the appendices to this paper, and in several cases there is a considerable amount of algebra involved, which was done using algebraic manipulator software [17].

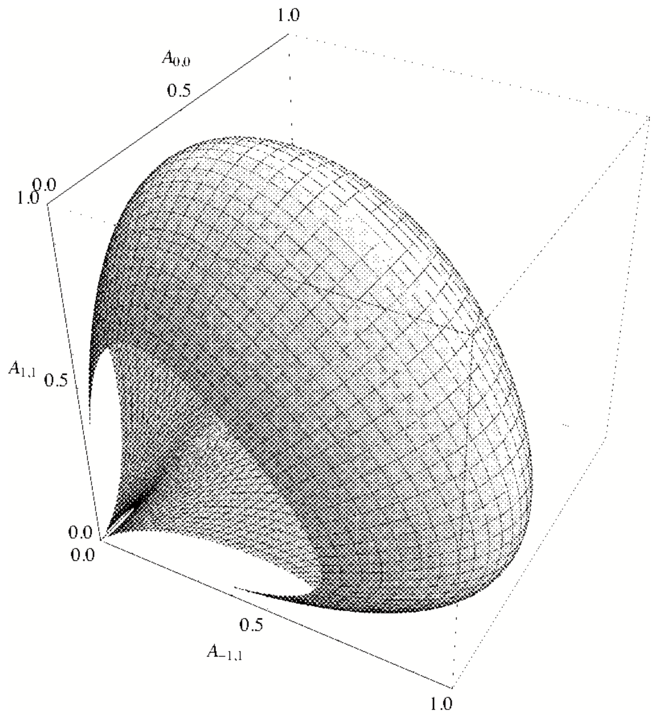


Figure 4: *Manifold formed when the 2-dimensional image of a target (a Gaussian profile with half-widths of one pixel in each direction) is moved around. Only the projection $A_{i,j}$ onto the pixels at $(i,j) = (-1,1)$, $(0,0)$ and $(1,1)$ is shown.*

II. BASIC THEORETICAL FRAMEWORK

The encoder model that is assumed throughout this paper is a 2-layer network of neurons. The state of the input layer is denoted as an input vector $\mathbf{x}$ (which is assumed in this paper to be a continuous activity pattern), and the state of the output layer is denoted as the output vector $\mathbf{y}$ (which is assumed in this paper to be a discrete pattern of firing events). The information content of the output state $\mathbf{y}$ may be used to draw inferences about the input state $\mathbf{x}$. This can be formalised by using Bayes' theorem in the form

$$\Pr(\mathbf{x}|\mathbf{y}) = \frac{\Pr(\mathbf{y}|\mathbf{x})\ \Pr(\mathbf{x})}{\int d\mathbf{x}'\ \Pr(\mathbf{y}|\mathbf{x}')\ \Pr(\mathbf{x}')} \tag{2.1}$$

where the PDF $\Pr(\mathbf{x}|\mathbf{y})$ of the input $\mathbf{x}$ given that the output $\mathbf{y}$ is known (i.e. the generative model) is completely determined by two quantities: the likelihood $\Pr(\mathbf{y}|\mathbf{x})$ that output $\mathbf{y}$ occurs when input $\mathbf{x}$ is present (i.e. the recognition model), and the prior PDF $\Pr(\mathbf{x})$ that input $\mathbf{x}$ could occur irrespective of whether $\mathbf{y}$ is being observed. However, for all but the most trivial situations, if the functional form of $\Pr(\mathbf{y}|\mathbf{x})$ is simple then the functional form of $\Pr(\mathbf{x}|\mathbf{y})$ is complicated (or vice versa, with the roles of $\Pr(\mathbf{y}|\mathbf{x})$ and $\Pr(\mathbf{x}|\mathbf{y})$ interchanged). In other words, if the recognition and generative models are strictly related by Bayes' theorem, then difficulties inevitably arise in analytic and numerical calculations.

A possible way around this problem is to use a network objective function D_0 that has a simple functional form for the $\Pr(\mathbf{y}|\mathbf{x})$, but has an approximation to the ideal $\Pr(\mathbf{x}|\mathbf{y})$ implied by Bayes' theorem (or vice versa). A convenient choice is

$$\begin{aligned} D_0 &\equiv -\int d\mathbf{x} \sum_{\mathbf{y}} \Pr(\mathbf{x},\mathbf{y})\ \log Q(\mathbf{x},\mathbf{y}) \\ &= -\int d\mathbf{x}\ \Pr(\mathbf{x}) \sum_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x})\ \log Q(\mathbf{x}|\mathbf{y}) - \sum_{\mathbf{y}} \Pr(\mathbf{y})\ \log Q(\mathbf{y}) \end{aligned} \tag{2.2}$$

$\Pr(\mathbf{x},\mathbf{y})$ is a joint probability that satisfies $\Pr(\mathbf{x},\mathbf{y}) = \Pr(\mathbf{y}|\mathbf{x})\ \Pr(\mathbf{x}) = \Pr(\mathbf{x}|\mathbf{y})\ \Pr(\mathbf{y})$ (i.e. Bayes' theorem holds), $Q(\mathbf{x},\mathbf{y})$ is an approximation to $\Pr(\mathbf{x},\mathbf{y})$ that satisfies the corresponding relationships $Q(\mathbf{x},\mathbf{y}) = Q(\mathbf{y}|\mathbf{x})\ Q(\mathbf{x}) = Q(\mathbf{x}|\mathbf{y})\ Q(\mathbf{y})$, $\int d\mathbf{x}\ \Pr(\mathbf{x})\ (\cdots)$ integrates over all the possible states of the input layer, $\sum_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x})\ (\cdots)$ sums over all the possible states of the output layer given that the state of the input layer is known, and $\sum_{\mathbf{y}} \Pr(\mathbf{y})\ (\cdots)$ sums over all the possible states of the output layer.

The objective function D_0 measures the average number of bits required when the approximate joint probability $Q(\mathbf{x},\mathbf{y})$ is used as a reference to encode each pair $(\mathbf{x},\mathbf{y})$ drawn randomly from the true joint probability $\Pr(\mathbf{x},\mathbf{y})$ [15], so D_0 belongs to the class of minimum description length (MDL) objective functions [14]. Strictly speaking, the number of bits depends on the accuracy with which the continuous-valued $\mathbf{x}$ is measured. However, this refinement is omitted from equation 2.2 because it does not affect the results in this paper, provided that the size of the quantisation cells into which $\mathbf{x}$ is binned is much smaller than the scale on which $\Pr(\mathbf{x}|\mathbf{y})$ and $Q(\mathbf{x}|\mathbf{y})$ fluctuate.

The objective function D_0 can be simplified if $Q(\mathbf{x},\mathbf{y})$

is assumed to have the following properties

$$\begin{aligned} Q(\mathbf{y}) &= \text{constant} \\ Q(\mathbf{x}|\mathbf{y}) &= \frac{1}{\left(\sqrt{2\pi}\,\sigma\right)^{\dim \mathbf{x}}} \exp\left(-\frac{\left\|\mathbf{x}-\mathbf{x}'(\mathbf{y})\right\|^2}{2\sigma^2}\right) \end{aligned} \tag{2.3}$$

where the approximation $Q(\mathbf{x}|\mathbf{y})$ to the true generative model $\Pr(\mathbf{x}|\mathbf{y})$ is a Gaussian PDF, and the prior probabilities $Q(\mathbf{y})$ are constrained to all be equal. If the value of σ is fixed, then D_0 may be replaced by the simpler, but equivalent, vector quantiser objective function D_{VQ}, which is defined as

$$D_{VQ} \equiv \int d\mathbf{x}\, \Pr(\mathbf{x}) \sum_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x})\, \left\|\mathbf{x}-\mathbf{x}'(\mathbf{y})\right\|^2 \tag{2.4}$$

where $\sum_{\mathbf{y}} \Pr(\mathbf{y}) = 1$ has been used to eliminate the $\sum_{\mathbf{y}} \Pr(\mathbf{y}) \log Q(\mathbf{y})$ term. This measures the average Euclidean distortion that occurs when the input $\mathbf{x}$ is probabilistically encoded as $\mathbf{y}$, and then subsequently reconstructed as $\mathbf{x}'(\mathbf{y})$. This is a soft version of the LBG vector quantiser objective function [7], in which $\mathbf{y}$ acts as a code index, $\Pr(\mathbf{y}|\mathbf{x})$ acts a soft encoding prescription for probabilistically transforming $\mathbf{x}$ into $\mathbf{y}$, and $\mathbf{x}'(\mathbf{y})$ acts as the corresponding code vector. The optimal $\Pr(\mathbf{y}|\mathbf{x})$ that minimises D_{VQ} is deterministic (i.e. each $\mathbf{x}$ is transformed to one, and only one, $\mathbf{y}$), so D_{VQ} actually leads to an LBG vector quantiser itself, rather than merely a probabilistic version thereof [9].

Under the same assumptions (see equation 2.3) that yielded the expression for D_{VQ}, the Helmholtz machine objective function [2] would reduce to

$$D_{HM} = D_{VQ} + \int d\mathbf{x}\, \Pr(\mathbf{x}) \sum_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x}) \log \Pr(\mathbf{y}|\mathbf{x}) \tag{2.5}$$

where the extra term is the so-called "bits-back" term, which is (minus) the entropy of the output $\mathbf{y}$ given that the input $\mathbf{x}$ is known, then averaged over all inputs. Thus D_{HM} does not directly penalise $\Pr(\mathbf{y}|\mathbf{x})$ that have a large entropy, or, in other words, it allows the recognition model $\Pr(\mathbf{y}|\mathbf{x})$ to be such that many output states $\mathbf{y}$ are permitted once the input state $\mathbf{x}$ is known. This means that the recognition models produced by a Helmholtz machine tend to be more stochastic than they would have been had the "bits-back" term been omitted from D_{HM}. Conversely, the objective function D_{VQ} that is used in this paper directly penalises $\Pr(\mathbf{y}|\mathbf{x})$ that have a large entropy, so the recognition models produced tend to be more deterministic than the stochastic ones that the Helmholtz machine would produce under equivalent circumstances. Thus using D_{VQ} tends to lead to sparse codes in which few neurons can fire, whereas using D_{HM} tends to lead to distributed codes in which many neurons can fire.

The chosen objective function has both an information theoretic interpretation (given by D_0 in equation 2.2), in which it seeks to minimise the number of bits required to encode $\Pr(\mathbf{x},\mathbf{y})$, and also an encoder/decoder interpretation (given by D_{VQ} in equation 2.4), in which it seeks to minimise the Euclidean distortion that arises when $\mathbf{x}$ is encoded as $\mathbf{y}$ and then subsequently reconstructed as $\mathbf{x}'(\mathbf{y})$. Also, using D_{VQ} as the network objective function ensures backward compatibility with preexisting results (e.g. [5, 7]).

An upper bound on the network objective function is introduced in section II A, and the stationarity conditions which must be satisfied for an optimal network behaviour are derived in section II B. Joint encoding on a 2-torus is discussed in section II C, and factorial encoding on a 2-torus is discussed in section II D.

A. Objective Function

In order to make progress it is necessary to make some assumptions about the network output state $\mathbf{y}$. Thus the output layer will be assumed to consist of M neurons that fire discretely in response to the input activity pattern $\mathbf{x}$. Furthermore, $\mathbf{y}$ will be assumed to be an n-dimensional vector, that consists of the observations of the locations $(y_1, y_2, \cdots, y_n)$ of the first n firing events that occur in response to input $\mathbf{x}$ (this is described in detail in [12]). Note that the individual y_i are scalars, but the generalisation to vector-valued $\mathbf{y}_i$ is straightforward.

For compatibility with results published earlier (e.g. [9, 12]), the objective function that will be used here is $D = 2D_{VQ}$, which has an upper bound $D_1 + D_2$ given by (see appendix A for a detailed derivation and discussion)

$$\begin{aligned} D_1 &\equiv \frac{2}{n} \int d\mathbf{x}\, \Pr(\mathbf{x}) \sum_{y=1}^{M} \Pr(y|\mathbf{x})\, \left\|\mathbf{x}-\mathbf{x}'(y)\right\|^2 \\ D_2 &\equiv \frac{2(n-1)}{n} \int d\mathbf{x}\, \Pr(\mathbf{x}) \left\|\mathbf{x} - \sum_{y=1}^{M} \Pr(y|\mathbf{x})\, \mathbf{x}'(y)\right\|^2 \end{aligned} \tag{2.6}$$

where $\Pr(y|\mathbf{x})$ is the probability that neuron y fires first in response to input $\mathbf{x}$, and $\mathbf{x}'(y)$ is a reference vector that is used by neuron y in its attempt to approximately reconstruct the input. In the limit $n = 1$ only D_1 contributes, and a standard LBG vector quantiser emerges when D_1 is minimised. As $n \to \infty$ only D_2 contributes, and a PCA encoder emerges when D_2 is minimised.

This upper bound $D_1 + D_2$ on the objective function D will be used to derive all of the results in this paper. Its functional form, in which $\Pr(y|\mathbf{x})$ appears only quadratically (unlike in equation 2.5 for D_{HM}), allows analytic results to be readily derived.

B. Stationarity Conditions

The upper bound $D_1 + D_2$ (see equation 2.6) on the objective function $D = 2D_{VQ}$ (see equation 2.4) needs to be minimised with respect to two types of parameter: posterior probabilities $\Pr(y|\mathbf{x})$ and reference vectors $\mathbf{x}'(y)$.

This could be done numerically for an arbitrary input PDF $\Pr(\mathbf{x})$ by using a gradient descent type of algorithm [12], but here $D_1 + D_2$ will be analytically minimised for some carefully chosen special cases of $\Pr(\mathbf{x})$.

The stationarity condition $\frac{\partial(D_1+D_2)}{\partial \mathbf{x}'(y)} = 0$ gives (see appendix B 1)

$$n \int d\mathbf{x} \Pr(\mathbf{x}|y)\ \mathbf{x} = \mathbf{x}'(y) + (n-1) \int d\mathbf{x} \Pr(\mathbf{x}|y) \sum_{y'=1}^{M} \Pr(y'|\mathbf{x})\ \mathbf{x}'(y') \tag{2.7}$$

where $\Pr(y) > 0$ has been assumed. The $\frac{\partial(D_1+D_2)}{\partial \mathbf{x}'(y)} = 0$ stationarity condition also has the solution $\Pr(y) = 0$, but this solution may be discarded because $\Pr(y) > 0$ is always the case in practice. The right hand side of the stationarity condition in equation 2.7 has two contributions: a D_1-like contribution which is a single reference vector $\mathbf{x}'(y)$, plus a D_2-like contribution which is $n-1$ times a sum of reference vectors $\sum_{y'=1}^{M} \left(\int d\mathbf{x} \Pr(y'|\mathbf{x}) \Pr(\mathbf{x}|y)\right) \mathbf{x}'(y')$, where the coefficient $\int d\mathbf{x} \Pr(y'|\mathbf{x}) \Pr(\mathbf{x}|y)$ accounts for the effect (at neuron y) of observing all pairs of firing events (y, y') for $y' = 1, 2, \cdots, M$. The sum of these two terms is n times the total reference vector that is effectively associated with neuron y, which is n times $\int d\mathbf{x} \Pr(\mathbf{x}|y)\ \mathbf{x}$ as given on the left hand side of equation 2.7.

The stationarity condition $\frac{\delta(D_1+D_2)}{\delta \log \Pr(y|\mathbf{x})} = 0$ gives (see appendix B 2)

$$\sum_{y'=1}^{M} \left(\Pr(y'|\mathbf{x}) - \delta_{y,y'}\right) \mathbf{x}'(y') \cdot \left(\frac{1}{2} \mathbf{x}'(y') - n\,\mathbf{x} + (n-1) \sum_{y''=1}^{M} \Pr(y''|\mathbf{x})\ \mathbf{x}'(y'') \right) = 0 \tag{2.8}$$

where the constraint $\sum_{y'=1}^{M} \Pr(y'|\mathbf{x}) = 1$ has been imposed, and $\Pr(\mathbf{x}) > 0$ and $\Pr(y|\mathbf{x}) > 0$ have been assumed. The $\frac{\delta(D_1+D_2)}{\delta \log \Pr(y|\mathbf{x})} = 0$ stationarity condition also has two other solutions: either $\Pr(\mathbf{x}) = 0$, or $\Pr(\mathbf{x}) > 0$ and $\Pr(y|\mathbf{x}) = 0$. Using the normalisation constraint $\sum_{y=1}^{M} \Pr(y|\mathbf{x}) = 1$, the last of these solutions ensures that $\Pr(y'|\mathbf{x}) \leq 1$ for $y' \neq y$, and when all values of y are considered the net effect is to constrain $\Pr(y|\mathbf{x})$ to the interval $0 \leq \Pr(y|\mathbf{x}) \leq 1$, as expected.

The solutions of the stationarity condition for $\Pr(y|\mathbf{x})$ in equation 2.8 are piecewise linear functions of $\mathbf{x}$. This piecewise linear property of $\Pr(y|\mathbf{x})$ (as discussed in appendix B 2) is an enormous simplification, because it means that rather than searching the infinite dimensional space of functions $\Pr(y|\mathbf{x})$ for the optimal ones that minimise $D_1 + D_2$, one needs only search a finite dimensional space of piecewise linear functions $\Pr(y|\mathbf{x})$ (subject to the constraints $0 \leq \Pr(y|\mathbf{x}) \leq 1$ and $\sum_{y=1}^{M} \Pr(y|\mathbf{x}) = 1$).

C. Joint Encoding

Joint encoding, as shown in figure 2(a), is characterised by a $\Pr(y|\mathbf{x})$ in which the neurons labelled by y form a discretised version of the manifold that $\mathbf{x}$ lives on. For instance, when $\mathbf{x}$ lives on a 2-torus, so that $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ where $\mathbf{x}_1 = (\cos\theta_1, \sin\theta_1)$ and $\mathbf{x}_2 = (\cos\theta_2, \sin\theta_2)$, where $0 \leq \theta_1 < 2\pi$ and $0 \leq \theta_2 < 2\pi$, the $\Pr(y|\mathbf{x})$ typically behave as shown in figure 2(a), where the 2-torus is tiled with encoding cells. When $n > 1$ neighbouring encoding cells overlap, so figure 2(a) does not then give an accurate representation of the encoding cells.

For joint encoding of a 2-torus, y must be replaced by the pair (y_1, y_2), where the y_1 index labels one direction around the toroidal lattice, and y_2 labels the other direction (this notation must not be confused with the $(y_1, y_2, \cdots, y_n)$ notation that was used in section II A). Thus $\Pr(y|\mathbf{x}) \to \Pr(y_1, y_2|\mathbf{x}_1, \mathbf{x}_2)$ with $1 \leq y_1 \leq \sqrt{M}$ and $1 \leq y_2 \leq \sqrt{M}$. For simplicity, assume $\Pr(\mathbf{x}_1, \mathbf{x}_2) = \Pr(\mathbf{x}_1) \Pr(\mathbf{x}_2)$, where $\Pr(\mathbf{x}_1)$ and $\Pr(\mathbf{x}_2)$ each define a uniform PDF on the input manifold. The following results for D_1 and D_2 may then be derived (see appendix C 1)

$$
\begin{aligned}
D_1 &= \frac{4}{n}\int d\mathbf{x}_1\,\Pr(\mathbf{x}_1)\sum_{y_1=1}^{\sqrt{M}}\Pr(y_1|\mathbf{x}_1)\,\left\|\mathbf{x}_1-\mathbf{x}_1'(y_1)\right\|^2 \\
D_2 &= \frac{4(n-1)}{n}\int d\mathbf{x}_1\,\Pr(\mathbf{x}_1)\left\|\mathbf{x}_1-\sum_{y_1=1}^{\sqrt{M}}\Pr(y_1|\mathbf{x}_1)\,\mathbf{x}_1'(y_1)\right\|^2
\end{aligned}
\qquad (2.9)
$$

These results for D_1 and D_2 show that, under the simplifying assumptions made above, the problem of optimising a joint encoder is equivalent to the problem of optimising an encoder for $\mathbf{x}_1$ alone (with the replacement $M \to \sqrt{M}$), and then multiplying the value of $D_1 + D_2$ by a factor 2 to account for $\mathbf{x}_2$ as well. This illustration of the behaviour of joint encoder posterior probabilities in the case of $\Pr(y_1, y_2|\mathbf{x}_1, \mathbf{x}_2)$ may readily be generalised to higher dimensions.

D. Factorial Encoding

Factorial encoding, as shown in figure 2(b), is characterised by a $\Pr(y|\mathbf{x})$ in which the neurons labelled by y are partitioned into a number of subsets, each of which forms a discretised version of a subspace of the manifold that $\mathbf{x}$ lives on. For instance, when $\mathbf{x}$ lives on a 2-torus, and the neurons are partitioned into two equal-sized subsets, the $\Pr(y|\mathbf{x})$ typically behave as shown in figure 2(b), where each of the two circular subspaces within the 2-torus is tiled with encoding cells, which overlap when $n > 1$.

For factorial encoding of a 2-torus $\Pr(y|\mathbf{x}) = \Pr(y|\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{2}\Pr(y|\mathbf{x}_1) + \frac{1}{2}\Pr(y|\mathbf{x}_2)$, where $\sum_{y=1}^{\frac{M}{2}}\Pr(y|\mathbf{x}_1) = 1$, $\sum_{y=\frac{M}{2}+1}^{M}\Pr(y|\mathbf{x}_2) = 1$, $\Pr(y|\mathbf{x}_1) = 0$ for $\frac{M}{2} + 1 \le y \le M$, and $\Pr(y|\mathbf{x}_2) = 0$ for $1 \le y \le \frac{M}{2}$. For simplicity, assume $\Pr(\mathbf{x}_1, \mathbf{x}_2) = \Pr(\mathbf{x}_1)\Pr(\mathbf{x}_2)$, where $\Pr(\mathbf{x}_1)$ and $\Pr(\mathbf{x}_2)$ each define a uniform PDF on the input manifold. The following results for D_1 and D_2 may then be derived (see appendix C 2)

$$
\begin{aligned}
D_1 &= \frac{2}{n}\left(\int d\mathbf{x}_1\,\Pr(\mathbf{x}_1)\sum_{y=1}^{\frac{M}{2}}\Pr(y|\mathbf{x}_1)\,\left\|\mathbf{x}_1-\mathbf{x}_1'(y)\right\|^2+\int d\mathbf{x}_2\,\Pr(\mathbf{x}_2)\,\left\|\mathbf{x}_2\right\|^2\right) \\
D_2 &= \frac{4(n-1)}{n}\int d\mathbf{x}_1\,\Pr(\mathbf{x}_1)\left\|\mathbf{x}_1-\frac{1}{2}\sum_{y=1}^{\frac{M}{2}}\Pr(y|\mathbf{x}_1)\,\mathbf{x}_1'(y)\right\|^2
\end{aligned}
\qquad (2.10)
$$

These results for D_1 and D_2 show that, under the simplifying assumptions made above, the problem of optimising a factorial encoder is closely related to the problem of optimising two 1-dimensional encoders. This illustration of the behaviour of factorial encoder posterior probabilities in the case of $\Pr(y|\mathbf{x}_1, \mathbf{x}_2)$ may readily be generalised to higher dimensions.

III. CIRCULAR MANIFOLD

The analysis of how to encode data that lives on a curved manifold begins with the case of data that lives on a circle. In particular, assume that the input vector $\mathbf{x}$ is uniformly distributed on the unit circle centred on the origin, so that $\mathbf{x}$ can be parameterised by a single angular variable θ, thus

$$
\begin{aligned}
\mathbf{x} &= (\cos\theta, \sin\theta) \\
\int d\mathbf{x}\,\Pr(\mathbf{x})\,(\cdots) &= \frac{1}{2\pi}\int_0^{2\pi} d\theta\,(\cdots)
\end{aligned}
\qquad (3.1)
$$

The posterior probability $\Pr(y|\mathbf{x})$ may thus be replaced by $\Pr(y|\theta)$, and for purely conventional reasons, the range of y is now chosen to be $y = 0, 1, \cdots, M-1$ rather than $y = 1, 2, \cdots, M$. The set of M posterior probabilities for $y = 0, 1, \cdots, M-1$ can be parameterised as

$$\Pr(y|\theta) = p\left(\theta - \frac{2\pi y}{M}\right) \qquad (3.2)$$

where $p(\theta)$ is the θ-dependence of the posterior probability associated with the $y = 0$ neuron. The θ-dependence of $p(\theta)$ must be piecewise sinusoidal (i.e.

made out of pieces that each have the functional form $a + b\cos\theta + c\sin\theta$) in order to ensure that $\Pr(y|\mathbf{x})$ is piecewise linear, as is required of solutions to equation B4. Similarly, the M corresponding reference vectors can be parameterised as

$$\mathbf{x}'(y) = r\left(\cos\left(\frac{2\pi y}{M}\right), \sin\left(\frac{2\pi y}{M}\right)\right) \tag{3.3}$$

which all have length r, and thus form a regular M-sided polygon.

It turns out that, for input vectors that live on a circular manifold, optimal joint encoding never causes more than 3 different neurons to fire in response to a given input (i.e. no more than 3 posterior probabilities overlap in input space). This severely limits the number of different piecewise functions that have to be manipulated when solving the $D_1 + D_2$ minimisation problem for input vectors that live on a circle. An analogous simplification also holds for joint and factorial encoding of a 2-torus. The case of 2 overlapping posterior probabilities can be optimised without too much difficulty, but the case of 3 overlapping posterior probabilities involves a prohibitively large amount of algebra, for which it is convenient to use an algebraic manipulator [17]. The calculations turn out to be highly structured, so the use of an algebraic manipulator could in principle be used to solve even more complicated analytic problems.

All of the results for encoding input data that lives on a circular manifold may be derived from the expression for $D_1 + D_2$ in equation 2.6 (and the corresponding stationarity conditions), with the replacement given in equation 3.1 to ensure that the input manifold corresponds to a uniform distribution of data around a unit circle, and the functional forms given in equation 3.2 and equation 3.3.

The corresponding results for joint encoding of data that lives on a 2-torus can be obtained directly from these results (see section II C). The expression for the minimum value of $D_1 + D_2$ for joint encoding a 2-torus using $\sqrt{M} \times \sqrt{M}$ neurons is obtained by making the replacement $M \to \sqrt{M}$ in the expression for the minimum value of $D_1 + D_2$ for encoding a circle using M neurons, and then multiplying this result by 2 in order to account for both the circles that form the 2-torus (see equation 2.9).

A. Two Overlapping Posterior Probabilities

A detailed derivation of the results reported in this section is given in appendix D 1. Because the neurons have an angular separation of $\frac{2\pi}{M}$ (see the form of the posterior probability given in equation 3.2), the functional form of $p(\theta)$ may be defined as

$$p(\theta) = \begin{cases} 1 & 0 \le |\theta| \le \frac{\pi}{M} - s \\ f(\theta) & \frac{\pi}{M} - s \le |\theta| \le \frac{\pi}{M} + s \\ 0 & |\theta| \ge \frac{\pi}{M} + s \end{cases} \tag{3.4}$$

where the s parameter is half the angular width of the overlap between the posterior probabilities of adjacent neurons on the unit circle, in which case $0 \le s \le \frac{\pi}{M}$ ensures that no more than two neurons can respond to a given input. Anticipating the optimum solution, a typical example of this type of posterior probability is shown in figure 5.

In order to guarantee that $\Pr(y|\mathbf{x})$ has a piecewise linear dependence on $\mathbf{x}$, as is required of solutions of equation 2.8, $f(\theta)$ must have the sinusoidal dependence $f(\theta) = a + b\cos\theta + c\sin|\theta|$, where the use of $|\theta|$ arises because $p(\theta) = p(-\theta)$. Note that the $\Pr(\mathbf{x}) = 0$ solution to the stationarity condition on $\Pr(y|\mathbf{x})$ (see equation 2.8) implies that $\Pr(y|\mathbf{x})$ is undefined for any $\mathbf{x}$ that does not lie on the unit circle. However, for those $\mathbf{x}$ that do lie on the unit circle, the a, b and c parameters can be determined by demanding continuity of $p(\theta)$ at the ends of its piecewise intervals (i.e. at $\theta = \frac{\pi}{M} - s$ and $\theta = \frac{\pi}{M} + s$), and by demanding that the total probability of any neuron firing first is unity (i.e. the total posterior probability is normalised such that $f(\theta) + f\left(\frac{2\pi}{M} - \theta\right) = 1$ in the interval $\frac{\pi}{M} - s \le \theta \le \frac{\pi}{M} + s$), to obtain

$$f(\theta) = \frac{1}{2} + \frac{1}{2}\frac{\sin\left(\frac{\pi}{M} - \theta\right)}{\sin s} \tag{3.5}$$

This corresponds to a piecewise linear contribution to $\Pr(y|\mathbf{x})$ whose gradient points in the $\left(-\sin\left(\frac{\pi}{M}\right), \cos\left(\frac{\pi}{M}\right)\right)$ direction. A typical example of this type of posterior probability is shown in figure 5.

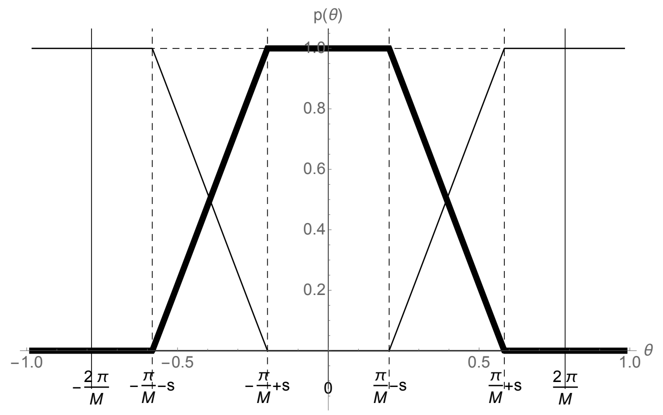


Figure 5: *Plot of the optimal neural posterior probability $p(\theta)$ for $M = 8$ and $n = 2$. The neighbouring posterior probabilities $p\left(\theta \pm \frac{2\pi}{M}\right)$ are also plotted. The optimal value of s is $s \approx 0.49\frac{\pi}{M}$. The departure of $p(\theta)$ from linearity in the interval $\frac{\pi}{M} - s \le \theta \le \frac{\pi}{M} + s$ is too small to be easily seen.*

Without loss of generality (because the solution is symmetric under rotations of θ which are multiples of $\frac{2\pi}{M}$) set $y = 0$ in equation 2.8, to obtain in the interval $\frac{\pi}{M} - s \le \theta \le \frac{\pi}{M} + s$

$$0 = r\ \csc^2 s\ \sin\left(\frac{\pi}{M}\right)\ \sin\left(\frac{\pi}{M} - \theta\right)\ \left(\sin s - \sin\left(\frac{\pi}{M} - \theta\right)\right)\ \left(n\ \sin s - (n-1)\ r\ \sin\left(\frac{\pi}{M}\right)\right) \tag{3.6}$$

which may be solved for the optimum length r of the reference vectors, to yield

$$r = \frac{n}{n-1}\ \frac{\sin s}{\sin\left(\frac{\pi}{M}\right)} \tag{3.7}$$

Set $y = 0$ in equation 2.7 to obtain a transcendental equation that must be satisfied by the optimum s

$$\frac{\sin s}{\sin\left(\frac{\pi}{M}\right)} - \frac{n-1}{n}\ \frac{M}{\pi}\ \sin\left(\frac{\pi}{M}\right)\ (\cos s + s\ \sin s) = 0 \tag{3.8}$$

The symmetry of the solution may be used to make the replacement $\frac{1}{2\pi}\int_0^{2\pi} d\theta\ (\cdots) \to \frac{M}{\pi}\int_0^{\frac{\pi}{M}} d\theta\ (\cdots)$ in the expressions for D_1 and D_2, which may then be evaluated and simplified to yield the minimum $D_1 + D_2$ as

$$D_1 + D_2 = 2 - \frac{n}{n-1}\ \frac{M}{2\pi}\ (2s + \sin(2s)) \tag{3.9}$$

The value of s which should be used in this expression for $D_1 + D_2$ is the solution of equation 3.8 for the chosen values of M and n.

Note that the expression for r in equation 3.7 and the expression for $D_1 + D_2$ in equation 3.9 both have a finite limits as $n \to 1$, because the limiting behaviour of the solution s of equation 3.8 is $s \to (n-1)\ \frac{M}{\pi}\ \sin^2\left(\frac{\pi}{M}\right)$ (see the asymptotic results in section VI), which contains a factor $n-1$ to cancel the $\frac{1}{n-1}$ factor that appears in both equation 3.7 and equation 3.9.

B. Three Overlapping Posterior Probabilities

A detailed derivation of the results reported in this section is given in appendix D 2. Because the neurons have an angular separation of $\frac{2\pi}{M}$, the functional form of $p(\theta)$ may be defined as

$$p(\theta) = \begin{cases} f_1(\theta) & 0 \le |\theta| \le -\frac{\pi}{M} + s \\ f_2(\theta) & -\frac{\pi}{M} + s \le |\theta| \le \frac{3\pi}{M} - s \\ f_3(\theta) & \frac{3\pi}{M} - s \le |\theta| \le \frac{\pi}{M} + s \\ 0 & |\theta| \ge \frac{\pi}{M} + s \end{cases} \tag{3.10}$$

where the s parameter is half the angular width of the overlap between the posterior probabilities of adjacent neurons on the unit circle, in which case $\frac{\pi}{M} \le s \le \frac{2\pi}{M}$ ensures that no more than 3 neurons can respond to a given input. Anticipating the optimum solution, a typical example of this type of posterior probability is shown in figure 6.

In order to guarantee that $\Pr(y|\mathbf{x})$ has a piecewise linear dependence on $\mathbf{x}$, the $f_i(\theta)$ must have the sinusoidal dependence $f_i(\theta) = a_i + b_i \cos\theta + c_i \sin|\theta|$ for $i = 1, 2, 3$. For those $\mathbf{x}$ that lie on the unit circle, the a_i, b_i and c_i parameters can be determined by imposing continuity of $p(\theta)$ at $\theta = -\frac{\pi}{M} + s$, $\theta = \frac{3\pi}{M} - s$ and $\theta = \frac{\pi}{M} + s$, and normalisation of the total posterior probability such that $f_1(\theta) + f_3\left(\frac{2\pi}{M} + \theta\right) + f_3\left(\frac{2\pi}{M} - \theta\right) = 1$ in the interval $0 \le \theta \le -\frac{\pi}{M} + s$, and $f_2(\theta) + f_2\left(\frac{2\pi}{M} - \theta\right) = 1$ in the interval $-\frac{\pi}{M} + s \le \theta \le \frac{3\pi}{M} - s$. Also, to satisfy the stationarity conditions, set $y = 0$ in equation 2.7, and also set $y = 0$ in equation 2.8 in each of the intervals $0 \le \theta \le -\frac{\pi}{M} + s$, $-\frac{\pi}{M} + s \le \theta \le \frac{3\pi}{M} - s$ and $\frac{3\pi}{M} - s \le \theta \le \frac{\pi}{M} + s$. These conditions are sufficient to solve for the optimum The $f_i(\theta)$ for $i = 1, 2, 3$, the optimum r, and the optimum s.

The optimum $f_i(\theta)$ are

$$\begin{aligned} f_1(\theta) &= -\frac{1}{4}\left(\cos\left(\frac{4\pi}{M} - s\right) + \cos s - 2\cos\left(\frac{\pi}{M}\right)\cos\theta\right)\csc^2\left(\frac{\pi}{M}\right)\sec\left(\frac{2\pi}{M} - s\right) \\ f_2(\theta) &= \frac{1}{2}\left(\cot\left(\frac{\pi}{M}\right)\sec\left(\frac{2\pi}{M} - s\right)\sin\left(\frac{\pi}{M} - \theta\right) + 1\right) \\ f_3(\theta) &= -\frac{1}{4}\csc^2\left(\frac{\pi}{M}\right)\left(\cos\left(\frac{3\pi}{M} - \theta\right)\sec\left(\frac{2\pi}{M} - s\right) - 1\right) \end{aligned} \tag{3.11}$$

which correspond to different piecewise linear contributions to $\Pr(y|\mathbf{x})$. The $f_1(\theta)$ piece has a gradient that points in the $(1, 0)$ direction, the $f_2(\theta)$ piece has a gradient that points in the $\left(-\sin\left(\frac{\pi}{M}\right), \cos\left(\frac{\pi}{M}\right)\right)$ direction, and the $f_3(\theta)$ piece has a gradient that points in the $\left(-\sin\left(\frac{3\pi}{M}\right), \cos\left(\frac{3\pi}{M}\right)\right)$ direction. The optimum r is

$$r = \frac{n}{n-1}\ \frac{\cos\left(\frac{2\pi}{M} - s\right)}{\cos\left(\frac{\pi}{M}\right)} \tag{3.12}$$

and the transcendental equation that must be satisfied by the optimum s (for $M = 4$ this reduces to equation 3.8) is

$$\frac{1}{n}\,\frac{\cos\left(\frac{2\pi}{M}-s\right)}{\cos\left(\frac{\pi}{M}\right)}-\frac{n-1}{n}\,\frac{M}{\pi}\cos\left(\frac{\pi}{M}\right)\left(\sin\left(\frac{2\pi}{M}-s\right)-\left(\frac{2\pi}{M}-s\right)\cos\left(\frac{2\pi}{M}-s\right)\right)=0 \tag{3.13}$$

and the minimum $D_1 + D_2$ may be obtained as

$$D_1+D_2=\frac{n\left((n-1)\left(2\,\frac{n-2}{n}-\frac{M}{\pi}\,s\right)-\sec^2\left(\frac{\pi}{M}\right)\right)}{2\,(n-1)^2}-\frac{n\left((n-1)\left(2-\frac{M}{\pi}\,s\right)+\sec^2\left(\frac{\pi}{M}\right)\right)}{2\,(n-1)^2}\cos\left(\frac{4\pi}{M}-2s\right) \tag{3.14}$$

As in section III A, the limit $n \to 1$ is well behaved because the limiting behaviour of the solution s of equation 3.13 contains a factor $n - 1$ (see the asymptotic results in section VI) to cancel the $\frac{1}{n-1}$ factor that appears in both equation 3.12 and equation 3.14.

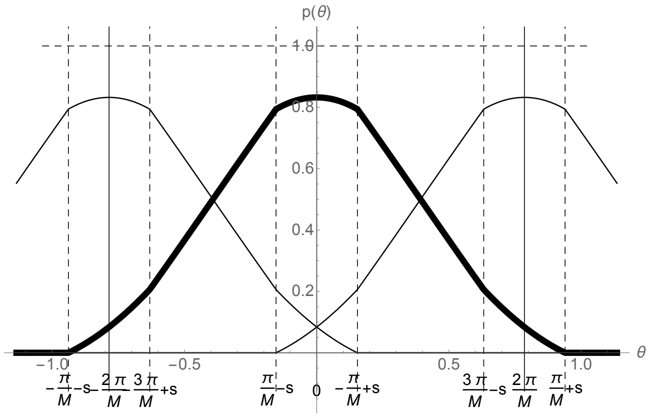


Figure 6: *Plot of the optimal neural posterior probability $p(\theta)$ for $M = 8$ and $n = 100$. The neighbouring posterior probabilities $p\left(\theta \pm \frac{2\pi}{M}\right)$ are also plotted. The optimal value of s is $s \approx 1.39\frac{\pi}{M}$.*

The results for the optimum value of s (i.e. equation 3.8 and equation 3.13) may be combined to yield the results shown in figure 7.

Asymptotically, as $M \to \infty$ and $n \to \infty$, the contour $s = \frac{\pi}{M}$ (the dashed line in figure 7), which is the boundary between the regions where 2 and 3 posterior probabilities overlap, is given by $n \approx 3\,\frac{M^2}{\pi^2}$ (see the asymptotic results in section VI).

The corresponding results for joint encoding of input vectors that live on a 2-torus are shown in figure 8.

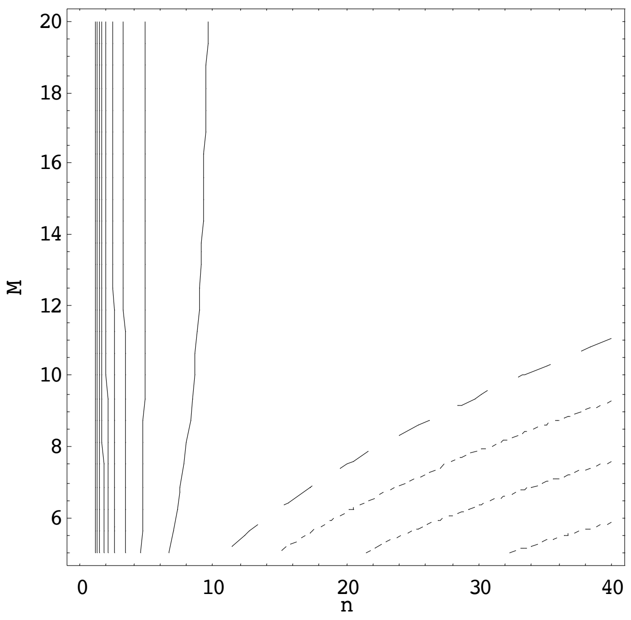


Figure 7: *Contour plot of the optimum value of sversus (n, M) for encoding of a circular manifold. The solid contours are for the interval $0 \leq s < \frac{\pi}{M}$, the dotted contours are for $\frac{\pi}{M} < s \leq \frac{2\pi}{M}$, and the dashed contour is for $s = \frac{\pi}{M}$ (this behaves asymptotically as $n \approx 3\,\frac{M^2}{\pi^2}$). The contours are all separated by intervals of $\frac{\pi}{10M}$.*

IV. TOROIDAL MANIFOLD: FACTORIAL ENCODING

All of the results for factorial encoding of input data that lives on a toroidal manifold may be derived from the expression for $D_1 + D_2$ in equation 2.10 (and the corresponding stationarity conditions), with the appropriate replacements for equations 3.1, 3.2 and 3.3.

The posterior probability $p(\theta)$ then has the same functional form as for a circular manifold, except that M is replaced by $\frac{M}{2}$ because each of the two dimensions uses exactly half of the total of M neurons, so these results are not quoted explicitly here. The steps in the derivation of the optimum values of r and s and the minimum value of $D_1 + D_2$ are analogous to the steps that appear in the derivation for a circular input manifold, and the results are sufficiently different from the ones that were obtained from a circular manifold that they are quoted explicitly here.

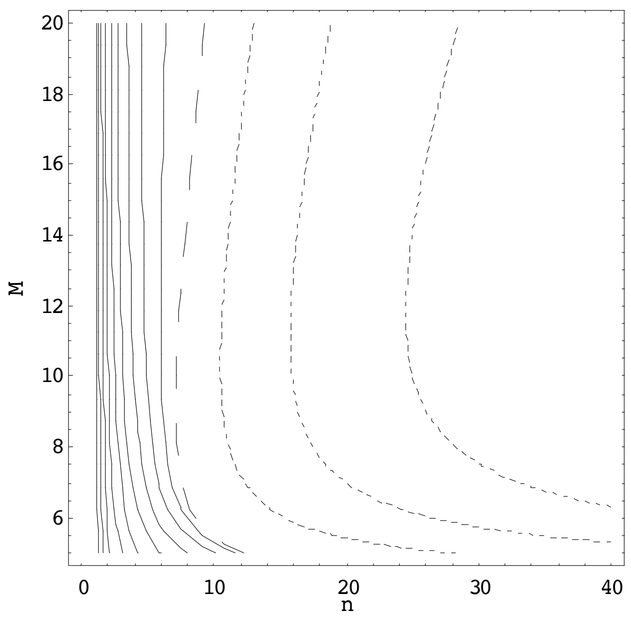


Figure 8: *Contour plot of the optimum value of s versus (n, M) for joint encoding of a toroidal manifold. The solid contours are for the interval $0 \leq s < \frac{\pi}{\sqrt{M}}$, the dotted contours are for $\frac{\pi}{\sqrt{M}} < s \leq \frac{2\pi}{\sqrt{M}}$, and the dashed contour is for $s = \frac{\pi}{\sqrt{M}}$ (this behaves asymptotically as $n \approx 3\,\frac{M}{\pi^2}$). The contours are all separated by intervals of $\frac{\pi}{10\sqrt{M}}$.*

A. Two Overlapping Posterior Probabilities

A detailed derivation of the results reported in this section is given in appendix D 3. The stationarity conditions yield the optimum r as

$$r = \frac{2n}{n-1}\,\frac{\sin s}{\sin\left(\frac{2\pi}{M}\right)} \tag{4.1}$$

The transcendental equation that must be satisfied by the optimum s is

$$\frac{\sin s}{\sin\left(\frac{2\pi}{M}\right)} - \frac{n-1}{n+1}\,\frac{M}{2\pi}\,\sin\left(\frac{2\pi}{M}\right)\,(\cos s + s\,\sin s) = 0 \tag{4.2}$$

The expression for the minimum $D_1 + D_2$ is

$$D_1 + D_2 = 4 - \frac{n}{n-1}\,\frac{M}{2\pi}\,(2s + \sin(2s)) \tag{4.3}$$

B. Three Overlapping Posterior Probabilities

A detailed derivation of the results reported in this section is given in appendix D 4. The stationarity conditions yield the optimum r as

$$r = \frac{2n}{n-1}\,\frac{\cos\left(\frac{4\pi}{M} - s\right)}{\cos\left(\frac{2\pi}{M}\right)} \tag{4.4}$$

The transcendental equation that must be satisfied by the optimum s is

$$\frac{1}{n}\,\frac{\cos\left(\frac{4\pi}{M} - s\right)}{\cos\left(\frac{2\pi}{M}\right)} - \frac{n-1}{2n}\,\frac{M}{2\pi}\,\cos\left(\frac{2\pi}{M}\right)\left(\sin\left(\frac{4\pi}{M} - s\right) - \left(\frac{4\pi}{M} - s\right)\cos\left(\frac{4\pi}{M} - s\right)\right) = 0 \tag{4.5}$$

The expression for the minimum $D_1 + D_2$ is

$$D_1 + D_2 = \frac{n\left((n-1)\left(2\,\frac{n-2}{n} - \frac{M}{2\pi}\,s\right) - 2\sec^2\left(\frac{2\pi}{M}\right)\right)}{(n-1)^2} - \frac{n\left((n-1)\left(2 - \frac{M}{2\pi}\,s\right) + 2\sec^2\left(\frac{2\pi}{M}\right)\right)}{(n-1)^2}\,\cos\left(\frac{8\pi}{M} - 2s\right) \tag{4.6}$$

The results for the optimum value of s (i.e. equation 4.2 and equation 4.5) may be combined to yield the results shown in figure 9.

V. JOINT VERSUS FACTORIAL ENCODING

The results in section III and section IV may be used to deduce when a factorial encoder is favoured with respect to a joint encoder (for input data that lives on a 2-torus). Firstly, equation 3.8 (with the replacement $M \to \sqrt{M}$, and setting $s = \frac{\pi}{\sqrt{M}}$) may be used to deduce the region of the (n, M) plane where joint encoding of a 2-torus involves no more that 2 overlapping posterior probabilities, and equation 4.2 (with $s = \frac{2\pi}{M}$) may be used to deduce the corresponding result for factorial encoding of a 2-torus. Once these regions have been established, it is then possible to decide which of equation 3.9 or equation 3.14 (with $M \to \sqrt{M}$ and then multiplied overall by 2) to use to calculate $D_1 + D_2$ in the case of joint encoding

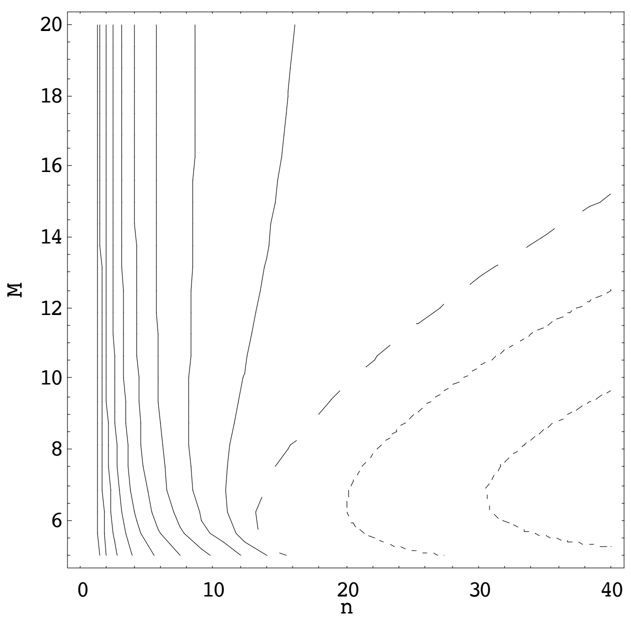


Figure 9: *Contour plot of the optimum value of s versus (n, M) for factorial encoding of a toroidal manifold. The solid contours are for the interval $0 \leq s < \frac{2\pi}{M}$, the dotted contours are for $\frac{2\pi}{M} < s \leq \frac{4\pi}{M}$, and the dashed contour is for $s = \frac{2\pi}{M}$ (this behaves asymptotically as $n \approx \frac{3}{2}\frac{M^2}{\pi^2}$). The contours are all separated by intervals of $\frac{\pi}{5M}$.*

a 2-torus, and which of equation 4.3 or equation 4.6 to use to calculate $D_1 + D_2$ in the case of factorial encoding a 2-torus. These results are gathered together in figure 10.

The need to derive results where up to 3 posterior probabilities overlap (which involves a large amount of algebra) is clear from the results shown in figure 10, where it may be seen that most of the region where the factorial encoder is favoured with respect to the joint encoder has up to 3 overlapping posterior probabilities. The degree to which a factorial encoder is favoured with respect to a joint encoder may be seen in figure 11.

If the number of neurons M is restricted (i.e. $M \lesssim 12$), then the joint encoding scheme in which the 2-torus is encoded using small encoding cells as shown in figure 2(a), is usually not as good as the factorial encoding scheme in which the 2-torus is encoded using the intersection of pairs of elongated encoding cells as shown in figure 2(b). This does require that the number of firing events n is sufficiently large that both subsets of $\frac{M}{2}$ neurons in the factorial encoder are virtually guaranteed to each receive at least 1 firing event, so that they can indeed approximate the input vector by the intersection of a pair of response regions.

If the number of neurons M is too large (i.e. $M \gtrsim 12$), then the joint encoding scheme is always favoured with respect to the factorial encoding scheme, because there are sufficient neurons to encode the 2-torus well using small response regions, as shown in figure 2(a). This includes the limiting case $M \to \infty$, where the curvature of the input manifold is not visible to each neuron separately, because each neuron then responds to an infinitesimally small angular interval of the input manifold. This result implies that joint encoding is always favoured when the input manifold is planar, as was discussed in figure 1 and figure 2.

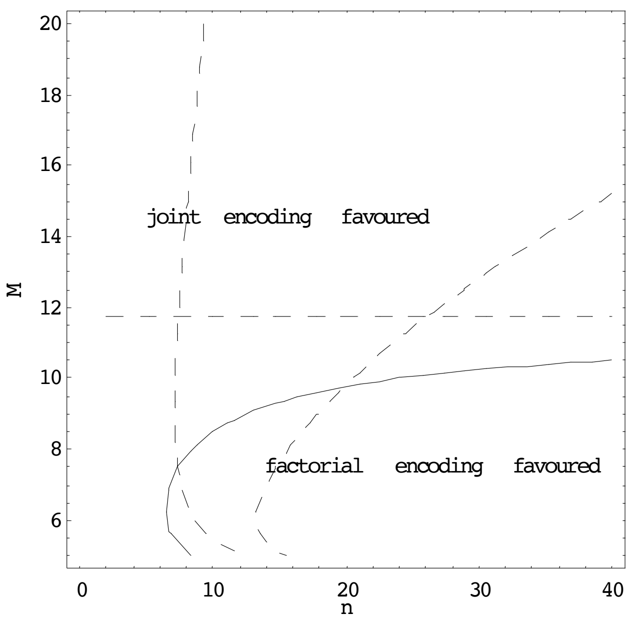


Figure 10: *The diagram shows various results pertaining to joint and factorial encoding of a 2-torus. The solid line is the boundary between the regions of the (n, M) plane where joint or factorial encoding are favoured, and the horizontal dashed line is the asymptotic limit $M \approx 12$ of this boundary as $n \to \infty$. The left hand dashed line is the boundary between the regions where 2 or 3 overlapping posterior probabilites occur in joint encoding, and the right hand dashed line is the corresponding boundary for factorial encoding.*

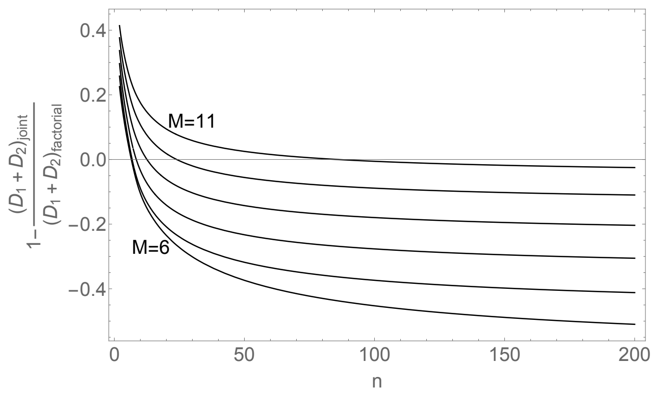


Figure 11: *Plots for $M = 6, 7, 8, 9, 10, 11$ of $(D_1 + D_2)_{factorial} - (D_1 + D_2)_{joint}$ in units in which $(D_1 + D_2)_{factorial} = 1$. This makes it clear that the degree to which a factorial encoder is favoured with respect to a joint encoder is quite significant for large n.*

Although not presented here, these results generalise readily to higher dimensional toruses, where factorial encoding is even more favoured, because (roughly speaking) the number of neurons required to do joint encoding with a given resolution increases exponentially with the dimensionality of the input, whereas the number of neurons required to do factorial encoding with a given resolution increases linearly with the dimensionality of the input (provided that enough firing events are observed).

VI. ASYMPTOTIC RESULTS

Referring to figure 10, the asymptotic behaviour as $M \to \infty$ lies in the region where two posterior probabilities overlap, and the asymptotic behaviour as $n \to \infty$ lies in the region where three posterior probabilities overlap, so care must be taken to use the appropriate results when deriving the various asymptotic approximations below. The boundary between the regions where two or three posterior probabilities overlap can be obtained for a circular input manifold by putting $s = \frac{\pi}{M}$ in equation 3.8 (or $s = \frac{2\pi}{M}$ in equation 4.2 in the case of a toroidal input manifold), and as $M \to \infty$ this is given by

$$n \approx \begin{cases} 3\,\frac{M^2}{\pi^2} & \text{circular manifold} \\ \frac{3}{2}\,\frac{M^2}{\pi^2} & \text{toroidal manifold (factorial encoding)} \end{cases} \tag{6.1}$$

As $M \to \infty$ the asymptotic behaviour of $D_1 + D_2$ for a circular input manifold may be obtained by asymptotically expanding the s dependence of equation 3.8 (or equation 4.2 in the case of a toroidal input manifold) in inverse powers of M, to yield

$$s \approx \begin{cases} \frac{n-1}{n}\,\frac{\pi}{M} + \frac{(n-1)\,(n^2-4n+2)}{3\,n^3}\,\frac{\pi^3}{M^3} & \text{circular manifold} \\ \frac{n-1}{n+1}\,\frac{2\pi}{M} + \frac{(n-1)\,(n^2-6n+1)}{3\,(n+1)^3}\left(\frac{2\pi}{M}\right)^3 & \text{toroidal manifold (factorial encoding)} \end{cases} \tag{6.2}$$

and substituting this solution into the appropriate expression for r to obtain

$$r \approx \begin{cases} 1 + \frac{(2n^2-6n+3)}{6\,n^2}\,\frac{\pi^2}{M^2} & \text{circular manifold} \\ \frac{2n}{n+1} + \frac{8n\,(n^2-4n+1)}{3\,(n+1)^3}\,\frac{\pi^2}{M^2} & \text{toroidal manifold (factorial encoding)} \end{cases} \tag{6.3}$$

and substituting this solution into the appropriate expression for $D_1 + D_2$ to obtain

$$D_1 + D_2 \approx \begin{cases} \frac{2\,(2n-1)}{3n^2}\,\frac{\pi^2}{M^2} & \text{circular manifold} \\ \frac{4}{n+1} + \frac{64n^2}{3\,(n+1)^3}\,\frac{\pi^2}{M^2} & \text{toroidal manifold (factorial encoding)} \end{cases} \tag{6.4}$$

The asymptotic result for a circular manifold may be used to determine the corresponding result for a linear manifold. Thus, if lengths are scaled so that the separation of the neurons (as measured around the circular manifold) becomes unity, which requires that all lengths are divided by $\frac{2\,\pi}{M}$, then asymptotically as $M \to \infty$ the circular manifold solution becomes identical to the solution for a linear manifold with neurons separated by unit distance. Thus the optimum solution for a linear manifold with neurons separated by unit distance is $s = \frac{n-1}{2n}$ and $D_1 + D_2 = \frac{2n-1}{6n^2}$ (note that $D_1 + D_2$ has the dimensions of (length)^2).

As $n \to 1$ (i.e. the LBG vector quantiser limit) the asymptotic behaviour of $D_1 + D_2$ for a circular input manifold may be obtained by expanding the s dependence of equation 3.8 about the point $s = 0$ (or equation 4.2 about the point $s = 0$ for a toroidal input manifold), to yield

$$s \approx \begin{cases} (n-1)\,\frac{M}{\pi}\,\sin^2\left(\frac{\pi}{M}\right) & \text{circular manifold} \\ \frac{n-1}{2}\,\frac{M}{2\pi}\,\sin^2\left(\frac{2\pi}{M}\right) & \text{toroidal manifold (factorial encoding)} \end{cases} \tag{6.5}$$

which gives $s = 0$ when $n = 1$, so there is no overlap between the posterior probabilities for different neurons, as would be expected in a vector quantiser where only one neuron is allowed to fire. Substitute this solution into the appropriate expression for r to obtain at $n = 1$

$$r = \begin{cases} \frac{M}{\pi}\,\sin\left(\frac{\pi}{M}\right) & \text{circular manifold} \\ \frac{M}{2\pi}\,\sin\left(\frac{2\pi}{M}\right) & \text{toroidal manifold (factorial encoding)} \end{cases} \tag{6.6}$$

which is the distance of the centroid of an arc of the unit circle (with angular length $\frac{2\pi}{M}$ for a circular manifold, or angular length $\frac{4\pi}{M}$ for a toroidal manifold) from the origin, as expected for a network in which only one neuron can fire. So the best reconstruction is the centroid of the inputs that could have caused the single firing event. These results may be substituted into the appropriate expression for $D_1 + D_2$ to obtain at $n = 1$

$$D_1 + D_2 = \begin{cases} 2 - 2\left(\frac{M}{\pi}\right)^2 \sin^2\left(\frac{\pi}{M}\right) & \text{circular manifold} \\ 4 - 2\left(\frac{M}{2\pi}\right)^2 \sin^2\left(\frac{2\pi}{M}\right) & \text{toroidal manifold (factorial encoding)} \end{cases} \tag{6.7}$$

These results for $D_1 + D_2$ have a simple geometrical interpretation. For a circular manifold $D_1 + D_2$ is (twice) the average squared distance from an arc with angular length $\frac{2\pi}{M}$ to its associated reference vector, which is exactly what would be expected. For a toroidal manifold $D_1 + D_2$ is the same result with $M \to \frac{M}{2}$, plus an extra contribution of 2, because a factorial encoder with only 1 firing event acts as a conventional encoder using $\frac{M}{2}$ neurons for the circular dimension that is fortunate enough to be associated with the firing event (hence the first contribution to $D_1 + D_2$), and acts as no encoder at all for the other circular dimension which is associated with no firing events (hence the extra contribution of 2 to $D_1 + D_2$).

As $n \to \infty$ the asymptotic behaviour of $D_1 + D_2$ for a circular input manifold may be obtained by expanding the s dependence of equation 3.13 about the point $s = \frac{2\pi}{M}$ (or equation 4.5 about the point $s = \frac{4\pi}{M}$ for a toroidal input manifold), to yield

$$s \approx \begin{cases} \frac{2\pi}{M} - \left(\frac{3\pi}{M\, n \cos^2\left(\frac{\pi}{M}\right)}\right)^{\frac{1}{3}} & \text{circular manifold} \\ \frac{4\pi}{M} - \left(\frac{12\pi}{M\, n \cos^2\left(\frac{2\pi}{M}\right)}\right)^{\frac{1}{3}} & \text{toroidal manifold (factorial encoding)} \end{cases} \tag{6.8}$$

where the limiting values of s as $n \to \infty$ (i.e. $s \to \frac{2\pi}{M}$ for a circular manifold, and $s \to \frac{4\pi}{M}$ for a toroidal manifold) stops just short of allowing four or more posterior probabilities to overlap. In this limit $D_1 = 0$, so for a circular manifold the network acts as a PCA encoder (see the discussion after equation 2.6) whose expansion coefficients sum to unity. In order to encode vectors on a unit circle without error three basis vectors are required; the expansion coefficients are probabilities which must sum to unity, so three basis vectors are required in order that there are two independent expansion coefficients. This is the reason why it is sufficient to consider no more than three overlapping posterior probabilities for encoding data that lives in a 2-dimensional manifold (this argument generalises straightforwardly to higher dimensions). The same argument applies to the case of factorial encoding of a toroidal manifold. Substitute this solution into the appropriate expression for r to obtain

$$r \approx \begin{cases} \frac{1}{2}\sec\left(\frac{\pi}{M}\right)\left(2 - \left(\frac{3\pi}{M\, n \cos^2\left(\frac{\pi}{M}\right)}\right)^{2/3}\right) & \text{circular manifold} \\ \sec\left(\frac{2\pi}{M}\right)\left(2 - \left(\frac{12\pi}{M\, n \cos^2\left(\frac{2\pi}{M}\right)}\right)^{2/3}\right) & \text{toroidal manifold (factorial encoding)} \end{cases} \tag{6.9}$$

and substitute these results into the appropriate expression for $D_1 + D_2$ to obtain

$$D_1 + D_2 \approx \begin{cases} \frac{2}{n}\tan^2\left(\frac{\pi}{M}\right) & \text{circular manifold} \\ \frac{4}{n}\left(2\sec^2\left(\frac{2\,\pi}{M}\right) - 1\right) & \text{toroidal manifold (factorial encoding)} \end{cases} \tag{6.10}$$

Thus as $n \to \infty$ it is possible to derive a value of M for which the asymptotic $D_1 + D_2$ is the same for joint and factorial encoding of a toroidal manifold. This value of M must satisfy $\frac{4}{n}\tan^2\left(\frac{\pi}{\sqrt{M}}\right) = \frac{4}{n}\left(2\sec^2\left(\frac{2\,\pi}{M}\right) - 1\right)$, which yields $M \approx 11.74$.

VII. APPROXIMATE THE POSTERIOR PROBABILITY

A posterior probability may always be written in the form

$$\Pr\left(y|\mathbf{x}\right) = \frac{Q\left(\mathbf{x}|y\right)}{\sum_{y'=0}^{M-1} Q\left(\mathbf{x}|y'\right)} \tag{7.1}$$

where $Q\left(\mathbf{x}|y\right) \geq 0$ (with $Q\left(\mathbf{x}|y\right) > 0$ for at least one value of y for each $\mathbf{x}$). If the neurons behaved in such a way that they produced independent Poissonian firing events in response to a given input, then $Q\left(\mathbf{x}|y\right)$ would be the firing rate (or activation function) of neuron y in response to input $\mathbf{x}$.

The optimum solution $p\left(\theta\right)$ (as given in equation 3.4 and equation 3.5) may be approximated on the unit circle (i.e. $\mathbf{x} = \left(\cos\theta, \sin\theta\right)$) by defining $Q\left(\mathbf{x}|y\right)$ as

$$\begin{aligned} Q\left(\mathbf{x}|y\right) &= \begin{cases} \mathbf{w}\cdot\mathbf{x} - a & \mathbf{w}\cdot\mathbf{x} \geq a \\ 0 & \mathbf{w}\cdot\mathbf{x} \leq a \end{cases} \\ \mathbf{w} &= \left(\cos\left(\frac{2\pi y}{M}\right), \sin\left(\frac{2\pi y}{M}\right)\right) \\ a &= \cos\left(\frac{\pi}{M}\right) - \sin\left(\frac{\pi}{M}\right)\sin s \end{aligned} \tag{7.2}$$

where a is a threshold parameter, and $\mathbf{w}$ is a unit weight vector. This is the form of the neural activation function that is used in [16]. This leads to a good approximation to the optimum solution $p\left(\theta\right)$ because

$$p\left(\theta\right) = \begin{cases} 0 & \theta \leq -\frac{\pi}{M} - s \\ \frac{Q(\mathbf{x}|y=0)}{Q(\mathbf{x}|y=0)+Q(\mathbf{x}|y=M-1)} + O\left(\left(\theta + \frac{\pi}{M}\right)^3\right) & -\frac{\pi}{M} - s \leq \theta \leq -\frac{\pi}{M} + s \\ 1 & -\frac{\pi}{M} + s \leq \theta \leq \frac{\pi}{M} - s \\ \frac{Q(\mathbf{x}|y=0)}{Q(\mathbf{x}|y=0)+Q(\mathbf{x}|y=1)} + O\left(\left(\theta - \frac{\pi}{M}\right)^3\right) & \frac{\pi}{M} - s \leq \theta \leq \frac{\pi}{M} + s \\ 0 & \theta \geq \frac{\pi}{M} + s \end{cases} \tag{7.3}$$

This approximation works well because curved input manifolds can be optimally encoded by using appropriate hyperplanes (as defined in equation 7.2) to slice off pieces of the manifold.

This approximation breaks down as $M \longrightarrow \infty$, as can be seen by inspecting the series expansion of $p\left(\theta\right)$ near $\theta = \frac{\pi}{M}$.

$$p\left(\theta\right) = \begin{cases} \frac{1}{2} - \frac{1}{2}\frac{1}{\sin s}\left(\theta - \frac{\pi}{M}\right) + \frac{1}{12}\frac{1}{\sin s}\left(\theta - \frac{\pi}{M}\right)^3 + O\left(\left(\theta - \frac{\pi}{M}\right)^4\right) & \text{exact} \\ \frac{1}{2} - \frac{1}{2}\frac{1}{\sin s}\left(\theta - \frac{\pi}{M}\right) + \frac{1}{12}\left(\frac{1}{\sin s} - \frac{3}{\tan\left(\frac{\pi}{M}\right)\sin^2 s}\right)\left(\theta - \frac{\pi}{M}\right)^3 + O\left(\left(\theta - \frac{\pi}{M}\right)^4\right) & \text{approximate} \end{cases} \tag{7.4}$$

which differ in the $O\left(\left(\theta - \frac{\pi}{M}\right)^3\right)$ term. In the limit $M \longrightarrow \infty$ the half-width parameter s behaves like M^{-1}, so the $O\left(\left(\theta - \frac{\pi}{M}\right)^3\right)$ term behaves like $M\left(\theta - \frac{\pi}{M}\right)^3$ in the exact case, and $M^3\left(\theta - \frac{\pi}{M}\right)^3$ in the approximate case because of the contribution from the $\frac{3}{\tan\left(\frac{\pi}{M}\right)\sin^2 s}$ term. As $M \longrightarrow \infty$ each neuron responds to a progressively smaller angular range of inputs on the unit circle, so from the point of view of each neuron the curvature of the input manifold becomes negligible (i.e. the input manifold appears to more and more closely approximate a straight line), which ultimately makes it impossible to use hyperplanes to slice off pieces of the manifold. In the $M \longrightarrow \infty$ limit, a better approximation to the posterior probability would be to use ball-shaped regions (e.g. a radial basis function network) to cut up the input manifold into pieces.

VIII. CONCLUSIONS

The results in this paper demonstrate that, for input data that lies on a curved manifold (specifically, a 2-torus), and for an objective function that measures the average reconstruction error (in the Euclidean sense) of a 2-layer neural network encoder, the type of encoder that is optimal depends on the total number of neurons and on the total number of observed firing events in the network output layer. There are two basic types of encoder: a joint encoder in which the network acts as a vector quantiser for the whole input space, and a factorial encoder in which the network breaks into a number of subnetworks, each of which acts as a vector quantiser for a subspace of the input space.

The particular conditions under which factorial encoding is favoured with respect to joint encoding arise when the input data is derived from a curved input manifold, provided that the number of neurons is not too large, and provided that the number of observed neural firing events is large enough. Factorial encoding does not emerge when the input manifold is insufficiently curved, or equivalently when there are too many neurons, because then each neuron does not have a sufficiently large encoding cell to be aware of the manifold's curvature.

Factorial encoding allows the input data to be encoded using a much smaller number of neurons than would be the case if joint encoding were used. Because only a small number of neurons is used, a factorial encoding scheme must be succinct, so it has to abstract the underlying degrees of freedom in the input manifold; this is a very useful side-effect of factorial encoding. This effect becomes stronger as the dimensionality of the curved input manifold is increased.

The main simplification that makes these calculations possible is that, in an optimal neural network, the form for the posterior probability is a piecewise linear function of the input vector. This leads to an enormous simplification in the mathematics, because only the space of piecewise linear functions needs to be searched for the optimal solution, rather than the whole space of functions (subject to normalisation and non-negativity constraints).

A convenient approximation to this type of factorial encoder is the partitioned mixture distribution (PMD) network [10], in which the individual subnetworks in the factorial encoder network are constrained to share parameters, which thus leads to an upper bound on the minimum value of the objective function that would have ideally been obtained with the unconstrained factorial encoder network.

IX. ACKNOWLEDGEMENTS

I thank Chris Webber for many useful conversations that we had during the course of this research.

Appendix A: Objective Function

The objective function $D = 2D_{VQ}$ is given by

$$D \equiv 2\int d\mathbf{x}\ \Pr(\mathbf{x}) \sum_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x})\ \left\|\mathbf{x} - \mathbf{x}'(\mathbf{y})\right\|^2 \tag{A1}$$

If the observed state of the output layer is the locations of n firing events on M neurons, then this expression for D can be manipulated into the following form [12]

$$D = 2\int d\mathbf{x}\ \Pr(\mathbf{x}) \sum_{y_1=1}^{M}\sum_{y_2=1}^{M}\cdots\sum_{y_n=1}^{M} \Pr(y_1, y_2, \cdots, y_n|\mathbf{x})\ \left\|\mathbf{x} - \mathbf{x}'(y_1, y_2, \cdots, y_n)\right\|^2 \tag{A2}$$

where $\Pr(\mathbf{y}|\mathbf{x})$ has now been replaced by the more explicit notation $\Pr(y_1, y_2, \cdots, y_n|\mathbf{x})$, and $\mathbf{x}'(y_1, y_2, \cdots, y_n)$ is a vector given by

$$\mathbf{x}'(y_1, y_2, \cdots, y_n) = \int d\mathbf{x}\ \Pr(\mathbf{x}|y_1, y_2, \cdots, y_n)\ \mathbf{x} \tag{A3}$$

where $\Pr(\mathbf{x}|y_1, y_2, \cdots, y_n)$ may be expressed in terms of $\Pr(\mathbf{x})$ and $\Pr(y_1, y_2, \cdots, y_n|\mathbf{x})$ by using Bayes' theorem in equation 2.1. The goal now is to minimise the expression for D in equation A2 with respect to the function $\Pr(y_1, y_2, \cdots, y_n|\mathbf{x})$. The correct value for $\mathbf{x}'(y_1, y_2, \cdots, y_n)$ may be determined by treating it as an unknown parameter that has to be adjusted to minimise D.

$\Pr(y_1, y_2, \cdots, y_n|\mathbf{x})$ may be interpreted as a recognition model which transforms the state of the input layer into (a probabilistic description of) the state of the output layer, and $\mathbf{x}'(y_1, y_2, \cdots, y_n)$ may be regarded as the corresponding generative model that transforms the state of the output layer into (an approximate reconstruction of) the state of the input layer.

There is so much flexibility in the choice of $\Pr(y_1, y_2, \cdots, y_n|\mathbf{x})$ (and the corresponding $\mathbf{x}'(y_1, y_2, \cdots, y_n)$) that even if D is minimised, it does not necessarily yield an encoded version of the input that is easily interpretable. One way in which a code can be encouraged to have a simple interpretation is to force $\mathbf{x}'(y_1, y_2, \cdots, y_n)$ (i.e. the generative model) to be parameterised thus [12]

$$\mathbf{x}'(y_1, y_2, \cdots, y_n) = \mathbf{x}'(y_1) + \mathbf{x}'(y_2) + \cdots + \mathbf{x}'(y_n) \tag{A4}$$

which is a (symmetric) superposition of reference vectors $\mathbf{x}'(y)$ from each neuron y that has been observed to fire. In this case each neuron has a clearly identifiable contribution to the reconstruction of the input, which makes it much easier to interpret what each neuron is doing. In this case the $\|\cdots\|^2$ term in D is symmetric under interchange of the $(y_1, y_2, \cdots, y_n)$, so only the symmetric part $S[\Pr(y_1, y_2, \cdots, y_n|\mathbf{x})]$ of $\Pr(y_1, y_2, \cdots, y_n|\mathbf{x})$ under interchange of the $(y_1, y_2, \cdots, y_n)$ contributes to D, because the symmetric summation $\sum_{y_1=1}^{M}\sum_{y_2=1}^{M}\cdots\sum_{y_n=1}^{M}(\cdots)$ then removes all non-symmetric contributions.

Define the marginal probabilities $\Pr(y_1|\mathbf{x})$ and $\Pr(y_1, y_2|\mathbf{x})$ of the symmetric part

$S\left[\Pr\left(y_1, y_2, \cdots, y_n|\mathbf{x}\right)\right]$ of $\Pr\left(y_1, y_2, \cdots, y_n|\mathbf{x}\right)$ under interchange of the $\left(y_1, y_2, \cdots, y_n\right)$ as

$$\begin{aligned}\Pr\left(y_1|\mathbf{x}\right) &= \sum_{y_2,y_3,y_4,\cdots,y_n=1}^{M} S\left[\Pr\left(y_1, y_2, \cdots, y_n|\mathbf{x}\right)\right] \\ \Pr\left(y_1, y_2|\mathbf{x}\right) &= \sum_{y_3,y_4,\cdots,y_n=1}^{M} S\left[\Pr\left(y_1, y_2, \cdots, y_n|\mathbf{x}\right)\right]\end{aligned} \tag{A5}$$

These marginal probabilities are for the case where n firing events have potentially been observed, but only the locations of 1 (or 2) firing event(s) chosen randomly from the total number n have actually been observed, with the locations of the other $n-1$ (or $n-2$) firing events having been averaged over.

If it is assumed that $\Pr\left(y_1|\mathbf{x}\right)$ and $\Pr\left(y_1, y_2|\mathbf{x}\right)$ are related by

$$\Pr\left(y_1, y_2|\mathbf{x}\right) = \Pr\left(y_1|\mathbf{x}\right)\,\Pr\left(y_2|\mathbf{x}\right) \tag{A6}$$

then the objective function D has an upper bound $D_1 + D_2$ given by [12]

$$\begin{aligned}&D \leq D_1 + D_2 \\ &D_1 \equiv \frac{2}{n}\int d\mathbf{x}\,\Pr\left(\mathbf{x}\right)\sum_{y=1}^{M}\Pr\left(y|\mathbf{x}\right)\,\left\|\mathbf{x}-\mathbf{x}'\left(y\right)\right\|^2 \\ &D_2 \equiv \frac{2\left(n-1\right)}{n}\int d\mathbf{x}\,\Pr\left(\mathbf{x}\right)\left\|\mathbf{x}-\sum_{y=1}^{M}\Pr\left(y|\mathbf{x}\right)\mathbf{x}'\left(y\right)\right\|^2\end{aligned} \tag{A7}$$

Each of the two marginal probabilities in equation A5 contributes to a different term in D_1+D_2; $\Pr\left(y_1|\mathbf{x}\right)$ contributes to D_1, whereas $\Pr\left(y_1, y_2|\mathbf{x}\right)$ contributes to D_2. Informally speaking, D_1 measures the information that a single firing event (out of n such events) contributes to the reconstruction of the input, whereas D_2 measures the information that pairs of firing events (out of n such events) contribute to the reconstruction of the input. D_1 is weighted by a factor $\frac{1}{n}$ which suppresses the single firing event contribution as $n \to \infty$, whereas D_2 is weighted by a factor $\frac{n-1}{n}$ which suppresses the double firing event contribution as $n \to 1$, as expected. If only the D_1 part of the objective function is used (i.e. $n = 1$), then a standard LBG vector quantiser [7] emerges which approximates the input by a single reference vector $\mathbf{x}'\left(y\right)$, whereas if only the D_2 part of the objective function is used (i.e. $n \to \infty$), then the network behaves essentially as a principal component analyser (PCA) which approximates the input by a sum of reference vectors $\sum_{y=1}^{M}\Pr\left(y|\mathbf{x}\right)\,\mathbf{x}'\left(y\right)$, where the $\Pr\left(y|\mathbf{x}\right)$ are expansion coefficients which sum to unity, and the $\mathbf{x}'\left(y\right)$ are basis vectors.

The upper bound $D_1 + D_2$ on D contains LBG encoding and PCA encoding as two limiting cases, and gives a principled way of interpolating between these extremes. This useful property has been bought at the cost of replacing D by an upper bound bound $D_1 + D_2$, which will yield only a suboptimal (from the point of view of D) encoder. However, this upper bound can be expected to be tight in cases where the input manifold can be modelled accurately using the parameteric form $\mathbf{x}'\left(y_1\right)+\mathbf{x}'\left(y_2\right)+\cdots+\mathbf{x}'\left(y_n\right)$. These conditions are well approximated in images which consist of a discrete number of constituents, each of which may be represented by an $\mathbf{x}'\left(y\right)$ for some choice of y. This model fails in situations where two or more constituents are placed so that they overlap, in which case the image will typically contain occluded objects, whereas the model assumes that the objects linearly superpose. Occlusion is not an easy situation to model, so it will be assumed that the image constituents are sufficiently sparse that they rarely occude each other.

Appendix B: Stationarity Conditions

The expression for $D_1 + D_2$ (see equation 2.6) has two types of parameters that need to be optimised: the reference vectors $\mathbf{x}'\left(y\right)$ and the posterior probabilities $\Pr\left(y|\mathbf{x}\right)$. In appendix B 1 the stationarity condition for $\mathbf{x}'\left(y\right)$ is derived, and in appendix B 2 the stationarity condition for $\Pr\left(y|\mathbf{x}\right)$ is derived, taking into account the constraints $0 \leq \Pr\left(y|\mathbf{x}\right) \leq 1$ and $\sum_{y=1}^{M}\Pr\left(y|\mathbf{x}\right) = 1$ which must be satisfied by probabilities.

1. Stationary $\mathbf{x}'\left(y\right)$

The stationarity condition $\frac{\partial\left(D_1+D_2\right)}{\partial\mathbf{x}'\left(y\right)} = 0$ for $\mathbf{x}'\left(y\right)$ was derived in [13]. Thus $\frac{\partial\left(D_1+D_2\right)}{\partial\mathbf{x}'\left(y\right)}$ can be written as

$$\frac{\partial\left(D_1+D_2\right)}{\partial\mathbf{x}'\left(y\right)} = -\frac{4}{n}\int d\mathbf{x}\,\Pr\left(\mathbf{x}\right)\,\Pr\left(y|\mathbf{x}\right)\left(\mathbf{x}-\mathbf{x}'\left(y\right)+\left(n-1\right)\sum_{y'=1}^{M}\Pr\left(y'|\mathbf{x}\right)\,\left(\mathbf{x}-\mathbf{x}'\left(y'\right)\right)\right) \tag{B1}$$

and, using Bayes' theorem in the form $\Pr(\mathbf{x}|y)\,\Pr(y) = \Pr(y|\mathbf{x})\,\Pr(\mathbf{x})$, this yields a matrix equation for the $\mathbf{x}'(y)$

$$0 = \Pr(y)\left(n\int d\mathbf{x}\,\Pr(\mathbf{x}|y)\,\mathbf{x} - (n-1)\sum_{y'=1}^{M}\left(\int d\mathbf{x}\,\Pr(\mathbf{x}|y)\,\Pr(y'|\mathbf{x})\right)\mathbf{x}'(y') - \mathbf{x}'(y)\right) \tag{B2}$$

There are two classes of solution to this stationarity condition, corresponding to one (or more) of the two factors in equation B2 being zero.

1. $\Pr(y) = 0$ (the first factor is zero). If the probability that neuron y fires is zero, then nothing can be deduced about $\mathbf{x}'(y)$, because there is no training data to explore this neuron's behaviour.

2. $n\int d\mathbf{x}\,\Pr(\mathbf{x}|y)\,\mathbf{x} = (n-1)\sum_{y'=1}^{M}\left(\int d\mathbf{x}\,\Pr(\mathbf{x}|y)\,\Pr(y'|\mathbf{x})\right)\mathbf{x}'(y') + \mathbf{x}'(y)$ (the second factor is zero). The solution to this matrix equation is the required $\mathbf{x}'(y)$.

2. Stationary $\Pr(y|\mathbf{x})$

The stationarity condition $\frac{\delta(D_1+D_2)}{\delta\log\Pr(y|\mathbf{x})}$ (with the normalisation constraint $\sum_{y'=1}^{M}\Pr(y'|\mathbf{x}) = 1$) for $\Pr(y|\mathbf{x})$ will now be derived. Thus functionally differentiate $D_1 + D_2$ with respect to $\log\Pr(y|\mathbf{x})$, where logarithmic differentation implicitly imposes the constraint $\Pr(y|\mathbf{x}) \geq 0$, and use a Lagrange multiplier term $L \equiv \int d\mathbf{x}'\,\lambda(\mathbf{x}')\sum_{y'=1}^{M}\Pr(y'|\mathbf{x}')$ to impose the normalisation constraint $\sum_{y=1}^{M}\Pr(y|\mathbf{x}) = 1$ for each $\mathbf{x}$, to obtain

$$\begin{aligned}\frac{\delta(D_1+D_2-L)}{\delta\log\Pr(y|\mathbf{x})} &= \frac{2}{n}\Pr(\mathbf{x})\,\Pr(y|\mathbf{x})\,\left\|\mathbf{x}-\mathbf{x}'(y)\right\|^2 \\ &\quad -\frac{4(n-1)}{n}\Pr(\mathbf{x})\,\Pr(y|\mathbf{x})\,\mathbf{x}'(y)\cdot\left(\mathbf{x}-\sum_{y=1}^{M}\Pr(y|\mathbf{x})\,\mathbf{x}'(y)\right) \\ &\quad -\lambda(\mathbf{x})\,\Pr(y|\mathbf{x})\end{aligned} \tag{B3}$$

The stationarity condition implies that $\sum_{y=1}^{M}\Pr(y|\mathbf{x})\,\frac{\delta(D_1+D_2-L)}{\delta\Pr(y|\mathbf{x})} = 0$, which may be used to determine the Lagrange multiplier function $\lambda(\mathbf{x})$. When $\lambda(\mathbf{x})$ is substituted back into the stationarity condition itself, it yields

$$0 = \Pr(\mathbf{x})\,\Pr(y|\mathbf{x})\sum_{y'=1}^{M}\left(\Pr(y'|\mathbf{x}) - \delta_{y,y'}\right)\mathbf{x}'(y')\cdot\left(\frac{1}{2}\mathbf{x}'(y') - n\,\mathbf{x} + (n-1)\sum_{y''=1}^{M}\Pr(y''|\mathbf{x})\,\mathbf{x}'(y'')\right) \tag{B4}$$

There are several classes of solution to this stationarity condition, corresponding to one (or more) of the three factors in equation B4 being zero.

1. $\Pr(\mathbf{x}) = 0$ (the first factor is zero). If the input PDF is zero at $\mathbf{x}$, then nothing can be deduced about $\Pr(y|\mathbf{x})$, because there is no training data to explore the network's behaviour at this point.

2. $\Pr(y|\mathbf{x}) = 0$ (the second factor is zero). This factor arises from the differentiation with respect to $\log\Pr(y|\mathbf{x})$, and it ensures that $\Pr(y|\mathbf{x}) < 0$ cannot be attained. The singularity in $\log\Pr(y|\mathbf{x})$ when $\Pr(y|\mathbf{x}) = 0$ is what causes this solution to emerge.

3. $\sum_{y'=1}^{M}\left(\Pr(y'|\mathbf{x}) - \delta_{y,y'}\right)\mathbf{x}'(y')\cdot(\cdots) = 0$ (the third factor is zero). The solution to this equation is a $\Pr(y|\mathbf{x})$ that has a piecewise linear dependence on $\mathbf{x}$. This result can be seen to be intuitively reasonable because $D_1 + D_2$ is of the form $\int d\mathbf{x}\,\Pr(\mathbf{x})\,f(\mathbf{x})$, where $f(\mathbf{x})$ is a linear combination of terms of the form $\mathbf{x}^i\,\Pr(y|\mathbf{x})^j$ (for $i = 0,1,2$ and $j = 0,1,2$), which is a quadratic form in $\mathbf{x}$ (ignoring the $\mathbf{x}$-dependence of $\Pr(y|\mathbf{x})$). However, the terms that appear in this linear combination are such that a $\Pr(y|\mathbf{x})$ that is a piecewise linear function of $\mathbf{x}$ guarantees that $f(\mathbf{x})$ is a piecewise linear combination of terms of the form $\mathbf{x}^i$ (for $i = 0,1,2$), which is a quadratic form in $\mathbf{x}$ (the normalisation constraint $\sum_{y=1}^{M}\Pr(y|\mathbf{x}) = 1$ is used to remove a contribution to that is potentially quartic in $\mathbf{x}$). Thus a piecewise linear dependence of $\Pr(y|\mathbf{x})$ on $\mathbf{x}$ does not lead to any dependencies on $\mathbf{x}$ that are not already explicitly present in $D_1 + D_2$. The stationarity condition on $\Pr(y|\mathbf{x})$ (see equation B4) then imposes conditions on the allowed piecewise

linearities that $\Pr(y|\mathbf{x})$ can have.

Appendix C: Simplified Expressions for $D_1 + D_2$

The expressions for D_1 and D_2 (see equation 2.6) may be simplified in the case of joint encoding and factorial encoding. The case of joint encoding is derived in appendix C 1, and the case of factorial encoding is derived in appendix C 2. In both cases it is assumed that $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ and $\Pr(\mathbf{x}_1, \mathbf{x}_2) = \Pr(\mathbf{x}_1)\Pr(\mathbf{x}_2)$ where $\Pr(\mathbf{x}_1)$ and $\Pr(\mathbf{x}_2)$ each define a uniform PDF on the input manifold.

1. Joint Encoding

The expressions for D_1 and D_2 may be simplified in the case of joint encoding, where $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$, $y = (y_1, y_2)$ for $1 \leq y_1 \leq \sqrt{M}$ and $1 \leq y_2 \leq \sqrt{M}$. In the following two derivations of the expressions for D_1 and D_2 the steps in the derivation use exactly the same sequence of manipulations.

The expression for D_1 is

$$D_1 = \frac{2}{n} \int d\mathbf{x}_1 \, d\mathbf{x}_2 \, \Pr(\mathbf{x}_1, \mathbf{x}_2) \sum_{y_1=1}^{\sqrt{M}} \sum_{y_2=1}^{\sqrt{M}} \Pr(y_1, y_2|\mathbf{x}_1, \mathbf{x}_2) \left\| \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} - \begin{pmatrix} \mathbf{x}_1'(y_1, y_2) \\ \mathbf{x}_2'(y_1, y_2) \end{pmatrix} \right\|^2 \tag{C1}$$

The assumed properties of $\Pr(\mathbf{x}_1, \mathbf{x}_2)$ imply that $\mathbf{x}_1'(y_1, y_2) = \mathbf{x}_1'(y_1)$ and $\mathbf{x}_2'(y_1, y_2) = \mathbf{x}_2'(y_2)$, which gives

$$D_1 = \frac{2}{n} \int d\mathbf{x}_1 \, d\mathbf{x}_2 \, \Pr(\mathbf{x}_1, \mathbf{x}_2) \sum_{y_1=1}^{\sqrt{M}} \sum_{y_2=1}^{\sqrt{M}} \Pr(y_1, y_2|\mathbf{x}_1, \mathbf{x}_2) \left(\|\mathbf{x}_1 - \mathbf{x}_1'(y_1)\|^2 + \|\mathbf{x}_2 - \mathbf{x}_2'(y_2)\|^2 \right) \tag{C2}$$

Marginalise $\Pr(y_1, y_2|\mathbf{x}_1, \mathbf{x}_2)$ where possible, using that $\sum_{y_1=1}^{\sqrt{M}} \Pr(y_1, y_2|\mathbf{x}_1, \mathbf{x}_2) = \Pr(y_2|\mathbf{x}_1, \mathbf{x}_2) = \Pr(y_2|\mathbf{x}_2)$ and $\sum_{y_2=1}^{\sqrt{M}} \Pr(y_1, y_2|\mathbf{x}_1, \mathbf{x}_2) = \Pr(y_1|\mathbf{x}_1, \mathbf{x}_2) = \Pr(y_1|\mathbf{x}_1)$, to obtain

$$D_1 = \frac{2}{n} \int d\mathbf{x}_1 \, d\mathbf{x}_2 \, \Pr(\mathbf{x}_1, \mathbf{x}_2) \left(\sum_{y_1=1}^{\sqrt{M}} \Pr(y_1|\mathbf{x}_1) \, \|\mathbf{x}_1 - \mathbf{x}_1'(y_1)\|^2 + \sum_{y_2=1}^{\sqrt{M}} \Pr(y_2|\mathbf{x}_2) \, \|\mathbf{x}_2 - \mathbf{x}_2'(y_2)\|^2 \right) \tag{C3}$$

Marginalise $\Pr(\mathbf{x}_1, \mathbf{x}_2)$ where possible, using that $\int d\mathbf{x}_1 \, \Pr(\mathbf{x}_1, \mathbf{x}_2) = \Pr(\mathbf{x}_2)$ and $\int d\mathbf{x}_2 \, \Pr(\mathbf{x}_1, \mathbf{x}_2) = \Pr(\mathbf{x}_1)$, to obtain

$$D_1 = \frac{2}{n} \int dx_1 \, \Pr(\mathbf{x}_1) \sum_{y_1=1}^{\sqrt{M}} \Pr(y_1|\mathbf{x}_1) \, \|\mathbf{x}_1 - \mathbf{x}_1'(y_1)\|^2 + \frac{2}{n} \int d\mathbf{x}_2 \, \Pr(\mathbf{x}_2) \sum_{y_2=1}^{\sqrt{M}} \Pr(y_2|\mathbf{x}_2) \, \|\mathbf{x}_2 - \mathbf{x}_2'(y_2)\|^2 \tag{C4}$$

Because of the assumed symmetry of the solution, these two terms are the same, which gives

$$D_1 = \frac{4}{n} \int d\mathbf{x}_1 \, \Pr(\mathbf{x}_1) \sum_{y_1=1}^{\sqrt{M}} \Pr(y_1|\mathbf{x}_1) \, \|\mathbf{x}_1 - \mathbf{x}_1'(y_1)\|^2 \tag{C5}$$

The expression for D_2 is

$$D_2 = \frac{2(n-1)}{n} \int d\mathbf{x}_1 \, d\mathbf{x}_2 \, \Pr(\mathbf{x}_1, \mathbf{x}_2) \left\| \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} - \sum_{y_1=1}^{\sqrt{M}} \sum_{y_2=1}^{\sqrt{M}} \Pr(y_1, y_2|\mathbf{x}_1, \mathbf{x}_2) \begin{pmatrix} \mathbf{x}_1'(y_1, y_2) \\ \mathbf{x}_2'(y_1, y_2) \end{pmatrix} \right\|^2 \tag{C6}$$

Use that $\mathbf{x}_1'(y_1, y_2) = \mathbf{x}_1'(y_1)$ and $\mathbf{x}_2'(y_1, y_2) = \mathbf{x}_2'(y_2)$.

$$\begin{aligned} D_2 &= \frac{2(n-1)}{n} \int d\mathbf{x}_1 \, d\mathbf{x}_2 \, \Pr(\mathbf{x}_1, \mathbf{x}_2) \\ &\times \left(\left\| \mathbf{x}_1 - \sum_{y_1=1}^{\sqrt{M}} \sum_{y_2=1}^{\sqrt{M}} \Pr(y_1, y_2|\mathbf{x}_1, \mathbf{x}_2) \, \mathbf{x}_1'(y_1) \right\|^2 + \left\| \mathbf{x}_2 - \sum_{y_1=1}^{\sqrt{M}} \sum_{y_2=1}^{\sqrt{M}} \Pr(y_1, y_2|\mathbf{x}_1, \mathbf{x}_2) \, \mathbf{x}_2'(y_2) \right\|^2 \right) \end{aligned} \tag{C7}$$

Use that $\sum_{y_1=1}^{\sqrt{M}} \Pr(y_1, y_2|\mathbf{x}_1, \mathbf{x}_2) = \Pr(y_2|\mathbf{x}_2)$ and $\sum_{y_2=1}^{\sqrt{M}} \Pr(y_1, y_2|\mathbf{x}_1, \mathbf{x}_2) = \Pr(y_1|\mathbf{x}_1)$.

$$D_2 = \frac{2(n-1)}{n} \int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1, \mathbf{x}_2) \left(\left\| \mathbf{x}_1 - \sum_{y_1=1}^{\sqrt{M}} \Pr(y_1|\mathbf{x}_1)\, \mathbf{x}'_1(y_1) \right\|^2 + \left\| \mathbf{x}_2 - \sum_{y_2=1}^{\sqrt{M}} \Pr(y_2|\mathbf{x}_2)\, \mathbf{x}'_2(y_2) \right\|^2 \right) \tag{C8}$$

Marginalise $\Pr(\mathbf{x}_1, \mathbf{x}_2)$.

$$D_2 = \frac{2(n-1)}{n} \int d\mathbf{x}_1\, \Pr(\mathbf{x}_1) \left\| \mathbf{x}_1 - \sum_{y_1=1}^{\sqrt{M}} \Pr(y_1|\mathbf{x}_1)\, \mathbf{x}'_1(y_1) \right\|^2 + \frac{2(n-1)}{n} \int d\mathbf{x}_2\, \Pr(\mathbf{x}_2) \left\| \mathbf{x}_2 - \sum_{y_2=1}^{\sqrt{M}} \Pr(y_2|\mathbf{x}_2)\, \mathbf{x}'_2(y_2) \right\|^2 \tag{C9}$$

Use symmetry

$$D_2 = \frac{4(n-1)}{n} \int d\mathbf{x}_1\, \Pr(\mathbf{x}_1) \left\| \mathbf{x}_1 - \sum_{y_1=1}^{\sqrt{M}} \Pr(y_1|\mathbf{x}_1)\, \mathbf{x}'_1(y_1) \right\|^2 \tag{C10}$$

These results may be combined to yield finally

$$D_1 + D_2 = \frac{4}{n} \int d\mathbf{x}_1\, \Pr(\mathbf{x}_1) \sum_{y_1=1}^{\sqrt{M}} \Pr(y_1|\mathbf{x}_1)\, \left\| \mathbf{x}_1 - \mathbf{x}'_1(y_1) \right\|^2 + \frac{4(n-1)}{n} \int d\mathbf{x}_1\, \Pr(\mathbf{x}_1) \left\| \mathbf{x}_1 - \sum_{y_1=1}^{\sqrt{M}} \Pr(y_1|\mathbf{x}_1)\, \mathbf{x}'_1(y_1) \right\|^2 \tag{C11}$$

which has the same form as $D_1 + D_2$ would have had for $\mathbf{x}_1$-space alone, with the replacement $M \to \frac{M}{2}$, followed by multiplication by a factor 2 overall. This implies that the problem of optimising a joint encoder is trivially related to the problem of optimising an encoder in the $\mathbf{x}_1$-space alone.

2. Factorial Encoding

The expressions for D_1 and D_2 may be simplified in the case of factorial encoding. In the following two derivations of the expressions for D_1 and D_2, the steps in the derivation use exactly the same sequence of manipulations, except that D_2 has one additional step which separates the contributions inside $\|\cdots\|^2$.

The expression for D_1 is

$$D_1 = \frac{2}{n} \int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1, \mathbf{x}_2) \sum_{y=1}^{M} \Pr(y|\mathbf{x}_1, \mathbf{x}_2) \left\| \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} - \begin{pmatrix} \mathbf{x}'_1(y) \\ \mathbf{x}'_2(y) \end{pmatrix} \right\|^2 \tag{C12}$$

Split up $\Pr(y|\mathbf{x}_1, \mathbf{x}_2)$, using that $\Pr(y|\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{2}\Pr(y|\mathbf{x}_1) + \frac{1}{2}\Pr(y|\mathbf{x}_2)$, which gives

$$D_1 = \frac{1}{n} \int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1, \mathbf{x}_2) \sum_{y=1}^{M} \left(\Pr(y|\mathbf{x}_1) + \Pr(y|\mathbf{x}_2)\right) \left\| \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} - \begin{pmatrix} \mathbf{x}'_1(y) \\ \mathbf{x}'_2(y) \end{pmatrix} \right\|^2 \tag{C13}$$

Assume that the input manifold is such that $\mathbf{x}'_1(y) = \mathbf{0}$ for $\frac{M}{2} + 1 \le y \le M$, and $\mathbf{x}'_2(y) = \mathbf{0}$ for $1 \le y \le \frac{M}{2}$. Also use that $\Pr(y|\mathbf{x}_1) = 0$ for $\frac{M}{2} + 1 \le y \le M$, and $\Pr(y|\mathbf{x}_2) = 0$ for $1 \le y \le \frac{M}{2}$, to obtain

$$\begin{aligned} D_1 &= \frac{1}{n} \int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1, \mathbf{x}_2) \sum_{y=1}^{\frac{M}{2}} \Pr(y|\mathbf{x}_1) \left\| \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} - \begin{pmatrix} \mathbf{x}'_1(y) \\ \mathbf{0} \end{pmatrix} \right\|^2 \\ &\quad + \frac{1}{n} \int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1, \mathbf{x}_2) \sum_{y=\frac{M}{2}+1}^{M} \Pr(y|\mathbf{x}_2) \left\| \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ \mathbf{x}'_2(y) \end{pmatrix} \right\|^2 \end{aligned} \tag{C14}$$

Because of the assumed symmetry of the solution, these two terms are the same, which gives

$$D_1 = \frac{2}{n} \int d\mathbf{x}_1\, d\mathbf{x}_2\, \Pr(\mathbf{x}_1, \mathbf{x}_2) \sum_{y=1}^{\frac{M}{2}} \Pr(y|\mathbf{x}_1) \left(\left\| \mathbf{x}_1 - \mathbf{x}'_1(y) \right\|^2 + \left\| \mathbf{x}_2 \right\|^2 \right) \tag{C15}$$

Marginalise $\Pr(\mathbf{x}_1, \mathbf{x}_2)$ where possible, using that $\int d\mathbf{x}_1 \Pr(\mathbf{x}_1, \mathbf{x}_2) = \Pr(\mathbf{x}_2)$ and $\int d\mathbf{x}_2 \Pr(\mathbf{x}_1, \mathbf{x}_2) = \Pr(\mathbf{x}_1)$, to obtain

$$D_1 = \frac{2}{n}\left(\int d\mathbf{x}_1 \Pr(\mathbf{x}_1) \sum_{y=1}^{\frac{M}{2}} \Pr(y|\mathbf{x}_1) \left\|\mathbf{x}_1 - \mathbf{x}_1'(y)\right\|^2 + \int d\mathbf{x}_2 \Pr(\mathbf{x}_2) \left\|\mathbf{x}_2\right\|^2\right) \tag{C16}$$

The expression for D_2 is

$$D_2 = \frac{2(n-1)}{n} \int d\mathbf{x}_1\, d\mathbf{x}_2 \Pr(\mathbf{x}_1, \mathbf{x}_2) \left\| \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} - \sum_{y=1}^{M} \Pr(y|\mathbf{x}_1, \mathbf{x}_2) \begin{pmatrix} \mathbf{x}_1'(y) \\ \mathbf{x}_2'(y) \end{pmatrix} \right\|^2 \tag{C17}$$

Use that $\Pr(y|\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{2}\Pr(y|\mathbf{x}_1) + \frac{1}{2}\Pr(y|\mathbf{x}_2)$.

$$D_2 = \frac{2(n-1)}{n} \int d\mathbf{x}_1\, d\mathbf{x}_2 \Pr(\mathbf{x}_1, \mathbf{x}_2) \left\| \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} - \frac{1}{2}\sum_{y=1}^{M} \left(\Pr(y|\mathbf{x}_1) + \Pr(y|\mathbf{x}_2)\right) \begin{pmatrix} \mathbf{x}_1'(y) \\ \mathbf{x}_2'(y) \end{pmatrix} \right\|^2 \tag{C18}$$

Separate the contributions from the upper and lower components inside $\left\|\cdots\right\|^2$, to obtain

$$D_2 = \frac{2(n-1)}{n} \int d\mathbf{x}_1\, d\mathbf{x}_2 \Pr(\mathbf{x}_1, \mathbf{x}_2) \left\| \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} - \frac{1}{2}\sum_{y=1}^{\frac{M}{2}} \Pr(y|\mathbf{x}_1) \begin{pmatrix} \mathbf{x}_1'(y) \\ \mathbf{0} \end{pmatrix} - \frac{1}{2}\sum_{y=\frac{M}{2}}^{M} \Pr(y|\mathbf{x}_2) \begin{pmatrix} \mathbf{0} \\ \mathbf{x}_2'(y) \end{pmatrix} \right\|^2 \tag{C19}$$

Use that $\mathbf{x}_1'(y) = \mathbf{0}$ for $\frac{M}{2}+1 \leq y \leq M$, and $\mathbf{x}_2'(y) = \mathbf{0}$ for $1 \leq y \leq \frac{M}{2}$. Also use that $\Pr(y|\mathbf{x}_1) = 0$ for $\frac{M}{2}+1 \leq y \leq M$, and $\Pr(y|\mathbf{x}_2) = 0$.

$$\begin{aligned} D_2 &= \frac{2(n-1)}{n} \int d\mathbf{x}_1\, d\mathbf{x}_2 \Pr(\mathbf{x}_1, \mathbf{x}_2) \left\| \mathbf{x}_1 - \frac{1}{2}\sum_{y=1}^{\frac{M}{2}} \Pr(y|\mathbf{x}_1)\, \mathbf{x}_1'(y) \right\|^2 \\ &\quad + \frac{2(n-1)}{n} \int d\mathbf{x}_1\, d\mathbf{x}_2 \Pr(\mathbf{x}_1, \mathbf{x}_2) \left\| \mathbf{x}_2 - \frac{1}{2}\sum_{y=\frac{M}{2}}^{M} \Pr(y|\mathbf{x}_2)\, \mathbf{x}_2'(y) \right\|^2 \end{aligned} \tag{C20}$$

Use symmetry.

$$D_2 = \frac{4(n-1)}{n} \int d\mathbf{x}_1\, d\mathbf{x}_2 \Pr(\mathbf{x}_1, \mathbf{x}_2) \left\| \mathbf{x}_1 - \frac{1}{2}\sum_{y=1}^{\frac{M}{2}} \Pr(y|\mathbf{x}_1)\, \mathbf{x}_1'(y) \right\|^2 \tag{C21}$$

Marginalise $\Pr(\mathbf{x}_1, \mathbf{x}_2)$.

$$D_2 = \frac{4(n-1)}{n} \int d\mathbf{x}_1 \Pr(\mathbf{x}_1) \left\| \mathbf{x}_1 - \frac{1}{2}\sum_{y=1}^{\frac{M}{2}} \Pr(y|\mathbf{x}_1)\, \mathbf{x}_1'(y) \right\|^2 \tag{C22}$$

These results may be combined to yield finally

$$\begin{aligned} D_1 + D_2 &= \frac{2}{n}\int d\mathbf{x}_1 \Pr(\mathbf{x}_1) \sum_{y=1}^{\frac{M}{2}} \Pr(y|\mathbf{x}_1) \left\|\mathbf{x}_1 - \mathbf{x}_1'(y)\right\|^2 + \frac{4(n-1)}{n} \int d\mathbf{x}_1 \Pr(\mathbf{x}_1) \left\| \mathbf{x}_1 - \frac{1}{2}\sum_{y=1}^{\frac{M}{2}} \Pr(y|\mathbf{x}_1)\, \mathbf{x}_1'(y) \right\|^2 \\ &\quad + \frac{2}{n}\int d\mathbf{x}_2 \Pr(\mathbf{x}_2) \left\|\mathbf{x}_2\right\|^2 \end{aligned} \tag{C23}$$

The stationarity conditions may be derived from this expression for the factorial encoding version of $D_1 + D_2$. The stationarity condition w.r.t. $\Pr(y|\mathbf{x}_1)$ is

$$\sum_{y'=1}^{\frac{M}{2}} \left(\Pr(y'|\mathbf{x}_1) - \delta_{y,y'}\right) \mathbf{x}_1'(y') \cdot \left(\frac{1}{2}\mathbf{x}_1'(y') - n\,\mathbf{x}_1 + \frac{n-1}{2} \sum_{y''=1}^{\frac{M}{2}} \Pr(y''|\mathbf{x}_1)\, \mathbf{x}_1'(y'') \right) = 0 \tag{C24}$$

and the stationarity condition w.r.t. $\mathbf{x}_1'(y)$ is

$$n\int d\mathbf{x}_1 \Pr(\mathbf{x}_1|y)\,\mathbf{x}_1 = \mathbf{x}_1'(y) + \frac{n-1}{2}\int d\mathbf{x}_1\,\Pr(\mathbf{x}_1|y)\sum_{y'=1}^{\frac{M}{2}}\Pr(y'|\mathbf{x}_1)\,\mathbf{x}_1'(y') \tag{C25}$$

Both of these stationarity conditions can be obtained from the standard ones by making the replacements $(n-1)\sum_{y'=1}^{M}\Pr(y'|\mathbf{x}_1)\,\mathbf{x}_1'(y') \to \frac{n-1}{2}\sum_{y'=1}^{M}\Pr(y'|\mathbf{x}_1)\,\mathbf{x}_1'(y')$ and $M \to \frac{M}{2}$.

Appendix D: Minimise $D_1 + D_2$

The expression for $D_1 + D_2$ needs to be minimised with respect to the reference vectors $\mathbf{x}'(y)$ and the posterior probabilities $\Pr(y|\mathbf{x})$. There are four cases to consider, which are various combinations of circular/toroidal input manifold (appendices D 1 and D 2/appendices D 3 and D 4) and two/three overlapping posterior probabilities (appendices D 1 and D 3/appendices D 2 and D 4). For a toroidal manifold it is not necessary to consider the case of joint encoding, because it is directly related to encoding a circular manifold, which is dealt with in appendices D 1 and D 2.

1. Circular Manifold: 2 Overlapping Posterior Probabilities

For $0 \le s \le \frac{\pi}{M}$ the functional form of $p(\theta)$ that ensures a piecewise linear $\Pr(y|\mathbf{x})$ is

$$p(\theta) = \begin{cases} 1 & 0 \le |\theta| \le \frac{\pi}{M} - s \\ f(\theta) & \frac{\pi}{M} - s \le |\theta| \le \frac{\pi}{M} + s \\ 0 & |\theta| \ge \frac{\pi}{M} + s \end{cases} \tag{D1}$$

where $f(\theta) = a + b\cos\theta + c\sin|\theta|$. Continuity of $p(\theta)$ gives $f\left(\frac{\pi}{M} - s\right) = 1$ and $f\left(\frac{\pi}{M} + s\right) = 0$. Normalisation of $p(\theta)$ in the interval $\frac{\pi}{M} - s \le \theta \le \frac{\pi}{M} + s$ requires that $f(\theta) + f\left(\frac{2\pi}{M} - \theta\right) = 1$. These yield $f(\theta)$ in the form

$$f(\theta) = \frac{1}{2} + \frac{1}{2}\frac{\sin\left(\frac{\pi}{M} - \theta\right)}{\sin s} \tag{D2}$$

$D_1 + D_2$ must be stationary w.r.t. variation of $p(\theta)$ in the interval $\frac{\pi}{M} - s \le \theta \le \frac{\pi}{M} + s$, which yields the condition

$$\begin{aligned} 0 =& r\csc^2 s\ \sin\left(\frac{\pi}{M}\right)\sin\left(\frac{\pi}{M} - \theta\right)\left(\sin s - \sin\left(\frac{\pi}{M} - \theta\right)\right) \\ & \times\left(n\sin s - (n-1)\ r\sin\left(\frac{\pi}{M}\right)\right) \end{aligned} \tag{D3}$$

$$0 = r\ \csc^2 s\ \sin\left(\frac{\pi}{M}\right)\ \sin\left(\frac{\pi}{M} - \theta\right)\ \left(\sin s - \sin\left(\frac{\pi}{M} - \theta\right)\right)\ \left(n\ \sin s - (n-1)\ r\ \sin\left(\frac{\pi}{M}\right)\right) \tag{D4}$$

which gives the optimum solution for r as

$$r = \frac{n}{n-1}\frac{\sin s}{\sin\left(\frac{\pi}{M}\right)} \tag{D5}$$

$D_1 + D_2$ must be stationary w.r.t. variation of r. This yields a transcendental equation that must be satisfied by the optimum solution for s as

$$\frac{\sin s}{\sin\left(\frac{\pi}{M}\right)} - \frac{n-1}{n}\frac{M}{\pi}\sin\left(\frac{\pi}{M}\right)(\cos s + s\ \sin s) = 0 \tag{D6}$$

D_1 and D_2 may be written out in full as (using $\mathbf{n}(\theta) \equiv (\cos\theta,\, \sin\theta)$)

$$D_1 = \frac{2M}{n\,\pi}\left(\int_0^{\frac{\pi}{M}-s} d\theta\ \|\mathbf{n}(\theta) - r\,\mathbf{n}(0)\|^2 + \int_{\frac{\pi}{M}-s}^{\frac{\pi}{M}} d\theta\, f(\theta)\ \|\mathbf{n}(\theta) - r\,\mathbf{n}(0)\|^2 + \int_{\frac{\pi}{M}-s}^{\frac{\pi}{M}} d\theta\, f\left(\frac{2\pi}{M} - \theta\right)\ \left\|\mathbf{n}(\theta) - r\,\mathbf{n}\left(\frac{2\pi}{M}\right)\right\|^2\right) \tag{D7}$$

$$D_2 = \frac{2\,(n-1)\,M}{n\,\pi}\left(\int_0^{\frac{\pi}{M}-s} d\theta\,\left\|\mathbf{n}(\theta) - r\,\mathbf{n}(0)\right\|^2 + \int_{\frac{\pi}{M}-s}^{\frac{\pi}{M}} d\theta\,\left\|\mathbf{n}(\theta) - r\,f(\theta)\,\mathbf{n}(0) - r\,f\left(\frac{2\pi}{M}-\theta\right)\mathbf{n}\left(\frac{2\pi}{M}\right)\right\|^2\right) \tag{D8}$$

The optimum $f(\theta)$ and r may be substituted into $D_1 + D_2$, the integrations evaluated, and then the condition that the optimum s must satisfy may be used to simplify the result, to yield the minimum $D_1 + D_2$ as

$$D_1 + D_2 = 2 - \frac{n}{n-1}\,\frac{M}{2\pi}\,\left(2s + \sin(2s)\right) \tag{D9}$$

2. Circular Manifold: 3 Overlapping Posterior Probabilities

For $\frac{\pi}{M} \leq s \leq \frac{2\pi}{M}$ the functional form of $p(\theta)$ that ensures a piecewise linear $\Pr(y|\mathbf{x})$ is

$$p(\theta) = \begin{cases} f_1(\theta) & 0 \leq |\theta| \leq -\frac{\pi}{M} + s \\ f_2(\theta) & -\frac{\pi}{M} + s \leq |\theta| \leq \frac{3\pi}{M} - s \\ f_3(\theta) & \frac{3\pi}{M} - s \leq |\theta| \leq \frac{\pi}{M} + s \\ 0 & |\theta| \geq \frac{\pi}{M} + s \end{cases} \tag{D10}$$

where $f_i(\theta) = a_i + b_i\cos\theta + c_i\sin|\theta|$ for $i = 1,2,3$. Continuity of $p(\theta)$ gives $f_1\left(-\frac{\pi}{M}+s\right) = f_2\left(-\frac{\pi}{M}+s\right)$, $f_2\left(\frac{3\pi}{M}-s\right) = f_3\left(\frac{3\pi}{M}-s\right)$ and $f_3\left(\frac{2\pi}{M}+s\right) = 0$. Normalisation of $p(\theta)$ in the interval $0 \leq \theta \leq -\frac{\pi}{M}+s$ requires that $f_1(\theta) + f_3\left(\frac{2\pi}{M}+\theta\right) + f_3\left(\frac{2\pi}{M}-\theta\right) = 1$, and normalisation of $p(\theta)$ in the interval $-\frac{\pi}{M}+s \leq \theta \leq \frac{3\pi}{M}-s$ requires that $f_2(\theta) + f_2\left(\frac{2\pi}{M}-\theta\right) = 1$. These conditions may be used to eliminate all but a pair of parameters in the $f_i(\theta)$, which may thus be written in the form

$$\begin{aligned} f_1(\theta) &= \frac{1}{2}\cos(\theta)\sec\left(\frac{\pi}{M}-s\right) + a_1\left(1-\cos(\theta)\sec\left(\frac{\pi}{M}-s\right)\right) + b_2\cos(\theta)\csc\left(\frac{\pi}{M}\right)\sin\left(\frac{2\pi}{M}-s\right)\sec\left(\frac{\pi}{M}-s\right) \\ f_2(\theta) &= \frac{1}{2} + b_2\left(\cos(\theta) - \cot\left(\frac{\pi}{M}\right)\sin(\theta)\right) \\ f_3(\theta) &= \frac{1}{2}\left(1-\csc\left(\frac{2\pi}{M}-2s\right)\sin\left(\frac{3\pi}{M}-s-\theta\right)\right) + \frac{1}{2}a_1\left(\cos\left(\frac{2\pi}{M}-\theta\right)\sec\left(\frac{\pi}{M}-s\right)-1\right) \\ &\quad + b_2\csc\left(\frac{\pi}{M}\right)\csc\left(\frac{2\pi}{M}-2s\right)\sin\left(\frac{2\pi}{M}-s\right)\sin\left(\frac{\pi}{M}+s-\theta\right) \end{aligned} \tag{D11}$$

$D_1 + D_2$ must be stationary w.r.t. variation of $p(\theta)$ in each of the 3 intervals $0 \leq \theta \leq -\frac{\pi}{M}+s$ (interval 1), $-\frac{\pi}{M}+s \leq \theta \leq \frac{3\pi}{M}-s$ (interval 2), and $\frac{3\pi}{M}-s \leq \theta \leq \frac{\pi}{M}+s$ (interval 3). The Fourier transform w.r.t. θ of each of these 3 stationarity conditions has 5 terms with basis functions $(1, \cos\theta, \sin\theta, \cos 2\theta, \sin 2\theta)$, and each of the total of 15 Fourier coefficients must be zero. There are only 3 free parameters a_1, b_2 and r, so only 3 of the 15 are actually independent; the particular 3 that are used are selected on the basis of ease of solution for the free parameters a_1, b_2 and r. The coefficient of the $\cos 2\theta$ term in interval 2 yields

$$b_2\,r\,\left(n + 2\,b_2\,r - 2\,b_2 r\,n\right)\cos\left(\frac{2\pi}{M}\right) = 0 \tag{D12}$$

which has the solution

$$b_2 = \frac{n}{2\,(n-1)\,r} \tag{D13}$$

which may be substituted back into the coefficient of the $\cos\theta$ term in interval 1 to yield

$$\begin{aligned} 0 &= r\sec\left(\frac{\pi}{M}-s\right)\sin\left(\frac{\pi}{M}\right) \\ &\quad\times\left((n-1)\left(-6\,a_1^2 + 7\,a_1 - 2\right)r\sin\left(\frac{\pi}{M}\right) + (n-1)\left(2\,a_1^2 - 3\,a_1 + 1\right)r\sin\left(\frac{3\pi}{M}\right)\right. \\ &\qquad\left. + n\left(a_1\sin\left(\frac{2\pi}{M}-s\right) + (1-a_1)\sin\left(\frac{4\pi}{M}-s\right)\right)\right) \end{aligned} \tag{D14}$$

and also substituted back into the coefficient of the $\sin\theta$ term in interval 3 to yield

$$\begin{aligned} 0 &= r\,\cos\left(\frac{\pi}{M}\right)\,\csc\left(\frac{\pi}{M}-s\right)\,\sec\left(\frac{\pi}{M}-s\right)\,\sin^2\left(\frac{\pi}{M}\right) \\ &\times\left(\begin{array}{l} -(n-1)\,r\left(\begin{array}{l} -2a_1\,(3\,a_1-2)\,\cos\left(\frac{2\pi}{M}-s\right) \\ \quad -2\,(a_1-1)\,a_1\,\cos\left(\frac{2\pi}{M}+s\right) \\ \quad\quad +\left(1-2a_1+2\,a_1^2\right)\,\cos\left(\frac{4\pi}{M}-s\right) \\ \quad\quad\quad +\left(1-4a_1+6\,a_1^2\right)\,\cos(s) \end{array}\right) \\ \quad +n\left(\begin{array}{l} (a_1+1)\,\cos\left(\frac{\pi}{M}\right)-(a_1-1)\,\cos\left(\frac{3\pi}{M}\right) \\ \quad -2a_1\,\sin\left(\frac{\pi}{M}\right)\,\sin\left(\frac{4\pi}{M}-2s\right) \end{array}\right) \end{array}\right) \end{aligned} \tag{D15}$$

These two conditions may be solved for a_1 and r to yield

$$a_1=\frac{\cos\left(\frac{2\pi}{M}\right)}{\cos\left(\frac{2\pi}{M}\right)-1} \tag{D16}$$

and

$$r=\frac{n}{n-1}\,\cos\left(\frac{2\pi}{M}-s\right)\,\sec\left(\frac{\pi}{M}\right) \tag{D17}$$

The solutions for a_1 and b_2 may be substituted back into the expressions for the $f_i(\theta)$ to reduce them to the form

$$\begin{aligned} f_1(\theta) &= -\frac{1}{4}\left(\cos\left(\frac{4\pi}{M}-s\right)+\cos s-2\,\cos\left(\frac{\pi}{M}\right)\,\cos\theta\right)\,\csc^2\left(\frac{\pi}{M}\right)\,\sec\left(\frac{2\pi}{M}-s\right) \\ f_2(\theta) &= \frac{1}{2}\left(\cot\left(\frac{\pi}{M}\right)\,\sec\left(\frac{2\pi}{M}-s\right)\,\sin\left(\frac{\pi}{M}-\theta\right)+1\right) \\ f_3(\theta) &= -\frac{1}{4}\,\csc^2\left(\frac{\pi}{M}\right)\,\left(\cos\left(\frac{3\pi}{M}-\theta\right)\,\sec\left(\frac{2\pi}{M}-s\right)-1\right) \end{aligned} \tag{D18}$$

D_1+D_2 must be stationary w.r.t. variation of r. This yields a transcendental equation that must be satisfied by the optimum solution for s as

$$\frac{1}{n}\,\frac{\cos\left(\frac{2\pi}{M}-s\right)}{\cos\left(\frac{\pi}{M}\right)}-\frac{n-1}{n}\,\frac{M}{\pi}\,\cos\left(\frac{\pi}{M}\right)\,\left(\sin\left(\frac{2\pi}{M}-s\right)-\left(\frac{2\pi}{M}-s\right)\,\cos\left(\frac{2\pi}{M}-s\right)\right)=0 \tag{D19}$$

D_1 and D_2 may be written out in full as

$$D_1=\frac{M}{n\,\pi}\left(\begin{array}{l} \int_0^{-\frac{\pi}{M}+s}d\theta\left(\begin{array}{l} f_1(\theta)\,\left\|\mathbf{n}(\theta)-r\,\mathbf{n}(0)\right\|^2 \\ \quad +f_3\left(\frac{2\pi}{M}-\theta\right)\,\left\|\mathbf{n}(\theta)-r\,\mathbf{n}\left(\frac{2\pi}{M}\right)\right\|^2 \\ \quad\quad +f_3\left(\frac{2\pi}{M}+\theta\right)\,\left\|\mathbf{n}(\theta)-r\,\mathbf{n}\left(-\frac{2\pi}{M}\right)\right\|^2 \end{array}\right) \\ \quad +\int_{-\frac{\pi}{M}+s}^{\frac{3\pi}{M}-s}d\theta\left(\begin{array}{l} f_2(\theta)\,\left\|\mathbf{n}(\theta)-r\,\mathbf{n}(0)\right\|^2 \\ \quad +f_2\left(\frac{2\pi}{M}-\theta\right)\,\left\|\mathbf{n}(\theta)-r\,\mathbf{n}\left(\frac{2\pi}{M}\right)\right\|^2 \end{array}\right) \\ \quad\quad +\int_{\frac{3\pi}{M}-s}^{\frac{2\pi}{M}}d\theta\left(\begin{array}{l} f_3(\theta)\,\left\|\mathbf{n}(\theta)-r\,\mathbf{n}(0)\right\|^2 \\ \quad +f_1\left(\frac{2\pi}{M}-\theta\right)\,\left\|\mathbf{n}(\theta)-r\,\mathbf{n}\left(\frac{2\pi}{M}\right)\right\|^2 \\ \quad\quad +f_3\left(\frac{4\pi}{M}-\theta\right)\,\left\|\mathbf{n}(\theta)-r\,\mathbf{n}\left(\frac{4\pi}{M}\right)\right\|^2 \end{array}\right) \end{array}\right) \tag{D20}$$

$$D_2=\frac{(n-1)\,M}{n\,\pi}\left(\begin{array}{l} \int_0^{-\frac{\pi}{M}+s}d\theta\left\|\begin{array}{l} \mathbf{n}(\theta)-f_1(\theta)\,r\,\mathbf{n}(0) \\ \quad -f_3\left(\frac{2\pi}{M}-\theta\right)\,r\,\mathbf{n}\left(\frac{2\pi}{M}\right) \\ \quad\quad -f_3\left(\frac{2\pi}{M}+\theta\right)\,r\,\mathbf{n}\left(-\frac{2\pi}{M}\right) \end{array}\right\|^2 \\ \quad +\int_{-\frac{\pi}{M}+s}^{\frac{3\pi}{M}-s}d\theta\left\|\begin{array}{l} \mathbf{n}(\theta)-f_2(\theta)\,r\,\mathbf{n}(0) \\ \quad -f_2\left(\frac{2\pi}{M}-\theta\right)\,r\,\mathbf{n}\left(\frac{2\pi}{M}\right) \end{array}\right\|^2 \\ \quad\quad +\int_{\frac{3\pi}{M}-s}^{\frac{2\pi}{M}}d\theta\left\|\begin{array}{l} \mathbf{n}(\theta)-f_3(\theta)\,r\,\mathbf{n}(0) \\ \quad -f_1\left(\frac{2\pi}{M}-\theta\right)\,r\,\mathbf{n}\left(\frac{2\pi}{M}\right) \\ \quad\quad -f_3\left(\frac{4\pi}{M}-\theta\right)\,r\,\mathbf{n}\left(\frac{4\pi}{M}\right) \end{array}\right\|^2 \end{array}\right) \tag{D21}$$

The optimum $f_i(\theta)$ and r may be substituted into D_1+D_2, the integrations evaluated, and then the condition that the optimum s must satisfy may be used to simplify the result, to yield the minimum D_1+D_2 as

$$D_1+D_2=\frac{n\left((n-1)\left(2\,\frac{n-2}{n}-\frac{M}{\pi}\,s\right)-\sec^2\left(\frac{\pi}{M}\right)\right)}{2\,(n-1)^2}-\frac{n\left((n-1)\left(2-\frac{M}{\pi}\,s\right)+\sec^2\left(\frac{\pi}{M}\right)\right)}{2\,(n-1)^2}\cos\left(\frac{4\pi}{M}-2s\right) \tag{D22}$$

3. Toroidal Manifold: 2 Overlapping Posterior Probabilities

For $0\leq s\leq\frac{2\pi}{M}$ the functional form of $p(\theta)$ may be obtained directly from the circular case with the replacement $M\to\frac{M}{2}$, so that

$$p(\theta)=\begin{cases}1 & 0\leq|\theta|\leq\frac{2\pi}{M}-s\\ f(\theta) & \frac{2\pi}{M}-s\leq|\theta|\leq\frac{2\pi}{M}+s\\ 0 & |\theta|\geq\frac{2\pi}{M}+s\end{cases} \tag{D23}$$

$$f(\theta)=\frac{1}{2}+\frac{1}{2}\,\frac{\sin\left(\frac{2\pi}{M}-\theta\right)}{\sin s} \tag{D24}$$

D_1+D_2 must be stationary w.r.t. variation of $p(\theta)$ in the interval $\frac{2\pi}{M}-s\leq\theta\leq\frac{2\pi}{M}+s$, which yields the condition

$$0=r\,\csc^2 s\,\sin\left(\frac{2\pi}{M}\right)\sin\left(\frac{2\pi}{M}-\theta\right)\left(\sin s-\sin\left(\frac{2\pi}{M}-\theta\right)\right)\left(2\,n\,\sin s-(n-1)\,r\sin\left(\frac{2\pi}{M}\right)\right) \tag{D25}$$

which has the same form as the circular case with the replacements $M\to\frac{M}{2}$ and $n\to\frac{2n}{n+1}$, which gives the optimum solution for r as

$$r=\frac{2n}{n-1}\,\frac{\sin s}{\sin\left(\frac{2\pi}{M}\right)} \tag{D26}$$

D_1+D_2 must be stationary w.r.t. variation of r. This yields a transcendental equation that must be satisfied by the optimum solution for s as

$$\frac{\sin s}{\sin\left(\frac{2\pi}{M}\right)}-\frac{n-1}{n+1}\,\frac{M}{2\pi}\,\sin\left(\frac{2\pi}{M}\right)(\cos s+s\,\sin s)=0 \tag{D27}$$

which has the same form as the circular case with the replacements $M\to\frac{M}{2}$ and $n\to\frac{n+1}{2}$. D_1 and D_2 may be written out in full as

$$D_1=\frac{M}{n\,\pi}\left(\begin{array}{l}\int_0^{\frac{2\pi}{M}-s}d\theta\,\left(1+\|\mathbf{n}(\theta)-r\,\mathbf{n}(0)\|^2\right)\\ \qquad+\int_{\frac{2\pi}{M}-s}^{\frac{2\pi}{M}}d\theta\,f(\theta)\,\left(1+\|\mathbf{n}(\theta)-r\,\mathbf{n}(0)\|^2\right)\\ \qquad\qquad+\int_{\frac{2\pi}{M}-s}^{\frac{2\pi}{M}}d\theta\,f\left(\frac{4\pi}{M}-\theta\right)\,\left(1+\left\|\mathbf{n}(\theta)-r\,\mathbf{n}\left(\frac{4\pi}{M}\right)\right\|^2\right)\end{array}\right) \tag{D28}$$

$$D_2=\frac{2\,(n-1)\,M}{n\,\pi}\left(\int_0^{\frac{2\pi}{M}-s}d\theta\,\left\|\mathbf{n}(\theta)-\frac{1}{2}r\,\mathbf{n}(0)\right\|^2+\int_{\frac{2\pi}{M}-s}^{\frac{2\pi}{M}}d\theta\,\left\|\mathbf{n}(\theta)-\frac{1}{2}r\,f(\theta)\,\mathbf{n}(0)-\frac{1}{2}r\,f\left(\frac{4\pi}{M}-\theta\right)\,\mathbf{n}\left(\frac{4\pi}{M}\right)\right\|^2\right) \tag{D29}$$

The optimum $f(\theta)$ and r may be substituted into D_1+D_2, the integrations evaluated, and then the condition that the optimum s must satisfy may be used to simplify the result, to yield the minimum D_1+D_2 as

$$D_1+D_2=4-\frac{n}{n-1}\,\frac{M}{2\pi}\,(2s+\sin(2s)) \tag{D30}$$

which has the same form as the circular case plus an extra contribution of 2.

4. Toroidal Manifold: 3 Overlapping Posterior Probabilities

For $\frac{\pi}{M} \leq s \leq \frac{2\pi}{M}$ the functional form of $p(\theta)$ may be obtained directly from the circular case with the replacement $M \to \frac{M}{2}$, so that

$$p(\theta) = \begin{cases} f_1(\theta) & 0 \leq |\theta| \leq -\frac{2\pi}{M} + s \\ f_2(\theta) & -\frac{2\pi}{M} + s \leq |\theta| \leq \frac{6\pi}{M} - s \\ f_3(\theta) & \frac{6\pi}{M} - s \leq |\theta| \leq \frac{2\pi}{M} + s \\ 0 & |\theta| \geq \frac{2\pi}{M} + s \end{cases} \tag{D31}$$

$$\begin{aligned} f_1(\theta) &= \frac{1}{2}\cos(\theta)\sec\left(\frac{2\pi}{M} - s\right) + a_1\left(1 - \cos(\theta)\sec\left(\frac{2\pi}{M} - s\right)\right) + b_2\cos(\theta)\csc\left(\frac{2\pi}{M}\right)\sin\left(\frac{4\pi}{M} - s\right)\sec\left(\frac{2\pi}{M} - s\right) \\ f_2(\theta) &= \frac{1}{2} + b_2\left(\cos(\theta) - \cot\left(\frac{2\pi}{M}\right)\sin(\theta)\right) \\ f_3(\theta) &= \frac{1}{2}\left(1 - \csc\left(\frac{4\pi}{M} - 2s\right)\sin\left(\frac{6\pi}{M} - s - \theta\right)\right) + \frac{1}{2}a_1\left(\cos\left(\frac{4\pi}{M} - \theta\right)\sec\left(\frac{2\pi}{M} - s\right) - 1\right) \\ &\quad + b_2\csc\left(\frac{2\pi}{M}\right)\csc\left(\frac{4\pi}{M} - 2s\right)\sin\left(\frac{4\pi}{M} - s\right)\sin\left(\frac{2\pi}{M} + s - \theta\right) \end{aligned} \tag{D32}$$

$D_1 + D_2$ must be stationary w.r.t. variation of $p(\theta)$ in each of the 3 intervals $0 \leq \theta \leq -\frac{2\pi}{M} + s$ (interval 1), $-\frac{2\pi}{M} + s \leq \theta \leq \frac{6\pi}{M} - s$ (interval 2), and $\frac{6\pi}{M} - s \leq \theta \leq \frac{2\pi}{M} + s$ (interval 3). The coefficient of the $\cos 2\theta$ term in interval 2 yields

$$b_2\, r\,(n + b_2\, r - b_2\, r\, n)\cos\left(\frac{4\pi}{M}\right) = 0 \tag{D33}$$

which has the same form as the circular case with the replacements $M \to \frac{M}{2}$ and $n \to \frac{2n}{n+1}$, which has the solution

$$b_2 = \frac{n}{(n-1)\, r} \tag{D34}$$

which may be substituted back into the coefficient of the $\cos\theta$ term in interval 1 to yield

$$0 = r\sec\left(\frac{2\pi}{M} - s\right)\sin\left(\frac{2\pi}{M}\right)\left(\begin{array}{l} (n-1)\left(-6a_1^2 + 7a_1 - 2\right) r\sin\left(\frac{2\pi}{M}\right) \\ \quad + (n-1)\left(2a_1^2 - 3a_1 + 1\right) r\sin\left(\frac{6\pi}{M}\right) \\ \quad\quad + 2n\left(a_1\sin\left(\frac{4\pi}{M} - s\right) + (1 - a_1)\sin\left(\frac{8\pi}{M} - s\right)\right) \end{array}\right) \tag{D35}$$

and also substituted back into the coefficient of the $\sin\theta$ term in interval 3 to yield

$$\begin{aligned} 0 &= r\cos\left(\frac{2\pi}{M}\right)\csc\left(\frac{2\pi}{M} - s\right)\sec\left(\frac{2\pi}{M} - s\right)\sin^2\left(\frac{2\pi}{M}\right) \\ &\times\left(\begin{array}{l} -(n-1)\, r\left(\begin{array}{l} -2a_1(3a_1 - 2)\cos\left(\frac{4\pi}{M} - s\right) \\ \quad -2(a_1 - 1)\, a_1\cos\left(\frac{4\pi}{M} + s\right) \\ \quad\quad + \left(1 - 2a_1 + 2a_1^2\right)\cos\left(\frac{8\pi}{M} - s\right) \\ \quad\quad\quad + \left(1 - 4a_1 + 6a_1^2\right)\cos(s) \end{array}\right) \\ \quad + 2n\left(\begin{array}{l} (a_1 + 1)\cos\left(\frac{2\pi}{M}\right) - (a_1 - 1)\cos\left(\frac{6\pi}{M}\right) \\ \quad -2a_1\sin\left(\frac{2\pi}{M}\right)\sin\left(\frac{8\pi}{M} - 2s\right) \end{array}\right) \end{array}\right) \end{aligned} \tag{D36}$$

both of which have the same form as the circular case with the replacements $M \to \frac{M}{2}$ and $n \to \frac{2n}{n+1}$. These two conditions may be solved for a_1 and r to yield

$$a_1 = \frac{\cos\left(\frac{4\pi}{M}\right)}{\cos\left(\frac{4\pi}{M}\right) - 1} \tag{D37}$$

and

$$r = \frac{2n}{n-1}\cos\left(\frac{4\pi}{M} - s\right)\sec\left(\frac{2\pi}{M}\right) \tag{D38}$$

The solutions for a_1 and b_2 may be substituted back into the expressions for the $f_i\left(\theta\right)$ to reduce them to the form

$$\begin{aligned}
f_1\left(\theta\right) &= -\frac{1}{4}\left(\cos\left(\frac{8\pi}{M}-s\right)+\cos s-2\cos\left(\frac{2\pi}{M}\right)\cos\theta\right)\csc^2\left(\frac{2\pi}{M}\right)\sec\left(\frac{4\pi}{M}-s\right)\\
f_2\left(\theta\right) &= \frac{1}{2}\left(\cot\left(\frac{2\pi}{M}\right)\sec\left(\frac{4\pi}{M}-s\right)\sin\left(\frac{2\pi}{M}-\theta\right)+1\right)\\
f_3\left(\theta\right) &= -\frac{1}{4}\csc^2\left(\frac{2\pi}{M}\right)\left(\cos\left(\frac{6\pi}{M}-\theta\right)\sec\left(\frac{6\pi}{M}-s\right)-1\right)
\end{aligned} \tag{D39}$$

which have the same form as the circular case with the replacement $M \to \frac{M}{2}$. $D_1 + D_2$ must be stationary w.r.t. variation of r. This yields a transcendental equation that must be satisfied by the optimum solution for s as

$$\frac{1}{n}\,\frac{\cos\left(\frac{4\pi}{M}-s\right)}{\cos\left(\frac{2\pi}{M}\right)}-\frac{n-1}{2n}\,\frac{M}{2\pi}\cos\left(\frac{2\pi}{M}\right)\left(\sin\left(\frac{4\pi}{M}-s\right)-\left(\frac{4\pi}{M}-s\right)\cos\left(\frac{4\pi}{M}-s\right)\right)=0 \tag{D40}$$

which has the same form as the circular case with the replacements $M \to \frac{M}{2}$ and $n \to \frac{n+1}{2}$. D_1 and D_2 may be written out in full as

$$D_1=\frac{M}{2\,n\,\pi}\left(\begin{array}{l}
\int_0^{-\frac{2\pi}{M}+s}d\theta\left(\begin{array}{l}
f_1\left(\theta\right)\left(1+\left\|\mathbf{n}\left(\theta\right)-r\,\mathbf{n}\left(0\right)\right\|^2\right)\\
\quad +f_3\left(\frac{4\pi}{M}-\theta\right)\left(1+\left\|\mathbf{n}\left(\theta\right)-r\,\mathbf{n}\left(\frac{4\pi}{M}\right)\right\|^2\right)\\
\quad\quad +f_3\left(\frac{4\pi}{M}+\theta\right)\left(1+\left\|\mathbf{n}\left(\theta\right)-r\,\mathbf{n}\left(-\frac{4\pi}{M}\right)\right\|^2\right)
\end{array}\right)\\
+\int_{-\frac{2\pi}{M}+s}^{\frac{6\pi}{M}-s}d\theta\left(\begin{array}{l}
f_2\left(\theta\right)\left(1+\left\|\mathbf{n}\left(\theta\right)-r\,\mathbf{n}\left(0\right)\right\|^2\right)\\
\quad +f_2\left(\frac{4\pi}{M}-\theta\right)\left(1+\left\|\mathbf{n}\left(\theta\right)-r\,\mathbf{n}\left(\frac{4\pi}{M}\right)\right\|^2\right)
\end{array}\right)\\
+\int_{\frac{6\pi}{M}-s}^{\frac{4\pi}{M}}d\theta\left(\begin{array}{l}
f_3\left(\theta\right)\left(1+\left\|\mathbf{n}\left(\theta\right)-r\,\mathbf{n}\left(0\right)\right\|^2\right)\\
\quad +f_1\left(\frac{4\pi}{M}-\theta\right)\left(1+\left\|\mathbf{n}\left(\theta\right)-r\,\mathbf{n}\left(\frac{4\pi}{M}\right)\right\|^2\right)\\
\quad\quad +f_3\left(\frac{8\pi}{M}-\theta\right)\left(1+\left\|\mathbf{n}\left(\theta\right)-r\,\mathbf{n}\left(\frac{8\pi}{M}\right)\right\|^2\right)
\end{array}\right)
\end{array}\right) \tag{D41}$$

$$D_2=\frac{\left(n-1\right)M}{n\,\pi}\left(\begin{array}{l}
\int_0^{-\frac{2\pi}{M}+s}d\theta\left\|\begin{array}{l}
\mathbf{n}\left(\theta\right)-\frac{1}{2}f_1\left(\theta\right)r\,\mathbf{n}\left(0\right)\\
\quad -\frac{1}{2}f_3\left(\frac{4\pi}{M}-\theta\right)r\,\mathbf{n}\left(\frac{4\pi}{M}\right)\\
\quad\quad -\frac{1}{2}f_3\left(\frac{4\pi}{M}+\theta\right)r\,\mathbf{n}\left(-\frac{4\pi}{M}\right)
\end{array}\right\|^2\\
+\int_{-\frac{2\pi}{M}+s}^{\frac{6\pi}{M}-s}d\theta\left\|\begin{array}{l}
\mathbf{n}\left(\theta\right)-\frac{1}{2}f_2\left(\theta\right)r\,\mathbf{n}\left(0\right)\\
\quad -\frac{1}{2}f_2\left(\frac{4\pi}{M}-\theta\right)r\,\mathbf{n}\left(\frac{4\pi}{M}\right)
\end{array}\right\|^2\\
+\int_{\frac{6\pi}{M}-s}^{\frac{4\pi}{M}}d\theta\left\|\begin{array}{l}
\mathbf{n}\left(\theta\right)-\frac{1}{2}f_3\left(\theta\right)r\,\mathbf{n}\left(0\right)\\
\quad -\frac{1}{2}f_1\left(\frac{4\pi}{M}-\theta\right)r\,\mathbf{n}\left(\frac{4\pi}{M}\right)\\
\quad\quad -\frac{1}{2}f_3\left(\frac{8\pi}{M}-\theta\right)r\,\mathbf{n}\left(\frac{8\pi}{M}\right)
\end{array}\right\|^2
\end{array}\right) \tag{D42}$$

The optimum $f_i\left(\theta\right)$ and r may be substituted into $D_1 + D_2$, the integrations evaluated, and then the condition that the optimum s must satisfy may be used to simplify the result, to yield the minimum $D_1 + D_2$ as

$$D_1+D_2=\frac{n\left(\left(n-1\right)\left(2\,\frac{n-2}{n}-\frac{M}{2\pi}\,s\right)-2\sec^2\left(\frac{2\pi}{M}\right)\right)}{\left(n-1\right)^2}-\frac{n\left(\left(n-1\right)\left(2-\frac{M}{2\pi}\,s\right)+2\sec^2\left(\frac{2\pi}{M}\right)\right)}{\left(n-1\right)^2}\cos\left(\frac{8\pi}{M}-2s\right) \tag{D43}$$

[1] M Burger, T Graepel, and K Obermayer, *Advances in neural information processing systems*, vol. 10, ch. An annealled self-organising map for source channel coding, pp. 430–436, MIT Press, 1998.

[2] P Dayan, G E Hinton, R M Neal, and R S Zemel, *The Helmholtz machine*, Neural Computation **7** (1995), no. 5, 889–904.

[3] N Farvardin, *A study of vector quantisation for noisy channels*, IEEE Transactions on Information Theory **36** (1990), no. 4, 799–809.

[4] T Graepel, M Burger, and K Obermayer, *Self-organising maps: generalisations and new optimisation techniques*, Neurocomputing **21** (1998), no. 1-3, 173–190.

[5] T Kohonen, *Self-organisation and associative memory*, Springer-Verlag, Berlin, 1984.

[6] H Kumazawa, M Kasahara, and T Namekawa, *A construction of vector quantisers for noisy channels*, Electronics and Engineering in Japan **67** (1984), no. 4, 39–47.

[7] Y Linde, A Buzo, and R M Gray, *An algorithm for vector quantiser design*, IEEE Transactions on Communications **28** (1980), no. 1, 84–95.

[8] S P Luttrell, *Derivation of a class of training algorithms*, IEEE Transactions on Neural Networks **1** (1990), no. 2, 229–232.

[9] ———, *A Bayesian analysis of self-organising maps*, Neural Computation **6** (1994), no. 5, 767–794.

[10] ———, *Partitioned mixture distribution: an adaptive Bayesian network for low-level image processing*, IEE Proceedings on Vision, Image, and Signal Processing **141** (1994), no. 4, 251–260.

[11] ———, *A discrete firing event analysis of the adaptive cluster expansion network*, Network: Computation in Neural Systems **7** (1996), no. 2, 285–290.

[12] ———, *Mathematics of neural networks: models, algorithms and applications*, ch. A theory of self-organising neural networks, pp. 240–244, Kluwer, Boston, 1997.

[13] ———, *Self-organisation of multiple winner-take-all neural networks*, Connection Science **9** (1997), no. 1, 11–30.

[14] J Rissanen, *Modelling by shortest data description*, Automatica **14** (1978), no. 5, 465–471.

[15] C E Shannon and W Weaver, *The mathematical theory of communication*, University of Illinois Press, Urbana, 1949.

[16] C J S Webber, *Self-organisation of transformation-invariant detectors for constituents of perceptual patterns*, Network: Computation in Neural Systems **5** (1994), no. 4, 471–496.

[17] S Wolfram, *The Mathematica book*, Wolfram Media and Cambridge University Press, 1996.

Self-Organising Stochastic Encoders [*]

S P Luttrell
DERA, Malvern

The processing of mega-dimensional data, such as images, scales linearly with image size only if fixed size processing windows are used. It would be very useful to be able to automate the process of sizing and interconnecting the processing windows. A stochastic encoder that is an extension of the standard Linde-Buzo-Gray vector quantiser, called a stochastic vector quantiser (SVQ), includes this required behaviour amongst its emergent properties, because it automatically splits the input space into statistically independent subspaces, which it then separately encodes.

Various optimal SVQs have been obtained, both analytically and numerically. Analytic solutions which demonstrate how the input space is split into independent subspaces may be obtained when an SVQ is used to encode data that lives on a 2-torus (e.g. the superposition of a pair of uncorrelated sinusoids). Many numerical solutions have also been obtained, using both SVQs and chains of linked SVQs: (1) images of multiple independent targets (encoders for single targets emerge), (2) images of multiple correlated targets (various types of encoder for single and multiple targets emerge), (3) superpositions of various waveforms (encoders for the separate waveforms emerge - this is a type of independent component analysis (ICA)), (4) maternal and foetal ECGs (another example of ICA), (5) images of textures (orientation maps and dominance stripes emerge).

Overall, SVQs exhibit a rich variety of self-organising behaviour, which effectively discovers the internal structure of the training data. This should have an immediate impact on "intelligent" computation, because it reduces the need for expert human intervention in the design of data processing algorithms.

I. STOCHASTIC VECTOR QUANTISER

A. Reference

Luttrell S P, 1997, *Mathematics of Neural Networks: Models, Algorithms and Applications*, Kluwer, Ellacott S W, Mason J C and Anderson I J (eds.), A theory of self-organising neural networks, 240-244.

B. Objective Function

1. Mean Euclidean Distortion

$$D = \int d\boldsymbol{x} \, \Pr(\boldsymbol{x}) \sum_{\boldsymbol{y}} \Pr(\boldsymbol{y}|\boldsymbol{x}) \int d\boldsymbol{x}' \, \Pr(\boldsymbol{x}'|\boldsymbol{y}) \, \|\boldsymbol{x}-\boldsymbol{x}'\|^2 \tag{1.1}$$

- Encode then decode: $\boldsymbol{x} \longrightarrow \boldsymbol{y} \longrightarrow \boldsymbol{x}'$.
- $\boldsymbol{x}$ = input vector; $\boldsymbol{y}$ = code; $\boldsymbol{x}'$ = reconstructed vector.
- Code vector $\boldsymbol{y} = (y_1, y_2, \cdots, y_n)$ for $1 \le y_i \le M$.
- $\Pr(\boldsymbol{x})$ = input PDF; $\Pr(\boldsymbol{y}|\boldsymbol{x})$ = stochastic encoder; $\Pr(\boldsymbol{x}'|\boldsymbol{y})$ = stochastic decoder.
- $\|\boldsymbol{x}-\boldsymbol{x}'\|^2$ = Euclidean reconstruction error.

2. Simplify

$$\begin{aligned} D &= 2\int d\boldsymbol{x} \, \Pr(\boldsymbol{x}) \sum_{\boldsymbol{y}} \Pr(\boldsymbol{y}|\boldsymbol{x}) \, \|\boldsymbol{x}-\boldsymbol{x}'(\boldsymbol{y})\|^2 \\ \boldsymbol{x}'(\boldsymbol{y}) &\equiv \int d\boldsymbol{x} \, \Pr(\boldsymbol{x}|\boldsymbol{y}) \, \boldsymbol{x} \\ &= \frac{\int d\boldsymbol{x} \, \Pr(\boldsymbol{x}) \, \Pr(\boldsymbol{y}|\boldsymbol{x}) \, \boldsymbol{x}}{\int d\boldsymbol{x} \, \Pr(\boldsymbol{x}) \, \Pr(\boldsymbol{y}|\boldsymbol{x})} \end{aligned} \tag{1.2}$$

- Do the $\int d\boldsymbol{x} \, \Pr(\boldsymbol{x}|\boldsymbol{y}) \, (\cdots)$ integration.
- $\boldsymbol{x}'(\boldsymbol{y})$ = reconstruction vector.
- $\boldsymbol{x}'(\boldsymbol{y})$ is the solution of $\frac{\partial D}{\partial \boldsymbol{x}'(\boldsymbol{y})} = 0$, so it can be deduced by optimisation.

3. Constrain

$$\begin{aligned} \Pr(\boldsymbol{y}|\boldsymbol{x}) &= \Pr(y_1|\boldsymbol{x}) \, \Pr(y_2|\boldsymbol{x}) \cdots \Pr(y_n|\boldsymbol{x}) \\ \boldsymbol{x}'(\boldsymbol{y}) &= \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}'(y_i) \end{aligned} \tag{1.3}$$

- $\Pr(\boldsymbol{y}|\boldsymbol{x})$ implies the components $(y_1, y_2, \cdots, y_n)$ of $\boldsymbol{y}$ are conditionally independent given $\boldsymbol{x}$.

[*] Typeset in LaTeX on May 9, 2019.
This is archived at http://arxiv.org/abs/1012.4126, 18 Dec 2010.
This unpublished talk was given at the Workshop on Self-Organising Systems - Future Prospects for Computing, 28-29 October 1999, Manchester, UK.

- $\boldsymbol{x}'(\boldsymbol{y})$ implies the reconstruction is a superposition of contributions $\boldsymbol{x}'(y_i)$ for $i = 1, 2, \cdots, n$.
- The stochastic encoder samples n times from the same $\Pr(y|\boldsymbol{x})$.

4. Upper Bound

$$\begin{aligned} D &\leq D_1 + D_2 \\ D_1 &\equiv \frac{2}{n}\int d\boldsymbol{x}\,\Pr(\boldsymbol{x})\sum_{y=1}^{M}\Pr(y|\boldsymbol{x})\,\|\boldsymbol{x}-\boldsymbol{x}'(y)\|^2 \qquad (1.4) \\ D_2 &\equiv \frac{2(n-1)}{n}\int d\boldsymbol{x}\,\Pr(\boldsymbol{x})\left\|\boldsymbol{x}-\sum_{y=1}^{M}\Pr(y|\boldsymbol{x})\,\boldsymbol{x}'(y)\right\|^2 \end{aligned}$$

- D_1 is a stochastic vector quantiser with the vector code $\boldsymbol{y}$ replaced by a scalar code y.
- D_2 is a non-linear (note $\Pr(y|\boldsymbol{x})$) encoder with a superposition term $\sum_{y=1}^{M}\Pr(y|\boldsymbol{x})\,\boldsymbol{x}'(y)$.
- $n \longrightarrow \infty$: the stochastic encoder measures $\Pr(y|\boldsymbol{x})$ accurately and D_2 dominates.
- $n \longrightarrow 1$: the stochastic encoder samples $\Pr(y|\boldsymbol{x})$ poorly and D_1 dominates.

II. ANALYTIC OPTIMISATION

A. References

Luttrell S P, 1999, *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*, 235-263, Springer-Verlag, Sharkey A J C (ed.), Self-organised modular neural networks for encoding data.

Luttrell S P, 1999, An Adaptive Network For Encoding Data Using Piecewise Linear Functions, *Proceedings of the 9th International Conference on Artificial Neural Networks (ICANN99)*, Edinburgh, 7-10 September 1999, 198-203.

B. Stationarity Conditions

1. Stationarity w.r.t. $\boldsymbol{x}'(y)$

$$n\int d\boldsymbol{x}\,\Pr(\boldsymbol{x}|y)\,\boldsymbol{x} = \boldsymbol{x}'(y) + (n-1)\int d\boldsymbol{x}\,\Pr(\boldsymbol{x}|y)\sum_{y'=1}^{M}\Pr(y'|\boldsymbol{x})\,\boldsymbol{x}'(y') \qquad (2.1)$$

- Stationarity condition is $\frac{\partial(D_1+D_2)}{\partial\boldsymbol{x}'(y)} = 0$.

2. Stationarity w.r.t. $\Pr(y'|\boldsymbol{x})$

$$\Pr(\boldsymbol{x})\,\Pr(y|\boldsymbol{x})\sum_{y'=1}^{M}\left(\Pr(y'|\boldsymbol{x}) - \delta_{y,y'}\right)\boldsymbol{x}'(y') . \left(\frac{1}{2}\boldsymbol{x}'(y') - n\,\boldsymbol{x} + (n-1)\sum_{y''=1}^{M}\Pr(y''|\boldsymbol{x})\,\boldsymbol{x}'(y'')\right) = 0 \qquad (2.2)$$

- Stationarity condition is $\frac{\partial(D_1+D_2)}{\partial\log\Pr(y|\boldsymbol{x})} = 0$, subject to $\sum_{y=1}^{M}\Pr(y|\boldsymbol{x}) = 1$.
- 3 types of solution: $\Pr(\boldsymbol{x}) = 0$ (trivial), $\Pr(y|\boldsymbol{x}) = 0$ (ensures $\Pr(y|\boldsymbol{x}) \geq 0$), and $\sum_{y'=1}^{M}\left(\Pr(y'|\boldsymbol{x}) - \delta_{y,y'}\right)(\cdots) = 0$.

C. Circle

1. Input vector uniformly distributed on a circle

$$\begin{aligned} \boldsymbol{x} &= (\cos\theta, \sin\theta) \\ \int d\boldsymbol{x}\,\Pr(\boldsymbol{x})\,(\cdots) &= \frac{1}{2\pi}\int_0^{2\pi} d\theta\,(\cdots) \qquad (2.3) \end{aligned}$$

2. Stochastic encoder PDFs symmetrically arranged around the circle

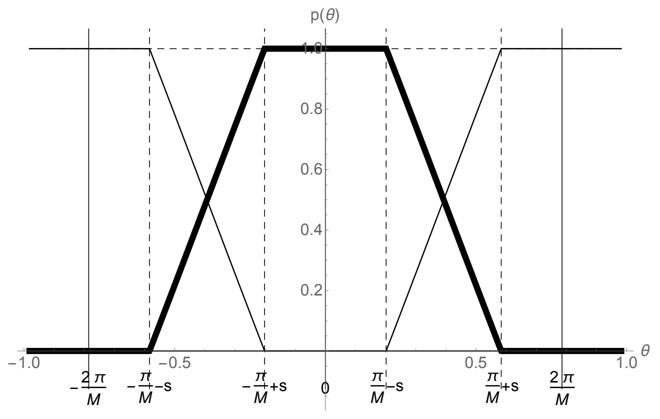
$$\Pr(y|\theta) = p\left(\theta - \frac{2\pi y}{M}\right) \qquad (2.4)$$

3. *Reconstruction vectors symmetrically arraged around the circle*

$$\boldsymbol{x}'(y) = r\left(\cos\left(\frac{2\pi y}{M}\right), \sin\left(\frac{2\pi y}{M}\right)\right) \qquad (2.5)$$

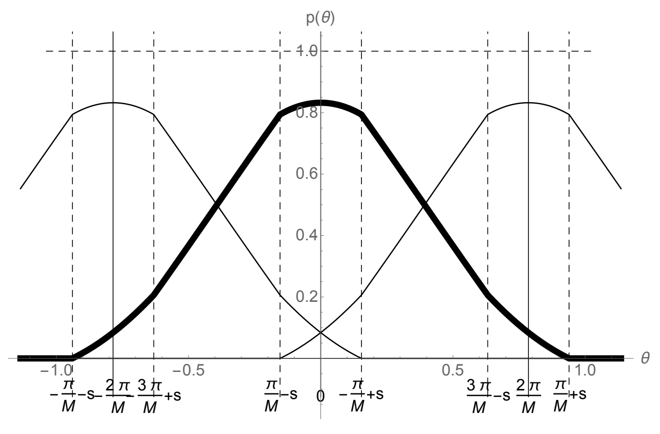
4. *Stochastic encoder PDFs overlap no more than 2 at a time*

$$p(\theta) = \begin{cases} 1 & 0 \le |\theta| \le \frac{\pi}{M} - s \\ f(\theta) & \frac{\pi}{M} - s \le |\theta| \le \frac{\pi}{M} + s \\ 0 & |\theta| \ge \frac{\pi}{M} + s \end{cases} \qquad (2.6)$$



5. *Stochastic encoder PDFs overlap no more than 3 at a time*

$$p(\theta) = \begin{cases} f_1(\theta) & 0 \le |\theta| \le -\frac{\pi}{M} + s \\ f_2(\theta) & -\frac{\pi}{M} + s \le |\theta| \le \frac{3\pi}{M} - s \\ f_3(\theta) & \frac{3\pi}{M} - s \le |\theta| \le \frac{\pi}{M} + s \\ 0 & |\theta| \ge \frac{\pi}{M} + s \end{cases} \qquad (2.7)$$



D. 2-Torus

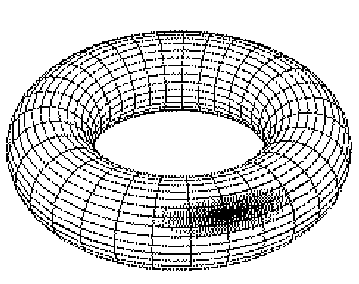
1. *Input vector uniformly distributed on a 2-torus*

$$\begin{aligned} \boldsymbol{x} &= (\boldsymbol{x}_1, \boldsymbol{x}_2) \\ \boldsymbol{x}_1 &= (\cos\theta_1, \sin\theta_1) \\ \boldsymbol{x}_2 &= (\cos\theta_2, \sin\theta_2) \\ \int d\boldsymbol{x}\, \Pr(\boldsymbol{x})(\cdots) &= \frac{1}{4\pi^2}\int_0^{2\pi} d\theta_1 \int_0^{2\pi} d\theta_2\, (\cdots) \end{aligned} \qquad (2.8)$$

2. *Joint encoding*

$$\Pr(y\,|\boldsymbol{x}) = \Pr(y\,|\boldsymbol{x}_1, \boldsymbol{x}_2) \qquad (2.9)$$

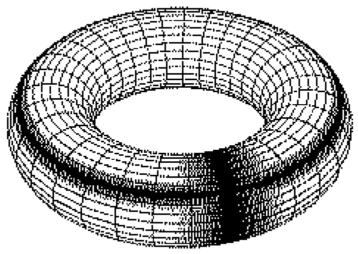
- $\Pr(y\,|\boldsymbol{x})$ depends jointly on $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.
- Requires $n = 1$ to encode $\boldsymbol{x}$.
- For a given resolution the size of the codebook increases exponentially with input dimension.



3. *Factorial encoding*

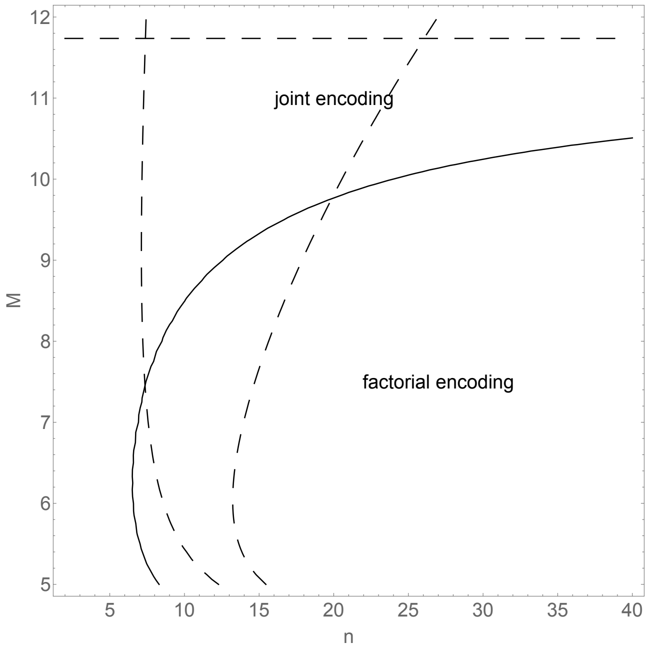
$$\Pr(y\,|\boldsymbol{x}) = \begin{cases} \Pr(y\,|\boldsymbol{x}_1) & y \in Y_1 \\ \Pr(y\,|\boldsymbol{x}_2) & y \in Y_2 \end{cases} \qquad (2.10)$$

- Y_1 and Y_2 are non-intersecting subsets of the allowed values of y.
- $\Pr(y\,|\boldsymbol{x})$ depends either on $\boldsymbol{x}_1$ or on $\boldsymbol{x}_2$, but not on both at the same time.
- Requires $n \gg 1$ to encode $\boldsymbol{x}$.
- For a given resolution the size of the codebook increases linearly with input dimension.



4. Stability diagram

- Fixed n, increasing M: joint encoding is eventually favoured because the size of the codebook is eventually large enough.
- Fixed M, increase n: factorial encoding is eventually favoured because the number of samples is eventually large enough.
- Factorial encoding is encouraged by using a small codebook and sampling a large number of times.



III. NUMERICAL OPTIMISATION

A. References

Luttrell S P, 1997, to appear in *Proceedings of the Conference on Information Theory and the Brain*, Newquay, 20-21 September 1996, The emergence of dominance stripes and orientation maps in a network of firing neurons.

Luttrell S P, 1997, *Mathematics of Neural Networks: Models, Algorithms and Applications*, Kluwer, Ellacott S W, Mason J C and Anderson I J (eds.), A theory of self-organising neural networks, 240-244.

Luttrell S P, 1999, submitted to a special issue of *IEEE Trans. Information Theory on Information-Theoretic Imaging*, Stochastic vector quantisers.

B. Gradient Descent

1. Posterior probability with infinite range neighbourhood

$$\Pr(y\,|\boldsymbol{x}\,) = \frac{Q(\boldsymbol{x}\,|y)}{\sum_{y'=1}^{M} Q(\boldsymbol{x}\,|y'\,)}$$

- $Q(\boldsymbol{x}\,|y) \geq 0$ is needed to ensure a valid $\Pr(y\,|\boldsymbol{x}\,)$.
- This does not restrict $\Pr(y\,|\boldsymbol{x}\,)$ in any way.

2. Posterior probability with finite range neighbourhood

$$\Pr(y\,|\boldsymbol{x};y'\,) \equiv \frac{Q(\boldsymbol{x}\,|y)\;\delta_{y\in\mathcal{N}(y')}}{\sum_{y''=\mathcal{N}(y')} Q(\boldsymbol{x}\,|y''\,)} \tag{3.1}$$

$$\Pr(y\,|\boldsymbol{x}\,) = \frac{1}{M}\sum_{y'=\mathcal{N}^{-1}(y)} \Pr(y\,|\boldsymbol{x};y'\,) = \frac{1}{M}\,Q(\boldsymbol{x}\,|y)\sum_{y'=\mathcal{N}^{-1}(y)} \frac{1}{\sum_{y''=\mathcal{N}(y')} Q(\boldsymbol{x}\,|y''\,)} \tag{3.2}$$

- $\mathcal{N}(y')$ is the set of neurons that lie in a predefined "neighbourhood" of y'.

- $\mathcal{N}^{-1}(y)$ is the "inverse neighbourhood" of y defined as $\mathcal{N}^{-1}(y) \equiv \{y' : y \in \mathcal{N}(y')\}$.
- Neighbourhood is used to introduce "lateral inhibition" between the firing neurons.
- This restricts $\Pr(y|\boldsymbol{x})$, but allows limited range lateral interactions to be used.

3. *Probability leakage*

$$\Pr(y|\boldsymbol{x}) \longrightarrow \sum_{y' \in \mathcal{L}^{-1}(y)} \Pr(y|y') \Pr(y'|\boldsymbol{x}) \tag{3.3}$$

- $\Pr(y|y')$ is the amount of probability that leaks from location y' to location y.
- $\mathcal{L}(y')$ is the "leakage neighbourhood" of y'.
- $\mathcal{L}^{-1}(y)$ is the "inverse leakage neighbourhood" of y defined as $\mathcal{L}^{-1}(y) \equiv \{y' : y \in \mathcal{L}(y')\}$.
- Leakage is to allow the network output to be "damaged" in a controlled way.
- When the network is optimised it automatically becomes robust with respect to such damage.
- Leakage leads to topographic ordering according to the defined neighbourhood.
- This restricts $\Pr(y|\boldsymbol{x})$, but allows topographic ordering to be obtained, and is faster to train.

4. *Shorthand notation*

$$\begin{aligned}
L_{y,y'} &\equiv \Pr(y|y') & P_{y,y'} &\equiv \Pr(y|\boldsymbol{x}; y') \\
p_y &\equiv \sum_{y' \in \mathcal{N}^{-1}(y)} P_{y',y} & \left(L^T p\right)_y &\equiv \sum_{y' \in \mathcal{L}^{-1}(y)} L_{y',y}\, p_{y'} \\
\boldsymbol{d}_y &\equiv \boldsymbol{x} - \boldsymbol{x}'(y) & (L\,\boldsymbol{d})_y &\equiv \sum_{y' \in \mathcal{L}(y)} L_{y',y}\, \boldsymbol{d}_{y'} \\
(P\,L\,\boldsymbol{d})_y &\equiv \sum_{y' \in \mathcal{N}(y)} P_{y,y'}\, (L\,\boldsymbol{d})_{y'} & \left(P^T P\,L\,\boldsymbol{d}\right)_y &\equiv \sum_{y' \in \mathcal{N}^{-1}(y)} P_{y',y}\, (P\,L\,\boldsymbol{d})_{y'} \\
e_y &\equiv \|\boldsymbol{x} - \boldsymbol{x}'(y)\|^2 & (L\,e)_y &\equiv \sum_{y' \in \mathcal{L}(y)} L_{y,y'}\, e_{y'} \\
(P\,L\,e)_y &\equiv \sum_{y' \in \mathcal{N}(y)} P_{y,y'}\, (L\,e)_{y'} & \left(P^T P\,L\,e\right)_y &\equiv \sum_{y' \in N^{-1}(y)} P_{y',y}\, (P\,L\,e)_{y'} \\
\bar{\boldsymbol{d}} &\equiv \sum_{y=1}^{M} \left(L^T p\right)_y \boldsymbol{d}_y & \text{or } \bar{\boldsymbol{d}} &\equiv \sum_{y=1}^{M} (P\,L\,\boldsymbol{d})_y
\end{aligned} \tag{3.4}$$

- This shorthand notation simplifies the appearance of the gradients of D_1 and D_2.
- For instance, $\Pr(y|\boldsymbol{x}) = \frac{1}{M}\left(L^T p\right)_y$.

5. *Derivatives w.r.t.* $\boldsymbol{x}'(y)$

$$\begin{aligned}
\frac{\partial D_1}{\boldsymbol{x}'(y)} &= -\frac{4}{n\,M} \int d\boldsymbol{x}\, \Pr(\boldsymbol{x}) \left(L^T p\right)_y \boldsymbol{d}_y \\
\frac{\partial D_2}{\boldsymbol{x}'(y)} &= -\frac{4(n-1)}{n\,M^2} \int d\boldsymbol{x}\, \Pr(\boldsymbol{x}) \left(L^T p\right)_y \bar{\boldsymbol{d}}
\end{aligned} \tag{3.5}$$

- The extra factor $\frac{1}{M}$ in $\frac{\partial D_2}{\boldsymbol{x}'(y)}$ arises because there is a $\sum_{y=1}^{M}(\cdots)$ hidden inside the $\bar{\boldsymbol{d}}$.

6. *Functional derivatives w.r.t.* $\log Q\left(\boldsymbol{x}\,|y\right)$

$$\begin{aligned}\delta D_1 &= \frac{2}{n\,M}\int d\boldsymbol{x}\;\Pr\left(\boldsymbol{x}\right)\sum_{y=1}^{M}\delta\log Q\left(\boldsymbol{x}\,|y\right)\,\left(p_y\,\left(L\,e\right)_y-\left(P^T\,P\,L\,e\right)_y\right) \\ \delta D_2 &= \frac{4\,(n-1)}{n\,M^2}\int d\boldsymbol{x}\;\Pr\left(\boldsymbol{x}\right)\sum_{y=1}^{M}\delta\log Q\left(\boldsymbol{x}\,|y\right)\,\left(p_y\,\left(L\,\boldsymbol{d}\right)_y-\left(P^T\,P\,L\,\boldsymbol{d}\right)_y\right).\bar{\boldsymbol{d}}\end{aligned} \tag{3.6}$$

- Differentiate w.r.t. $\log Q\left(\boldsymbol{x}\,|y\right)$ because $Q\left(\boldsymbol{x}\,|y\right)\geq 0$.

7. *Neural response model*

$$Q\left(\boldsymbol{x}\,|y\right)=\frac{1}{1+\exp\left(-\boldsymbol{w}\left(y\right).\boldsymbol{x}-b\left(y\right)\right)} \tag{3.7}$$

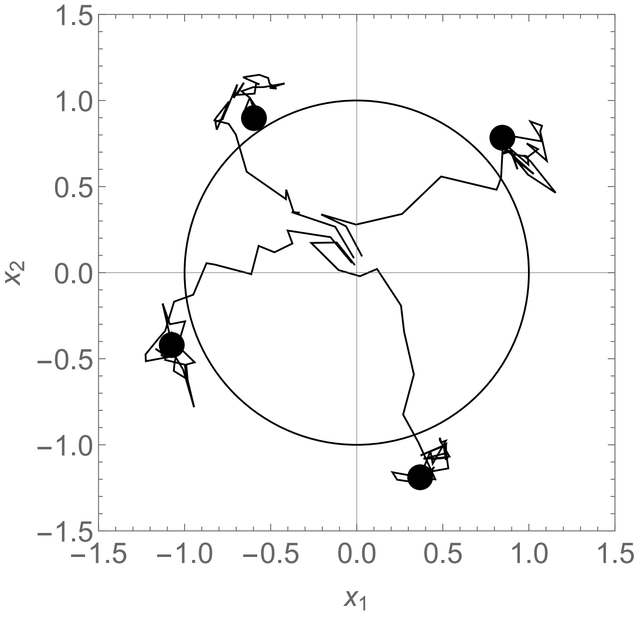
- This is a standard "sigmoid" function.
- This restricts $\Pr\left(y\,|\boldsymbol{x}\right)$, but it is easy to implement, and leads to results similar to the ideal analytic results.

8. *Derivatives w.r.t.* $\boldsymbol{w}\left(y\right)$ *and* $b\left(y\right)$

$$\begin{aligned}\frac{\partial D_1}{\partial\begin{pmatrix}b\left(y\right)\\ \boldsymbol{w}\left(y\right)\end{pmatrix}} &= \frac{2}{n\,M}\int d\boldsymbol{x}\;\Pr\left(\boldsymbol{x}\right)\,\left(p_y\,\left(L\,e\right)_y-\left(P^T\,P\,L\,e\right)_y\right)\,\left(1-Q\left(\boldsymbol{x}\,|y\right)\right)\begin{pmatrix}1\\ \boldsymbol{x}\end{pmatrix} \\ \frac{\partial D_2}{\partial\begin{pmatrix}b\left(y\right)\\ \boldsymbol{w}\left(y\right)\end{pmatrix}} &= \frac{4\,(n-1)}{n\,M^2}\int d\boldsymbol{x}\;\Pr\left(\boldsymbol{x}\right)\,\left(p_y\,\left(L\,\boldsymbol{d}\right)_y-\left(P^T\,P\,L\,\boldsymbol{d}\right)_y\right).\bar{\boldsymbol{d}}\,\left(1-Q\left(\boldsymbol{x}\,|y\right)\right)\begin{pmatrix}1\\ \boldsymbol{x}\end{pmatrix}\end{aligned} \tag{3.8}$$

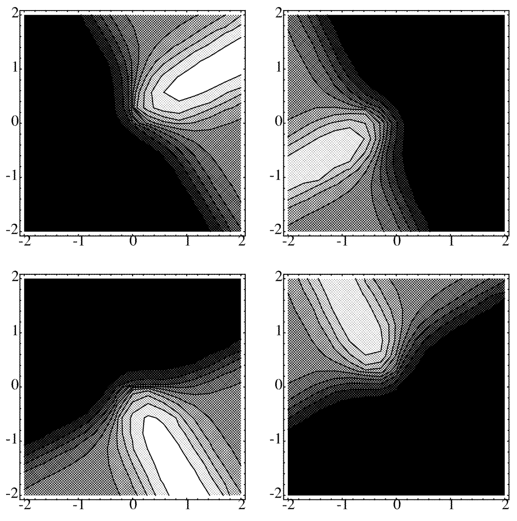
C. Circle

1. *Training history*



- $M=4$ and $n=10$ were used.
- The reference vectors $\boldsymbol{x}'\left(y\right)$ (for $y=1,2,3,4$) are initialised close to the origin.
- The training history leads to stationary $\boldsymbol{x}'\left(y\right)$ just outside the unit circle.

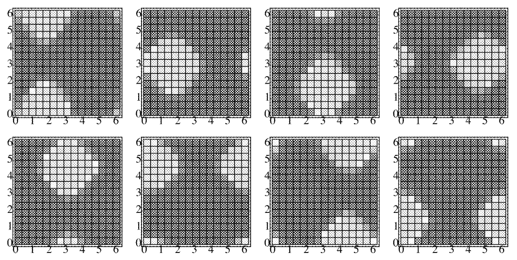
2. *Posterior probabilities*



- Each of the posterior probabilities $\Pr(y\,|\boldsymbol{x})$ (for $y = 1, 2, 3, 4$) is large mainly in a $\frac{\pi}{2}$ radian arc of the circle.
- There is some overlap between the $\Pr(y\,|\boldsymbol{x})$.

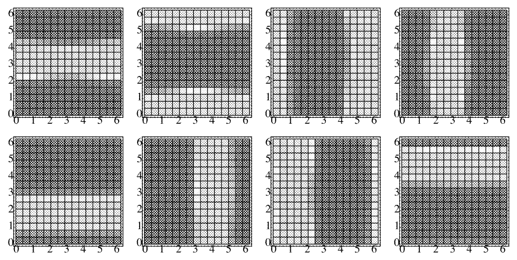
D. 2-Torus

1. *Posterior probabilities: joint encoding*



- $M = 8$ and $n = 5$ were used, which lies inside the joint encoding region of the stability diagram.
- Each of the posterior probabilities $\Pr(y\,|\boldsymbol{x})$ is large mainly in a localised region of the torus.
- There is some overlap between the $\Pr(y\,|\boldsymbol{x})$.

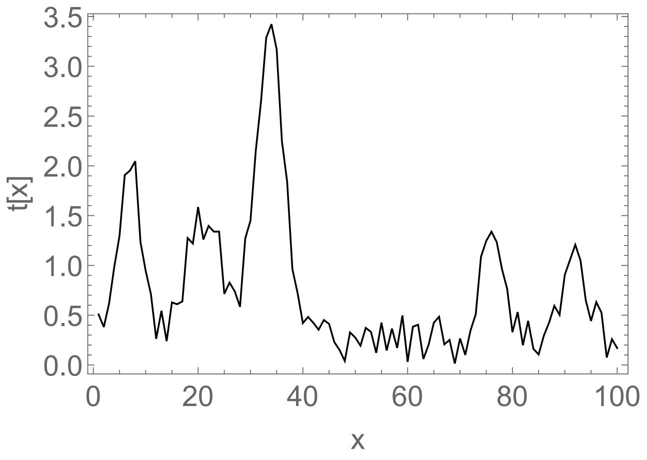
2. *Posterior probabilities: factorial encoding*



- $M = 8$ and $n = 20$ were used, which lies inside the factorial encoding region of the stability diagram.
- Each of the posterior probabilities $\Pr(y\,|\boldsymbol{x})$ is large mainly in a collar-shaped region of the torus; half circle one way round the torus, and half the other way.
- There is some overlap between the $\Pr(y\,|\boldsymbol{x})$ that circle the same way round the torus.
- There is a localised region of overlap between a pair of $\Pr(y\,|\boldsymbol{x})$ that circle the opposite way round the torus.
- These localised overlap regions are the mechanism by which factorial encoding has a small reconstruction distortion.

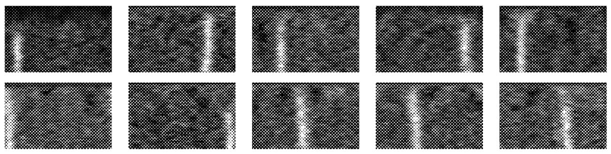
E. Multiple Independent Targets

1. *Training data*



- The targets were unit height Gaussian bumps with $\sigma = 2$.
- The additive noise was uniformly distributed variables in $[0, 0.5]$.

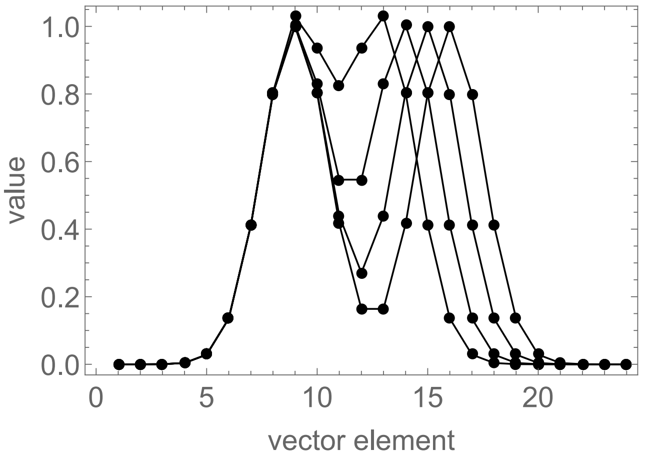
2. Factorial encoding



- $M = 10$ and $n = 10$ were used.
- Each of the reference vectors $\boldsymbol{x}'(y)$ becomes large in a localised region.
- Each input vector causes a subset of the neurons to fire corresponding to locations of the targets.
- This is a factorial encoder because each neuron responds to only a subspace of the input.

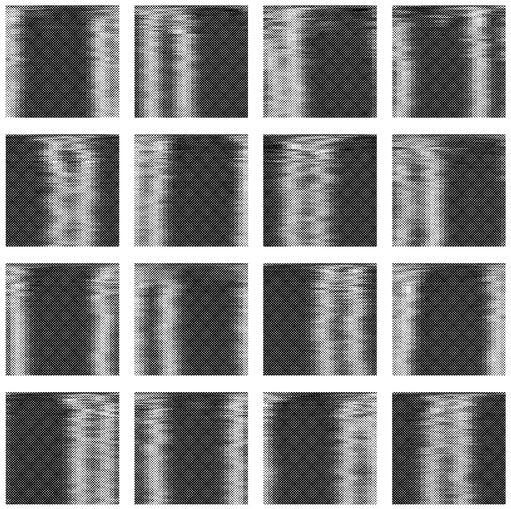
F. Pair of Correlated Targets

1. Training data



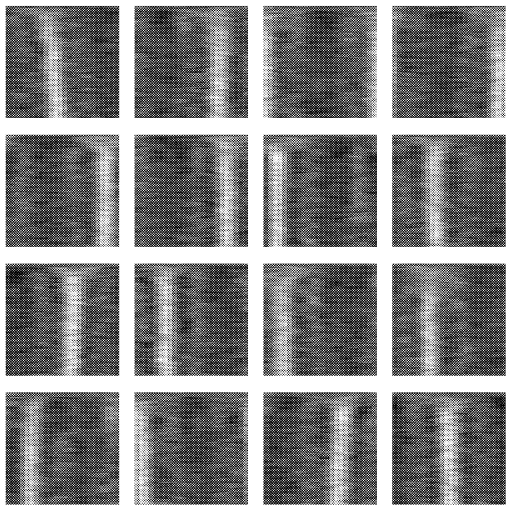
- The targets were unit height Gaussian bumps with $\sigma = 1.5$.

2. Training history: joint encoding



- $M = 16$ and $n = 3$ were used.
- Each of the reference vectors $\boldsymbol{x}'(y)$ becomes large in a pair of localised regions.
- Each neuron responds to a small range of positions and separations of the pair of targets.
- The neurons respond *jointly* to the position and separation of the targets.

3. Training history: factorial encoding



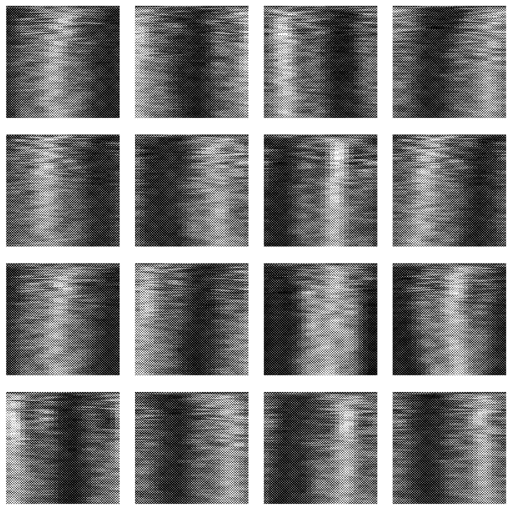
- $M = \{16, 16\}$ and $n = \{20, 20\}$ were used; this is a 2-stage encoder.
- The second encoder uses as input the posterior probability output by the first encoder.

- The objective function is the sum of the separate encoder objective functions (with equal weighting given to each).

- The presence of the second encoder affects the optimisation of the first encoder via "self-supervision".

- Each of the reference vectors $\boldsymbol{x}'(y)$ becomes large in a single localised region.

4. *Training history: invariant encoding*



- $M = \{16, 16\}$ and $n = \{3, 3\}$ were used; this is a 2-stage encoder.

- During training the ratio of the weighting assigned to the first and second encoders is increased from 1:5 to 1:40.

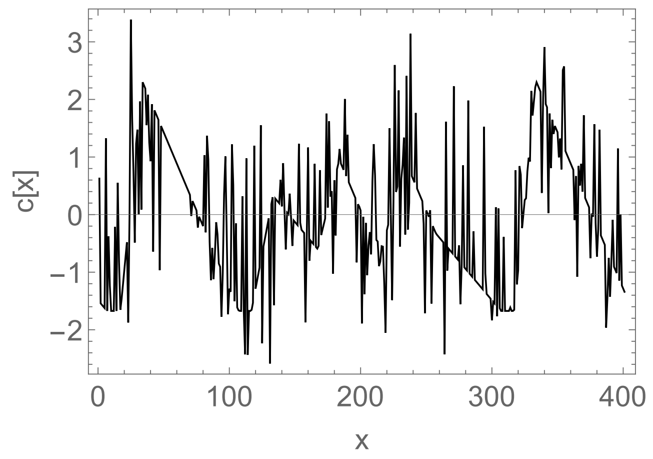
- Each of the reference vectors $\boldsymbol{x}'(y)$ becomes large in a single broad region.

- Each neuron responds only the position (and not the separation) of the pair of targets.

- The response of the neurons is *invariant* w.r.t. the separation of the targets.

G. Separating Different Waveforms

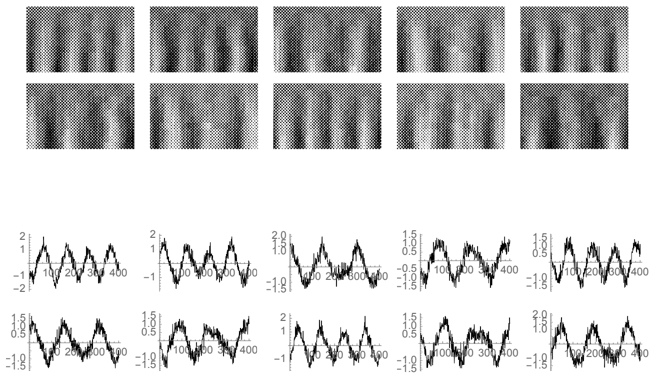
1. *Training data*



- This data is the superposition of a pair of waveforms plus noise.

- In each training vector the relative phase of the two waveforms is randomly selected.

2. *Training history: factorial encoding*



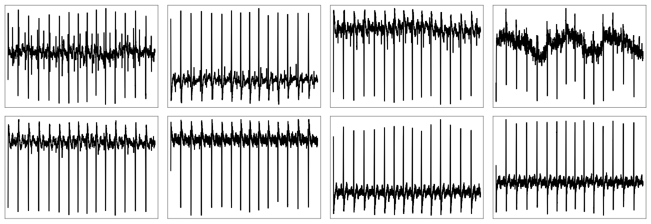
- $M = 10$ and $n = 20$ were used.

- Each of the reference vectors $\boldsymbol{x}'(y)$ becomes one or other of the two waveforms, and has a definite phase.

- Each neuron responds to only one of the waveforms, and then only when its phase is in a localised range.

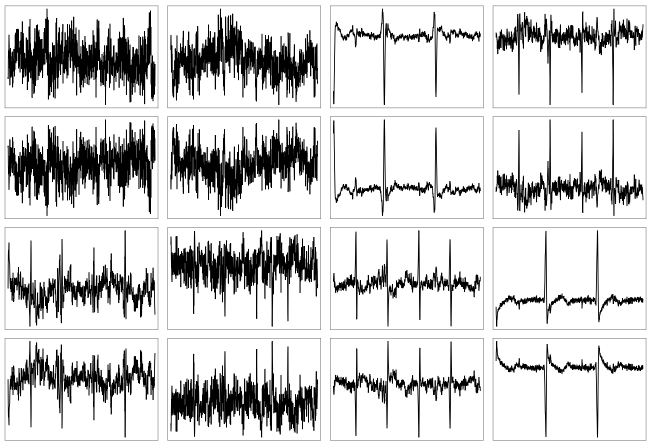
H. Maternal + Foetal ECG

1. Training data



- This data is an 8-channel ECG recording taken from a pregnant woman.
- The large spikes are the woman's heart beat.
- The noise masks the foetus' heartbeat.
- This data was whitened before training the neural network.

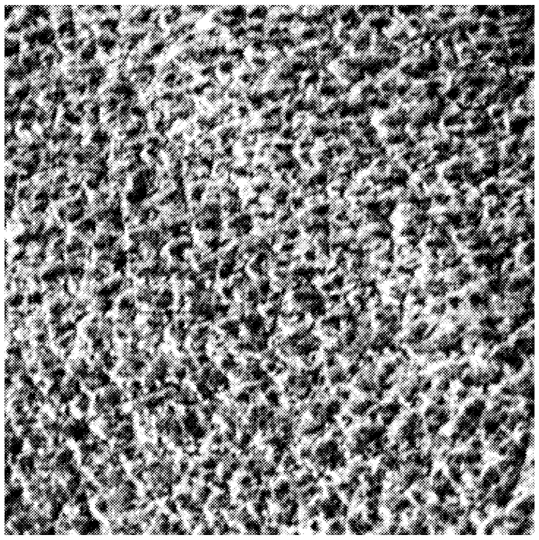
2. Factorial Encoding



- $M = 16$ and $n = 20$ were used.
- The results shown are $\boldsymbol{w}(y).\boldsymbol{x}$ computed for all neurons ($y = 1, 2, \cdots, 8$) for each 8-dimensional input vector $\boldsymbol{x}$.
- After limited training some, but not all, of the neurons have converged.
- The broadly separated spikes indicate a neuron that responds to the mother's heartbeat.
- The closely separated spikes indicate a neuron that responds to the foetus' heartbeat.

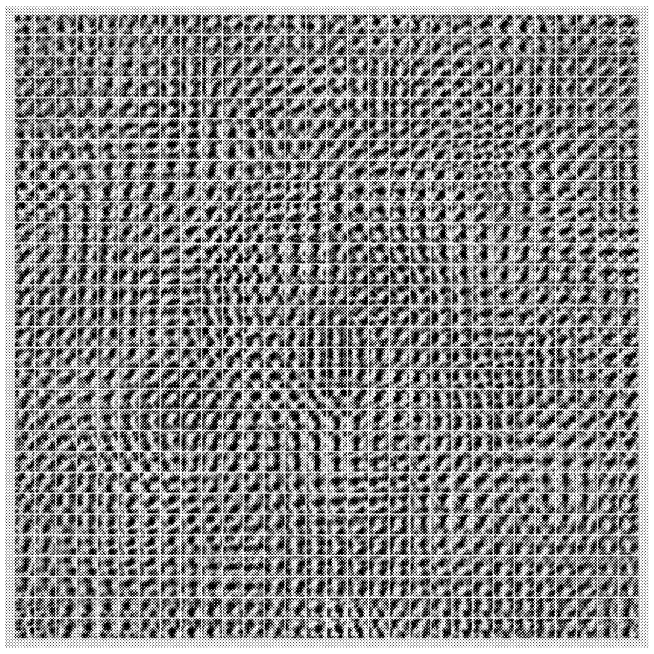
I. Visual Cortex Network (VICON)

1. Training Data



- This is a Brodatz texture image, whose spatial correlation length is 5-10 pixels.

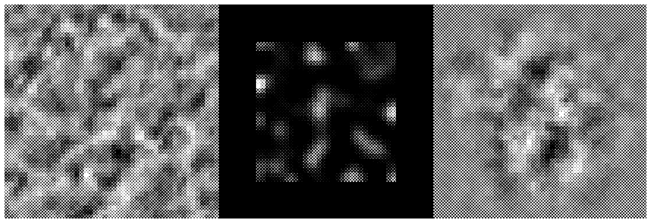
2. Orientation map



- $M = 30 \times 30$ and $n = 1$ were used.
- Input window size $= 17 \times 17$, neighbourhood size $= 9 \times 9$, leakage neighbourhood size $= 3 \times 3$ were used.
- Leakage probability was sampled from a 2-dimensional Gaussian PDF, with $\sigma = 1$ in each direction.

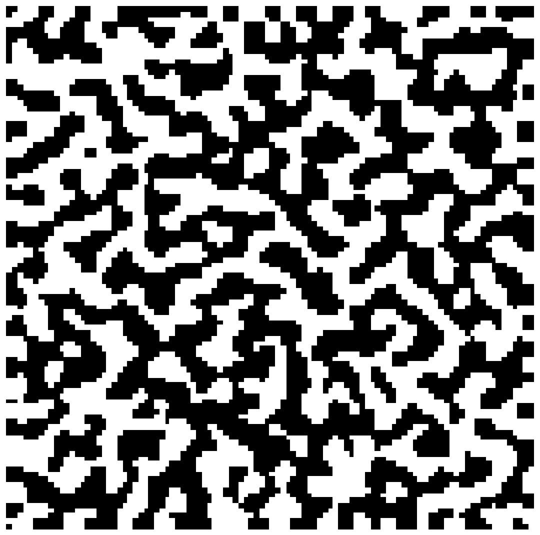
- Each of the reference vectors $\boldsymbol{x}'(y)$ typically looks like a small patch of image.
- Leakage induces topographic ordering across the array of neurons
- This makes the array of reference vectors look like an "orientation map".

3. *Sparse coding*



- The trained network is used to encode and decode a typical input image.
- Left image = input.
- Middle image = posterior probability. This shows "sparse coding" with a small number of "activity bubbles".
- Right image = reconstruction. Apart from edge effects, this is a low resolution version of the input.

4. *Dominance stripes*



- Interdigitate a pair of training images, so that one occupies on the black squares, and the other the white squares, of a "chess board".
- Preprocess this interdigitated image to locally normalise it using a finite range neighbourhood.
- $M = 100 \times 100$ and $n = 1$ were used.
- Input window size = 3×3, neighbourhood size = 5×5, leakage neighbourhood size = 3×3 were used.
- Leakage probability was sampled from a 2-dimensional Gaussian PDF, with $\sigma = 1$ in each direction.
- The dominance stripe map records for each neuron which of the 2 interdigitated images causes it to respond more strongly.
- The dominance stripes tend to run perpendicularly into the boundaries, because the neighbourhood window is truncated at the edge of the array.

IV. REFERENCES

[1][2][3][4][5]

[1] S P Luttrell, *Mathematics of neural networks: models, algorithms and applications*, ch. A theory of self-organising neural networks, pp. 240–244, Kluwer, Boston, 1997.

[2] ______, *An adaptive network for encoding data using piecewise linear functions*, Proceedings of international confererence on artificial neural networks (Edinburgh),

IEE, 1999, pp. 198–203.

[3] ———, *Combining artificial neural nets: Ensemble and modular multi-net systems*, Perspectives in Neural Computing, ch. Self-organised modular neural networks for encoding data, pp. 235–263, Springer-Verlag, London, 1999.

[4] ———, *Information theory and the brain*, ch. The emergence of dominance stripes and orientation maps in a network of firing neurons, pp. 101–121, Cambridge University Press, 2000.

[5] ———, *Stochastic vector quantisers*, (2000).

Stochastic Vector Quantisers [*]

S P Luttrell[†]

Defence Evaluation and Research Agency, St Andrews Rd, Malvern, Worcs, WR14 3PS, United Kingdom, tel: +44 (0) 1684-894046, fax: +44 (0) 1684-894384

In this paper a stochastic generalisation of the standard Linde-Buzo-Gray (LBG) approach to vector quantiser (VQ) design is presented, in which the encoder is implemented as the sampling of a vector of code indices from a probability distribution derived from the input vector, and the decoder is implemented as a superposition of reconstruction vectors, and the stochastic VQ is optimised using a minimum mean Euclidean reconstruction distortion criterion, as in the LBG case. Numerical simulations are used to demonstrate how this leads to self-organisation of the stochastic VQ, where different stochastically sampled code indices become associated with different input subspaces. This property may be used to automate the process of splitting high-dimensional input vectors into low-dimensional blocks before encoding them.

I. INTRODUCTION

In vector quantisation a code book is used to encode each input vector as a corresponding code index, which is then decoded (again, using the codebook) to produce an approximate reconstruction of the original input vector [2, 3]. The purpose of this paper is to generalise the standard approach to vector quantiser (VQ) design [7], so that each input vector is encoded as a *vector* of code indices that are stochastically sampled from a probability distribution that depends on the input vector, rather than as a *single* code index that is the deterministic outcome of finding which entry in a code book is closest to the input vector. This will be called a stochastic VQ (SVQ), and it includes the standard VQ as a special case. Note that this approach is different from the various soft competition and stochastic relaxation schemes that are used to train VQs (see e.g. [13]), because here the probability distribution is an essential part of the encoder, both during and after training.

One advantage of using the stochastic approach, which will be demonstrated in this paper, is that it automates the process of splitting high-dimensional input vectors into low-dimensional blocks before encoding them, because minimising the mean Euclidean reconstruction error can encourage different stochastically sampled code indices to become associated with different input subspaces [11]. Another advantage is that it is very easy to connect SVQs together, by using the vector of code index probabilities computed by one SVQ as the input vector to another SVQ [10].

In section II various pieces of previously published theory are unified to give a coherent account of SVQs. In section III the results of some new numerical simulations are presented, which demonstrate how the code indices in a SVQ can become associated in various ways with input subspaces.

II. THEORY

In this section various pieces of previously published theory are unified to establish a coherent framework for modelling SVQs.

In section II A the basic theory of folded Markov chains (FMC) is given [8], and in section II B it is extended to the case of high-dimensional input data [9]. In section II C some properties of the solutions that emerge when the input vector lives on a 2-torus are summarised [11]. Finally, in section II D the theory is further generalised to chains of linked FMCs [10].

A. Folded Markov Chains

The basic building block of the encoder/decoder model used in this paper is the folded Markov chain (FMC) [8]. Thus an input vector $\mathbf{x}$ is encoded as a code index vector $\mathbf{y}$, which is then subsequently decoded as a reconstruction $\mathbf{x}'$ of the input vector. Both the encoding and decoding operations are allowed to be probabilistic, in the sense that $\mathbf{y}$ is a sample drawn from $\Pr(\mathbf{y}|\mathbf{x})$, and $\mathbf{x}'$ is a sample drawn from $\Pr(\mathbf{x}'|\mathbf{y})$, where $\Pr(\mathbf{y}|\mathbf{x})$ and $\Pr(\mathbf{x}|\mathbf{y})$ are Bayes' inverses of each other, as given by

$$\Pr(\mathbf{x}|\mathbf{y}) = \frac{\Pr(\mathbf{y}|\mathbf{x})\ \Pr(\mathbf{x})}{\int d\mathbf{z}\ \Pr(\mathbf{y}|\mathbf{z})\ \Pr(\mathbf{z})} \tag{2.1}$$

and $\Pr(\mathbf{x})$ is the prior probability from which $\mathbf{x}$ was sampled.

Because the chain of dependences in passing from $\mathbf{x}$ to $\mathbf{y}$ and then to $\mathbf{x}'$ is first order Markov (i.e. it is described by the directed graph $\mathbf{x} \longrightarrow \mathbf{y} \longrightarrow \mathbf{x}'$), and because the two ends of this Markov chain (i.e. $\mathbf{x}$ and $\mathbf{x}'$) live in the same vector space, it is called a *folded* Markov

[*]Typeset in LaTeX on May 9, 2019.
This is archived at http://arxiv.org/abs/1012.3705, 16 Dec 2010. This paper was submitted to Special Issue of IEEE Trans. IT on Information-Theoretic Imaging on 30 July 1999. It was not accepted for publication, but it underpins several subsequently published papers. ©1999 British Crown Copyright/DERA
[†]Electronic address: luttrell@signal.dera.gov.uk

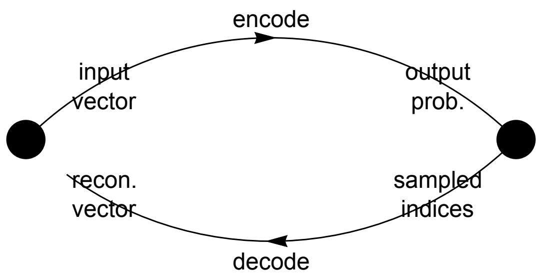


Figure 1: A folded Markov chain (FMC) in which an input vector $\mathbf{x}$ is encoded as a code index vector $\mathbf{y}$ that is drawn from a conditional probability $\Pr(\mathbf{y}|\mathbf{x})$, which is then decoded as a reconstruction vector $\mathbf{x}'$ drawn from the Bayes' inverse conditional probability $\Pr(\mathbf{x}'|\mathbf{y})$.

chain (FMC). The operations that occur in an FMC are summarised in figure 1.

In order to ensure that the FMC encodes the input vector optimally, a measure of the reconstruction error must be minimised. There are many possible ways to define this measure, but one that is consistent with many previous results, and which also leads to many new results, is the mean Euclidean reconstruction error measure D, which is defined as

$$D \equiv \int d\mathbf{x} \Pr(\mathbf{x}) \sum_{\mathbf{y}=1}^{\mathbf{M}} \Pr(\mathbf{y}|\mathbf{x}) \int d\mathbf{x}' \Pr(\mathbf{x}'|\mathbf{y}) \left\|\mathbf{x}-\mathbf{x}'\right\|^2 \tag{2.2}$$

where $\Pr(\mathbf{x}) \Pr(\mathbf{y}|\mathbf{x}) \Pr(\mathbf{x}'|\mathbf{y})$ is the joint probability that the FMC has state $(\mathbf{x},\mathbf{y},\mathbf{x}')$, $\left\|\mathbf{x}-\mathbf{x}'\right\|^2$ is the Euclidean reconstruction error, and $\int d\mathbf{x} \sum_{\mathbf{y}=1}^{\mathbf{M}} \int d\mathbf{x}' (\cdots)$ sums over all possible states of the FMC (weighted by the joint probability). The code index vector $\mathbf{y}$ is assumed to lie on a rectangular lattice of size $\mathbf{M}$.

The Bayes' inverse probability $\Pr(\mathbf{x}'|\mathbf{y})$ may be integrated out of this expression for D to yield

$$D = 2\int d\mathbf{x} \Pr(\mathbf{x}) \sum_{\mathbf{y}=1}^{\mathbf{M}} \Pr(\mathbf{y}|\mathbf{x}) \left\|\mathbf{x}-\mathbf{x}'(\mathbf{y})\right\|^2 \tag{2.3}$$

where the reconstruction vector $\mathbf{x}'(\mathbf{y})$ is defined as $\mathbf{x}'(\mathbf{y}) \equiv \int d\mathbf{x} \Pr(\mathbf{x}|\mathbf{y})\, \mathbf{x}$. Because of the quadratic form of the objective function, it turns out that $\mathbf{x}'(\mathbf{y})$ may be treated as a free parameter whose optimum value (i.e. the solution of $\frac{\partial D}{\partial \mathbf{x}'(\mathbf{y})} = 0$) is $\int d\mathbf{x} \Pr(\mathbf{x}|\mathbf{y})\, \mathbf{x}$, as required.

If D is now minimised with respect to the probabilistic encoder $\Pr(\mathbf{y}|\mathbf{x})$ and the reconstruction vector $\mathbf{x}'(\mathbf{y})$, then the optimum has the form

$$\begin{aligned} \Pr(\mathbf{y}|\mathbf{x}) &= \delta_{\mathbf{y},\mathbf{y}(\mathbf{x})} \\ \mathbf{y}(\mathbf{x}) &= \underset{\mathbf{y}}{\arg\min} \left\|\mathbf{x}-\mathbf{x}'(\mathbf{y})\right\|^2 \\ \mathbf{x}'(\mathbf{y}) &= \int d\mathbf{x} \Pr(\mathbf{x}|\mathbf{y})\, \mathbf{x} \end{aligned} \tag{2.4}$$

where $\Pr(\mathbf{y}|\mathbf{x})$ has reduced to a deterministic encoder (as described by the Kronecker delta $\delta_{\mathbf{y},\mathbf{y}(\mathbf{x})}$), $\mathbf{y}(\mathbf{x})$ is a nearest neighbour encoding algorithm using code vectors $\mathbf{x}'(\mathbf{y})$ to partition the input space into code cells, and (in an optimised configuration) the $\mathbf{x}'(\mathbf{y})$ are the centroids of these code cells. This is equivalent to a standard VQ [7].

An extension of the standard VQ to the case of where the code index is transmitted along a noisy communication channel before the reconstruction is attempted [1, 6] was derived in [8], and was shown to lead to a good approximation to a topographic mapping neural network [5].

B. High Dimensional Input Spaces

A problem with the standard VQ is that its code book grows exponentially in size as the dimensionality of the input vector is increased, assuming that the contribution to the reconstruction error from each input dimension is held constant. This means that such VQs are useless for encoding extremely high dimensional input vectors, such as images. The usual solution to this problem is to partition the input space into a number of lower dimensional subspaces (or blocks), and then to encode each of these subspaces separately. However, this produces an undesirable side effect, where the boundaries of the blocks are clearly visible in the reconstruction; this is the origin of the blocky appearance of reconstructed images, for instance.

There is also a much more fundamental objection to this type of partitioning, because the choice of blocks should ideally be decided in such a way that the correlations *within* a block are much stronger than the correlations *between* blocks. In the case of image encoding, this will usually be the case if the partitioning is that each block consists of contiguous image pixels. However, more generally, there is no guarantee that the input vector statistics will respect the partitioning in this convenient way. Thus it will be necessary to deduce the best choice of blocks from the training data.

In order to solve the problem of finding the best partitioning, consider the following constraint on $\Pr(\mathbf{y}|\mathbf{x})$ and $\mathbf{x}'(\mathbf{y})$ in the FMC objective function in equation 2.3

$$\begin{aligned} \mathbf{y} &= (y_1, y_2, \cdots, y_n) \qquad 1 \leq y_i \leq M \\ \Pr(\mathbf{y}|\mathbf{x}) &= \Pr(y_1|\mathbf{x}) \Pr(y_2|\mathbf{x}) \cdots \Pr(y_n|\mathbf{x}) \\ \mathbf{x}'(\mathbf{y}) &= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}'(y_i) \end{aligned} \tag{2.5}$$

Thus the code index vector $\mathbf{y}$ is assumed to be n-dimensional, each component y_i (for $i = 1, 2, \cdots, n$ and $1 \leq y_i \leq M$) is an independent sample drawn from $\Pr(y|\mathbf{x})$, and the reconstruction vector $\mathbf{x}'(\mathbf{y})$ (vector argument) is assumed to be a superposition of n contributions $\mathbf{x}'(y_i)$ (scalar argument) for $i = 1, 2, \cdots, n$. As D is minimised, this constraint allows partitioned solutions to emerge by a process of self-organisation.

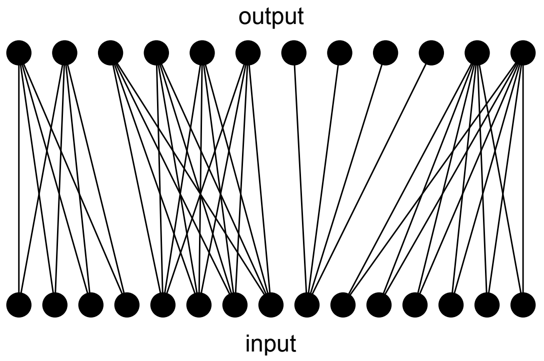


Figure 2: A typical solution in which the components of the input vector $\mathbf{x}$ and the range of values of the output code index y are both partitioned into blocks. These input and output blocks are connected together as illustrated. More generally, the blocks may overlap each other.

For instance, solutions can have the structure illustrated in figure 2. This type of structure is summarised as follows

$$\begin{aligned} \mathbf{x} &= (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N) \\ \Pr(\mathbf{y}|\mathbf{x}) &= \Pr\left(y_1|\mathbf{x}_{p(y_1)}\right) \Pr\left(y_2|\mathbf{x}_{p(y_2)}\right) \cdots \Pr\left(y_n|\mathbf{x}_{p(y_n)}\right) \end{aligned} \quad (2.6)$$

In this type of solution the input vector $\mathbf{x}$ is partitioned as $(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N)$, the probability $\Pr(y|\mathbf{x})$ reduces to $\Pr\left(y|\mathbf{x}_{p(y)}\right)$ which depends only on $\mathbf{x}_{p(y)}$, where the function $p(y)$ computes the index of the block which code index y inhabits. There is not an exact correspondence between this type of partitioning and that used in the standard approach to encoding image blocks, because here the n code indices are spread at random over the N image blocks, which does not guarantee that every block is encoded. Although, for a given N, if n is chosen to be suffiently large, then there is a virtual certainty that every block is encoded. This is the price that has to be paid when it is assumed that the code indices are drawn independently.

The constraints in equation 2.5 prevent the full space of possible values of $\Pr(\mathbf{y}|\mathbf{x})$ or $\mathbf{x}'(\mathbf{y})$ from being explored as D is minimised, so they lead to an *upper bound* D_1+D_2 on the FMC objective function D (i.e. $D \le D_1 + D_2$), which may be derived as [9]

$$\begin{aligned} D_1 &\equiv \frac{2}{n} \int d\mathbf{x} \Pr(\mathbf{x}) \sum_{y=1}^{M} \Pr(y|\mathbf{x}) \left\|\mathbf{x} - \mathbf{x}'(y)\right\|^2 \\ D_2 &\equiv \frac{2(n-1)}{n} \int d\mathbf{x} \Pr(\mathbf{x}) \left\|\mathbf{x} - \sum_{y=1}^{M} \Pr(y|\mathbf{x})\, \mathbf{x}'(y)\right\|^2 \end{aligned} \quad (2.7)$$

Note that M and n are model order parameters, whose values need to be chosen appropriately for each encoder optimisation problem.

For $n = 1$ only the D_1 term contributes, and it is equivalent to the FMC objective function D in equation 2.3 with the vector code index $\mathbf{y}$ replaced by a scalar code index y, so its minimisation leads to a standard vector quantiser as in equation 2.4, in which each input vector is approximated by a *single* reconstruction vector $\mathbf{x}'(y)$.

When n becomes large enough that D_2 dominates over D_1, the optimisation problem reduces to minimisation of the mean Euclidean reconstruction error (approximately). This encourages the approximation $\mathbf{x} \approx \sum_{y=1}^{M} \Pr(y|\mathbf{x})\, \mathbf{x}'(y)$ to hold, in which the input vector $\mathbf{x}$ is approximated as a weighted (using weights $\Pr(y|\mathbf{x})$) sum of *many* reconstruction vectors $\mathbf{x}'(y)$. In numerical simulations this has invariably led to solutions which are a type of principal components analysis (PCA) of the input vectors, where the expansion coefficients $\Pr(y|\mathbf{x})$ are constrained to be non-negative and sum to unity. Also, the approximation $\mathbf{x} \approx \sum_{y=1}^{M} \Pr(y|\mathbf{x})\, \mathbf{x}'(y)$ is very good for solutions in which $\Pr(y|\mathbf{x})$ depends on the whole of $\mathbf{x}$, rather than merely on a subspace of $\mathbf{x}$, so this does not lead to a partitioned solution.

For intermediate values of n, where both D_1 and D_2 are comparable in size, partitioned solutions can emerge. However, in this intermediate region the properties of the optimum solution depend critically on the interplay between the statistical properties of the training data and the model order parameters M and n. To illustrate how the choice of M and n affects the solution, the case of input vectors that live on a 2-torus is summarised in section II C.

When the full FMC objective function in equation 2.3 is optimised it leads to the standard (deterministic) VQ in equation 2.4. However, it turns out that the constrained FMC objective function $D_1 + D_2$ in equation 2.7 does not allow a deterministic VQ to emerge (except in the case $n = 1$), because a more accurate reconstruction can be obtained by allowing more than one code index to be sampled for each input vector. Because of this behaviour, in which the encoder is stochastic both during and after training, this type of constrained FMC will be called a SVQ.

C. Example: 2-Torus Case

A scene is defined as a number of objects at specified positions and orientations, so is it specified by a low-dimensional vector of scene coordinates, which are effectively the intrinsic coordinates of a low-dimensional manifold. An image of that scene is an embedding of the low-dimensional manifold in the high-dimensional space of image pixel values. Because the image pixel values are non-linearly related to the vector of scene coordinates, this embedding operation distorts the manifold so that it becomes curved. The problem of finding the optimal way to encode images may thus be viewed as the problem of finding the optimal way to encode curved manifolds, where the instrinsic dimensionality of the manifold is the same as the dimensionality of the vector of scene coordinates.

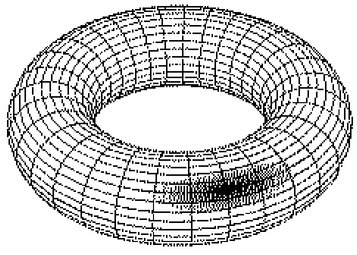


Figure 3: A typical probability $\Pr(y|\mathbf{x})$ (only one y value is illustrated) for encoding a 2-torus using a small value of n. This defines a smoothly tapered localised region on the 2-torus. The toroidal mesh serves only to help visualise the 2-torus.

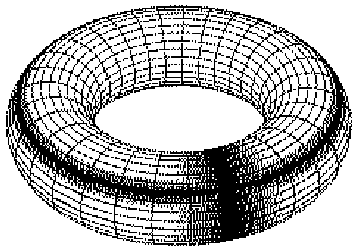


Figure 4: Two typical probabilities $\Pr(y_1|\mathbf{x})$ and $\Pr(y_2|\mathbf{x})$ for encoding a 2-torus using a large value of n. Each separately defines a smoothly tapered collar-shaped region on the 2-torus. However, taken together, their region of intersection defines a smoothly tapered localised region on the 2-torus. The toroidal mesh serves only to help visualise the 2-torus.

The simplest curved manifold is the circle, and the next most simple curved manifold is the 2-torus (which has 2 intrinsic circular coordinates). By making extensive use of the fact that the optimal form of $\Pr(y|\mathbf{x})$ must be a piecewise linear function of the input vector $\mathbf{x}$ [10], the optimal encoders for these manifolds have been derived analytically [11]. The toroidal case is very interesting because it demonstrates the transition between the unpartitioned and partitioned optimum solutions as n is increased. A 2-torus is a realistic model of the manifold generated by the linear superposition of 2 sine waves, having fixed amplitudes and wavenumbers. The phases of the 2 sine waves are then the 2 intrinsic circular coordinates of this manifold, and if these phases are both uniformly distributed on the circle, then $\Pr(y|\mathbf{x})$ defines a constant probability density on the 2-torus.

A typical $\Pr(y|\mathbf{x})$ for small n is illustrated in figure 3. Only in the case where $n = 1$ would $\Pr(y|\mathbf{x})$ correspond to a sharply defined code cell; for $n > 1$ the edges of the code cells are tapered so that they overlap with one other. The 2-torus is covered with a large number of these overlapping code cells, and when a code index y is sampled from $\Pr(y|\mathbf{x})$, it allows a reconstruction of the input to be made to within an uncertainty area commensurate with the size of a code cell. This type of encoding is called *joint* encoding, because the 2 intrinsic dimensions of the 2-torus are *simultaneously* encoded by y.

A typical pair of $\Pr(y|\mathbf{x})$ (i.e. $\Pr(y_1|\mathbf{x})$ and $\Pr(y_2|\mathbf{x})$) for large n is illustrated in figure 4. The partitioning splits the code indices into two different types: one type encodes one of the intrinsic dimensions of the 2-torus, and the other type encodes the other intrinsic dimension of the 2-torus, so each code cell is a tapered collar-shaped region. When the code indices $(y_1, y_2, \cdots, y_n)$ are sampled from $\Pr(y|\mathbf{x})$, they allow a reconstruction of the input to be made to within an uncertainty area commensurate with the size of the region of intersection of a pair of orthogonal code cells, as illustrated in figure 4. This type of encoding is called *factorial* encoding, because the 2 intrinsic dimensions of the 2-torus are *separately* encoded in $(y_1, y_2, \cdots, y_n)$.

For a 2-torus there is an upper limit $M \approx 12$ beyond which the optimum solution is always a joint encoder (as shown in figure 3). This limit arises because when $M \gtrsim 12$ the code book is sufficiently large that the joint encoder gives the best reconstruction for all values of n. This result critically depends on the fact that as n is increased the code cells overlap progressively more and more, so the accuracy of the reconstruction progressively increases. For $M \gtrsim 12$ the rate at which the accuracy of the joint encoder improves (as n increases) is sufficiently great that it is always better than that of the factorial encoder (which also improves as n increases).

D. Chains of Linked FMCs

Thus far it has been shown that the FMC objective function in equation 2.3, with the constraints imposed in equation 2.5, leads to useful properties, such as the automatic partitioning of the code book to yield the factorial encoder, such as that illustrated in figure 4 (and more generally, as illustrated in figure 2). The free parameters (M, n) (i.e. the size of the code book, and the number of code indices sampled) can be adjusted to obtain an optimal solution that has the desired properties (e.g. a joint or a factorial encoder, as in figures 3 and 4, respectively). However, since there are only 2 free parameters, there is a limit to the variety of types of properties that the optimal solution can have. It would thus be very useful to introduce more free parameters.

The FMC illustrated in figure 1 may be generalised to a chain of linked FMCs as shown in figure 5. Each stage in this chain is an FMC of the type shown in figure 1, and the vector of probabilities (for all values of the

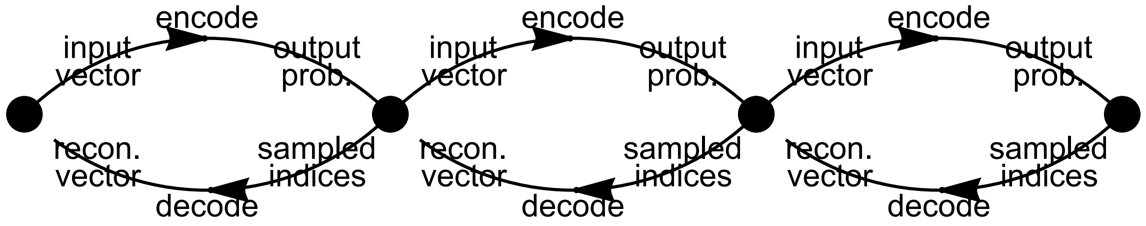


Figure 5: A chain of linked FMCs, in which the output from each stage is its vector of posterior probabilities (for all values of the code index), which is then used as the input to the next stage. Only 3 stages are shown, but any number may be used. More generally, any acyclically linked network of FMCs may be used.

code index) computed by each stage is used as the input vector to the next stage; there are other ways of linking the stages together, but this is the simplest possibility. The overall objective function is a weighted sum of the FMC objective functions derived from each stage. The total number of free parameters in an L stage chain is $3L-1$, which is the sum of 2 free parameters for each of the L stages, plus $L-1$ weighting coefficients; there are $L-1$ rather than L weighting coefficients because the overall normalisation of the objective function does not affect the optimum solution.

The chain of linked FMCs may be expressed mathematically by first of all introducing an index l to allow different stages of the chain to be distinguished thus

$$\begin{array}{ll} M \longrightarrow M^{(l)} & y \longrightarrow y^{(l)} \\ \mathbf{x} \longrightarrow \mathbf{x}^{(l)} & \mathbf{x}' \longrightarrow \mathbf{x}^{(l)\prime} \\ n \longrightarrow n^{(l)} & D \longrightarrow D^{(l)} \\ D_1 \longrightarrow D_1^{(l)} & D_2 \longrightarrow D_2^{(l)} \end{array} \tag{2.8}$$

The stages are then defined and linked together thus (the detailed are given only as far as the input to the third stage)

$$\begin{aligned} \mathbf{x}^{(1)} &\longrightarrow y^{(1)} \longrightarrow \mathbf{x}^{(1)\prime} \\ \mathbf{x}^{(2)} &= \left(x_1^{(2)}, x_2^{(2)}, \cdots, x_{M^{(1)}}^{(2)}\right) \\ x_i^{(2)} &= \Pr\left(y^{(1)} = i|\mathbf{x}^{(1)}\right) \qquad 1 \leq i \leq M^{(1)} \\ \mathbf{x}^{(2)} &\longrightarrow y^{(2)} \longrightarrow \mathbf{x}^{(2)\prime} \\ \mathbf{x}^{(3)} &= \left(x_1^{(3)}, x_2^{(3)}, \cdots, x_{M^{(2)}}^{(3)}\right) \\ x_i^{(3)} &= \Pr\left(y^{(2)} = i|\mathbf{x}^{(2)}\right) \qquad 1 \leq i \leq M^{(2)} \end{aligned} \tag{2.9}$$

The objective function and its upper bound are then given by

$$\begin{aligned} D &= \sum_{l=1}^{L} s^{(l)} D^{(l)} \\ &\leq D_1 + D_2 \\ &= \sum_{l=1}^{L} s^{(l)} \left(D_1^{(l)} + D_2^{(l)}\right) \end{aligned} \tag{2.10}$$

where $s^{(l)} \geq 0$ is the weighting that is applied to the contribution of stage l of the chain to the overall objective function.

The piecewise linearity property enjoyed by $\Pr(y|\mathbf{x})$ in a single stage chain also holds for all of the $\Pr\left(y^{(l)}|\mathbf{x}^{(l)}\right)$ in a multi-stage chain, provided that the stages are linked together as prescribed in equation 2.9 [10]. This will allow optimum analytic solutions to be derived by an extension of the single stage methods used in [11].

III. SIMULATIONS

In this section the results of various numerical simulations are presented, which demonstrate some of the types of behaviour exhibited by an encoder that consists of a chain of linked FMCs. Synthetic, rather than real, training data are used in all of the simulations, because this allows the basic types of behaviour to be cleanly demonstrated.

In section III A the training algorithm is presented. In section III B the training data is described. In section III C a single stage encoder is trained on data that is a superposition of two randomly positioned objects. In section III D this is generalised to objects with correlated positions, and three different types of behaviour are demonstrated: factorial encoding using both a 1-stage and a 2-stage encoders (section III D 1), joint encoding using a 1-stage encoder (section III D 2), and invariant encoding (i.e. ignoring a subspace of the input space altogether) using a 2-stage encoder (section III D 3).

A. Training Algorithm

Assuming that $\Pr(y|\mathbf{x})$ is modelled as in appendix A (i.e. $\Pr(y|\mathbf{x}) = \frac{Q(y|\mathbf{x})}{\sum_{y'=1}^{M} Q(y'|\mathbf{x})}$ and $Q(y|\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}(y)\cdot\mathbf{x}-b(y))}$), then the partial derivatives of $D_1 + D_2$ with respect to the 3 types of parameters in a single stage of the encoder may be denoted as

$$\begin{aligned} \mathbf{g}_w(y) &\equiv \frac{\partial(D_1+D_2)}{\partial \mathbf{w}(y)} \\ g_b(y) &\equiv \frac{\partial(D_1+D_2)}{\partial b(y)} \\ \mathbf{g}_x(y) &\equiv \frac{\partial(D_1+D_2)}{\partial \mathbf{x}'(y)} \end{aligned} \quad (3.1)$$

This may be generalised to each stage of a multi-stage encoder by including an (l) superscript, and ensuring that for each stage the partial derivatives include the additional contributions that arise from forward propagation through later stages; this is essentially an application of the chain rule of differentiation, using the derivatives $\frac{\partial \mathbf{x}^{(l+1)}}{\partial \mathbf{w}^{(l)}(y^{(l)})}$ and $\frac{\partial \mathbf{x}^{(l+1)}}{\partial b^{(l)}(y^{(l)})}$ to link the stages together (see appendix A).

A simple algorithm for updating these parameters is (omitting the (l) superscript, for clarity)

$$\begin{aligned} \mathbf{w}(y) &\longrightarrow \mathbf{w}(y) - \varepsilon\,\frac{\mathbf{g}_w(y)}{g_{w,0}} \\ b(y) &\longrightarrow b(y) - \varepsilon\,\frac{g_b(y)}{g_{b,0}} \\ \mathbf{x}'(y) &\longrightarrow \mathbf{x}'(y) - \varepsilon\,\frac{\mathbf{g}_x(y)}{g_{x,0}} \end{aligned} \quad (3.2)$$

where ε is a small update step size parameter, and the three normalisation factors are defined as

$$\begin{aligned} g_{w,0} &\equiv \max_{y} \sqrt{\frac{\|\mathbf{g}_w(y)\|^2}{\dim \mathbf{x}}} \\ g_{b,0} &\equiv \max_{y} |b(y)| \\ g_{x,0} &\equiv \max_{y} \sqrt{\frac{\|\mathbf{g}_x(y)\|^2}{\dim \mathbf{x}}} \end{aligned} \quad (3.3)$$

The $\frac{\mathbf{g}_w(y)}{g_{w,0}}$ and $\frac{\mathbf{g}_x(y)}{g_{x,0}}$ factors ensure that the maximum update step size for $\mathbf{w}(y)$ and $\mathbf{x}'(y)$ is $\varepsilon \dim \mathbf{x}$ (i.e. ε per dimension), and the $\frac{g_b(y)}{g_{b,0}}$ factor ensures that the maximum update step size for $b(y)$ is ε. This update algorithm can be generalised to use a different ε for each stage of the encoder, and also to allow a different ε to be used for each of the 3 types of parameter. Furthermore, the size of ε can be varied as training proceeds, usually starting with a large value, and then gradually reducing its size as the solution converges. It is not possible to give general rules for exactly how to do this, because training conditions depend very much on the statistical properties of the training set.

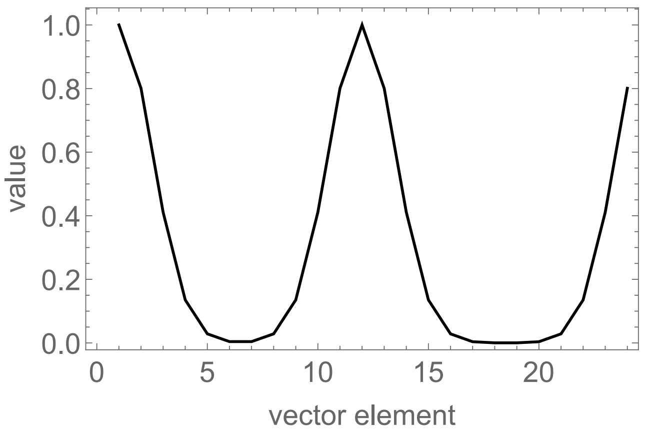


Figure 6: An example of a typical training vector for $M = 24$. Each object is a Gaussian hump with a half-width of 1.5 units, and peak amplitude of 1. The overall input vector is formed as a linear superposition of the 2 objects. Note that the input vector is wrapped around circularly to remove minor edge effects that would otherwise arise.

B. Training Data

The key property that this type of self-organising encoder exhibits is its ability to automatically split up high-dimensional input spaces into lower-dimensional subspaces, each of which is separately encoded. For instance, see section II B for a summary of the analytically solved case of training data that lives on a simple curved manifold (i.e. a 2-torus). This self-organisation manifests itself in many different ways, depending on the interplay between the statistical properties of the training data, and the 3 free parameters (i.e. the code book size M, the number of code indices sampled n, and the stage weighting s) per stage of the encoder (see section II D). However, it turns out that the joint and factorial encoders (of the same general type as those obtained in the case of a 2-torus) are also the optimum solutions for more general curved manifolds.

In order to demonstrate the various different basic types of self-organisation it is necessary to use synthetic training data with controlled properties. All of the types of self-organisation that will be demonstrated in this paper may be obtained by training a 1-stage or 2-stage encoder on 24-dimensional data (i.e. $M = 24$) that consists of a superposition of a pair of identical objects (with circular wraparound to remove edge effects), such as is shown in figure 6.

The training data is thus uniformly distributed on a manifold with 2 intrinsic circular coordinates, which is then embedded in a 24-dimensional image space. The embedding is a curved manifold, but is *not* a 2-torus, and there are two reasons for this. Firstly, even though the manifold has 2 intrinsic circular coordinates, the non-linear embedding distorts these circles in the 24-dimensional embedding space so that they are not planar (i.e. the profile of each object lives in the *full* 24-dimensional embedding space). Secondly, unlike a 2-torus, each point on the manifold maps to itself under interchange of the pair of circular coordinates, so the manifold is covered twice by a 2-torus (i.e. the objects are identical, so it makes no difference if they are swapped over). However, these differences do not destroy the general character of the joint and factorial encoder solutions that were obtained in section II B.

In the simulations presented below, two different methods of selecting the object positions are used: either the positions are statistically independent, or they are correlated. In the independent case, each object position is a random integer in the interval $[1, 24]$. In the correlated case, the first object position is a random integer in the interval $[1, 24]$, and the second object position is chosen *relative to* the first one as an integer in the range $[4, 8]$, so that the mean object separation is 6 units.

C. Independent Objects

The simplest demonstration is to let a single stage encoder discover the fact that the training data consists of a superposition of a pair of objects, which is an example of independent component analysis (ICA) or blind signal separation (BSS) [4]. This may readily be done by setting the parameters values as follows: code book size $M = 16$, number of code indices sampled $n = 20$, $\varepsilon = 0.2$ for 250 training steps, $\varepsilon = 0.1$ for a further 250 training steps.

The self-organisation of the 16 reconstruction vectors as training progresses (measured down the page) is shown in figure 7.

After some initial confusion, the reconstruction vectors self-organise so that each code index corresponds to a *single* object at a well defined location. This behaviour is non-trivial, because each training vector is a superposition of a *pair* of objects at independent locations, so typically more than one code index must be sampled by the encoder, which is made possible by the relatively large choice $n = 20$. This result is a factorial encoder, because the objects are encoded separately. This is a rudimentary example of the type of solution that was illustrated in figure 2, although here the blocks overlap each other.

The case of a joint encoder requires a rather large code book when the objects are independent. However, when correlations between the objects are introduced then the code book can be reduced to a manageable size, as will be demonstrated in the next section.

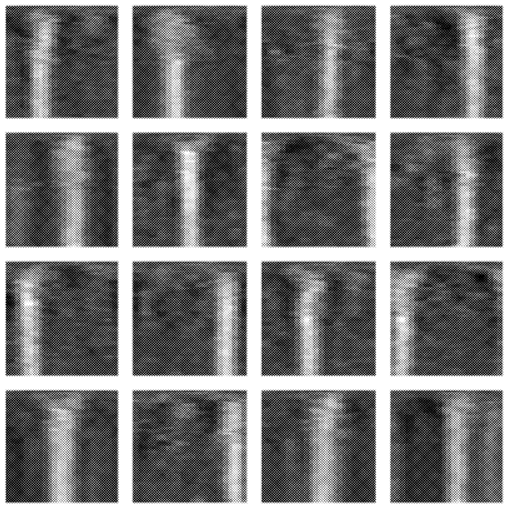


Figure 7: A factorial encoder emerges when a single stage encoder is trained on data that is a superposition of 2 objects in independent locations.

D. Correlated Objects

A more interesting situation arises if the positions of the pair of objects are mutually correlated, so that the training data is non-uniformly distributed on a manifold with 2 intrinsic circular coordinates. The pair of objects can then be encoded in 3 fundamentally different ways:

1. Factorial encoder. This encoder ignores the correlations between the objects, and encodes them as if they were 2 independent objects. Each code index would thus encode a single object position, so many code indices must be sampled in order to virtually guarantee that both object positions are encoded. This result is a type of independent component analysis (ICA) [4].

2. Joint encoder. This encoder regards each possible joint placement of the 2 objects as a distinct configuration. Each code index would thus encode a pair of object positions, so only one code index needs to be sampled in order to guarantee that both object positions are encoded. This result is basically the same as what would be obtained by using a standard VQ [7].

3. Invariant encoder. This encoder regards each possible placement of the centroid of the 2 objects as a distinct configuration, but regards all possible object separations (for a given centroid) as being equivalent. Each code index would thus encode only the centroid of the pair of objects. This type of encoder does not arise when the objects are independent. This is similar to self-organising transformation invariant detectors described in [12].

Each of these 3 possibilities is shown in figure 8, where the diagrams are meant only to be illustrative. The correlated variables live in the large 2-dimensional rectangular region extending from bottom-left to top-right of each diagram. For data of the type shown in figure 6, the rectangular region is in reality the curved manifold generated by varying the pair of object coordinates, and the invariance of the data under interchange of the pair of object coodinates means that the upper left and lower right halves of each diagram cover the manifold twice.

The factorial encoder has two orthogonal sets of long thin rectangular code cells, and the diagram shows how a pair of such cells intersect to define a small square code cell. The joint encoder behaves as a standard vector quantiser, and is illustrated as having a set of square code cells, although their shapes will not be as simple as this in practice. The invariant encoder has a set of long thin rectangular code cells that encode only the long diagonal dimension.

In all 3 cases there is overlap between code cells. In the case of the factorial and joint encoders the overlap tends to be only between nearby code cells, whereas in the case of an invariant encoder the range of the overlap is usually much greater, as will be seen in the numerical simulations below. In practice the optimum encoder may not be a clean example of one of the types illustrated in figure 8, as will also be seen in the numerical simulations below.

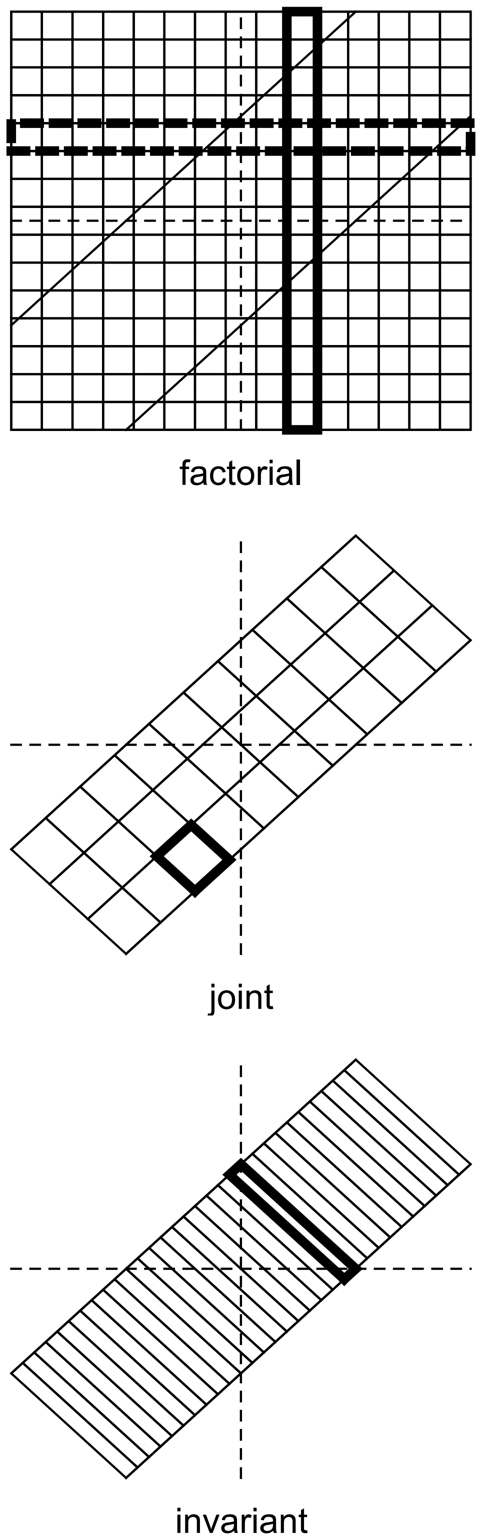


Figure 8: Three alternative ways of using 30 code indices to encode a pair of correlated variables. The typical code cells are shown in bold.

1. Factorial Encoding

A factorial encoder may be trained by setting the parameter values as follows: code book size $M = 16$, number of code indices sampled $n = 20$, $\varepsilon = 0.2$ for 500 training steps, $\varepsilon = 0.1$ for a further 500 training steps. This is the same as in the case of independent objects, except that the number of training steps has been doubled.

The result is shown in figure 9 which should be compared with the result for independent objects in figure 7. The presence of correlations degrades the quality of this factorial code relative to the case of independent objects. The contamination of the factorial code takes the form of a few code indices which respond jointly to the pair of objects.

The joint coding contamination of the factorial code can be reduced by using a 2-stage encoder, in which the second stage has the same values of M and n as the first stage (although identical parameter values are not necessary), and (in this case) both stages have the same weighting in the objective function (see equation 2.10).

The results are shown in figure 10. The reason that the second stage encourages the first to adopt a pure factorial code is quite subtle. The result shown in figure 10 will lead to the first stage producing an output in which 2 code indices (one for each object) typically have probability $\frac{1}{2}$ of being sampled, and all of the remaining code indices have a very small probability (this

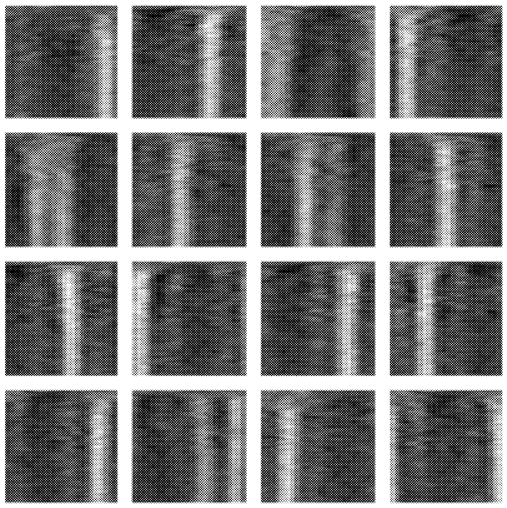


Figure 9: A factorial encoder emerges when a single stage encoder is trained on data that is a superposition of 2 objects in correlated locations.

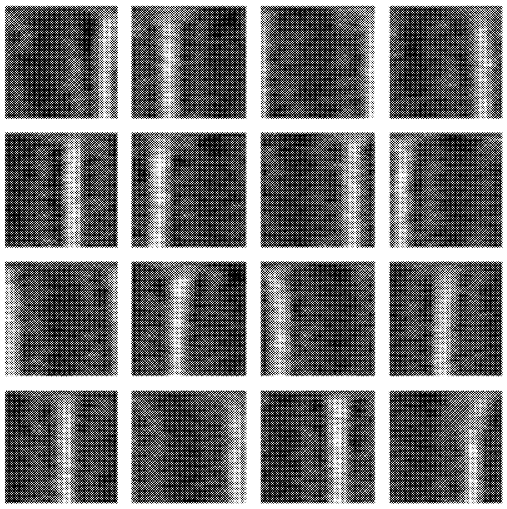


Figure 10: The factorial encoder is improved, by the removal of the joint encoding contamination, when a 2-stage encoder is used.

is an approximation which ignores the fact that the code cells overlap). On the other hand, figure 9 will lead to an output in which the probability is sometimes concentrated on a single code index. However, the contribution of the second stage to the overall objective function encourages it to encode the vector of probabilities output by the first stage with minimum Euclidean reconstruction error, which is easier to do if the situation is as in figure 10 rather than as in figure 9. In effect, the second stage likes to see an output from the first stage in which a large number of code indices are each sampled with a low probability, which favours factorial coding over joint encoding.

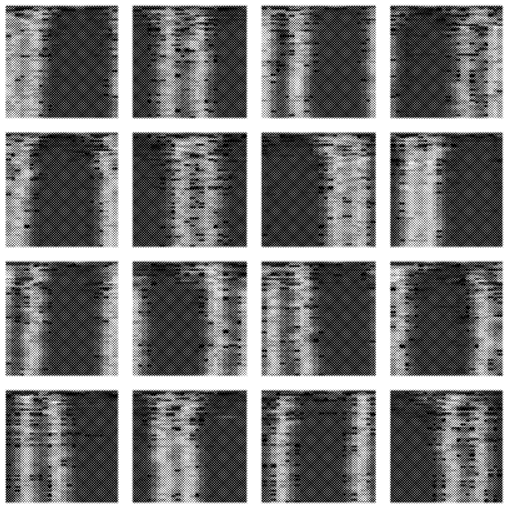


Figure 11: A joint encoder emerges when a single stage encoder is trained on data that is a superposition of 2 objects in correlated locations.

2. Joint Encoding

A joint encoder may be trained by setting the parameter values as follows: code book size $M = 16$, number of code indices sampled $n = 3$, $\varepsilon = 0.2$ for 500 training steps, $\varepsilon = 0.1$ for a further 500 training steps, $\varepsilon = 0.05$ for a further 1000 training steps. This is the same as the parameter values for the factorial encoder above, except that n has been reduced to $n = 3$, and the training schedule has been extended.

The result is shown in figure 11. After some initial confusion, the reconstruction vectors self-organise so that each code index corresponds to a *pair* of objects at well defined locations, so the code index jointly encodes the pair of object positions; this is a joint encoder. The small value of n prevents a factorial encoder from emerging.

3. Invariant Encoding

An invariant encoder may be trained by using a 2-stage encoder, and setting the parameter values identically in each stage as follows (where the weighting of the second stage relative to the first is denoted as s): code book size $M = 16$, number of code indices sampled $n = 3$, $\varepsilon = 0.2$ and $s = 5$ for 500 training steps, $\varepsilon = 0.1$ and $s = 10$ for a further 500 training steps, $\varepsilon = 0.05$ and $s = 20$ for a further 500 training steps, $\varepsilon = 0.05$ and $s = 40$ for a further 500 training steps. This is basically the same as the parameter values used for the joint encoder above, except that there are now 2 stages, and the weighting of the second stage is progressively increased throughout the training schedule. Note that the large value that is used for s is offset to a certain extent by the fact that the ratio of the normalisation of the inputs to the first

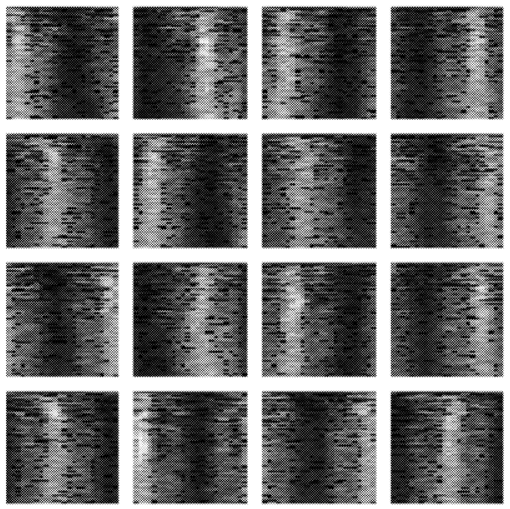


Figure 12: An invariant encoder emerges when 2-stage encoder is trained on data that is a superposition of 2 objects in correlated locations.

and second stages is very large; the anomalous normalisation of the input to the first stage could be removed by insisting that the input to the first stage is a vector of probabilities, but that is not done in these simulations.

The result is shown in figure 12. During the early part of the training schedule the weighting of the second stage is still relatively small, so it has the effect of turning what would otherwise have been a joint encoder into a factorial encoder; this is analogous to the effect observed when figure 9 becomes figure 10. However, as the training schedule progresses the weighting of the second stage increases further, and the reconstruction vectors self-organise so that each code index corresponds to a *pair* of objects with a well defined centroid but indeterminate separation. Thus each code index encodes only the centroid of the pair of objects and ignores their separation. This is a new type of encoder that arises when the objects are correlated, and it will be called an *invariant* encoder, in recognition of the fact that its output is invariant with respect to the separation of the objects.

Note that in these results there is a large amount of overlap between the code cells, which should be taken into account when interpreting the illustration in figure 8.

IV. CONCLUSIONS

The numerical results presented in this paper show that a stochastic vector quantiser (VQ) can be trained to find a variety of different types of way of encoding high-dimensional input vectors. These input vectors are generated in two stages. Firstly, a low-dimensional manifold is created whose intrinsic coordinates are the positions of the objects in the scene; this corresponds to generating the scene itself. Secondly, this manifold is non-linearly embedded to create a curved manifold that lives in a high-dimensional space of image pixel values; this corresponds to imaging the generated scene.

Three fundamentally different types of encoder have been demonstrated, which differ in the way that they build a reconstruction that approximates the input vector:

1. A factorial encoder uses a reconstruction that is superposition of a *number* of vectors that each lives in a well defined input subspace, which is useful for discovering constituent objects in the input vector. This result is a type of independent component analysis (ICA) [4].

2. A joint encoder uses a reconstruction that is a *single* vector that lives in the whole input space. This result is basically the same as what would be obtained by using a standard VQ [7].

3. An invariant encoder uses a reconstruction that is a *single* vector that lives in a subspace of the whole input space, so it ignores some dimensions of the input vector, which is therefore useful for discovering correlated objects whilst rejecting uninteresting fluctuations in their relative coordinates. This is similar to self-organising transformation invariant detectors described in [12].

More generally, the encoder will be a hybrid of these basic types, depending on the interplay between the statistical properties of the input vector and the parameter settings of the SVQ.

V. ACKNOWLEDGEMENT

I thank Chris Webber for many useful conversations that we had during the course of this research.

Appendix A: Derivatives of the Objective Function

In order to minimise $D_1 + D_2$ it is necessary to compute its derivatives. The derivatives were presented in detail in [9] for a single stage chain (i.e. a single FMC). The purpose of this appendix is to extend this derivation to a multi-stage chain of linked FMCs. In order to write the various expressions compactly, infinitesimal variations will be used thoughout this appendix, so that $\delta(uv) = \delta u\, v + u\, \delta v$ will be written rather than $\frac{\partial(uv)}{\partial\theta} = \frac{\partial u}{\partial\theta} v + u \frac{\partial v}{\partial\theta}$ (for some parameter θ). The calculation will be done in a top-down fashion, differentiating the objective function first, then differentiating anything that the objective function depends on, and so on following the dependencies down until only constants are left (this is essentially the chain rule of differentiation).

The derivative of $D_1 + D_2$ (defined in equation 2.10) is given by

$$\delta \sum_{l=1}^{L} s^{(l)} \left(D_1^{(l)} + D_2^{(l)}\right) = \sum_{l=1}^{L} s^{(l)} \left(\delta D_1^{(l)} + \delta D_2^{(l)}\right) \tag{A1}$$

The derivatives of the $D_1^{(l)}$ and $D_2^{(l)}$ parts (defined in equation 2.7, with appropriate (l) superscripts added) of the contribution of stage l to $D_1 + D_2$ are given by (dropping the (l) superscripts again, for clarity)

$$\begin{aligned} \delta D_1 &= \frac{2}{n} \int d\mathbf{x} \, \Pr(\mathbf{x}) \sum_{y=1}^{M} \left(\delta \Pr(y|\mathbf{x}) \, \|\mathbf{x} - \mathbf{x}'(y)\|^2 + 2 \Pr(y|\mathbf{x}) \, (\delta\mathbf{x} - \delta\mathbf{x}'(y)) \cdot (\mathbf{x} - \mathbf{x}'(y))\right) \\ \delta D_2 &= \frac{4(n-1)}{n} \int d\mathbf{x} \, \Pr(\mathbf{x}) \sum_{y=1}^{M} \left(\delta\mathbf{x} - \sum_{y'=1}^{M} \left(\delta \Pr(y'|\mathbf{x}) \, \mathbf{x}'(y') + \Pr(y'|\mathbf{x}) \, \delta\mathbf{x}'(y')\right)\right) \cdot \left(\mathbf{x} - \sum_{y'=1}^{M} \Pr(y'|\mathbf{x}) \, \mathbf{x}'(y')\right) \end{aligned} \tag{A2}$$

In numerical simulations the exact piecewise linear solution for the optimum $\Pr(y|\mathbf{x})$ (see section II C) will *not* be sought, rather $\Pr(y|\mathbf{x})$ will be modelled using a simple parametric form, and then the parameters will be optimised. This model of $\Pr(y|\mathbf{x})$ will not in general include the ideal piecewise linear optimum solution, so using it amounts to replacing $D_1 + D_2$, which is an upper bound on the objective function D (see equation 2.10), by an even weaker upper bound on D. The justification for using this approach rests on the quality of the results that are obtained from the resulting numerical simulations (see section III).

The first step in modelling $\Pr(y|\mathbf{x})$ is to explicitly state the fact that it is a probability, which is a normalised quantity. This may be done as follows

$$\Pr(y|\mathbf{x}) = \frac{Q(y|\mathbf{x})}{\sum_{y'=1}^{M} Q(y'|\mathbf{x})} \tag{A3}$$

where $Q(y|\mathbf{x}) \geq 0$ (note that there is a slight change of notation compared with [9], because $Q(y|\mathbf{x})$ rather than $Q(\mathbf{x}|y)$ is written, but the results are equivalent). The $Q(y|\mathbf{x})$ are thus unnormalised probabilities, and $\sum_{y'=1}^{M} Q(y'|\mathbf{x})$ is the normalisation factor. The derivative of $\Pr(y|\mathbf{x})$ is given by

$$\frac{\delta \Pr(y|\mathbf{x})}{\Pr(y|\mathbf{x})} = \frac{1}{Q(y|\mathbf{x})} \left(\delta Q(y|\mathbf{x}) - \Pr(y|\mathbf{x}) \sum_{y'=1}^{M} \delta Q(y'|\mathbf{x})\right) \tag{A4}$$

The second step in modelling $\Pr(y|\mathbf{x})$ is to introduce an explicit parameteric form for $Q(y|\mathbf{x})$. The following sigmoidal function will be used in this paper

$$Q(y|\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}(y) \cdot \mathbf{x} - b(y))} \tag{A5}$$

where $\mathbf{w}(y)$ is a weight vector and $b(y)$ is a bias. The derivative of $Q(y|\mathbf{x})$ is given by

$$\delta Q(y|\mathbf{x}) = Q(y|\mathbf{x}) \, (1 - Q(y|\mathbf{x})) \, (\delta\mathbf{w}(y) \cdot \mathbf{x} + \mathbf{w}(y) \cdot \delta\mathbf{x} + \delta b(y)) \tag{A6}$$

There are also $\delta\mathbf{x}$ derivatives in equation A2 and equation A6. The $\delta\mathbf{x}$ derivative arises only in multi-stage chains of FMCs, and because of the way in which stages of the chain are linked together (see equation 2.9) it is equal to the derivative of the vector of probabilities output by the previous stage. Thus the $\delta\mathbf{x}$ derivative may be obtained by following its dependencies back through the stages of the chain until the first layer is reached; this is essentially the chain rule of differentiation. This ensures that for each stage the partial derivatives include the additional contributions that arise from forward propagation through later stages, as described in section III A.

There are also $\delta\mathbf{x}'(y)$ derivatives in equation A2, but these require no further simplification.

[1] N Farvardin, *A study of vector quantisation for noisy channels*, IEEE Transactions on Information Theory **36** (1990), no. 4, 799–809.

[2] A Gersho and R M Gray, *Vector quantisation and signal compression*, Kluwer, Norwell, 1992.

[3] R M Gray, *Vector quantisation*, IEEE Transactions on Acoustics, Speech, and Signal Processing Magazine **1** (1984), no. 2, 4–29.

[4] A Hyvärinen, *Survey on independent component analysis*, Neural Computing Surveys **2** (1999), 94–128.

[5] T Kohonen, *Self-organisation and associative memory*, Springer-Verlag, Berlin, 1984.

[6] H Kumazawa, M Kasahara, and T Namekawa, *A construction of vector quantisers for noisy channels*, Electronics and Engineering in Japan **67** (1984), no. 4, 39–47.

[7] Y Linde, A Buzo, and R M Gray, *An algorithm for vector quantiser design*, IEEE Transactions on Communications **28** (1980), no. 1, 84–95.

[8] S P Luttrell, *A Bayesian analysis of self-organising maps*, Neural Computation **6** (1994), no. 5, 767–794.

[9] ______, *Mathematics of neural networks: models, algorithms and applications*, ch. A theory of self-organising neural networks, pp. 240–244, Kluwer, Boston, 1997.

[10] ______, *An adaptive network for encoding data using piecewise linear functions*, Proceedings of international confererence on artificial neural networks (Edinburgh), IEE, 1999, pp. 198–203.

[11] ______, *Combining artificial neural nets: Ensemble and modular multi-net systems*, Perspectives in Neural Computing, ch. Self-organised modular neural networks for encoding data, pp. 235–263, Springer-Verlag, London, 1999.

[12] C J S Webber, *Self-organisation of transformation-invariant detectors for constituents of perceptual patterns*, Network: Computation in Neural Systems **5** (1994), no. 4, 471–496.

[13] E Yair, K Zeger, and A Gersho, *Competitive learning and soft competition for vector quantiser design*, IEEE Transactions on Signal Processing **40** (1992), no. 2, 294–309.

Using Stochastic Encoders to Discover Structure in Data [*]

S P Luttrell[†]
DERA, St Andrews Road, Malvern, Worcs, WR14 3PS, UK

In this paper a stochastic generalisation of the standard Linde-Buzo-Gray (LBG) approach to vector quantiser (VQ) design is presented, in which the encoder is implemented as the sampling of a vector of code indices from a probability distribution derived from the input vector, and the decoder is implemented as a superposition of reconstruction vectors. This stochastic VQ (SVQ) is optimised using a minimum mean Euclidean reconstruction distortion criterion, as in the LBG case. Numerical simulations are used to demonstrate how this leads to self-organisation of the SVQ, where different stochastically sampled code indices become associated with different input subspaces.

I. INTRODUCTION

In vector quantisation a code book is used to encode each input vector as a corresponding code index, which is then decoded (again, using the codebook) to produce an approximate reconstruction of the original input vector [2, 3]. The purpose of this paper is to generalise the standard approach to vector quantiser (VQ) design [7], so that each input vector is encoded as a vector of code indices that are stochastically sampled from a probability distribution that depends on the input vector, rather than as a single code index that is the deterministic outcome of finding which entry in a code book is closest to the input vector. This will be called a stochastic VQ (SVQ), and it includes the standard VQ as a special case. Note that this approach is different from the various stochastic approches that are used to train VQs (see e.g. [12, 14, 15]), because here the codebook itself is stochastic, so the use of probability distributions is essential both during and after training.

One advantage of using the stochastic approach, which will be demonstrated in this paper, is that it automates the process of splitting high-dimensional input vectors into low-dimensional blocks before encoding them, because minimising the mean Euclidean reconstruction error can encourage different stochastically sampled code indices to become associated with different input subspaces [11]. Another advantage is that it is very easy to connect SVQs together, by using the vector of code index probabilities computed by one SVQ as the input vector to another SVQ [10].

In Section II various pieces of previously published theory are unified to give a coherent account of SVQs. In Section III the results of some new numerical simulations are presented, which demonstrate how the code indices in a SVQ can become associated in various ways with input subspaces. In the appendices various derivations relating to the detailed training of an SVQ are presented.

II. THEORY

In this section various pieces of previously published theory are unified to establish a coherent framework for modelling SVQs. In Section II A the basic theory of folded Markov chains (FMC) is given [8], and in Section II B it is extended to the case of high-dimensional input data [9]. Finally, in Section II C the theory is further generalised to chains of linked FMCs [10].

A. Folded Markov Chains

The basic building block of the encoder/decoder model used in this paper is the folded Markov chain (FMC) [8]. Thus an input vector $\boldsymbol{x}$ is encoded as a code index vector $\boldsymbol{y}$, which is then subsequently decoded as a reconstruction $\boldsymbol{x}'$ of the input vector. Both the encoding and decoding operations are allowed to be probabilistic, in the sense that $\boldsymbol{y}$ is a sample drawn from $\Pr(\boldsymbol{y}|\boldsymbol{x})$, and $\boldsymbol{x}'$ is a sample drawn from $\Pr(\boldsymbol{x}'|\boldsymbol{y})$, where $\Pr(\boldsymbol{y}|\boldsymbol{x})$ and $\Pr(\boldsymbol{x}'|\boldsymbol{y})$ are Bayes' inverses of each other, as given by $\Pr(\boldsymbol{x}'|\boldsymbol{y}) = \frac{\Pr(\boldsymbol{y}|\boldsymbol{x}) \Pr(\boldsymbol{x})}{\int d\boldsymbol{z} \Pr(\boldsymbol{y}|\boldsymbol{z}) \Pr(\boldsymbol{z})}$, and $\Pr(\boldsymbol{x})$ is the prior probability from which $\boldsymbol{x}$ was sampled. Because the chain of dependences in passing from $\boldsymbol{x}$ to $\boldsymbol{y}$ and then to $\boldsymbol{x}'$ is first order Markov (i.e. it is described by the directed graph $(\boldsymbol{x} \longrightarrow \boldsymbol{y} \longrightarrow \boldsymbol{x}')$, and because the two ends of this Markov chain (i.e. $\boldsymbol{x}$ and $\boldsymbol{x}'$) live in the same vector space, it is called a *folded* Markov chain (FMC). The operations that occur in an FMC are summarised in Figure 1.

[*]Typeset in LaTeX on May 9, 2019.
This is archived at http://arxiv.org/abs/cs/0408049, 21 Aug 2004. Full version of a short paper that was published in the Digest of the 5th IMA International Conference on Mathematics in Signal Processing, 18-20 December 2000, Warwick University, UK. © British Crown Copyright 2000 / DERA.
[†]Electronic address: luttrell@signal.dera.gov.uk

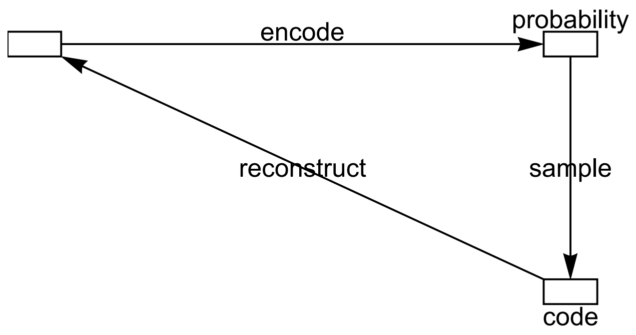


Figure 1: A folded Markov chain (FMC) in which an input vector $\boldsymbol{x}$ is encoded as a code index vector $\boldsymbol{y}$ that is drawn from a conditional probability $\Pr(\boldsymbol{y}|\boldsymbol{x})$, which is then decoded as a reconstruction vector $\boldsymbol{x}'$ drawn from the Bayes' inverse conditional probability $\Pr(\boldsymbol{x}'|\boldsymbol{y})$.

In order to ensure that the FMC encodes the input vector optimally, a measure of the reconstruction error must be minimised. There are many possible ways to define this measure, but one that is consistent with many previous results, and which also leads to many new results, is the mean Euclidean reconstruction error measure D, which is defined as

$$D \equiv \int d\boldsymbol{x}\ \Pr(\boldsymbol{x}) \sum_{y_1=1}^{M} \sum_{y_2=1}^{M} \cdots \sum_{y_n=1}^{M} \Pr(\boldsymbol{y}|\boldsymbol{x}) \int d\boldsymbol{x}'\ \Pr(\boldsymbol{x}'|\boldsymbol{y})\, ||\boldsymbol{x}-\boldsymbol{x}'||^2 \tag{1}$$

where $\boldsymbol{y} = (y_1, y_2, \cdots, y_n)$ $1 \leq y_i \leq M$ is assumed, $\Pr(\boldsymbol{x})\ \Pr(\boldsymbol{y}|\boldsymbol{x})\ \Pr(\boldsymbol{x}'|\boldsymbol{y})$ is the joint probability that the FMC has state $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{x}')$, $||\boldsymbol{x}-\boldsymbol{x}'||^2$ is the Euclidean reconstruction error, and $\int d\boldsymbol{x} \sum_{y_1=1}^{M} \sum_{y_2=1}^{M} \cdots \sum_{y_n=1}^{M} \int d\boldsymbol{x}'\,(\cdots)$ sums over all possible states of the FMC (weighted by the joint probability).

The Bayes' inverse probability $\Pr(\boldsymbol{x}'|\boldsymbol{y})$ may be integrated out of this expression for D to yield

$$D = 2 \int d\boldsymbol{x} \Pr(\boldsymbol{x}) \sum_{y_1=1}^{M} \sum_{y_2=1}^{M} \cdots \sum_{y_n=1}^{M} \Pr(\boldsymbol{y}|\boldsymbol{x})\, ||\boldsymbol{x}-\boldsymbol{x}'(\boldsymbol{y})||^2 \tag{2}$$

where the reconstruction vector $\boldsymbol{x}'(\boldsymbol{y})$ is defined as $\boldsymbol{x}'(\boldsymbol{y}) \equiv \int d\boldsymbol{x}\ \Pr(\boldsymbol{x}|\boldsymbol{y})\, \boldsymbol{x}$. Because of the quadratic form of the objective function, it turns out that $\boldsymbol{x}'(\boldsymbol{y})$ may be treated as a free parameter whose optimum value (i.e. the solution of $\frac{\partial D}{\partial \boldsymbol{x}'(\boldsymbol{y})} = 0$) is $\int d\boldsymbol{x}\ \Pr(\boldsymbol{x})\, \boldsymbol{x}$, as required.

It was shown in [8] that the standard VQ [7] and topograpic mappings [5] automatically emerge as special cases when D is minimised. In this approach, topographic mappings emerge as the optimal coding scheme when the code is to be transmitted along a noisy communication channel before being decoded [1, 6].

B. High Dimensional Input Spaces

A problem with the standard VQ is that its code book grows exponentially in size as the dimensionality of the input vector is increased, assuming that the contribution to the reconstruction error from each input dimension is held constant. This means that such VQs are useless for encoding extremely high dimensional input vectors, such as images. The usual solution to this problem is to manually partition the input space into a number of lower dimensional subspaces, and then to encode each of these subspaces separately. However, it would be very useful if this partitioning could be done automatically, in such a way that typically the correlations *within* each subspace were much stronger than the correlations *between* subspaces, so that the subspaces were approximately statistically independent of each other. The purpose of this paper is to present a solution to this problem.

The key step in solving this problem is to constrain the minimisation of D in such a way as to encourage the formation of code schemes in which each component of the code vector $\boldsymbol{y}$ codes a different subspace of the input vector $\boldsymbol{x}$. There are two related constraints that may be imposed on $\Pr(\boldsymbol{y}|\boldsymbol{x})$ and $\boldsymbol{x}'(\boldsymbol{y})$ which may be summarised as

$$\begin{aligned} \Pr(\boldsymbol{y}|\boldsymbol{x}) &= \Pr(y_1|\boldsymbol{x})\ \Pr(y_2|\boldsymbol{x}) \cdots \Pr(y_n|\boldsymbol{x}) \\ \boldsymbol{x}'(\boldsymbol{y}) &= \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}'(y_i) \end{aligned} \tag{3}$$

Thus each component y_i (for $i = 1, 2, \cdots, n$ and $1 \leq y_i \leq M$) is an *independent* sample drawn from the codebook using $\Pr(y_i|\boldsymbol{x})$ (which is assumed to be the same function for all i), and the reconstruction vector $\boldsymbol{x}'(\boldsymbol{y})$ (vector argument) is assumed to be a *superposition* of n contributions $\boldsymbol{x}'(y_i)$ (scalar argument) for $i = 1, 2, \cdots, n$. Taken together, these constraints encourage the forma-

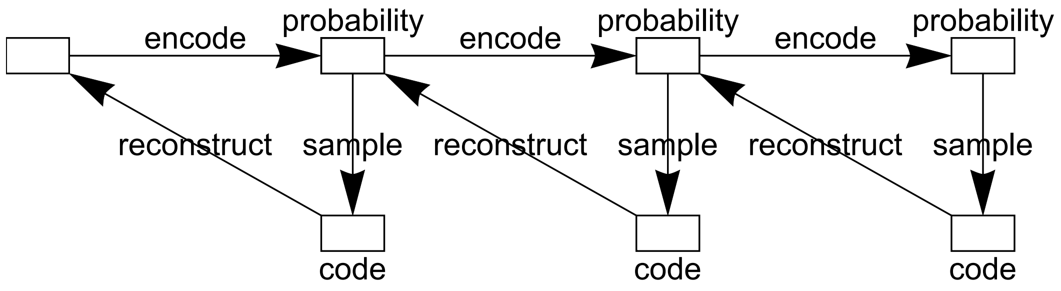


Figure 2: A chain of linked FMCs, in which the output from each stage is its vector of posterior probabilities (for all values of the code index), which is then used as the input to the next stage. Only 3 stages are shown, but any number may be used. More generally, any acyclically linked network of FMCs may be used.

tion of coding schemes in which independent subspaces are separately coded, as required.

The constraints in Equation 3 prevent the full space of possible values of $\Pr(\boldsymbol{y}|\boldsymbol{x})$ or $\boldsymbol{x}'(\boldsymbol{y})$ from being explored as D is minimised, so they lead to an *upper bound* $D_1 + D_2$ on the FMC objective function D (i.e. $D \leq D_1 + D_2$), which may be derived as [9]

$$\begin{aligned} D_1 &\equiv \tfrac{2}{n} \int d\boldsymbol{x} \, \Pr(\boldsymbol{x}) \sum_{y=1}^{M} \Pr(y|\boldsymbol{x}) \, \|\boldsymbol{x} - \boldsymbol{x}'(y)\|^2 \\ D_2 &\equiv \tfrac{2(n-1)}{n} \int d\boldsymbol{x} \, \Pr(\boldsymbol{x}) \left\| \boldsymbol{x} - \sum_{y=1}^{M} \Pr(y|\boldsymbol{x}) \, \boldsymbol{x}'(y) \right\|^2 \end{aligned} \tag{4}$$

Note that M (size of codebook) and n (number of samples drawn from codebook using $\Pr(y|\boldsymbol{x})$) are effectively model order parameters, whose values need to be chosen appropriately for each encoder optimisation problem. The properties of the optimum solution depend critically on the interplay between the statistical properties of the training data and the model order parameters M and n, as will be seen in the simulations in Section III.

C. Chains of Linked FMCs

The FMC illustrated in Figure 1 may be generalised to a chain of linked FMCs as shown in Figure 2. Each stage in this chain is an FMC of the type shown in Figure 1, and the vector of probabilities (for all values of the code index) computed by each stage is used as the input vector to the next stage; there are other ways of linking the stages together, but this is the simplest possibility. The overall objective function is a weighted sum of the FMC objective functions derived from each stage. The total number of free parameters in an L stage chain is $3L - 1$, which is the sum of 2 free parameters for each of the L stages, plus $L - 1$ weighting coefficients; there are $L - 1$ rather than L weighting coefficients because the overall normalisation of the objective function does not affect the optimum solution.

The chain of linked FMCs may be expressed mathematically by first of all introducing an index l to allow different stages of the chain to be distinguished thus

$$\begin{aligned} M &\longrightarrow M^{(l)} \\ \boldsymbol{x} &\longrightarrow \boldsymbol{x}^{(l)} \\ y &\longrightarrow y^{(l)} \\ \boldsymbol{x}' &\longrightarrow \boldsymbol{x}'^{(l)} \\ n &\longrightarrow n^{(l)} \\ D &\longrightarrow D^{(l)} \\ D_1 &\longrightarrow D_1^{(l)} \\ D_2 &\longrightarrow D_2^{(l)} \end{aligned} \tag{5}$$

The stages are then defined and linked together thus

$$\begin{aligned} \boldsymbol{x}^{(l)} &\longrightarrow y^{(l)} \longrightarrow \boldsymbol{x}'^{(l)} \\ \boldsymbol{x}^{(l+1)} &= \left(x_1^{(l+1)}, x_2^{(l+1)}, \cdots, x_{M^{(l)}}^{(l+1)} \right) \\ x_i^{(l+1)} &= \Pr(y^{(l)} = i|\boldsymbol{x}^{(l)}) \qquad 1 \leq i \leq M^{(l)} \end{aligned} \tag{6}$$

The objective function and its upper bound are then given by

$$\begin{aligned} D &= \sum_{l=1}^{L} s^{(l)} \, D^{(l)} \\ &\leq D_1 + D_2 \\ &= \sum_{l=1}^{L} s^{(l)} \left(D_1^{(l)} + D_2^{(l)} \right) \end{aligned} \tag{7}$$

where $s^{(l)} \geq 0$ is the weighting that is applied to the contribution of stage l of the chain to the overall objective function.

III. SIMULATIONS

In this section the results of various simulations are presented, which demonstrate some of the types of self-organising behaviour exhibited by an encoder that consists of a chain of linked FMCs. Synthetic, rather than real, training data are used in all of the simulations, because this allows the basic types of behaviour to be cleanly demonstrated.

In Section III A the training data is described. In Section III B a single stage encoder is trained on data that is a superposition of two randomly positioned objects. In Section III C this is generalised to objects with correlated positions, and three different types of behaviour are demonstrated: factorial encoding using both a 1-stage and a 2-stage encoders (Section III D), joint encoding using a 1-stage encoder (Section III E), and invariant encoding using a 2-stage encoder (Section III F).

In Appendix A the derivatives of the objective function are derived, and in Appendix B a gradient descent training algorithm based on these derivatives is presented.

A. Training Data

The key property that this type of self-organising encoder exhibits is its ability to automatically split up high-dimensional input spaces into lower-dimensional subspaces, each of which is separately encoded. This self-organisation manifests itself in many different ways, depending on the interplay between the statistical properties of the training data, and the 3 free parameters (i.e. the code book size M, the number of code indices sampled n, and the stage weighting s) per stage of the encoder (see Section II C).

In order to demonstrate the various different basic types of self-organisation it is necessary to use synthetic training data with controlled properties. All of the types of self-organisation that will be demonstrated in this paper may be obtained by training a 1-stage or 2-stage encoder on 24-dimensional data (i.e. $M = 24$) that consists of a superposition of a pair of identical objects (with circular wraparound to remove edge effects), such as is shown in Figure 3.

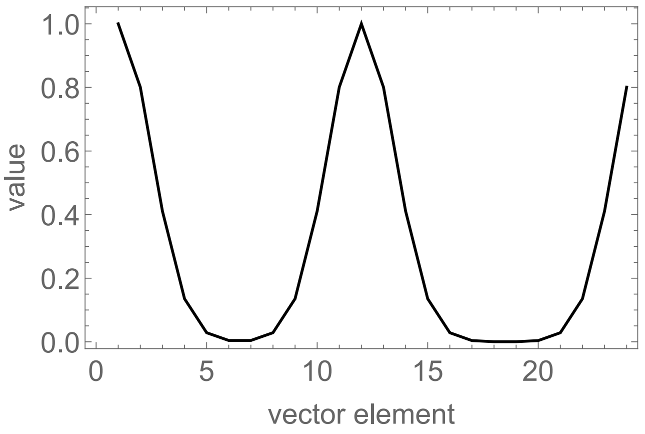


Figure 3: An example of a typical training vector for $M = 24$. Each object is a Gaussian hump with a half-width of 1.5 units, and peak amplitude of 1. The overall input vector is formed as a linear superposition of the 2 objects. Note that the input vector is wrapped around circularly to remove minor edge effects that would otherwise arise.

In the simulations presented below, two different methods of selecting the object positions are used: either the positions are statistically independent, or they are correlated. In the independent case, each object position is a random integer in the interval $[1, 24]$. In the correlated case, the first object position is a random integer in the interval $[1, 24]$, and the second object position is chosen *relative to* the first one as an integer in the range $[4, 8]$, so that the mean object separation is 6 units.

B. Independent Objects

The simplest demonstration is to let a single stage encoder discover the fact that the training data consists of a superposition of a pair of objects, which is a type of independent component analysis (ICA) [4]. This may readily be done by setting the parameter values as follows: code book size $M = 16$, number of code indices sampled $n = 20$, $\varepsilon = 0.2$ for 500 training steps, $\varepsilon = 0.1$ for a further 500 training steps.

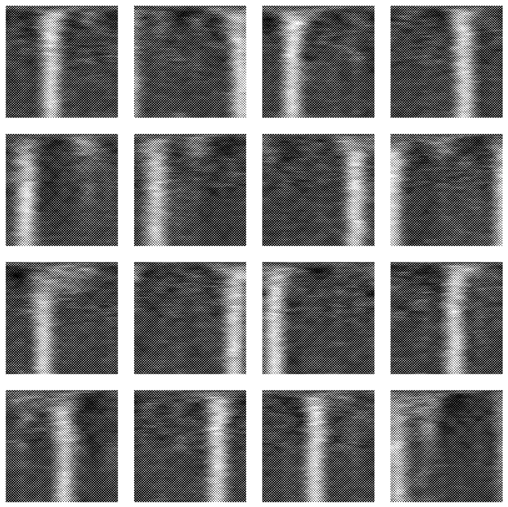


Figure 4: A factorial encoder emerges when a single stage encoder is trained on data that is a superposition of 2 objects in independent locations.

The self-organisation of each of the 16 reconstruction vectors as training progresses (measured down the page) is shown in Figure 4. After some initial confusion, the reconstruction vectors self-organise so that each code index corresponds to a *single* object at a well defined location, whose width automatically adjusts itself so that the M reconstruction vectors cover the whole input space. This behaviour is non-trivial, because each training vector is a superposition of a *pair* of objects at independent locations, so two different code index values must be sampled by the encoder (assuming that the two objects are not at the same location); the relatively large choice $n = 20$ ensures that it is highly likely that both code index values will be amongst the n random samples [11]. This result is called a factorial encoder, because the objects are encoded separately.

The case of a joint encoder, where each code index corresponds to a *pair* of objects at well defined locations, requires a rather large code book when the objects are independent. However, when correlations between the objects are introduced then the code book can be reduced to a manageable size, as will be demonstrated in the next section.

C. Correlated Objects

If the positions of the pair of objects are mutually correlated, then they can be encoded in 3 fundamentally different ways:

1. Factorial encoder. This encoder ignores the correlations between the objects, and encodes them as if they were 2 independent objects. Each code index thus encodes a single object position, so many code indices must be sampled in order to virtually guarantee that both object positions are encoded [11]. This result is a type of independent component analysis (ICA) [4].
2. Joint encoder. This encoder regards each possible joint placement of the 2 objects as a distinct configuration. Each code index thus encodes a pair of object positions, so only one code index needs to be sampled in order to guarantee that both object positions are encoded [11]. This result is basically the same as what would be obtained by using a standard VQ [7].
3. Invariant encoder. This encoder regards each possible placement of the centroid of the 2 objects as a distinct configuration, but regards all possible object separations (for a given centroid) as being equivalent. Each code index thus encodes only the centroid of the pair of objects. This type of encoder does not arise when the objects are independent. This is similar to self-organising transformation invariant detectors described in [13].

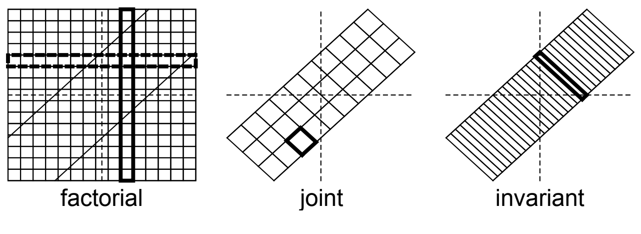


Figure 5: Three alternative ways of using 30 code indices to encode a pair of correlated variables. The typical code cells are shown in bold.

Each of these 3 possibilities is shown in Figure 5, where the diagrams are meant only to be illustrative. The correlated variables live in the large 2-dimensional rectangular region extending from bottom-left to top-right of each diagram.

The factorial encoder has two orthogonal sets of long thin rectangular code cells, and the diagram shows how a pair of such cells intersect to define a small square code cell. The joint encoder behaves as a standard vector quantiser, and is illustrated as having a set of square code cells, although their shapes will not be as simple as this in practice. The invariant encoder ideally has a set of long thin rectangular code cells that encode only the long diagonal dimension.

In all 3 cases there is overlap between code cells. In the case of the factorial and joint encoders the overlap tends to be only between nearby code cells, whereas in the case of an invariant encoder the range of the overlap is usually much greater, as will be seen in the numerical simulations below. In practice the optimum encoder may not be a clean example of one of the types illustrated in Figure 5, as will also be seen in the numerical simulations below.

D. Factorial Encoding

A factorial encoder may be trained by setting the parameter values as follows: code book size $M = 16$, number of code indices sampled $n = 20$, $\varepsilon = 0.2$ for 500 training steps, $\varepsilon = 0.1$ for a further 500 training steps.

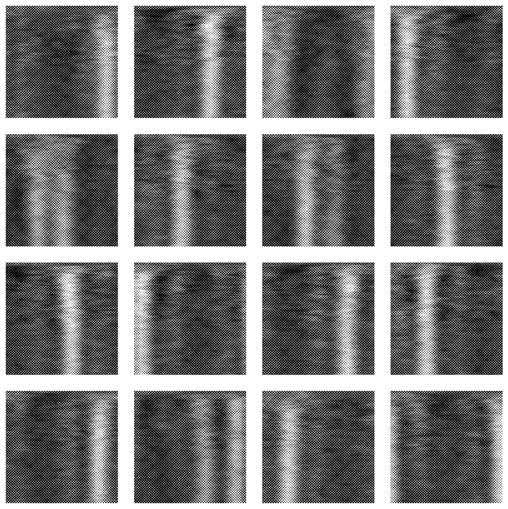


Figure 6: A factorial encoder emerges when a single stage encoder is trained on data that is a superposition of 2 objects in correlated locations.

The result is shown in Figure 6 which should be compared with the result for independent objects in Figure 4. The presence of correlations degrades the quality of this factorial code relative to the case of independent objects. The contamination of the factorial code takes the form of a few code indices which respond jointly to the pair of objects.

The joint coding contamination of the factorial code can be reduced by using a 2-stage encoder, in which the second stage has the same values of M and n as the first stage (although identical parameter values are not necessary), and (in this case) both stages have the same weighting in the objective function (see Equation 7).

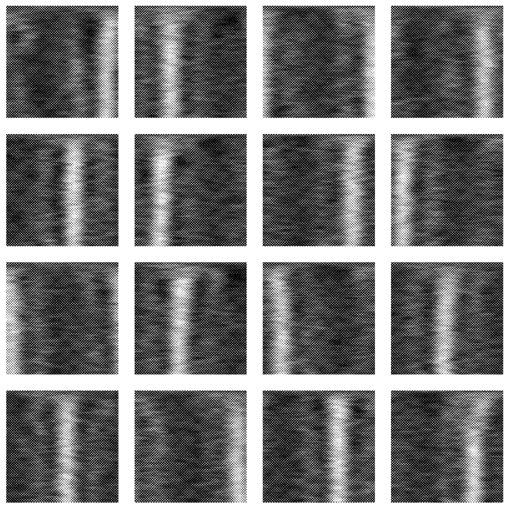


Figure 7: The factorial encoder is improved, by the removal of the joint encoding contamination, when a 2-stage encoder is used.

The results are shown in Figure 7. The reason that the second stage encourages the first to adopt a pure factorial code is quite subtle. The result shown in Figure 7 will lead to the first stage producing an output in which 2 code indices (one for each object) each typically have probability $\frac{1}{2}$ of being sampled, and all of the remaining code indices have a very small probability (this is an approximation which ignores the fact that the code cells overlap). On the other hand, Figure 6 will lead to an output in which the probability can be concentrated on a single code index, if it can jointly code the pair of objects. However, the contribution of the second stage to the overall objective function encourages it to encode the vector of probabilities output by the first stage with minimum Euclidean reconstruction error, which is easier to do if the situation is as in Figure 7 rather than as in Figure 6. In effect, the second stage likes to see an output from the first stage in which more than one code index has a significant probability of being sampled, which favours factorial coding over joint encoding.

E. Joint Encoding

A joint encoder may be trained by setting the parameter values as follows: code book size $M = 16$, number of code indices sampled $n = 3$, $\varepsilon = 0.2$ for 500 training steps, $\varepsilon = 0.1$ for a further 500 training steps, $\varepsilon = 0.05$ for a further 1000 training steps. This is the same as the parameter values for the factorial encoder above, except that n has been reduced to $n = 3$, and the training schedule has been extended.

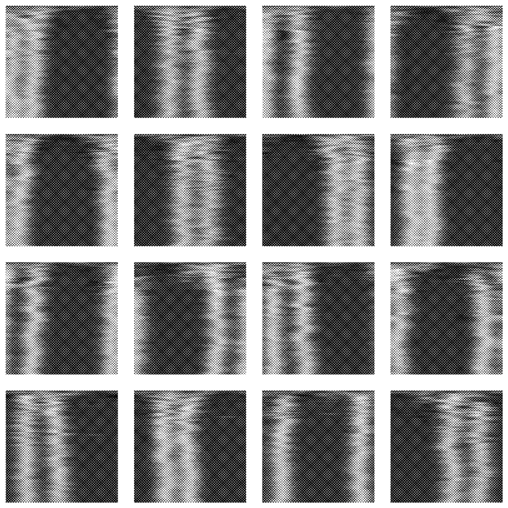


Figure 8: A joint encoder emerges when a single stage encoder is trained on data that is a superposition of 2 objects in correlated locations.

The result is shown in Figure 8. After some initial confusion, the reconstruction vectors self-organise so that each code index corresponds to a *pair* of objects at well defined locations, so the code index jointly encodes the pair of object positions; this is a joint encoder. The small value of n prevents a factorial encoder from emerging [11].

F. Invariant Encoding

An invariant encoder may be trained by using a 2-stage encoder, and setting the parameter values identically in each stage as follows (where the weighting of the second stage relative to the first is denoted as s): code book size $M = 16$, number of code indices sampled $n = 3$, $\varepsilon = 0.2$ and $s = 5$ for 500 training steps, $\varepsilon = 0.1$ and $s = 10$ for a further 500 training steps, $\varepsilon = 0.05$ and $s = 20$ for a further 500 training steps, $\varepsilon = 0.05$ and $s = 40$ for a further 500 training steps. This is basically the same as the parameter values used for the joint encoder above, except that there are now 2 stages, and the weighting of the second stage is progressively increased throughout the training schedule. Note that the large value that is used for s is offset to a certain extent by the fact that the ratio of the normalisation of the inputs to the first and second stages is very large; the anomalous normalisation of the input to the first stage could be removed by insisting that the input to the first stage is a vector of probabilities, but that is not done in these simulations.

The result is shown in Figure 9. During the early part of the training schedule the weighting of the second stage is still relatively small, so it has the effect of turning what would otherwise have been a joint encoder into a factorial encoder; this is analogous to the effect ob-

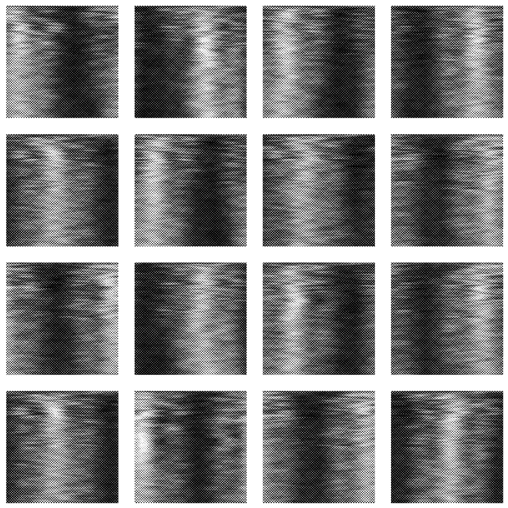


Figure 9: An invariant encoder emerges when 2-stage encoder is trained on data that is a superposition of 2 objects in correlated locations.

served when Figure 6 becomes Figure 7. However, as the training schedule progresses the weighting of the second stage increases further, and the reconstruction vectors self-organise so that each code index corresponds to a *pair* of objects with a well defined centroid but indeterminate separation. Thus each code index encodes only the centroid of the pair of objects and ignores their separation. This is a new type of encoder that arises when the objects are correlated, and it will be called an *invariant* encoder, in recognition of the fact that its output is invariant with respect to the separation of the objects.

Note that in these results there is a large amount of overlap between the code cells, which should be taken into account when interpreting the illustration in Figure 5. This is an extreme example the second stage preferring an output from the first stage in which more than one code index has a significant probability of being sampled; the large amount of overlap between code cells means that many code indices have a significant probability of being sampled.

IV. CONCLUSIONS

The numerical results presented in this paper show that a stochastic vector quantiser (SVQ) can self-organise to find a variety of different types of way of encoding high-dimensional input vectors. Three fundamentally different types of encoder have been demonstrated, which differ in the way that they build a reconstruction that approximates the input vector:

1. A factorial encoder uses a reconstruction that is superposition of a *number* of vectors that each lives in a well defined input subspace, which is useful for discovering constituent objects in the input vector. This result is a type of independent component analysis (ICA) [4].

2. A joint encoder uses a reconstruction that is a *single* vector that lives in the whole input space. This result is basically the same as what would be obtained by using a standard VQ [7].

3. An invariant encoder uses a reconstruction that is a *single* vector that lives in a subspace of the whole input space, so it ignores some dimensions of the input vector, which is therefore useful for discovering correlated objects whilst rejecting uninteresting fluctuations in their relative coordinates. This is similar to self-organising transformation invariant detectors described in [13].

More generally, the encoder will be a hybrid of these basic types, depending on the interplay between the statistical properties of the input vector and the parameter settings of the SVQ.

Appendix A: Derivatives of the Objective Function

In order to minimise $D_1 + D_2$ it is necessary to compute its derivatives. The derivatives were presented in detail in [9] for a single stage chain (i.e. a single FMC). The purpose of this appendix is to extend this derivation to a multi-stage chain of linked FMCs. In order to write the various expressions compactly, infinitesimal variations will be used thoughout this appendix, so that $\delta(uv) = \delta u v + u \delta v$ will be written rather than $\frac{\partial(uv)}{\partial\theta} = \frac{\partial u}{\partial\theta} v + u \frac{\partial v}{\partial\theta}$ (for some parameter θ). The calculation will be done in a top-down fashion, differentiating the objective function first, then differentiating anything that the objective function depends on, and so on following the dependencies down until only constants are left (this is essentially the chain rule of differentiation).

The derivative of $D_1 + D_2$ (defined in Equation 7) is given by

$$\delta \sum_{l=1}^{L} s^{(l)} \left(D_1^{(l)} + D_2^{(l)} \right) = \sum_{l=1}^{L} s^{(l)} \left(\delta D_1^{(l)} + \delta D_2^{(l)} \right) \tag{A1}$$

The derivatives of the $D_1^{(l)}$ and $D_2^{(l)}$ parts (defined in Equation 4, with appropriate (l) superscripts added) of the contribution of stage l to $D_1 + D_2$ are given by (dropping the (l) superscripts again, for clarity)

$$\begin{aligned}
\delta D_1 &= \frac{2}{n}\int d\boldsymbol{x}\,\Pr(\boldsymbol{x})\sum_{y=1}^{M}\Big(\delta Pr(y|\boldsymbol{x})\,||\boldsymbol{x}-\boldsymbol{x}'(y)||^2 + 2\Pr(y|\boldsymbol{x})\,\left(\delta\boldsymbol{x}-\delta\boldsymbol{x}'(y)\right)\Big).(\boldsymbol{x}-\boldsymbol{x}'(y)) \\
\delta D_2 &= \frac{4(n-1)}{n}\int d\boldsymbol{x}\,\Pr(\boldsymbol{x})\left(\delta\boldsymbol{x}-\sum_{y=1}^{M}\left(\delta Pr(y|\boldsymbol{x})\,\boldsymbol{x}'(y)+\Pr(y|\boldsymbol{x})\,\delta\boldsymbol{x}'(y)\right)\right).\left(\boldsymbol{x}-\sum_{y'=1}^{M}\Pr(y'|\boldsymbol{x})\,\boldsymbol{x}'(y')\right)
\end{aligned} \tag{A2}$$

The first step in modelling $\Pr(y|\boldsymbol{x})$ is to explicitly state the fact that it is a probability, which is a non-negative normalised quantity. This may be done as follows

$$\Pr(y|\boldsymbol{x}) = \frac{Q(y|\boldsymbol{x})}{\sum_{y'=1}^{M} Q(y'|\boldsymbol{x})} \tag{A3}$$

where $Q(y|\boldsymbol{x}) \geq 0$. The $Q(y|\boldsymbol{x})$ are unnormalised probabilities, and $\sum_{y'=1}^{M} Q(y'|\boldsymbol{x})$ is the normalisation factor. The derivative of $\Pr(y|\boldsymbol{x})$ is given by

$$\frac{\delta Pr(y|\boldsymbol{x})}{\Pr(y|\boldsymbol{x})} = \frac{1}{Q(y|\boldsymbol{x})}\left(\delta Q(y|\boldsymbol{x}) - \Pr(y|\boldsymbol{x})\sum_{y'=1}^{M}\delta Q(y'|\boldsymbol{x})\right) \tag{A4}$$

The second step in modelling $\Pr(y|\boldsymbol{x})$ is to introduce an explicit parameteric form for $Q(y|\boldsymbol{x})$. The following sigmoidal function will be used in this paper

$$Q(y|\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{w}(y).\boldsymbol{x}-b(y))} \tag{A5}$$

where $\boldsymbol{w}(y)$ is a weight vector and $b(y)$ is a bias. The derivative of $Q(y|\boldsymbol{x})$ is given by

$$\delta Q(y|\boldsymbol{x}) = Q(y|\boldsymbol{x})\,(1-Q(y|\boldsymbol{x}))\,(\delta\boldsymbol{w}(y).\boldsymbol{x}+\boldsymbol{w}(y).\delta\boldsymbol{x}+\delta b(y)) \tag{A6}$$

This has reduced the δD_1 and δD_2 derivatives to $\delta\boldsymbol{w}(y)$, $\delta b(y)$, $\delta\boldsymbol{x}'(y)$ and $\delta\boldsymbol{x}$ derivatives. The $\delta\boldsymbol{w}(y)$, $\delta b(y)$ and $\delta\boldsymbol{x}'(y)$ derivatives relate directly to the parameters being optimised and thus need no further simplification, however the $\delta\boldsymbol{x}$ derivatives in Equation A2 and Equation A6 need some further attention. The $\delta\boldsymbol{x}$ derivative arises only in multi-stage chains of FMCs, and because of the way in which stages of the chain are linked together (see Equation 6) it is equal to the derivative of the vector of probabilities output by the previous stage. Thus the $\delta\boldsymbol{x}$ derivative may be obtained by following its dependencies back through the stages of the chain until the first layer is reached; this is essentially the chain rule of differentiation. This ensures that for each stage the partial derivatives include the additional contributions that arise from forward propagation through later stages, as described in Appendix B.

Appendix B: Training Algorithm

Assuming that $\Pr(y|\boldsymbol{x})$ is modelled as in appendix A (i.e. $\Pr(y|\boldsymbol{x}) = \frac{Q(y|\boldsymbol{x})}{\sum_{y'=1}^{M} Q(y'|\boldsymbol{x})}$ and $Q(y|\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{w}(y).\boldsymbol{x}-b(y))}$), then the partial derivatives of $D_1 + D_2$ with respect to the 3 types of parameters in a single stage of the encoder may be denoted as

$$\begin{aligned}
\boldsymbol{g}_w(y) &\equiv \tfrac{\partial(D_1+D_2)}{\partial\boldsymbol{w}(y)} \\
g_b(y) &\equiv \tfrac{\partial(D_1+D_2)}{\partial b(y)} \\
\boldsymbol{g}_x(y) &\equiv \tfrac{\partial(D_1+D_2)}{\partial\boldsymbol{x}'(y)}
\end{aligned} \tag{B1}$$

This may be generalised to each stage of a multi-stage encoder by including an (l) superscript, and ensuring that for each stage the partial derivatives include the additional contributions that arise from forward propagation through later stages; this is essentially an application of the chain rule of differentiation, using the derivatives $\frac{\partial\boldsymbol{x}^{(l+1)}}{\partial\boldsymbol{w}^{(l)}(y^{(l)})}$ and $\frac{\partial\boldsymbol{x}^{(l+1)}}{\partial b^{(l)}(y^{(l)})}$ to link the stages together (see appendix A).

A simple algorithm for updating these parameters is (omitting the (l) superscript, for clarity)

$$\begin{aligned} \boldsymbol{w}(y) &\longrightarrow \boldsymbol{w}(y) - \varepsilon \frac{\boldsymbol{g}_w(y)}{g_{w,0}} \\ b(y) &\longrightarrow b(y) - \varepsilon \frac{g_b(y)}{g_{b,0}} \\ \boldsymbol{x}'(y) &\longrightarrow \boldsymbol{x}'(y) - \varepsilon \frac{\boldsymbol{g}_x(y)}{g_{x,0}} \end{aligned} \tag{B2}$$

where ε is a small update step size parameter, and the three normalisation factors are defined as

$$\begin{aligned} g_{w,0} &\equiv \max_y \sqrt{\frac{\|\boldsymbol{g}_w(y)\|^2}{\dim \boldsymbol{x}}} \\ g_{b,0} &\equiv \max_y |b(y)| \\ g_{x,0} &\equiv \max_y \sqrt{\frac{\|\boldsymbol{g}_x(y)\|^2}{\dim \boldsymbol{x}}} \end{aligned} \tag{B3}$$

The $\frac{g_w(y)}{g_{w,0}}$ and $\frac{g_x(y)}{g_{x,0}}$ factors ensure that the maximum update step size for $\boldsymbol{w}(y)$ and $\boldsymbol{x}'(y)$ is $\varepsilon \dim \boldsymbol{x}$ (i.e. ε per dimension), and the $\frac{g_b(y)}{g_{b,0}}$ factor ensures that the maximum update step size for $b(y)$ is ε. When a stationary point of $D_1 + D_2$ is reached, the finite size of ε prevents the parmater values from converging to a perfectly stationary solution, and instead they jump around in its neighbourhood.

This update algorithm can be generalised to use a different ε for each stage of the encoder, and also to allow a different ε to be used for each of the 3 types of parameter. Furthermore, the size of ε can be varied as training proceeds, usually starting with a large value, and then gradually reducing its size to obtain an accurate estimate of the stationary solution. It is not possible to give general rules for exactly how to do this, because training conditions depend very much on the statistical properties of the training set.

[1] N Farvardin, *A study of vector quantisation for noisy channels*, IEEE Transactions on Information Theory **36** (1990), no. 4, 799–809.

[2] A Gersho and R M Gray, *Vector quantisation and signal compression*, Kluwer, Norwell, 1992.

[3] R M Gray, *Vector quantisation*, IEEE Transactions on Acoustics, Speech, and Signal Processing Magazine **1** (1984), no. 2, 4–29.

[4] A Hyvärinen, *Survey on independent component analysis*, Neural Computing Surveys **2** (1999), 94–128.

[5] T Kohonen, *Self-organisation and associative memory*, Springer-Verlag, Berlin, 1984.

[6] H Kumazawa, M Kasahara, and T Namekawa, *A construction of vector quantisers for noisy channels*, Electronics and Engineering in Japan **67** (1984), no. 4, 39–47.

[7] Y Linde, A Buzo, and R M Gray, *An algorithm for vector quantiser design*, IEEE Transactions on Communications **28** (1980), no. 1, 84–95.

[8] S P Luttrell, *A Bayesian analysis of self-organising maps*, Neural Computation **6** (1994), no. 5, 767–794.

[9] ———, *Mathematics of neural networks: models, algorithms and applications*, ch. A theory of self-organising neural networks, pp. 240–244, Kluwer, Boston, 1997.

[10] ———, *An adaptive network for encoding data using piecewise linear functions*, Proceedings of international confererence on artificial neural networks (Edinburgh), IEE, 1999, pp. 198–203.

[11] ———, *Combining artificial neural nets: Ensemble and modular multi-net systems*, Perspectives in Neural Computing, ch. Self-organised modular neural networks for encoding data, pp. 235–263, Springer-Verlag, London, 1999.

[12] L Torres, J R Casas, and E Arias, *Stochastic vector quantisation of images*, Signal Processing **62** (1997), no. 3, 291–301.

[13] C J S Webber, *Self-organisation of transformation-invariant detectors for constituents of perceptual patterns*, Network: Computation in Neural Systems **5** (1994), no. 4, 471–496.

[14] E Yair, K Zeger, and A Gersho, *Competitive learning and soft competition for vector quantiser design*, IEEE Transactions on Signal Processing **40** (1992), no. 2, 294–309.

[15] K Zeger and A Gersho, *Stochastic relaxation algorithm for improved vector quantiser design*, Electronics Letters **25** (1989), no. 14, 896–898.

Invariant Stochastic Encoders [*]

S P Luttrell[†]
DERA, St Andrews Road, Malvern, Worcs, WR14 3PS, UK

The theory of stochastic vector quantisers (SVQ) has been extended to allow the quantiser to develop invariances, so that only "large" degrees of freedom in the input vector are represented in the code. This has been applied to the problem of encoding data vectors which are a superposition of a "large" jammer and a "small" signal, so that only the jammer is represented in the code. This allows the jammer to be subtracted from the total input vector (i.e. the jammer is nulled), leaving a residual that contains only the underlying signal. The main advantage of this approach to jammer nulling is that little prior knowledge of the jammer is assumed, because these properties are automatically discovered by the SVQ as it is trained on examples of input vectors.

I. INTRODUCTION

In vector quantisation a code book is used to encode each input vector as a corresponding code index, which is then decoded (again, using the codebook) to produce an approximate reconstruction of the original input vector [3, 4]. The standard approach to vector quantiser (VQ) design [5] may be generalised [7] so that each input vector is encoded as a *vector* of code indices that are stochastically sampled from a probability distribution that depends on the input vector, rather than as a *single* code index that is the deterministic outcome of finding which entry in a code book is closest to the input vector. This will be called a stochastic VQ (SVQ), and it includes the standard VQ as a special case.

One advantage of using the stochastic approach is that it automates the process of splitting high-dimensional input vectors into low-dimensional blocks before encoding them, because minimising the mean Euclidean reconstruction error can encourage different stochastically sampled code indices to become associated with different input subspaces [8, 10]. Another advantage is that it is very easy to connect SVQs together, by using the vector of code index probabilities computed by one SVQ as the input vector to another SVQ [9].

SVQ theory will be extended to the case of encoding noisy (or distorted) data, with the intention of subsequently reconstructing an approximation to the noiseless data. This theory is then applied to the problem of encoding data vectors which are a superposition of a "large" jammer and a "small" signal, where the signal is regarded as a distortion superimposed on the jammer, rather than the other way around. The reconstruction is then an approximation to the jammer, which can thus be subtracted from the original data to reveal the underlying signal of interest.

In Section II the underlying theory of SVQs is developed together with its extension to the encoding of noisy data, and in Section III some simulations illustrating the application of SVQs to the nulling of jammers are presented.

II. STOCHASTIC VECTOR QUANTISER THEORY

In Section II A the basic theory of folded Markov chains (FMC) is given [6], in Section II B FMC theory is extended to the case of encoding noisy or distorted data with the intention of eventually recovering the undistorted data, in Section II C this extended theory is applied to the problem of encoding data that contain unwanted "nuisance degrees of freedom", in Section II D some constraints (including the threshold trick of [11]) on the optimisation of the encoder are introduced to encourage the encoder to disregard the nuisance degrees of freedom (i.e. discover invariances), and finally in Section II E this invariant encoder theory is applied to the problem of encoding and subsequently nulling "large" jammers that obscure "small" signals.

A. Folded Markov Chains

The basic building block of the SVQ used in this paper is the folded Markov chain (FMC) [6]. An input vector $\boldsymbol{x}$ is encoded as a code index vector $\boldsymbol{y}$, which is then subsequently decoded as a reconstruction $\boldsymbol{x}'$ of the input vector. Both the encoding and decoding operations are allowed to be probabilistic, in the sense that $\boldsymbol{y}$ is a sample drawn from $\Pr(\boldsymbol{y}|\boldsymbol{x})$, and $\boldsymbol{x}'$ is a sample drawn from $\Pr(\boldsymbol{x}'|\boldsymbol{y})$, where $\Pr(\boldsymbol{y}|\boldsymbol{x})$ and $\Pr(\boldsymbol{x}'|\boldsymbol{y})$ are Bayes' inverses of each other, as given by $\Pr(\boldsymbol{x}'|\boldsymbol{y}) = \frac{\Pr(\boldsymbol{y}|\boldsymbol{x})\ \Pr(\boldsymbol{x})}{\int d\boldsymbol{z}\ \Pr(\boldsymbol{y}|\boldsymbol{z})\ \Pr(\boldsymbol{z})}$, and $\Pr(\boldsymbol{x})$ is the prior probability from which $\boldsymbol{x}$ is sampled.

In order to ensure that the FMC encodes the input vector optimally, a measure of the reconstruction error must be minimised. There are many possible ways to define

[*] Typeset in LaTeX on May 9, 2019.
This is archived at http://arxiv.org/abs/cs/0408050, 21 Aug 2004. Full version of a short paper that was published in the Digest of the 5th IMA International Conference on Mathematics in Signal Processing, 18-20 December 2000, Warwick University, UK. © British Crown Copyright 2000 / DERA
[†] Electronic address: luttrell@signal.dera.gov.uk

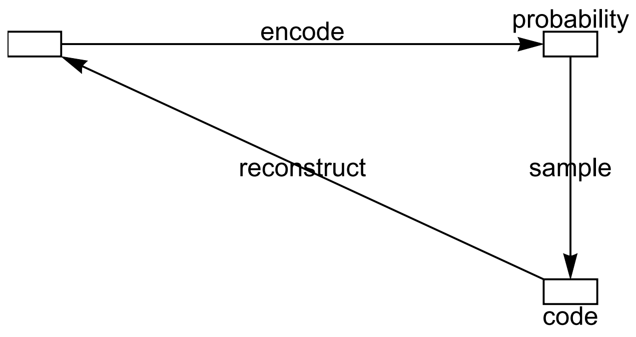


Figure 1: A folded Markov chain (FMC) in which an input vector $\boldsymbol{x}$ is encoded as a code index vector $\boldsymbol{y}$ that is drawn from a conditional probability $\Pr(\boldsymbol{y}|\boldsymbol{x})$, which is then decoded as a reconstruction vector $\boldsymbol{x}'$ drawn from the Bayes' inverse conditional probability $\Pr(\boldsymbol{x}'|\boldsymbol{y})$.

this measure, but one that is consistent with many previous results, and which also leads to many new results, is the mean Euclidean reconstruction error measure D, which is defined as [6]

$$D \equiv \int d\boldsymbol{x}\ \Pr(\boldsymbol{x}) \sum_{y_1=1}^{M} \sum_{y_2=1}^{M} \cdots \sum_{y_n=1}^{M} \Pr(\boldsymbol{y}|\boldsymbol{x}) \int d\boldsymbol{x}'\ \Pr(\boldsymbol{x}'|\boldsymbol{y})\, \|\boldsymbol{x}-\boldsymbol{x}'\|^2 \tag{1}$$

where $\boldsymbol{y} = (y_1, y_2, \cdots, y_n)$ $1 \le y_i \le M$ is assumed, $\Pr(\boldsymbol{x})\ \Pr(\boldsymbol{y}|\boldsymbol{x})\ \Pr(\boldsymbol{x}'|\boldsymbol{y})$ is the joint probability that the FMC has state $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{x}')$, $\|\boldsymbol{x}-\boldsymbol{x}'\|^2$ is the Euclidean reconstruction error, and $\int d\boldsymbol{x} \sum_{y_1=1}^{M} \sum_{y_2=1}^{M} \cdots \sum_{y_n=1}^{M} \int d\boldsymbol{x}' \,(\cdots)$ sums over all possible states of the FMC (weighted by the joint probability).

The Bayes' inverse probability $\Pr(\boldsymbol{x}'|\boldsymbol{y})$ may be integrated out of this expression for D to yield [6]

$$D = 2 \int d\boldsymbol{x}\ \Pr(\boldsymbol{x}) \sum_{y_1=1}^{M} \sum_{y_2=1}^{M} \cdots \sum_{y_n=1}^{M} \Pr(\boldsymbol{y}|\boldsymbol{x})\, \|\boldsymbol{x}-\boldsymbol{x}'(\boldsymbol{y})\|^2 \tag{2}$$

where the reconstruction vector $\boldsymbol{x}'(\boldsymbol{y})$ is defined as $\boldsymbol{x}'(\boldsymbol{y}) \equiv \int d\boldsymbol{x}\ \Pr(\boldsymbol{x}|\boldsymbol{y})\,\boldsymbol{x}$. Because of the quadratic form of the objective function, it turns out that $\boldsymbol{x}'(\boldsymbol{y})$ may be treated as a free parameter whose optimum value (i.e. the solution of $\frac{\partial D}{\partial \boldsymbol{x}'(\boldsymbol{y})} = 0$) is $\int d\boldsymbol{x}\ \Pr(\boldsymbol{x}|\boldsymbol{y})\,\boldsymbol{x}$, as required.

B. Noisy Data

The FMC approach can be generalised to the problem of encoding noisy or distorted data, with the intention of eventually recovering the undistorted data. This generalisation is based on the results reported in [2]. The input vector is $\boldsymbol{x}_0$, which is converted into the distorted input vector $\boldsymbol{x}$ by a distortion process $\Pr(\boldsymbol{x}|\boldsymbol{x}_0)$, which is then encoded as a code index vector $\boldsymbol{y}$, which is then subsequently decoded as a reconstruction $\boldsymbol{x}_0{}'$ of the original input vector. This is described by the directed graph $\boldsymbol{x}_0 \longrightarrow \boldsymbol{x} \longrightarrow \boldsymbol{y} \longrightarrow \boldsymbol{x}_0{}'$. The operations that occur are summarised in Figure 2.

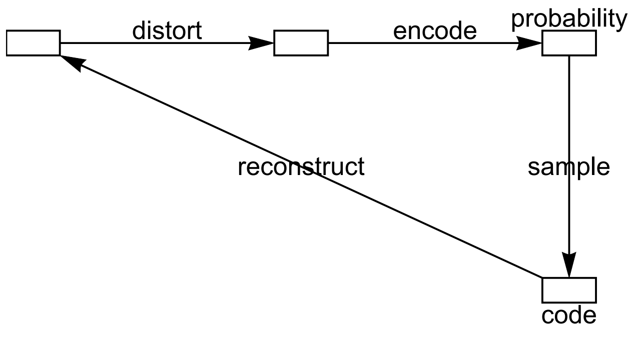


Figure 2: A folded Markov chain (FMC) in which an input vector $\boldsymbol{x}_0$ is first distorted into $\boldsymbol{x}$, which is then encoded as a code index vector $\boldsymbol{y}$ that is drawn from a conditional probability $\Pr(\boldsymbol{y}|\boldsymbol{x})$, which is then decoded as a reconstruction vector $\boldsymbol{x}_0{}'$ drawn from the Bayes' inverse conditional probability $\Pr(\boldsymbol{x}_0{}'|\boldsymbol{y})$.

The mean Euclidean reconstruction error measure D becomes (compare Equation 1)

$$D = \int d\boldsymbol{x}_0 \, \Pr(\boldsymbol{x}_0) \int d\boldsymbol{x} \, \Pr(\boldsymbol{x}|\boldsymbol{x}_0) \sum_{y_1=1}^{M} \sum_{y_2=1}^{M} \cdots \sum_{y_n=1}^{M} \Pr(\boldsymbol{y}|\boldsymbol{x}) \int d\boldsymbol{x}' \, \Pr(\boldsymbol{x}_0{}'|\boldsymbol{y}) \left\| \boldsymbol{x}_0 - \boldsymbol{x}_0{}' \right\|^2 \tag{3}$$

The Bayes' inverse probability $\Pr(\boldsymbol{x}_0{}'|\boldsymbol{y})$ may be integrated out of this expression for D to yield (compare Equation 2)

$$D = 2 \int d\boldsymbol{x}_0 \, \Pr(\boldsymbol{x}_0) \int d\boldsymbol{x} \, \Pr(\boldsymbol{x}|\boldsymbol{x}_0) \sum_{y_1=1}^{M} \sum_{y_2=1}^{M} \cdots \sum_{y_n=1}^{M} \Pr(\boldsymbol{y}|\boldsymbol{x}) \left\| \boldsymbol{x}_0 - \boldsymbol{x}_0{}'(\boldsymbol{y}) \right\|^2 \tag{4}$$

where the reconstruction vector $\boldsymbol{x}_0{}'(\boldsymbol{y})$ is defined as $\boldsymbol{x}_0{}'(\boldsymbol{y}) \equiv \int d\boldsymbol{x}_0 \, \Pr(\boldsymbol{x}_0 \, |\boldsymbol{y}) \, \boldsymbol{x}_0$, which may be treated as a free parameter.

Bayes' theorem $\Pr(\boldsymbol{x}_0) \, \Pr(\boldsymbol{x}|\boldsymbol{x}_0) = \Pr(\boldsymbol{x}) \, \Pr(\boldsymbol{x}_0|\boldsymbol{x})$ may be used to integrate out $\boldsymbol{x}_0$ to yield

$$D = 2 \int d\boldsymbol{x} \, \Pr(\boldsymbol{x}) \sum_{y_1=1}^{M} \sum_{y_2=1}^{M} \cdots \sum_{y_n=1}^{M} \Pr(\boldsymbol{y}|\boldsymbol{x}) \left\| \boldsymbol{x}_0(\boldsymbol{x}) - \boldsymbol{x}_0{}'(\boldsymbol{y}) \right\|^2 + \text{constant} \tag{5}$$

where $\boldsymbol{x}_0(\boldsymbol{x})$ is defined as $\boldsymbol{x}_0(\boldsymbol{x}) \equiv \int d\boldsymbol{x}_0 \, \Pr(\boldsymbol{x}_0|\boldsymbol{x}) \, \boldsymbol{x}_0$.

It is much more difficult to optimise this version of the objective function than the version in Equation 2, because the $\boldsymbol{x}_0(\boldsymbol{x})$ term is in general a non-linear function of $\boldsymbol{x}$. Worse still, the expression for $\boldsymbol{x}_0(\boldsymbol{x})$ involves $\Pr(\boldsymbol{x}_0|\boldsymbol{x})$, which depends on the unknown $\Pr(\boldsymbol{x}_0)$, so $x_0(\boldsymbol{x})$ cannot be computed analytically anyway. The situation looks irretrievable, but it turns out that some progress can be made by conceptually splitting $\boldsymbol{x}$ into "signal" and "noise" subspaces, as will be shown in Section II C.

C. Nuisance Degrees of Freedom

For convenience, split up the input space into (possibly non-orthogonal) subspaces as $(\boldsymbol{x}_0, \boldsymbol{x}_\perp)$, where all of the distortion is contained in $\boldsymbol{x}_\perp$, which requires that any distortion that lies *in* the $\boldsymbol{x}_0$ subspace is regarded as part of the undistorted input. The directed graph becomes $(\boldsymbol{x}_0, \boldsymbol{0}) \longrightarrow (\boldsymbol{x}_0, \boldsymbol{x}_\perp) \longrightarrow \boldsymbol{y} \longrightarrow \boldsymbol{x}_0{}'$ as shown in Figure 3.

The expression for D becomes (compare Equation 4).

$$D = 2 \int d\boldsymbol{x}_0 \, \Pr(\boldsymbol{x}_0) \int d\boldsymbol{x}_\perp \, \Pr(\boldsymbol{x}_\perp|\boldsymbol{x}_0) \sum_{y_1=1}^{M} \sum_{y_2=1}^{M} \cdots \sum_{y_n=1}^{M} \Pr(\boldsymbol{y}|\boldsymbol{x}_0, \boldsymbol{x}_\perp) \left\| \boldsymbol{x}_0 - \boldsymbol{x}_0{}'(\boldsymbol{y}) \right\|^2 \tag{6}$$

Consider the related optimisation problem in which an attempt to to reconstruct $(\boldsymbol{x}_0, \boldsymbol{x}_\perp)$ is made, as shown in Figure 4.

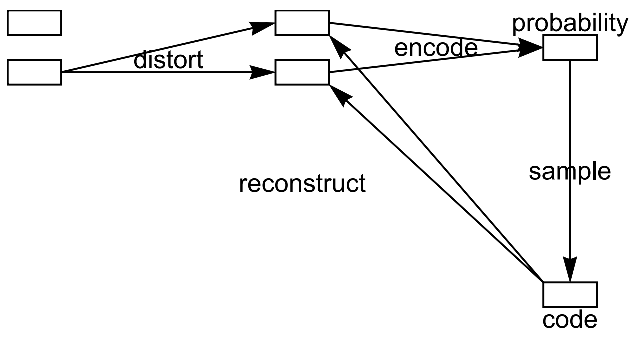


Figure 4: Modified version of Figure 3 in which the reconstruction link is switched from the original undistorted signal to the full signal+distortion.

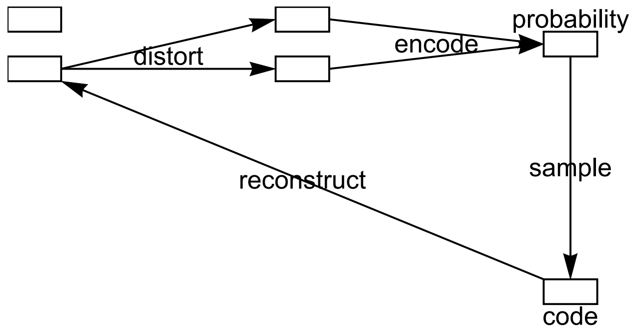


Figure 3: A folded Markov chain (FMC) in which an input vector $(\boldsymbol{x}_0, \mathbf{0})$ is first distorted into $(\boldsymbol{x}_0, \boldsymbol{x}_\perp)$, which is then encoded as a code index vector $\boldsymbol{y}$ that is drawn from a conditional probability $\Pr(\boldsymbol{y}|\boldsymbol{x}_0, \boldsymbol{x}_\perp)$, which is then decoded as a reconstruction vector $\boldsymbol{x}_0{}'$ drawn from the Bayes' inverse conditional probability $\Pr(\boldsymbol{x}_0{}'|\boldsymbol{y})$.

The corresponding objective function may be obtained by modifying Equation 6, where the cross-term arising from non-orthogonal $(\boldsymbol{x}_0, \boldsymbol{x}_\perp)$ is omitted.

$$D = 2\int d\boldsymbol{x}_0 \Pr(\boldsymbol{x}_0) \int d\boldsymbol{x}_\perp \Pr(\boldsymbol{x}_\perp|\boldsymbol{x}_0) \sum_{y_1=1}^{M}\sum_{y_2=1}^{M}\cdots\sum_{y_n=1}^{M} \Pr(\boldsymbol{y}|\boldsymbol{x}_0, \boldsymbol{x}_\perp) \left(||\boldsymbol{x}_0 - \boldsymbol{x}_0{}'(\boldsymbol{y})||^2 + ||\boldsymbol{x}_\perp - \boldsymbol{x}_\perp{}'(\boldsymbol{y})||^2\right) \quad (7)$$

Assume for now (to be justified below) that some of the links in Figure 4 are broken as shown in Figure 5.

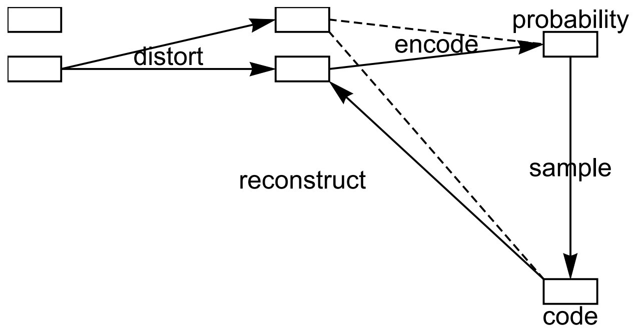


Figure 5: Modified version of Figure 4 in which the encoder (and reconstruction) links from (and to) the distortion subspace are deleted (as indicated by the dashed lines).

Because the distortion subspace is not involved in the computations in Figure 5, it may be redrawn as shown in Figure 6.

This is the same as Figure 3, except that the encoder now disregards (or is invariant with respect to) the nuisance degrees of freedom.

In order to break the links as shown in Figure 5 the following argument is required:

1. Assume that the encoder is independent of $\boldsymbol{x}_\perp$, so that $\Pr(\boldsymbol{y}|\boldsymbol{x}_0, \boldsymbol{x}_\perp) = \Pr(\boldsymbol{y}|\boldsymbol{x}_0)$.

2. The $||\boldsymbol{x}_\perp - \boldsymbol{x}_\perp{}'(\boldsymbol{y})||^2$ term in D needs to simplify to a constant.

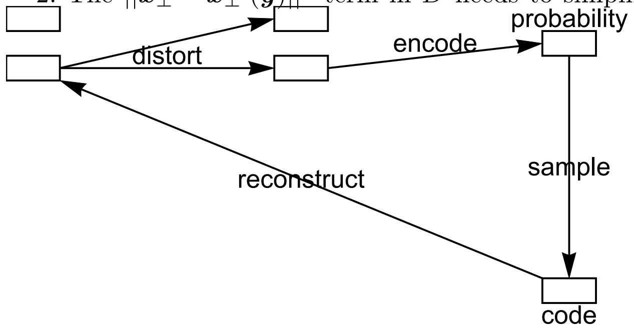


Figure 6: Equivalent version of Figure 5 in which the reconstruction link is moved to an equivalent position.

3. This requires that

$$\int d\boldsymbol{x}_0 \Pr(\boldsymbol{x}_0) \int d\boldsymbol{x}_\perp \Pr(\boldsymbol{x}_\perp|\boldsymbol{x}_0) \sum_{y_1=1}^{M}\sum_{y_2=1}^{M}\cdots\sum_{y_n=1}^{M} \Pr(\boldsymbol{y}|\boldsymbol{x}_0)\, ||\boldsymbol{x}_\perp - \boldsymbol{x}_\perp{}'(\boldsymbol{y})||^2 = \text{constant}.$$

4. To guarantee this constant, it is sufficient to have $\int d\boldsymbol{x}_\perp \Pr(\boldsymbol{x}_\perp|\boldsymbol{x}_0)\, ||\boldsymbol{x}_\perp - \boldsymbol{x}_\perp{}'(\boldsymbol{y})||^2$ = constant in-

dependent of $\boldsymbol{x}_0$ and $\boldsymbol{y}$.

5. To guarantee this constant independent of $\boldsymbol{x}_0$ and $\boldsymbol{y}$, it is sufficient to have $\Pr(\boldsymbol{x}_\perp|\boldsymbol{x}_0) = \Pr(\boldsymbol{x}_\perp)$.

6. Given that $\Pr(\boldsymbol{x}_\perp|\boldsymbol{x}_0) = \Pr(\boldsymbol{x}_\perp)$ and $\Pr(\boldsymbol{y}|\boldsymbol{x}_0,\boldsymbol{x}_\perp) = \Pr(\boldsymbol{y}|\boldsymbol{x}_0)$, then making the replacement $\boldsymbol{x}_\perp{}'(\boldsymbol{y}) \longrightarrow \langle\boldsymbol{x}_\perp\rangle$ in D will give an objective function with the same stationary points as D, because $\boldsymbol{x}_\perp{}'(\boldsymbol{y}) = \langle\boldsymbol{x}_\perp\rangle$ is the stationary point of D with respect to $\boldsymbol{x}_\perp{}'(\boldsymbol{y})$.

7. Given that $\Pr(\boldsymbol{x}_\perp|\boldsymbol{x}_0) = \Pr(\boldsymbol{x}_\perp)$ and $\boldsymbol{x}_\perp{}'(\boldsymbol{y}) = \langle\boldsymbol{x}_\perp\rangle$, the result $\int d\boldsymbol{x}_\perp \Pr(\boldsymbol{x}_\perp|\boldsymbol{x}_0)\,||\boldsymbol{x}_\perp - \boldsymbol{x}_\perp{}'(\boldsymbol{y})||^2 = \text{constant}$ independent of $\boldsymbol{x}_0$ and $\boldsymbol{y}$ follows automatically.

The assumptions may be summarised as

$$\begin{aligned}\Pr(\boldsymbol{y}|\boldsymbol{x}_0,\boldsymbol{x}_\perp) &= \Pr(\boldsymbol{y}|\boldsymbol{x}_0)\\ \Pr(\boldsymbol{x}_\perp|\boldsymbol{x}_0) &= \Pr(\boldsymbol{x}_\perp)\end{aligned} \tag{8}$$

which allow the objective function D (see Equation 7) to be replaced by the equivalent objective function

$$D = 2\int d\boldsymbol{x}_0 \Pr(\boldsymbol{x}_0)\sum_{y_1=1}^{M}\sum_{y_2=1}^{M}\cdots\sum_{y_n=1}^{M}\Pr(\boldsymbol{y}|\boldsymbol{x}_0)\,||\boldsymbol{x}_0 - \boldsymbol{x}_0{}'(\boldsymbol{y})||^2 + \text{constant} \tag{9}$$

This is the standard FMC objective function (compare Equation 2) for encoding and reconstructing the undistorted input, for which the directed graph is $\boldsymbol{x}_0 \longrightarrow \boldsymbol{y} \longrightarrow \boldsymbol{x}_0{}'$. Note that, under the stated assumptions, the simplification in Equation 9 occurs even if the two subspaces are not orthogonal to each other, the potential cross-term $\int d\boldsymbol{x}_\perp \Pr(\boldsymbol{x}_\perp|\boldsymbol{x}_0)\,(\boldsymbol{x}_0 - \boldsymbol{x}_0{}'(\boldsymbol{y})).(\boldsymbol{x}_\perp - \boldsymbol{x}_\perp{}'(\boldsymbol{y}))$ in Equation 7 is zero.

In summary, the encoder has access only to the signal + distortion $(\boldsymbol{x}_0,\boldsymbol{x}_\perp)$ (see Figure 4 and Equation 7), but the assumptions in Equation 8 force the encoder to disregard the distortion (see Figure 6 and Equation 9). In practice, it is not possible to satisfy these assumptions in general, because it is not known in advance how to extract orthogonal signal and distortion subspaces $(\boldsymbol{x}_0,\boldsymbol{x}_\perp)$ given examples of only the distorted signal. However, these assumptions may be encouraged to hold true by minimising D (as defined in Equation 7) under certain constraints, in which case Figure 6 and Equation 9 follow automatically from Figure 4 and Equation 7, respectively. These constraints are discussed in Section II D.

This type of encoder, in which the large degrees of freedom are preferentially encoded, can be used as the basis of a so-called "residual vector quantiser" [1], in which (quoting from [1]) "the quantiser has a sequence of encoding stages, where each stage encodes the residual (error) vector of the prior stage". Note that a residual vector quantiser is a special case of the type of multistage encoder discussed in [9].

D. Optimisation Constraints

Henceforth, only the scalar case will be considered, so the vector $\boldsymbol{y}$ is now replaced by the scalar y ($1 \le y \le M$). In order to implement a practical optimisation procedure for minimising D it is necessary to introduce a variety of assumptions and constraints.

Because $\Pr(y|\boldsymbol{x})$ is a probability it satisfies $\Pr(y|\boldsymbol{x}) \ge 0$ and $\sum_{y=1}^{M}\Pr(y|\boldsymbol{x}) = 1$, which is guaranteed if $\Pr(y|\boldsymbol{x})$ is written as

$$\Pr(y|\boldsymbol{x}) = \frac{Q(y|\boldsymbol{x})}{\sum_{y'=1}^{M} Q(y'|\boldsymbol{x})} \tag{10}$$

where $Q(y|\boldsymbol{x}) \ge 0$. This removes the need to explicitly impose the constraint $\sum_{y=1}^{M}\Pr(y|\boldsymbol{x})$ during optimisation. The $Q(y|\boldsymbol{x})$ are the unnormalised *likelihoods* of sampling code index y from the code book.

However, $Q(y|\boldsymbol{x})$ itself needs to be described by a finite number of parameters in order that the values that minimise D may be derived from a finite amount of training data. It can be shown that the optimal form of $\Pr(y|\boldsymbol{x})$ is piecewise linear in $\boldsymbol{x}$ [9], and that for training data that lie on smooth curved manifolds the form of this solution is well approximated by a piecewise linear $Q(y|\boldsymbol{x})$ of the form [10]

$$Q(y|\boldsymbol{x}) = \begin{cases}\boldsymbol{w}(y).\boldsymbol{x} - a(y) & \boldsymbol{w}(y).\boldsymbol{x} \ge a(y)\\ 0 & \boldsymbol{w}(y).\boldsymbol{x} \le a(y)\end{cases} \tag{11}$$

which is the same as the functional form used for the neural response in [11]. However, the precise functional form of $Q(y|\boldsymbol{x})$ needs to exhibit this behaviour only in the vicinity of the data manifold, so in particular it can be allowed to saturate (i.e. $Q(y|\boldsymbol{x}) \longrightarrow 1$) as $\boldsymbol{w}(y).\boldsymbol{x} \longrightarrow \infty$. A convenient functional form that achieves this is the sigmoid, which is defined as

$$Q(y|\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}(y).\boldsymbol{x} - b(y))} \tag{12}$$

This reduces the problem of minimising D to one of finding the optimal values of the $\boldsymbol{w}(y)$, $b(y)$ and $\boldsymbol{x}'(y)$. This may be done by using the gradient descent procedure described in [7].

If the input is an *undistorted* signal (i.e. $\boldsymbol{x} = (\boldsymbol{x}_0, \boldsymbol{0})$) which lies on a smooth curved manifold, then the sigmoids can cooperate in encoding this input as illustrated in Figure 7, where the sigmoid threshold planes $\boldsymbol{w}(y).\boldsymbol{x} + b(y) = 0$ are shown slicing pieces off the curved manifold [10].

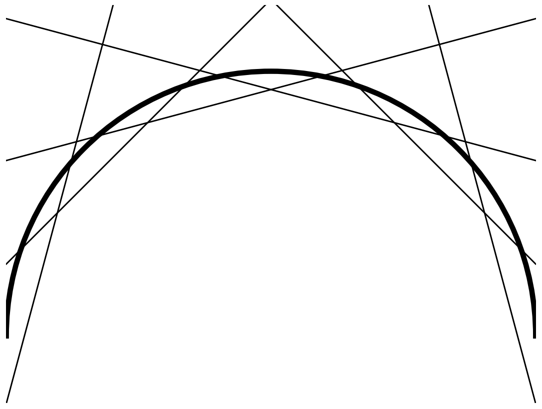


Figure 7: Illustration of how a number of sigmoids can cooperate to slice pieces off a signal manifold.

The additional constraints that are required in order to implement the behaviour described in Section II C will now be described. Thus the constraints must be such that the encoder disregards (or is invariant with respect to) the nuisance degrees of freedom $\boldsymbol{x}_\perp$ in the full input vector $(\boldsymbol{x}_0, \boldsymbol{x}_\perp)$. However, without knowing $\Pr(\boldsymbol{x}_0, \boldsymbol{x}_\perp)$ in advance (which would allow $\boldsymbol{x}_0(\boldsymbol{x})$ in Equation 5 to be calculated), it is not possible to give a general approach that works in all cases. At best, an empirical approach must be used.

A very simple and useful constraint is to impose a threshold constraint on the sigmoid function, which forces the value of the sigmoid to lie exactly halfway up its slope when the norm of its input vector is θ. This is achieved by choosing $b(y) = -\theta|\boldsymbol{w}(y)|$, so that

$$Q(y|\boldsymbol{x}) = \frac{1}{1 + \exp\left(-\left(\hat{\boldsymbol{w}}(y).\boldsymbol{x} - \theta\right) ||\boldsymbol{w}(y)||\right)} \tag{13}$$

where $||\boldsymbol{w}(y)|| \equiv \sqrt{\boldsymbol{w}(y).\boldsymbol{w}(y)}$ and $\hat{\boldsymbol{w}}(y) \equiv \frac{\boldsymbol{w}(y)}{||\boldsymbol{w}(y)||}$.

If the input is a *distorted* signal (i.e. $\boldsymbol{x} = (\boldsymbol{x}_0, \boldsymbol{x}_\perp)$) which lies on a "thickened" version of the smooth curved manifold of Figure 7 (the thickness represents the nuisance degrees of freedom), then the sigmoids can cooperate in encoding this input as illustrated in Figure 8, where the sigmoid threshold planes $\hat{\boldsymbol{w}}(y).\boldsymbol{x} = \theta$ are shown slicing pieces off the curved manifold in a way that disregards the nuisance degrees of freedom.

Note that in Figure 8 the representation of thickening is not complete, because it can actually occur in *any* direction orthogonal to the manifold, including directions

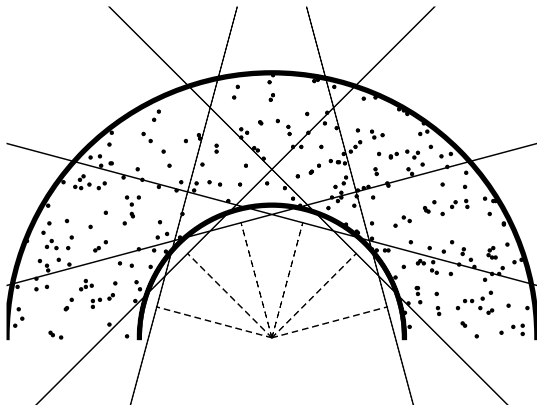


Figure 8: Illustration of how a number of sigmoids can cooperate to slice pieces off a signal manifold thickened by nuisance degrees of freedom.

orthogonal to the space in which the manifold is embedded; the radial direction in Figure 8 does not include this latter possibility.

In practice, for numerical efficiency and to encourage the optimisation procedure to locate the global minimum of D, it is useful to introduce two additional constraints. Firstly, because optimal solutions typically satisfy $\boldsymbol{x}'(y) \approx \boldsymbol{w}(y)$ up to a multiplicative constant, each reconstruction vector $\boldsymbol{x}'(y)$ can be forced to lie parallel to the corresponding weight vector $\boldsymbol{w}(y)$, so that $\boldsymbol{x}'(y) \propto \boldsymbol{w}(y)$; this constraint was also used in [12], but there it was a necessary part of the optimisation procedure, whereas here it merely encourages faster convergence. Secondly, the norm of the weight vectors $||\boldsymbol{w}(y)||$ can be constrained as $||\boldsymbol{w}(y)|| = w_0$, in order to avoid situations where they grow to rather large values which make $Q(y|\boldsymbol{x})$ (and hence $\Pr(y|\boldsymbol{x})$) depend very strongly on $\boldsymbol{x}$ in some regions. Both of these constraints speed up convergence to the global minimum of D, can finally be lifted in the vicinity of an optimal solution to obtain complete convergence.

E. Jammer Nulling

A number of examples of typical behaviours of $\Pr(y|\boldsymbol{x})$ are shown in Figure 9.

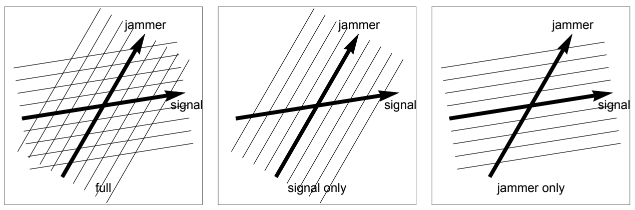


Figure 9: Examples of the response of $\Pr(y|\boldsymbol{x})$ to signal and jammer subspaces.

In Figure 9 the signal and jammer degrees of freedom generate a pair of non-orthogonal subspaces, whose axes are indicated in bold. The response contours of a variety of possible $\Pr(y|\boldsymbol{x})$ are shown. In the "full" case a pair of $\Pr(y|\boldsymbol{x})$ respond to the signal and jammer subspaces respectively. In the "signal" case a $\Pr(y|\boldsymbol{x})$ responds to only the signal subspace, and is thus invariant over the jammer subspace. In the "jammer" case the situation is the reverse of the "signal" case. This argument may readily be generalised to any number of $\Pr(y|\boldsymbol{x})$.

If it is assumed that the jammer is the "large" degree of freedom and the signal is the "small" degree of freedom, the signal and jammer subspaces may be separated by adjusting the threshold parameter θ so that in Figure 9 the "jammer" case is obtained, in which case the $\Pr(y|\boldsymbol{x})$ for $y = 1, 2, \cdots, M$ will all become invariant over the signal subspace. The jammer subspace is then spanned by the set of gradient vectors $\boldsymbol{\nabla} \Pr(y|\boldsymbol{x})$ for $y = 1, 2, \cdots, M$, which can thus be used to construct a projection operator $\boldsymbol{J}$ onto the jammer subspace, and a projection operator $\boldsymbol{1} - \boldsymbol{J}$ onto the signal subspace. This definition of the projection operator may also be used in cases where the jammer and signal subspaces are curved, so that the directions of their axes are functions of $\boldsymbol{x}$, and all of the straight lines in Figure 9 are replaced by curves defining a curvilinear coordinate system and its coordinate surfaces. Note that curved subspaces are the norm rather than the exception.

III. JAMMER NULLING SIMULATIONS

The optimisation of the encoder may be done by minimising D using gradient descent [7], using the sigmoid function in Equation 13 to constrain the optimisation so that it encodes only the jammer subspace.

In these simulations the input vector $\boldsymbol{x}$ is 100-dimensional so that $\boldsymbol{x} = (x_1, x_2, \cdots, x_{100})$, and each vector in the training set is independently generated as a superposition of a pair of response functions

$$x_i = a_s \frac{\sin\left(\frac{i-i_s}{\sigma}\right)}{\frac{i-i_s}{\sigma}} + a_j \frac{\sin\left(\frac{i-i_j}{\sigma}\right)}{\frac{i-i_j}{\sigma}} \tag{14}$$

where a_s is the signal amplitude that is uniformly distributed in the interval $[-\sqrt{10^{-3}}, \sqrt{10^{-3}}]$ (this correponds to a signal level of -30dB), a_j is the jammer amplitude that is uniformly distributed in the interval $[-1, 1]$ (this correponds to a jammer level of 0dB), i_s is the signal location that is chosen to be 50, i_j is the jammer location that is uniformly distributed in the interval $[38 - \Delta, 38 + \Delta]$ ($\Delta = 0, 2, 4$ is used in the simulations), and σ is the width of the response function that is chosen to be 2. The peak and the first zero of the sinc function are separated by $\pi\sigma$, which defines the resolution cell size. The mean jammer position and the signal position satisfy $i_s - < i_j >= 12$, which corresponds to a separation of $\frac{12}{\pi\sigma} \approx 2$ resolution cells. Random noise uniformly distributed in the interval $[-\sqrt{10^{-5}}, \sqrt{10^{-5}}]$ (this correponds to a noise level of -50dB) is also added to each component of the training vector.

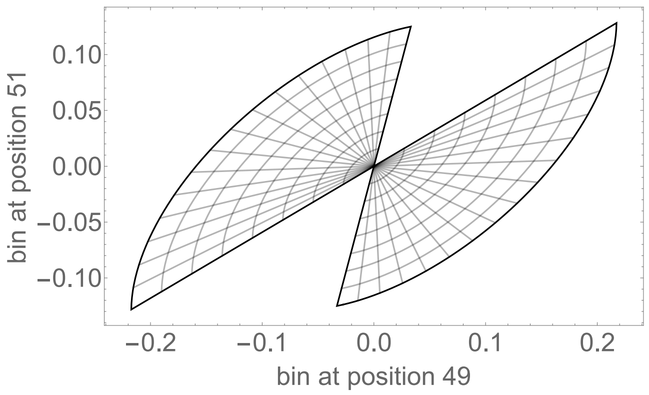


Figure 10: A two-dimensional projection of the curved manifold generated by the jammer when $\Delta = 2$.

In Figure 10 the 2-dimensional manifold generated by varying the jammer position over the interval $[38-\Delta, 38+\Delta]$ (for $\Delta = 2$), and varying the jammer amplitude over the interval $[-1, 1]$, is shown. Because the input vector $\boldsymbol{x}$ is 100-dimensional, only a low-dimensional projection can be visualised, and the 2-dimensional vector (x_{49}, x_{51}) is displayed here. The curvilinear grid traces out the coordinate surfaces of jammer position i_j and jammer amplitude a_j, and the whole diagram shows how this grid is embedded in (x_{49}, x_{51})-space. Note that the a_j dimension behaves as a "radial" coordinate (straight lines), whereas the i_j dimension behaves as an "angular" coordinate (curved lines).

In Figure 11 an encoder is trained on three different jammer scenarios $\Delta = 0, 2, 4$. After training the encoder is tested for how well it can be used to null a pure jammer (i.e. with no signal or noise added), where the degree of nulling is defined as the ratio of the squared lengths of the nulled input vector and the original input vector. This is a good test of the ability of the encoder to simultaneously learn the profile of the jammer and the shape of the jammer manifold which is generated by sweeping this profile over the interval $[38 - \Delta, 38 + \Delta]$. When $\Delta = 0$ there is a sharp minimum at the jammer location $i_j = 38$, as expected. When $\Delta = 2$ the minimum becomes spread over the jammer locations $i_j \in [36, 40]$, and when $\Delta = 4$ the minimum becomes spread even more broadly over the jammer locations $i_j \in [34, 42]$. All of these results are as expected.

In Figure 12 typical examples of an input vector together with how it appears after jammer nulling are shown for each of the jammer scenarios considered in Figure 11. In every case the signal is clearly revealed at its correct location after nulling the jammer.

In all of these training scenarios, one could envisage further constraining some of the properties of the encoder, in order to introduce prior knowledge of the form of the jammer and/or signal subspaces, and to

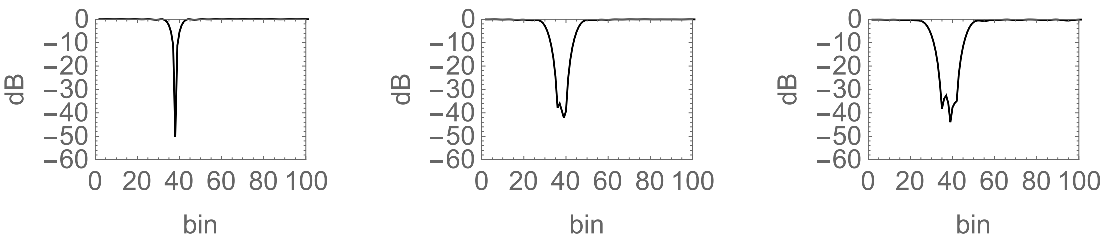


Figure 11: Plot of degree of nulling against nominal jammer location, for jammer locations that are spread over the intervals $[38, 38]$ using $M = 2$, $[36, 40]$ using $M = 4$, and $[34, 42]$ using $M = 6$.

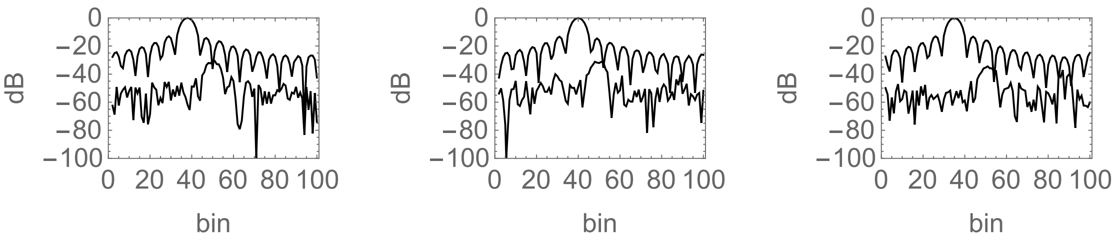


Figure 12: Plot of a typical input vector before and after jammer nulling for each of the scenarios in Figure 11.

thereby reduce the computational complexity of the jammer nulling. For instance, the signal subspace could be predefined, as in conventional algorithms which hold constant the response in a predefined "look direction". Similarly, the jammer subspace could be built out of prefined subspaces which are optimised so as to maximally null the jammer(s), as in conventional algorithms in which a number of jammer "templates" are used to remove the jammer(s). In general, by choosing appropriate additional constraints, the SVQ approach to jammer nulling can be made backwardly compatible with conventional approaches.

IV. CONCLUSIONS

The theory of stochastic vector quantisers (SVQ) [7] has been extended to allow the quantiser to develop invariances, so that only "large" degrees of freedom in the input vector are represented in the code. This has been applied to the problem of encoding data vectors which are a superposition of a "large" jammer and a "small" signal, so that only the jammer is represented in the code. This allows the jammer to be subtracted from the total input vector (i.e. the jammer is nulled), leaving a residual that contains only the underlying signal. Several numerical simulations have shown how that idea works in practice, even when the jammer location is uncertain so that the jammer subspace is curved.

The main advantage of this approach to jammer nulling is that little prior knowledge of the jammer is assumed, because these properties are automatically discovered by the SVQ as it is trained on examples of input vectors. Provided that the signal is much weaker than the jammer, the SVQ acquires an internal representation of the jammer and signal manifolds, in which its code is invariant with respect to the signal. In a sense, the SVQ regards the "large" jammer as the normal type of input that it expects to receive, whereas it regards the "small" signal as an anomaly.

[1] C F Barnes, S A Rizvi, and N M Nasrabadi, *Advances in residual vector quantisation: a review*, IEEE Transactions on Image Processing **5** (1996), no. 2, 226–262.

[2] Y Ephraim and R M Gray, *A unified approach for encoding clean and noisy sources by means of waveform and autoregressive model vector quantisation*, IEEE Transactions on Information Theory **34** (1988), no. 4, 826–834.

[3] A Gersho and R M Gray, *Vector quantisation and signal compression*, Kluwer, Norwell, 1992.

[4] R M Gray, *Vector quantisation*, IEEE Transactions on Acoustics, Speech, and Signal Processing Magazine **1** (1984), no. 2, 4–29.

[5] Y Linde, A Buzo, and R M Gray, *An algorithm for vector quantiser design*, IEEE Transactions on Communications **28** (1980), no. 1, 84–95.

[6] S P Luttrell, *A Bayesian analysis of self-organising maps*, Neural Computation **6** (1994), no. 5, 767–794.

[7] ———, *Mathematics of neural networks: models, algorithms and applications*, ch. A theory of self-organising neural networks, pp. 240–244, Kluwer, Boston, 1997.

[8] ———, *Self-organisation of multiple winner-take-all neural networks*, Connection Science **9** (1997), no. 1, 11–30.

[9] ———, *An adaptive network for encoding data using piecewise linear functions*, Proceedings of international confererence on artificial neural networks (Edinburgh), IEE, 1999, pp. 198–203.

[10] ———, *Combining artificial neural nets: Ensemble and modular multi-net systems*, Perspectives in Neural Computing, ch. Self-organised modular neural networks for encoding data, pp. 235–263, Springer-Verlag, London, 1999.

[11] C J S Webber, *Self-organisation of transformation-invariant detectors for constituents of perceptual patterns*, Network: Computation in Neural Systems **5** (1994), no. 4, 471–496.

[12] ———, *Emergent componential coding of a handwritten image database by neural self-organisation*, Network: Computation in Neural Systems **9** (1998), no. 4, 433–447.

Using Self-Organising Mappings to Learn the Structure of Data Manifolds *

S P Luttrell
Room EX21, QinetiQ, Malvern Technology Centre

In this paper it is shown how to map a data manifold into a simpler form by progressively discarding small degrees of freedom. This is the key to self-organising data fusion, where the raw data is embedded in a very high-dimensional space (e.g. the pixel values of one or more images), and the requirement is to isolate the important degrees of freedom which lie on a low-dimensional manifold. A useful advantage of the approach used in this paper is that the computations are arranged as a feed-forward processing chain, where all the details of the processing in each stage of the chain are learnt by self-organisation. This approach is demonstrated using hierarchically correlated data, which causes the processing chain to split the data into separate processing channels, and then to progressively merge these channels wherever they are correlated with each other. This is the key to self-organising data fusion.

I. INTRODUCTION

The aim of this paper is to illustrate an approach that maps raw data into a representation that reveals its internal structure. The raw data is a high-dimensional vector of sample values output by a sensor such as the samples of a time series or the pixel values of an image, and the representation is typically a lower dimensional vector that retains some or all of the information content of the raw data. There are many ways of achieving this type of data reduction, and this paper will focus on methods that learn from examples of the raw data alone.

A key approach to data reduction is the self-organising map (SOM) [2]. There are many variants of the SOM approach which may be used to map raw data into a lower dimensional space that retains some or all of its information content. In order to increase the variety of mappings SOMs can learn some of these variants use quite sophisticated learning algorithms. For instance, the topology of a SOM can be learnt by the neural gas approach [9], or the topology of the network connecting several SOMs can be learnt by the growing hierarchical self-organising map (GHSOM) approach [1].

The approach used in this paper aims to achieve a similar type of result to the GHSOM approach. GHSOM is a top-down coarse-to-fine approach to optimising a tree structured network of SOMs, whereas in this paper a bottom-up fine-to-coarse approach will be used that learns a tree structure where appropriate. The choice of a fine-to-coarse rather than coarse-to-fine approach is made in order to obtain networks that can be readily applied to data fusion problems, where the goal is to progressively discard noise (and irrelevant degrees of freedom) as the data passes along the processing chain, thus gradually reducing its dimensionality to eventually obtain a low-dimensional representation of the original raw data.

The basis for the approach used in this paper is a Bayesian theory of SOMs [4] in which a SOM is modelled as an encoder/decoder pair, where the decoder is the Bayes inverse of the encoder. In this approach the encoder is modelled as a conditional probability over all possible codes given the input, and the code that is actually used is a *single* sample drawn from this conditional probability (i.e. a winner-take-all code). When the conditional probability is optimised to minimise the average Euclidean distortion between the original input and its reconstruction this leads to a network that has properties very similar to a Kohonen SOM.

The basic approach [4] needs to be extended in two separate ways [8]. Firstly, to encourage the self-organisation of a processing chain leading from raw data to a higher level representation, the single encoder/decoder is extended to become a Markov chain of connected encoders, where each encoder feeds its output into the next encoder in the chain. Secondly, to encourage the self-organisation of each encoder into a number of separate smaller encoders and thus to learn tree-structured networks where appropriate, each encoder is generalised to use codes that make simultaneous use of *several* samples from the conditional probability rather than only a *single* winner-take-all sample.

The goal of the approach used in this paper is similar to that of the multiple cause vector quantisation approach [10], because the common aim is to split data into its separate components (or causes). However, the approach used in this paper aims to minimise the amount of manual intervention in the training of the network, and thus allow the structure of the data to determine the structure of the network. This is made possible by using codes that consist of *several* samples from a *single* conditional probability, which allows each encoder to decide for itself how to split into a number of separate smaller encoders. Also the approach used in this paper does *not* make explicit use of a generative model of the data, because the aim is only to map raw data into a representation that clarifies its internal structure (i.e. build a recognition model), for which a generative model may be sufficient

*Typeset in LaTeX on May 9, 2019.
This is archived at http://arxiv.org/abs/cs/0406017, 8 Jun 2004. This paper was submitted to J. Machine Learning Research on 9 December 2003. Paper No. 03-592. It was not accepted for publication, but it underpins several subsequently published papers. © Copyright QinetiQ Ltd 2003

but is actually not necessary.

This paper is organised as follows. In Section II the structure of data is represented as smooth curved manifolds, and encoders are represented as hyperplanes that slice through these manifolds. In Section III the theory of a single encoder is developed by extending a Bayesian theory of SOMs [4] from winner-take-all encoders to multiple output encoders, and this theory is further extended to Markov chains of connected encoders [8]. In Section IV these results are used to train a network on some hierarchically correlated data to demonstrate the self-organisation of a tree-structured network for processing the data.

II. DATA MANIFOLDS

In order to represent the structure of data a flexible framework needs to be used. In this paper an approach will be used in which the structure of the data manifold is of primary importance, and the aim is to split apart the manifold in such a way as to reveal how its overall structure is composed. This approach must take account of the relative amplitude of the various contributions, so that a high resolution representation would include even the smallest amplitude contributions to the manifold, and a low resolution representation would retain only the largest amplitude contributions. More generally, it would be useful to construct a sequence of representations, each with a lower resolution than the previous one in the sequence. This could be achieved by progressively discarding the smallest degree of freedom to gradually lower the resolution of the representation. In effect, the representation will become increasingly abstract as it becomes more and more invariant to the fine details of the original data manifold.

In Section II A the basic notation used to describe manifolds is presented, and in Section II B the process of splitting a manifold into its component pieces and then reassembling these to form an approximation to the manifold is described.

A. Representation of Data Manifolds

Assume that the raw data vector $\boldsymbol{x}$ lies on a smooth manifold $\boldsymbol{x}(\boldsymbol{u})$, parameterised by $\boldsymbol{u}$ which is a vector of co-ordinates in the manifold. Usually, though not invariably, $\boldsymbol{x}$ is a high-dimensional vector (e.g. an image comprising an array of pixel values) and $\boldsymbol{u}$ is a low-dimensional vector (e.g. a vector of object positions), in which case the space in which $\boldsymbol{x}$ lives is a high-dimensional embedding space for a low-dimensional manifold. Typically, $\boldsymbol{u}$ represents the underlying degrees of freedom (e.g. object co-ordinates), whereas $\boldsymbol{x}$ represents the observed degrees of freedom (e.g. sensor measurements). Usually, $\boldsymbol{u}$ will contain some noise degrees of freedom, but these can be handled in exactly the same way as other degrees of freedom by splitting $\boldsymbol{u}$ as $\boldsymbol{u} = (\boldsymbol{u}_s, \boldsymbol{u}_n)$ where $\boldsymbol{u}_s$ is signal and $\boldsymbol{u}_n$ is noise. The probability density function (PDF) $\Pr(\boldsymbol{u})$ describes how the manifold is populated and $\Pr(\boldsymbol{x})$ (where $\Pr(\boldsymbol{x}) = \int d\boldsymbol{u} \, \Pr(\boldsymbol{u}) \, \delta(\boldsymbol{x} - \boldsymbol{x}(\boldsymbol{u}))$) describes how the embedding space is populated.

In general, $\boldsymbol{x}(\boldsymbol{u})$ is a non-linear function of $\boldsymbol{u}$ so the manifold is curved, and thus occupies more linear dimensions of the embedding space than would be the case if the manifold were not curved. It is commonplace for a 1-dimensional manifold (i.e. $\boldsymbol{u}$ is a scalar) to be curved so as to occupy *all* of the linear dimensions of the embedding space (e.g. the manifold of images generated by moving an object along a 1-dimensional line of positions).

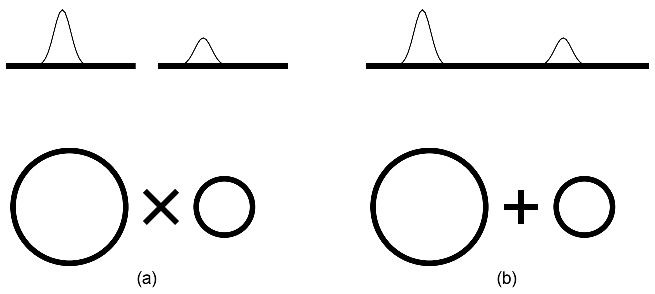


Figure 1: Examples of manifolds generated by images of a pair of objects. In each of the two diagrams the upper half shows the sensor data, and the lower half shows the low-dimensional manifold *topology* assuming that the sensor data have circular wraparound. The high-dimensional manifold *geometry* has a number of dimensions equal to the number of pixels in the corresponding sensor. (a) Tensor product of manifolds: 2-torus topology. This is generated by observing each object using a separate sensor. (b) Superposition (or mixture) of manifolds: This is generated by observing both objects using the same sensor, so that there is the possibility of overlap (possibly with obscuration) of the sensor data from the two objects. In the limit where the objects overlap infrequently case (b) closely approximates case (a).

If $\boldsymbol{x}(\boldsymbol{u})$ can be written as $\boldsymbol{x}(\boldsymbol{u}) = (\boldsymbol{x}_1(\boldsymbol{u}_1), \boldsymbol{x}_2(\boldsymbol{u}_2))$, where $\boldsymbol{x}_1(\boldsymbol{u}_1)$ and $\boldsymbol{x}_2(\boldsymbol{u}_2)$ are independently parameterised manifolds living in separate subspaces of the embedding space (where $\dim \boldsymbol{x} = \dim \boldsymbol{x}_1 + \dim \boldsymbol{x}_2$ and $\dim \boldsymbol{u} = \dim \boldsymbol{u}_1 + \dim \boldsymbol{u}_2$), then $\boldsymbol{x}(\boldsymbol{u})$ describes a tensor product of manifolds as shown in Figure 1a. This type of manifold arises when the underlying degrees of freedom are measured by separate sensors. This parameterisation can readily be generalised to $\boldsymbol{x}(\boldsymbol{u}) = (\boldsymbol{x}_1(\boldsymbol{u}_1), \boldsymbol{x}_2(\boldsymbol{u}_2), \cdots, \boldsymbol{x}_k(\boldsymbol{u}_k))$ for $k > 2$. For independently populated manifolds $\Pr(\boldsymbol{u})$ factorises as $\Pr(\boldsymbol{u}) = \Pr(\boldsymbol{u}_1) \Pr(\boldsymbol{u}_2)$, and if $\boldsymbol{x}(\boldsymbol{u}) = (\boldsymbol{x}_1(\boldsymbol{u}_1), \boldsymbol{x}_2(\boldsymbol{u}_2))$ then $\Pr(\boldsymbol{x}) = \Pr(\boldsymbol{x}_1) \Pr(\boldsymbol{x}_2)$ where $\Pr(\boldsymbol{x}_i) = \int d\boldsymbol{u}_i \, \Pr(\boldsymbol{u}_i) \, \delta(\boldsymbol{x}_i - \boldsymbol{x}_i(\boldsymbol{u}_i))$ for $i = 1, 2$. For manifolds that are populated in a correlated way (i.e. $\Pr(\boldsymbol{u}) \neq \Pr(\boldsymbol{u}_1) \Pr(\boldsymbol{u}_2)$) no such simple result holds.

If $\boldsymbol{x}(\boldsymbol{u})$ can be written as $\boldsymbol{x}(\boldsymbol{u}) = \boldsymbol{x}_1(\boldsymbol{u}_1) + \boldsymbol{x}_2(\boldsymbol{u}_2)$

(where $\dim \boldsymbol{x} = \dim \boldsymbol{x}_1 = \dim \boldsymbol{x}_2$), then $\boldsymbol{x}(\boldsymbol{u})$ describes a superposition (or mixture) of manifolds as shown in Figure 1b. This type of manifold arises when the underlying degrees of freedom are simultaneously measured by the same sensor. If there is little or no overlap between $\boldsymbol{x}_1(\boldsymbol{u}_1)$ and $\boldsymbol{x}_2(\boldsymbol{u}_2)$ then this is approximately equivalent to the case $\boldsymbol{x}(\boldsymbol{u}) = (\boldsymbol{x}_1(\boldsymbol{u}_1), \boldsymbol{x}_2(\boldsymbol{u}_2))$ (where $\dim \boldsymbol{x} = \dim \boldsymbol{x}_1 + \dim \boldsymbol{x}_2$ and $\dim \boldsymbol{u} = \dim \boldsymbol{u}_1 + \dim \boldsymbol{u}_2$). On the other hand, where there is a significant amount of overlap so that $\boldsymbol{x}_1(\boldsymbol{u}_1).\boldsymbol{x}_2(\boldsymbol{u}_2) > 0$, there is no such correspondence. Assuming $\Pr(\boldsymbol{u}) = \Pr(\boldsymbol{u}_1)\Pr(\boldsymbol{u}_2)$ then $\Pr(\boldsymbol{x})$ is given by $\Pr(\boldsymbol{x}) = \int d\boldsymbol{u}_1\, d\boldsymbol{u}_2 \Pr(\boldsymbol{u}_1)\Pr(\boldsymbol{u}_2)\,\delta(\boldsymbol{x} - \boldsymbol{x}_1(\boldsymbol{u}_1) - \boldsymbol{x}_2(\boldsymbol{u}_2))$.

More generally, $\boldsymbol{x}(\boldsymbol{u})$ can be written as $\boldsymbol{x}(\boldsymbol{u}) = \boldsymbol{x}(\boldsymbol{u}_1, \boldsymbol{u}_2)$ where $\boldsymbol{x}(\boldsymbol{u}_1, \boldsymbol{u}_2)$ has no special dependence on $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$. Although the manifolds are independently parameterised by $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$, when they are mapped to $\boldsymbol{x}$ their tensor product structure is disguised by the mapping function $\boldsymbol{x}(\boldsymbol{u}_1, \boldsymbol{u}_2)$ which is usually not invertible. The superposition of manifolds $\boldsymbol{x}(\boldsymbol{u}) = \boldsymbol{x}_1(\boldsymbol{u}_1) + \boldsymbol{x}_2(\boldsymbol{u}_2)$ is a special case of this effect.

B. Mapping of Data Manifolds

Given examples of the raw data $\boldsymbol{x}$ how can an approximation to the mapping function $\boldsymbol{x}(\boldsymbol{u})$ be constructed? The detailed approach will be described in Section III, but the basic geometric ideas will be described here. The basic idea is to cut the manifold into pieces whilst retaining only a limited amount of information about each piece, and then to reassemble these pieces to reconstruct an approximation to the manifold. This process is imperfect because it is disrupted by discarding some of the information about each piece, so the reconstructed manifold is not a perfect copy of the original manifold. This loss of information is critical to the success of this process, because if perfect information were retained then there would be no need to discover a clever way of cutting the manifold into pieces, and thus no possibility of discovering the structure of the manifold (e.g. whether it is a simple tensor product). The information that is preserved depends on exactly how the curved manifold is mapped to a new representation (see Section III for details).

Figure 2a shows an example of how a convex 1-dimensional manifold can be cut into overlapping pieces by a set of lines, and Figure 2b shows the generalisation to the 2-dimensional case. This process is considerably simplified if the manifold is convex because then the hyperplane slices off a localised piece of the manifold, as required.

Figure 3 shows an example of how the convexity assumption can break down. The full embedding space contains the vector formed from an array of samples of a 1-dimensional object function, but only three dimensions of the full embedding space are shown in Figure 3. Several scenarios are shown ranging from a narrow

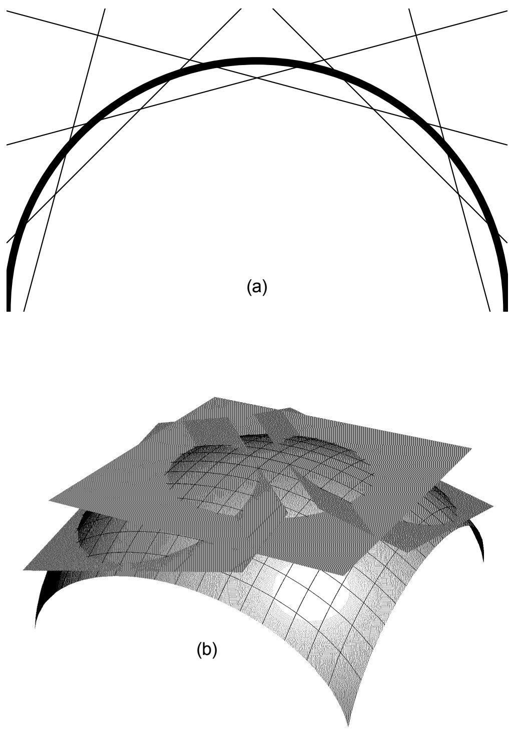


Figure 2: Using hyperplanes to slice pieces off convex curved manifolds. Slicing a curved manifold into pieces prepares it for mapping to another representation. (a) 1-dimensional manifold with arcs being sliced off by chords. (b) 2-dimensional manifold with caps being sliced off by planes (only a few of these are shown in order to keep the diagram simple).

object (i.e. undersampled) to a broad object (i.e. oversampled). Oversampling leads to a smooth convex manifold, whereas undersampling leads to a concave manifold with cusps. Typically, convex manifolds occur in signal and image processing where the raw data are sampled at a high enough rate, and non-convex manifolds occur when the raw data has already been processed into a low-dimensional form, such as when some underlying degrees of freedom (or features) have already been extracted from the raw data.

Figure 4 shows the results obtained by for a circular manifold using the stochastic vector quantiser (SVQ) approach of Section III. The results correspond to Figure 2a, except that now the slicing is done softly in order to preserve additional information about the manifold, and to ensure that the reconstructed manifold does not show artefacts when the slices are reassembled.

Figure 5 shows an example of how a convex $(1+\epsilon)$-dimensional manifold (a small length extracted from a cylindrical surface) can be cut into overlapping pieces by a set of planes. The 1 in $(1+\epsilon)$ is a large degree of freedom (arc length around the cylinder)), whereas the ϵ in $(1+\epsilon)$ is a small degree of freedom (length along the cylinder) because it has a small amplitude compared to

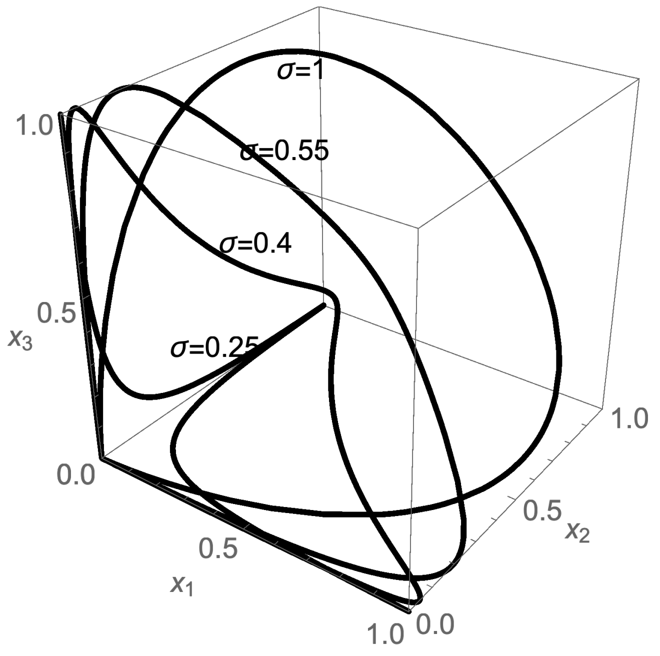


Figure 3: Manifolds generated by a 1-dimensional object. The data vector is $\boldsymbol{x} = (\cdots, x_{-2}, x_{-1}, x_0, x_1, x_2, \cdots)$ where $x_i = \exp\left(-\frac{(i-a)^2}{2\sigma^2}\right)$, σ is the width of the object function, a ($-\infty < a < \infty$) is the position of the object, and i ($i = 0, \pm 1, \pm 2, \cdots$) is the location of the points where the object amplitude is sampled. The manifolds shown are 3-dimensional embeddings (x_1, x_2, x_3) of the 1-dimensional curved manifolds generated as a varies for a variety of object widths σ. For $\sigma = 0.25$ (i.e. a narrow object function) the manifold is concave with cusps, as σ is increased the concavity and the cusps become less pronounced until the manifold crosses the border between being concave and being convex, and for $\sigma = 1$ (i.e. a wide object function) the manifold is smoothly convex. Concave manifolds with cusps are *not* well suited to being sliced apart by hyperplanes whereas smoothly convex manifolds *are* well suited, and this type of convex manifold is typical of high-dimensional data which also has a high resolution so that each object covers several sample points.

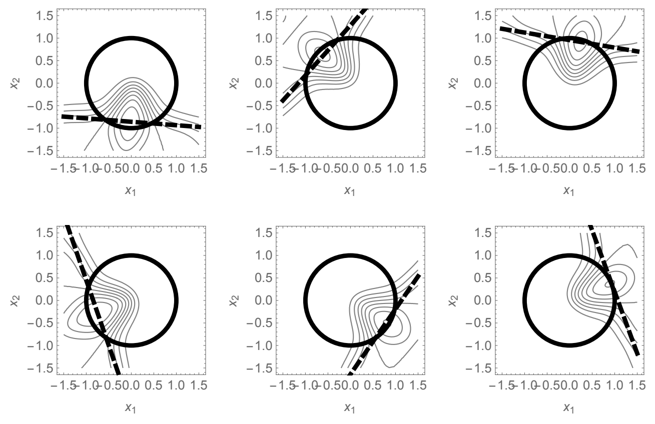


Figure 4: Using a stochastic vector quantiser (SVQ) to map a curved manifold to a new representation (see Section III for details). The manifold is the unit circle which is softly sliced up by the SVQ posterior probabilities, which are defined as the (normalised) outputs of a set of sigmoid functions, which in turn depend on a set of weight vectors and biases. For each sigmoid function a dashed line is drawn to show where its (unnormalised) output is $\frac{1}{2}$, although here it is the curved contours of the posterior probabilities (rather than the dashed lines) that are actually used to slice up the manifold.

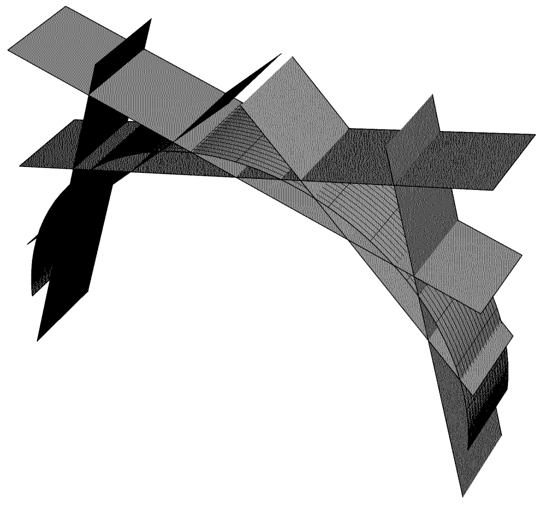


Figure 5: Using hyperplanes to slice pieces off a curved manifold. The manifold is 2-dimensional with a large and a small degree of freedom. Each hyperplane slices through the manifold in such a way that it cuts off a piece of the manifold that has a *limited* range of values of the large degree of freedom but *all* possible values of the small degree of freedom. This is the basic means by which a manifold can be mapped to a new representation.

the large degree of freedom. Because of the orientation of the planes they are insensitive to the small degree of freedom, so the reconstructed manifold is 1-dimensional manifold (i.e. the ϵ component has been discarded). The orientation of the planes may be used in various ways to control their sensitivity to the manifold, and some quite sophisticated examples of this will be discussed in Section IV. These results generalise to soft slicing as used in Figure 4.

III. LEARNING A MANIFOLD

In order to learn how to represent the structure of a data manifold a flexible framework needs to be used. In this paper an approach will be used in which the manifold is mapped to a lower resolution representation in such a way that a good approximation to the original manifold can be reconstructed. Key requirements are that these mappings can be cascaded to form sequences of representations of progressively lower resolution having more and more invariance with respect to details in the original manifold, and that the mappings can learn to represent tensor products of manifolds so that the representation of the manifold can split into separate channels. To achieve this it is sufficient to use a variant [8] of the standard vector quantiser [3] to gradually compress the data.

In Section III A the theory of stochastic vector quantisers (SVQ) is presented, and in Section III B it is extended to chains of linked SVQs.

A. Stochastic Vector Quantiser

As was discussed in Section II a procedure is needed for cutting a manifold into pieces and then reassembling these pieces to reconstruct the manifold. It turns out that all of the required properties emerge automatically from vector quantisers (VQ), and their generalisation to stochastic vector quantisers (SVQ).

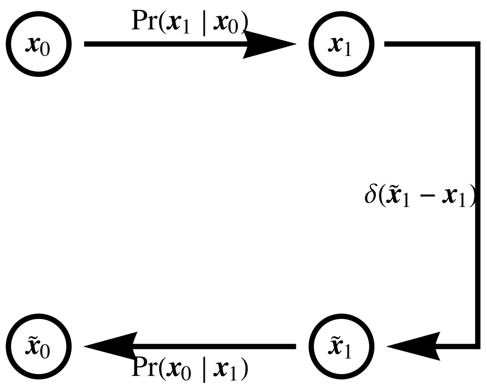


Figure 6: A matched encoder/decoder pair represented as a folded Markov chain (FMC) $\boldsymbol{x}_0 \longrightarrow \boldsymbol{x}_1 \longrightarrow \boldsymbol{x}_1 \longrightarrow \tilde{\boldsymbol{x}}_0$. The input $\boldsymbol{x}_0$ is encoded as $\boldsymbol{x}_1$ which is then passed along a distortionless communication channel to become $\boldsymbol{x}_1$ which is then decoded as $\tilde{\boldsymbol{x}}_0$. The encoder is modelled using the conditional probability $\Pr(\boldsymbol{x}_1|\boldsymbol{x}_0)$ to allow for the possibility that the encoder is stochastic, and the corresponding decoder is modelled using the Bayes inverse conditional probability $\Pr(\boldsymbol{x}_0|\boldsymbol{x}_1)$. The distortionless communication channel is modelled using the delta function $\delta(\tilde{\boldsymbol{x}}_1 - \boldsymbol{x}_1)$.

Figure 6 shows a folded Markov chain (FMC) as described in [4]. An FMC encodes its input $\boldsymbol{x}_0$ (i.e. cuts the input manifold into pieces using the conditional PDF $\Pr(\boldsymbol{x}_1|\boldsymbol{x}_0)$) and then reconstructs an approximation to its input $\tilde{\boldsymbol{x}}_0$ (i.e. reassembling the pieces to reconstruct the input manifold using the Bayes inverse PDF $\Pr(\tilde{\boldsymbol{x}}_0|\boldsymbol{x}_1) = \frac{\Pr(\boldsymbol{x}_1|\tilde{\boldsymbol{x}}_0)\,\Pr(\tilde{\boldsymbol{x}}_0)}{\Pr(\boldsymbol{x}_1)}$), so it is ideally suited to the task at hand. An objective function D needs to be defined to measure how accurately the reconstruction $\tilde{\boldsymbol{x}}_0$ approximates the original input $\boldsymbol{x}_0$.

It is simplest to use a Euclidean objective function that measures the average squared (i.e. L^2) distance $\|\boldsymbol{x}_0 - \tilde{\boldsymbol{x}}_0\|^2$, and which must be minimised with respect to the encoder $\Pr(\boldsymbol{x}_1|\boldsymbol{x}_0)$ (note that the decoder $\Pr(\boldsymbol{x}_0|\boldsymbol{x}_1)$ is then completely determined by Bayes' theorem).

$$D = \int d\boldsymbol{x}_0\, d\boldsymbol{x}_1\, d\tilde{\boldsymbol{x}}_0\, d\tilde{\boldsymbol{x}}_1\ \Pr(\boldsymbol{x}_0)\ \Pr(\boldsymbol{x}_1|\boldsymbol{x}_0)\,\delta(\tilde{\boldsymbol{x}}_1 - \boldsymbol{x}_1)\ \Pr(\boldsymbol{x}_0|\boldsymbol{x}_1)\,\|\boldsymbol{x}_0 - \tilde{\boldsymbol{x}}_0\|^2 \tag{3.1}$$

Using Bayes' theorem Equation 3.1 can be manipulated into the form [4]

$$D = 2\int d\boldsymbol{x}_0\, d\boldsymbol{x}_1\ \Pr(\boldsymbol{x}_0)\ \Pr(\boldsymbol{x}_1|\boldsymbol{x}_0)\,\|\boldsymbol{x}_0 - \boldsymbol{x}_0(\boldsymbol{x}_1)\|^2 \tag{3.2}$$

where D must be minimised with respect to both the encoder $\Pr(\boldsymbol{x}_1|\boldsymbol{x}_0)$ and the reconstruction vector $\boldsymbol{x}_0(\boldsymbol{x}_1)$. Note that this simplification of Equation 3.1 into Equation 3.2 depends critically on the Euclidean form of the objective function.

Figure 7 is a transformed version of Figure 6 that reflects the transformation of Equation 3.1 into Equation 3.2. The encoder $\Pr(\boldsymbol{x}_1|\boldsymbol{x}_0)$ is the most important part of this diagram, whereas the reconstruction vector $\boldsymbol{x}_0(\boldsymbol{x}_1)$ is less important so it is shown as a dashed line.

Thus far a non-parametric representation of $\Pr(\boldsymbol{x}_1|\boldsymbol{x}_0)$ and $\boldsymbol{x}_0(\boldsymbol{x}_1)$ has been used, so analytic minimisation of D [4] leads to $\Pr(\boldsymbol{x}_1|\boldsymbol{x}_0) \longrightarrow \delta(\boldsymbol{x}_1 - \boldsymbol{x}_1(\boldsymbol{x}_0))$, in which case the encoder could be perfect (i.e. lossless) and it would not be possible to discover the structure of the input manifold, as discussed in Section II. To make progress constrained forms of $\Pr(\boldsymbol{x}_1|\boldsymbol{x}_0)$ and $\boldsymbol{x}_0(\boldsymbol{x}_1)$ must be used in order to limit the resources available to the encoder/decoder, and thus force it to discover clever ways of mapping the input manifold to reduce the damage caused by having only limited coding resources.

One way of constraining the encoder/decoder is for $\boldsymbol{x}_1$

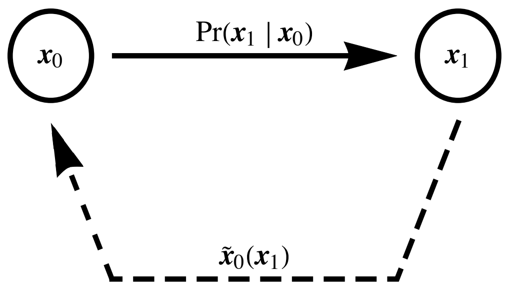


Figure 7: An encoder/decoder pair represented as the chain $\boldsymbol{x}_0 \longrightarrow \boldsymbol{x}_1 \longrightarrow \boldsymbol{x}_0(\boldsymbol{x}_1)$. This contains only those parts of the FMC that affect the Euclidean distortion objective function.

to be a *scalar* index y_1 where $y_1 = 1, 2, \cdots, m_1$ (m_1 is the size of the code book), which is a *single* sample drawn from the encoder $\Pr(\boldsymbol{x}_1|\boldsymbol{x}_0)$. Analytic minimisation of D [4] now leads to $\Pr(\boldsymbol{x}_1|\boldsymbol{x}_0) \longrightarrow \delta_{y_1, y_1(\boldsymbol{x}_0)}$ so that $D = 2\int d\boldsymbol{x}_0 \Pr(\boldsymbol{x}_0) \left\|\boldsymbol{x}_0 - \boldsymbol{x}_0(y_1(\boldsymbol{x}_0))\right\|^2$ which is the objective function for a standard least squares vector quantiser [3].

A better way of constraining the encoder/decoder is for $\boldsymbol{x}_1$ to be the *histogram* $(\nu_1, \nu_2, \cdots, \nu_{m_1})$ of counts of independent samples of the scalar index y_1 ($n_1 = \sum_{y_1=1}^{m_1} \nu_{y_1}$ is the total number of samples), and for $\boldsymbol{x}_0(\boldsymbol{x}_1)$ to be approximated as $\boldsymbol{x}_0(\boldsymbol{x}_1) \approx \sum_{y_1=1}^{m_1} \frac{\nu_{y_1}}{n_1} \boldsymbol{x}_0(y_1)$ (rather than using the full functional form $\boldsymbol{x}_0(\nu_1, \nu_2, \cdots, \nu_{m_1})$). Although it is still possible to obtain analytic results it usually requires a lot of calculation [6], and it is generally better to use a numerical optimisation approach. An upper bound for the objective function D is then given by [5]

$$D \leq \frac{2}{n_1} \int d\boldsymbol{x}_0 \Pr(\boldsymbol{x}_0) \sum_{y_1=1}^{m_1} \Pr(y_1|\boldsymbol{x}_0) \left\|\boldsymbol{x}_0 - \boldsymbol{x}_0(y_1)\right\|^2 + \frac{2(n_1-1)}{n_1} \int d\boldsymbol{x}_0 \Pr(\boldsymbol{x}_0) \left\|\boldsymbol{x}_0 - \sum_{y_1=1}^{m_1} \Pr(y_1|\boldsymbol{x}_0)\, \boldsymbol{x}_0(y_1)\right\|^2 \quad (3.3)$$

where the unconstrained encoder/decoder corresponds to the left hand side of Equation 3.3 and the constrained encoder/decoder corresponds to the right hand side of Equation 3.3. Note how the random fluctuations in the *multiple* sample histogram are analytically summed over in Equation 3.3, leaving only the *single* sample encoder $\Pr(y_1|\boldsymbol{x}_0)$ to be optimised.

A further constraint is to assume that $\Pr(y_1|\boldsymbol{x}_0)$ is parameterised as the normalised output of a set of sigmoid functions

$$\begin{aligned} \Pr(y_1|\boldsymbol{x}_0) &= \frac{Q(y_1|\boldsymbol{x}_0)}{\sum_{y_1'=1}^{m_1} Q(y_1'|\boldsymbol{x}_0)} \\ Q(y_1|\boldsymbol{x}_0) &= \frac{1}{1+\exp(-\boldsymbol{w}_{10}(y_1).\boldsymbol{x}_0 - b_1(y_1))} \end{aligned} \quad (3.4)$$

$$\begin{aligned} \Pr(y_1|\boldsymbol{x}_0) &= \frac{Q(y_1|\boldsymbol{x}_0)}{\sum_{y_1'=1}^{m_1} Q(y_1'|\boldsymbol{x}_0)} \\ Q(y_1|\boldsymbol{x}_0) &= \frac{1}{1+\exp(-\boldsymbol{w}_{10}(y_1).\boldsymbol{x}_0 - b_1(y_1))} \end{aligned} \quad (3.5)$$

where $Q(y_1|\boldsymbol{x}_0)$ is the unnormalised output from code index y_1, depending on the weight vector $\boldsymbol{w}_{10}(y_1)$ and the bias $b_1(y_1)$. This parameterisation of $\Pr(y_1|\boldsymbol{x}_0)$ ensures that it can be used to slice pieces off convex manifolds as illustrated in Figure 4. Optimisation of the objective function is then achieved by gradient descent variation of the three sets of parameters $\boldsymbol{w}_{10}(y_1)$, $b_1(y_1)$, and $\boldsymbol{x}_0(y_1)$. These (and other) derivatives of the objective function were given in [5].

The constrained objective function in Equation 3.3 and Equation 3.4 yields a great variety of useful results, such as the simple result shown in Figure 4 which used $m_1 = 6$, $n_1 = 20$, and $\boldsymbol{x}_0 = (\cos\theta, \sin\theta)$ with θ uniformly distributed in $[0, 2\pi]$, to learn a mapping from the 1-dimensional input manifold embeded in a 2-dimensional space ($\dim \boldsymbol{x}_0 = 2$) to a 6-dimensional space ($m_1 = 6$). This is the key objective function that can be used to optimise the mapping of the input manifold to a new representation $\Pr(y_1|\boldsymbol{x}_0)$ for $y_1 = 1, 2, \cdots, m_1$. Minimising the Euclidean distortion ensures that $\Pr(y_1|\boldsymbol{x}_0)$ defines an optimal mapping of the input manifold, such that when n_1 samples are drawn from $\Pr(y_1|\boldsymbol{x}_0)$ they contain enough information to form an accurate reconstruction of $\boldsymbol{x}_0$.

Figure 8 is a transformed version of Figure 7 that shows an example of the structure of some types of optimal solution that are obtained by minimising the constrained objective function in Equation 3.3. The tensor product structure of the input manifold is revealed in this type of solution, because the input vector $\boldsymbol{x}_0$ splits into two parts as $\boldsymbol{x}_0 = (x_0^a, x_0^b)$ each of which is separately encoded/decoded. This type of factorial encoder is favoured by limiting the size of the code book m and by using an intermediate number of samples n_1 ($n_1 = 1$ leads to a standard VQ, and $n_1 \longrightarrow \infty$ allows too many coding resources to lead to clever encoding schemes).

The self-organised emergence of factorial encoders is one of the major strengths of the SVQ approach. It allows the code book to split into two or more separate smaller code books in a data driven way rather than being hard-wired into the code book at the outset (e.g. [10]).

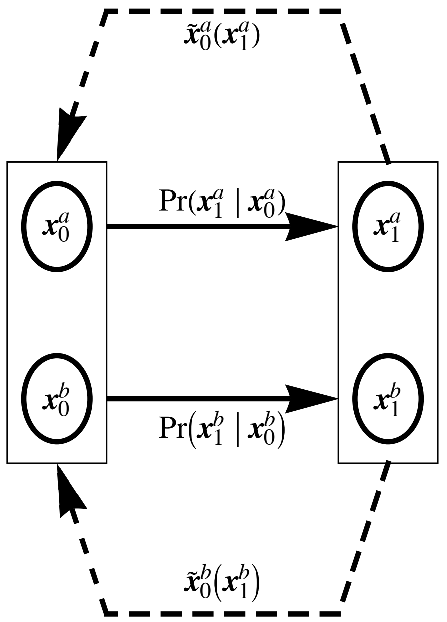


Figure 8: A factorial encoder/decoder pair represented as the pair of disconnected chains $x_0^a \longrightarrow x_1^a \longrightarrow x_0^a(x_1^a)$ and $x_0^b \longrightarrow x_1^b \longrightarrow x_0^b(x_1^b)$. The input vector is $\boldsymbol{x}_0 = (x_0^a, x_0^b)$ highlighted by the left hand rectangle, the code is $\boldsymbol{x}_1 = (x_1^a, x_1^b)$ highlighted by the right hand rectangle, and the reconstruction is $\boldsymbol{x}_0 = (x_0^a, x_0^b)$. The dependencies amongst the variables is indicated by the arrows in the diagram, which shows that subspaces a and b are *independently* encoded/decoded.

B. Chain of Stochastic Vector Quantisers

The encoder/decoder in Figure 7 leads to useful results for mapping the input data manifold when its operation is constrained in various ways. A much larger variety of mappings may be constructed if the encoder/decoder is viewed as a basic module, and then networks of linked modules are used to process the data [8]. It is simplest to regard this type of network as progressively mapping the input manifold as it flows through the network modules.

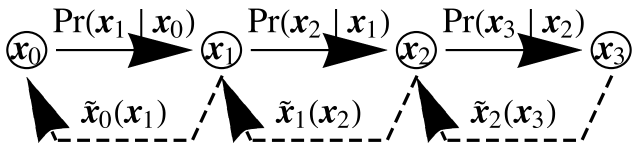


Figure 9: A 3-stage chain of linked SVQs. The l^{th} encoder is modelled using the conditional probability $\Pr_{l,l-1}(\boldsymbol{x}_l|\boldsymbol{x}_{l-1})$, and the corresponding decoder is modelled using the reconstruction vectors $\tilde{\boldsymbol{x}}_{l-1}(\boldsymbol{x}_l)$, where each $\boldsymbol{x}_l$ is a histogram of samples.

Figure 9 shows a 3-stage chain of linked encoder/decoders of the type shown in Figure 7. The important part of this diagram is the processing chain which is the solid line flowing from left to right at the top of the diagram creating the Markov chain $\boldsymbol{x}_0 \xrightarrow{\Pr(\boldsymbol{x}_1|\boldsymbol{x}_0)} \boldsymbol{x}_1 \xrightarrow{\Pr(\boldsymbol{x}_2|\boldsymbol{x}_1)} \boldsymbol{x}_2 \xrightarrow{\Pr(\boldsymbol{x}_3|\boldsymbol{x}_1)} \boldsymbol{x}_3$. The reconstruction vectors $\boldsymbol{x}_{l-1}(\boldsymbol{x}_l)$ for $l = 1, 2, 3$ are the dashed lines flowing from right to left.

The state $\boldsymbol{x}_l$ of layer l of the chain is the histogram $(\nu_1, \nu_2, \cdots, \nu_{m_l})$ of counts of samples drawn from $\Pr(y_l|\boldsymbol{x}_{l-1})$ ($n_l = \sum_{y_l=1}^{m_l} \nu_{y_l}$ is the total number of samples). In numerical implementations $\boldsymbol{x}_l$ is chosen to be the (normalised) histogram for an infinite number of samples (i.e. the relative frequencies implied by $\Pr(y_l|\boldsymbol{x}_{l-1})$), and $\boldsymbol{x}_{l-1}(\boldsymbol{x}_l)$ is chosen to depend on only a finite number m_l of samples randomly selected from this histogram $\boldsymbol{x}_{l-1}(\boldsymbol{x}_l) \approx \sum_{y_l=1}^{m_l} \frac{\nu_{y_l}}{n_l} \boldsymbol{x}_{l-1}(y_l)$. This choice of how to operate the network is not unique but it has the advantage of simplifying the computations. The *infinite* number of samples used in $\boldsymbol{x}_l$ ensures that the $\boldsymbol{x}_l$ do not randomly fluctuate, so no Monte Carlo simulations are required to implement the feed-forward flow through the network. The *finite* number of samples n_l used in $\boldsymbol{x}_{l-1}(\boldsymbol{x}_l)$ leads to exactly the same objective function as in Equation 3.3 where the random fluctuations are analytically summed over, which ensures that each decoder has limited resources and thus forces the optimisation of the network to discover intelligent ways of encoding the data. This type of network reduces to a standard way of using a Markov chain when only a single sample is drawn from each of the $\Pr(y_l|\boldsymbol{x}_{l-1})$.

Each stage of the chain corresponds to an objective function of the form shown in Equation 3.3 but applied to the l^{th} stage of the chain. The total objective function is a weighted sum of these individual contributions. This encourages all of the mappings in the chain to minimise their average Euclidean reconstruction error, which gives a progressive mapping of the input manifold along the processing chain. However, the relative weighting of the later stages of the chain must not be too great otherwise they force the earlier mappings in the chain to become singular (e.g. all inputs mapped to the same output), because the output of a singular mapping can be mapped with little or no more contribution to the overall objective function further along the chain. A less extreme form of this phenomenon can be used to encourage factorial encoders to emerge, because they produce a (normalised) histogram output state that has a smaller volume (in the Euclidean sense) than a non-factorial encoder, which reduces the size of the contribution to the overall objective function from the next stage in the chain.

If the chain network topology in Figure 9 is combined with the factorial encoder/decoder property of SVQs shown in Figure 8 then all acyclic network topologies are possible. This can be seen intuitively because flow through the chain corresponds to flow along the time-like direction in an acyclic network (i.e. following the directed links), and multiple parallel branches occur wherever there is an SVQ factorial encoder in the chain. An example of the emergence of this type of network topology will be shown in Section IV.

IV. LEARNING A HIERARCHICAL NETWORK

The purpose of this section is to demonstrate the self-organised emergence of a hierarchical network topology starting from a chain-like topology of the type shown in Figure 9. For the purpose of this demonstration the raw data must have an appropriate correlation structure, which will be achieved by generating the data as a set of hierarchically correlated phases. Thus each data vector is a 4-dimensional vector of phases $\boldsymbol{\phi} = (\phi_1, \phi_2, \phi_2, \phi_4)$, where the ϕ_i are the leaf nodes of a binary tree of phases, where the binary splitting rule used is $\phi \longrightarrow (\phi-\alpha, \phi+\beta)$ with α and β being independently and uniformly sampled from the interval $[0, \frac{\pi}{2}]$, and the phase of the root node is uniformly distributed in the interval $[0, 2\pi]$. This will lead to each of the ϕ_i being uniformly distributed phase variables thus uniformly occupying a circular manifold. However, because the ϕ_i are correlated due to the way that they are generated by the binary splitting process, the $\boldsymbol{\phi}$ do *not* uniformly populate a 4-torus manifold (i.e. tensor product of 4 circles).

Figure 5 showed an example of what the manifold of a pair of correlated variables looks like, with a large degree of freedom (e.g. $\phi_1 + \phi_2$) and a small degree of freedom (e.g. $\phi_1 - \phi_2$), and the hyperplanes encoding the manifold in such a way as to discard information about the small degree of freedom (e.g. $\phi_1 - \phi_2$). In this way the 3-stage chain in Figure 9 can progressively discard information about small degrees of freedom in $\boldsymbol{\phi}$, starting with a 4-torus manifold (non-uniformly populated) and ending up with a circular manifold (uniformly populated), as will be seen below. This is the basic idea behind using this type of self-organising network for data fusion.

Figure 10 shows the co-occurrence matrices of pairs of the ϕ_i displayed as scatter plots. The bands in these plots wrap around circularly and correspond to the manifold shown in Figure 5. Because ϕ_1 and ϕ_2 (and also ϕ_3 and ϕ_4) lie close to each other in the hierarchy, $\Pr(\phi_1,\phi_2)$ and $\Pr(\phi_3, \phi_4)$ have a narrower band than $\Pr(\phi_1,\phi_3)$, $\Pr(\phi_1,\phi_4)$, $\Pr(\phi_2,\phi_3)$ and $\Pr(\phi_2,\phi_4)$.

A 3-stage chain of linked SVQs of the type shown in Figure 9 is now trained, where each stage contributes an objective function of the form shown in Equation 3.3 and Equation 3.4. The sizes M of each of the 4 network layers are $\boldsymbol{M} = (8, 16, 8, 4)$. The size of layer 0 (the input layer) is determined by the dimensionality of the input data, whereas the sizes of each of the other layers is chosen to be 4 times the number of phase variables that each is expected to use in its encoding of the input data, which encourages the progressive removal of small degrees of freedom from the data as it flows along the chain into ever smaller layers. The number of samples n used for each of the 3 SVQ stages are $\boldsymbol{n} = (20, 20, 20)$,

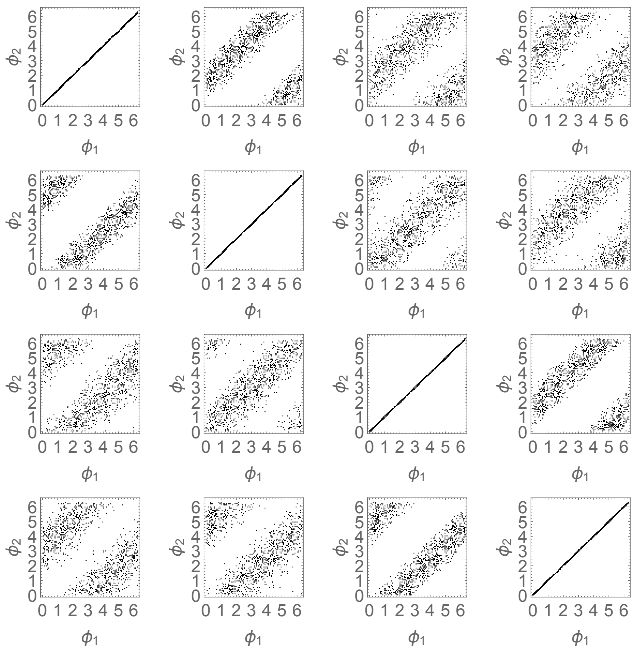


Figure 10: Co-occurrence matrices of all pairs of phases. Note that the block structure is symmetric so each off-diagonal co-occurrence matrix appears twice. The hierarchical correlations cause $\phi_1$and ϕ_2 (and similarly ϕ_3 and ϕ_4) to be more strongly correlated with each other than ϕ_2 and ϕ_3.

which are large enough to allow each SVQ to develop into a factorial encoder, so that the processing can proceed in parallel along several paths (which are progressively fused) along the chain. The relative weightings λ assigned to the objective functions contributed by each of the 3 SVQ stages are $\boldsymbol{\lambda} = (1, 5, 0.1)$, where a large weighting is assigned to the stage 2 SVQ to encourage the stage 1 SVQ to develop into a factorial encoder, and a small weighting is assigned to the stage 3 SVQ because the stage 2 SVQ needs no additional encouragement to develop into a factorial encoder.

The network was trained by a gradient descent on the overall network objective function, using a step size chosen separately for each SVQ stage and separately for each of the 3 parameter types $\boldsymbol{w}_{l,l-1}(y_l)$, $b_l(y_l)$, and $\boldsymbol{x}_{l-1,l}(y_l)$ in each SVQ stage (for $l = 1, 2, 3$). The size of all of these step size parameters was chosen to be large at the start of the training schedule, and then gradually reduced as training progressed, with the relative rate of reduction being chosen to encourage earlier SVQ stages to converge before later SVQ stages. In general, different choices of network parameters and training conditions lead to different types of trained network, and since there is no prior reason for choosing one particular solution in preference to another the choice must be left up to the user. All the components of the weight vectors, biases, and reconstruction vectors are initialised to random numbers uniformly distributed in the interval $[-0.1, 0.1]$.

Figure 11 shows the reconstruction vectors $\boldsymbol{x}_{l-1,l}(y_l)$ (for $l = 1, 2, 3$) in a trained 3-stage chain of linked

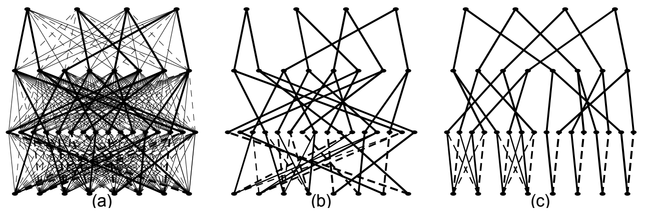


Figure 11: Reconstruction vectors $x_{l-1,l}(y_l)$ (for $l = 1, 2, 3$) after training a 3-stage network of linked SVQs on the hierarchically correlated phases data. These diagrams are rotated 90° anticlockwise relative to Figure 9, so the processing chain runs from bottom to top of each diagram. Line thickness indicates the size of a reconstruction vector component, and dashing indicates that the component is negative. (a) All reconstruction vector components. (b) Largest reconstruction vector components obtained by applying a threshold to the magnitude of each component. (c) The same as (b) except that the positions of the nodes in all layers (other than the input layer) have been permuted (along with the connections between layers) to make the network topology clearer.

SVQs of the type shown in Figure 9. Reconstruction vectors are displayed because they are easier to interpret than weight vectors. The data $\boldsymbol{\phi} = (\phi_1, \phi_2, \phi_2, \phi_4)$ is embedded in an 8-dimensional input space as $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ where $(x_{2i-1}, x_{2i}) = (\cos\phi_i, \sin\phi_i)$. The key diagram is Figure 11c which shows the largest components of the reconstruction vectors, and has been reordered to make the hierarchical network topology clear. Each of the first two stages of this network has learnt to operate as two or more encoder/decoders (i.e. a factorial encoder/decoder) as in Figure 8. The first stage of the network breaks into 4 encoder/decoders that encode each of the ϕ_i (see the results in Figure 14 and Figure 15 for justification of this), the second stage of the network breaks into 2 encoder/decoders that encode $\phi_1 + \phi_2$ and $\phi_3 + \phi_4$ (see the results in Figure 16 and Figure 17 for justification of this), and the third stage of the network is a single encoder that encodes $\phi_1 + \phi_2 + \phi_3 + \phi_4$ (see the results in Figure 18 and Figure 19 for justification of this). The connectivity in the stage 1 SVQ is not the same for all of the ϕ_i because of the interaction between the thresholding prescription used to create Figure 11 and the different orientation of each of the 4 parts of the stage 1 factorial encoder with respect to each of the 4 corresponding circular input manifolds.

Although the network in Figure 11 computes using continuous-valued numbers, the thresholded reconstruction vectors in Figure 11b may be inspected to reveal the symbolic logic expressions that approximate to each of the (thresholded) outputs $O_i(\boldsymbol{x})$ for $i = 1, 2, 3, 4$ from the highest layer of the network (using logical negation $\overline{x}_i$ to denote $-x_i$ because the inputs lie in the range $[-1, 1]$).

$$\begin{aligned} O_1(\boldsymbol{x}) &= \overline{x}_2 \cap x_3 \cap x_5 \cap x_8 \\ O_2(\boldsymbol{x}) &= x_2 \cap \overline{x}_3 \cap \overline{x}_5 \cap \overline{x}_8 \\ &= \overline{O}_1(\boldsymbol{x}) \\ O_3(\boldsymbol{x}) &= \overline{x}_1 \cap \overline{x}_4 \cap \overline{x}_6 \cap x_7 \\ O_4(\boldsymbol{x}) &= x_1 \cap x_4 \cap x_6 \cap \overline{x}_7 \\ &= \overline{O}_3(\boldsymbol{x}) \end{aligned} \tag{4.1}$$

In order to keep these expressions short they use a slightly higher threshold than was used to create Figure 11b, because this suppresses some of the reconstruction vector components linked to (x_1, x_2, x_3, x_4). In this particularly simple example it is possible to obtain very short symbolic expressions, but more generally continuous-valued computations would be needed to obtain good approximations to the network outputs.

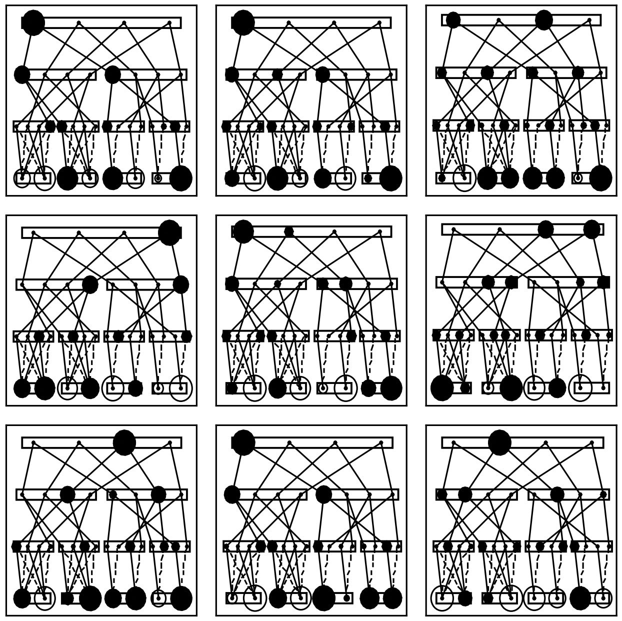


Figure 12: Some typical examples of node activities $\Pr(y_l|\boldsymbol{x}_{l-1})$ (for $l = 1, 2, 3$) in the trained 3-stage network of linked SVQs. The permuted version of the network is used to make the results easier to interpret. The area of each filled circle is proportional to the activity it represents, and the negative values that occur in the input layer are represented by unfilled circles. The hierarchical structure of the network is indicated by drawing a box around each part of each network layer that acts as a separate encoder, so that typically there are one or two active nodes within every box.

Figure 12 shows some examples of the node activities $\Pr(y_l|\boldsymbol{x}_{l-1})$ (for $l = 1, 2, 3$) in the trained network shown in Figure 11c. The individual pieces of each factorial encoder are indicated by the boxes, and the patterns of activity are such that every box contains one or more active nodes, as would be expected if each box were acting as a separate encoder.

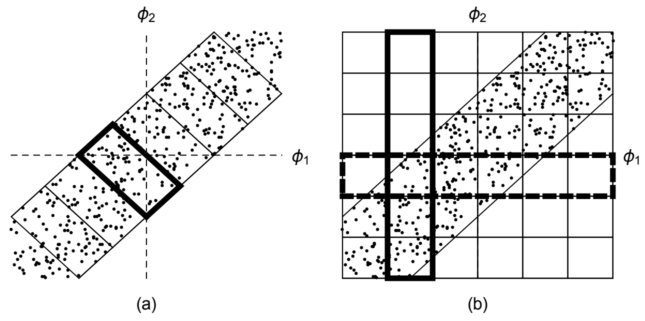


Figure 13: Two ways of encoding a pair of correlated phases ϕ_1 and ϕ_2. The co-occurrence matrix of ϕ_1 and ϕ_2 is represented as a narrow band that is populated by data points, so that the correlation manifests itself as $\phi_1 \approx \phi_2$. (a) This shows how an invariant encoder operates, in which the response region of each node is oriented so that it has high resolution for the $\phi_1 + \phi_2$ but is completely insensitive to $\phi_1 - \phi_2$. This does not encode information about the small degree of freedom measured *across* the band of the co-occurrence matrix. (b) This shows how a factorial encoder operates, in which the response region of each node is highly anisotropic, with high resolution for one of the phases but completely insensitive to the other phase. Accurate encoding is achieved by using the nodes in *pairs* with orthogonally intersecting response regions, as shown in the example highlighted in the diagram.

Figure 13 shows simplified versions of the two types of encoder/decoder that occur in Figure 11 overlaid on a co-occurrence matrix of the type shown in Figure 10.

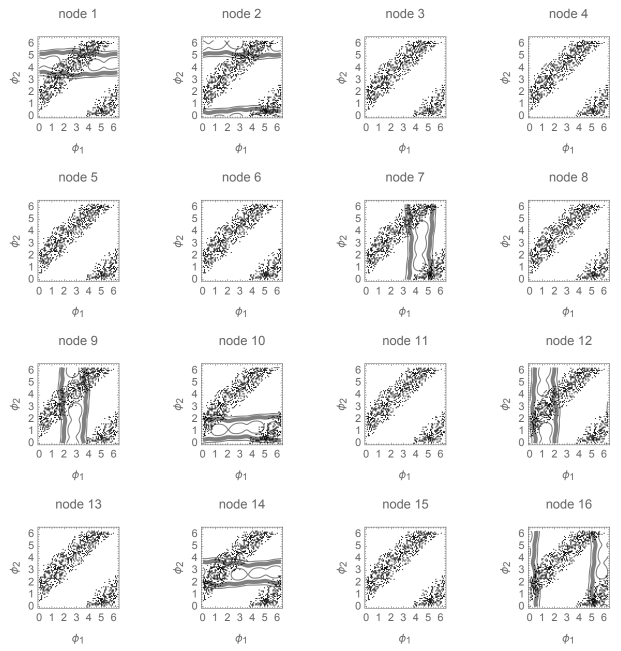


Figure 15: Node activities in layer 1 as a function of the inputs ϕ_3 and ϕ_4 (with $\phi_1 = \phi_2 = 0$) which has the properties of a factorial encoder. The non-zero responses correspond to the 8 nodes in layer 1 that are strongly connected to ϕ_3 and ϕ_4.

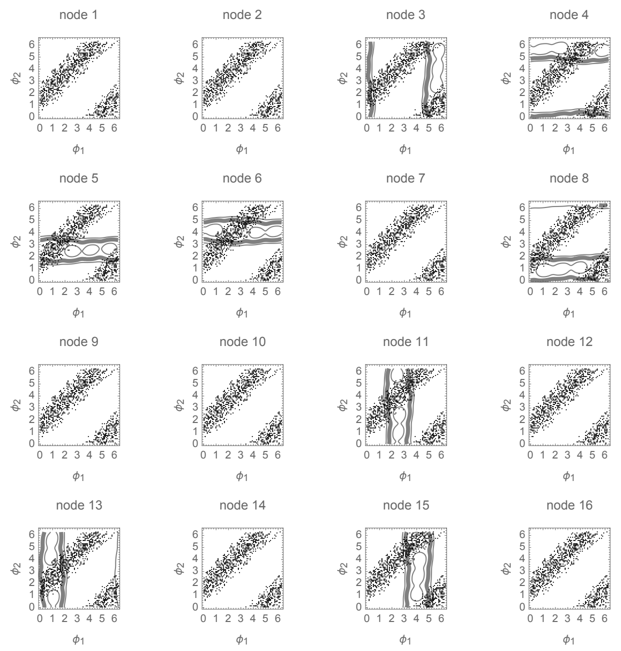


Figure 14: Node activities in layer 1 as a function of the inputs ϕ_1 and ϕ_2 (with $\phi_3 = \phi_4 = 0$) which has the properties of a factorial encoder. In each plot the contours representing the node reponse are overlaid on the co-occurrence matrix of the pair of inputs ϕ_1 and ϕ_2. One of the contour heights is drawn bold to highlight the region where the node response is large. Half of the nodes do not respond at all, and the other half split into two subsets of equal size, one with high resolution in ϕ_1 but completely insensitive to ϕ_2, and the other with high resolution in ϕ_2 but completely insensitive to ϕ_1. The response is sensitive to the small degree of freedom measured *across* the band of the co-occurrence matrix. The non-zero responses correspond to the 8 nodes in layer 1 that are strongly connected to ϕ_1 and ϕ_2.

Figure 14 and Figure 15 show the encoders that occur in stage 1 of Figure 11. These are all factorial encoders of the type shown in Figure 13b, as can be seen from the orientation of the response regions for the various nodes which cuts across the band of the co-occurrence matrix in the same way as in Figure 13b. This corresponds to the connectivity seen in Figure 11c where each ϕ_i has its own encoder.

Figure 16 and Figure 17 show the encoders that occur in stage 2 of Figure 11. These are invariant encoders of the type shown in Figure 13a, as can be seen from the orientation of the response regions for the various nodes which cuts across the band of the co-occurrence matrix in the same way as in Figure 13a. This corresponds to the connectivity seen in Figure 11c where each of $\phi_1 + \phi_2$ and $\phi_3 + \phi_4$ has its own encoder.

Figure 18 and Figure 19 show the encoder that occurs in stage 3 of Figure 11. This is an invariant encoder of the type shown in Figure 13a, which corresponds to the connectivity seen in Figure 11c.

The diagrams in this section show how a 3-stage chain

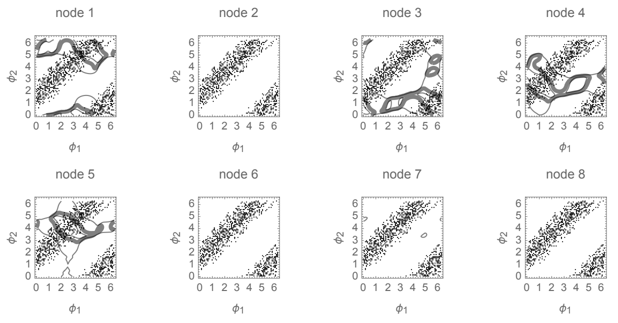


Figure 16: Node activities in layer 2 as a function of the inputs ϕ_1 and ϕ_2 (with $\phi_3 = \phi_4 = 0$) which has the properties of an invariant encoder. Half of the nodes do not respond at all, and the other half respond to well-defined regions in ϕ_1 and ϕ_2. The contours representing the node response are overlaid on the co-occurrence matrix of the pair of inputs ϕ_1 and ϕ_2. One of the contour heights is drawn bold to highlight the region where the node response is large. This shows that each node responds to a local region of the *populated* region of the co-occurrence matrix, and to a limited extent generalises outside this region. The response is invariant with respect to the small degree of freedom measured *across* the band of the co-occurrence matrix, which demonstrates that layer 2 has acquired an invariance that was absent in layer 1. There are also non-zero responses in the *unpopulated* region of the co-occurrence matrix which arise because the sum of the node activities is normalised. The non-zero responses correspond to the 4 nodes in layer 2 that are strongly connected to ϕ_1 and ϕ_2.

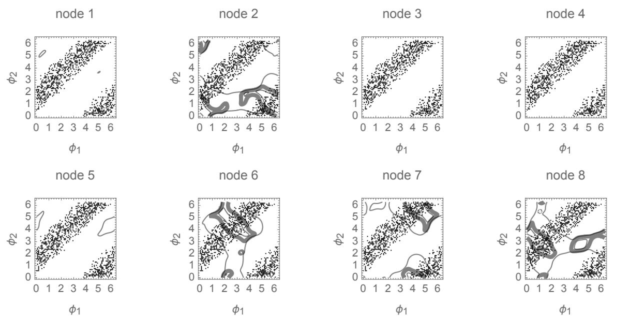


Figure 17: Node activities in layer 2 as a function of the inputs ϕ_3 and ϕ_4 (with $\phi_1 = \phi_2 = 0$) which has the properties of an invariant encoder. The non-zero responses correspond to the 4 nodes in layer 2 that are strongly connected to ϕ_3 and ϕ_4.

of linked SVQs of the type shown in Figure 9 self-organises to process hierarchically correlated phase data. Stage 1 makes an approximate copy of the data where each phase is separately encoded, then stage 2 encodes the output of stage 1 discarding the smallest degrees of freedom, and finally stage 3 encodes the output of stage 2 discarding the next smallest degree of freedom. The chain of SVQs has thus split itself into a hierarchical network of linked encoders that is optimally matched to the task of mapping from the original data at the input to the chain to the compressed representation at the output

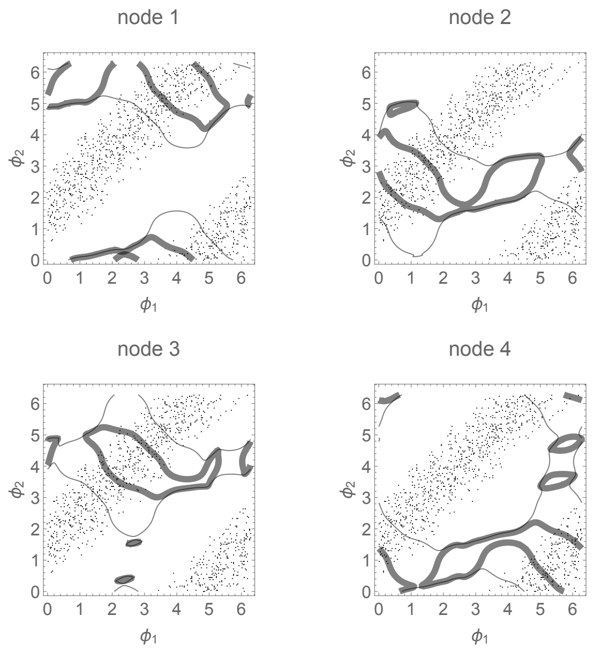


Figure 18: Node activities in layer 3 as a function of the inputs ϕ_1 and ϕ_2 (with $\phi_3 = \phi_4 = 0$) which has the properties of an invariant encoder.

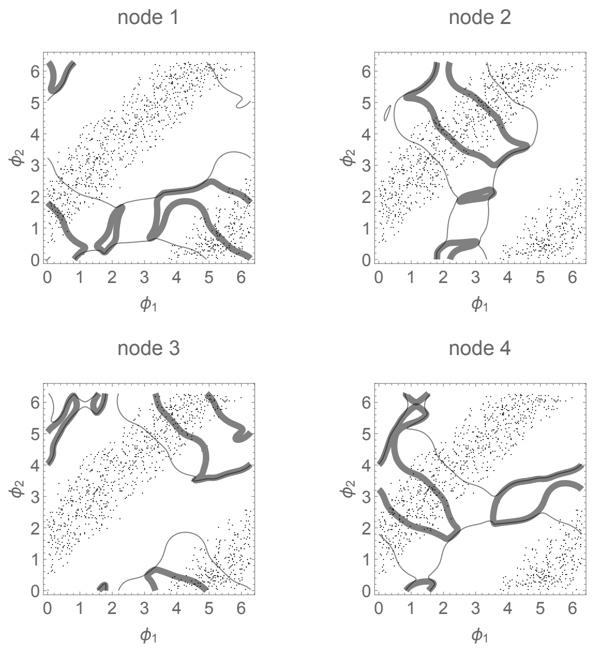


Figure 19: Node activities in layer 3 as a function of the inputs ϕ_3 and ϕ_4 (with $\phi_1 = \phi_2 = 0$) which has the properties of an invariant encoder.

of the chain. This is true self-organisation of multiple encoders unlike the hard-wiring of encoders that is used in other approaches (e.g. [10]).

V. CONCLUSIONS

This paper has shown how it is possible to map a data manifold into a simpler form by progressively discarding small degrees of freedom. This is the key to self-organising data fusion, where the raw data is embedded in a very high-dimensional space (e.g. the pixel values of one or more images), and the requirement is to isolate the important degrees of freedom which lie on a low-dimensional manifold. A useful advantage of the approach used in this paper is that it assumes only that the mapping from manifold to manifold is organised in a chain-like topology, and that all the other details of the processing in each stage of the chain are to be learnt by self-organisation. The types of application for which this approach is well-suited are ones in which separation of small and large degrees of freedom is desirable. For instance, separation of targets (small) and jammers (large) is relatively straightforward using this approach [7].

Data that is not embedded in a higher-dimensional space is usually not suitable for processing with the approach used in this paper. For instance, categorical (or symbolic) data that has one of only a few possible states is not suitable, but a smoothly variable array of pixel values is suitable. This type of network is intended to operate on raw sensor data rather than pre-processed data, and typically will use high-dimensional intermediate representations in its processing chain. Because of its ability to compress the raw data into a much simpler form, this type of network would typically be used as a bridge between the sub-symbolic raw sensor data and the symbolic higher level representation of that data.

Although the full connectivity between adjacent layers of the chain implies that the computations can be expensive, after some initial training the factorial structure of the various encoders becomes clear and can be used to prune the connections to keep only the ones that are actually used (i.e. usually only a small proportion of the total number). When the chain is fully trained each code index typically depends on only a small number of contributing inputs (i.e. a receptive field) from the previous stage of the chain. Furthermore, because of the normalisation used in each layer, the size and shape of the receptive fields mutually interact (i.e. there is a fixed total amount of activity in each layer), so the raw receptive fields (i.e. as defined by the feed-forward network weights) are *different from* the renormalised receptive fields (i.e. after taking account of normalisation).

The network described in this paper passes information along the processing chain in a deterministic fashion, because it uses (hypothetical) histograms containing an *infinite* number of samples, which thus do not randomly fluctuate. This was done for computational convenience (i.e. to avoid Monte Carlo simulations) and is *not* a fundamental limitation of the approach used. With additional computational effort it is possible to operate the network as a (non-deterministic) Markov chain in which the histograms contain only a *finite* number of samples, which therefore randomly fluctuate and explore network states in the vicinity of the deterministic state used in this paper.

VI. ACKNOWLEDGEMENTS

The research presented in this paper was supported by the United Kingdom's MoD Corporate Research Programme.

[1] M Dittenbach, D Merkl, and A Rauber, *The growing hierarchical self-organising map*, Proceedings of international joint conference on neural networks (Como) (S-I Amari, C L Giles, M Gori, and V Piuri, eds.), IEEE Computer Society, 2000, pp. 15–19.

[2] T Kohonen, *Self-organising maps*, Springer-Verlag, Berlin, 2001.

[3] Y Linde, A Buzo, and R M Gray, *An algorithm for vector quantiser design*, IEEE Transactions on Communications **28** (1980), no. 1, 84–95.

[4] S P Luttrell, *A Bayesian analysis of self-organising maps*, Neural Computation **6** (1994), no. 5, 767–794.

[5] ———, *Mathematics of neural networks: models, algorithms and applications*, ch. A theory of self-organising neural networks, pp. 240–244, Kluwer, Boston, 1997.

[6] ———, *Combining artificial neural nets: Ensemble and modular multi-net systems*, Perspectives in Neural Computing, ch. Self-organised modular neural networks for encoding data, pp. 235–263, Springer-Verlag, London, 1999.

[7] ———, *Mathematics in signal processing*, vol. 5, ch. Using stochastic vector quantisers to characterise signal and noise subspaces, pp. 193–204, Oxford University Press, 2002.

[8] ———, *A Markov chain approach to multiple classifier fusion*, Proceedings of international workshop on multiple classifier fusion (London) (T Windeatt and F Roli, eds.), Springer-Verlag, 2003, pp. 217–226.

[9] T M Martinez and K J Schulten, *Artificial neural networks*, ch. A "neural-gas" network learns topologies, pp. 397–402, North-Holland, Amsterdam, 1991.

[10] D A Ross and R S Zemel, *Advances in neural information processing systems*, vol. 15, ch. Multiple-cause vector quantisation, pp. 1041–1046, MIT Press, Cambridge, 2002.

Discrete Network Dynamics. Part 1: Operator Theory [*]

S P Luttrell
Room EX21, QinetiQ, Malvern Technology Centre

An operator algebra implementation of Markov chain Monte Carlo algorithms for simulating Markov random fields is proposed. It allows the dynamics of networks whose nodes have discrete state spaces to be specified by the action of an update operator that is composed of creation and annihilation operators. This formulation of discrete network dynamics has properties that are similar to those of a quantum field theory of bosons, which allows reuse of many conceptual and theoretical structures from QFT. The equilibrium behaviour of one of these generalised MRFs and of the adaptive cluster expansion network (ACEnet) are shown to be equivalent, which provides a way of unifying these two theories.

I. INTRODUCTION

The aim of this paper is to present a theoretical framework for building recurrent network models where the states of the network nodes are discrete-valued, which will define a general framework for discrete information processing that can be implemented in various computational architectures. The introduction of recurrence into networks makes them much more difficult to analyse and control than feed-forward networks. The basic reason for these difficulties is that loopy propagation in recurrent networks causes each network observable to be a sum of an infinite (or, at least, a very large) number of contributions.

One type of network that can be modelled using this framework is a network of spiking neurons, where the presence or absence of a spike is a binary quantity (i.e. it is discrete-valued). However, in this paper, there is no specific aim to model biological information processing, but there will nevertheless be points of contact between the general information processing framework presented here and the specific details of biological information processing.

The only consistent way of processing information is to use Bayesian methods [2], which represent information by using the joint probability of the states of the network nodes, and process information (or make inferences) by manipulating these joint probabilities according to well-defined rules such as Bayes theorem. The Bayesian approach achieves its consistency by *not* discarding any of the various alternative inferences that can be made, and by following up the consequences of all of the alternatives it ensures that there are never any of the contradictions that would otherwise occur, such as reaching conclusions that depend on which route one takes through the maze of inferences.

Bayesian information processing needs a flexible way of representing and manipulating joint probabilities. An ideal framework for this is Markov random field (MRF) theory [8], because it allows one to systematically build up a joint probability model out of pieces that have a simple functional dependence on the underlying state variables. For networks that have a finite number of nodes, each of which has a finite number of states, the MRF approach allows *all* possible joint probability models to be constructed, so use of the MRF framework imposes no artificial constraints. Because the MRF approach constructs a joint probability model, it can be cleanly coupled to any other probability modelling approach.

The implementation of MRFs is usually done using stochastic Markov chain Monte Carlo (MCMC) computations, unless the MRF happens to have a particularly simple topology which allows a simpler deterministic implementation to be achieved (e.g. a tree-like topology allows exact computations to be done). In this paper no simplifying assumptions will be made about the network topology, in order to create the most general possible theoretical framework for discrete information processing. The simplest type of MCMC computation stochastically updates the joint state of the MRF, so that it moves around its joint state space visiting every joint state with a frequency that is proportional to the joint probability specified by the MRF. More sophisticated MCMC computations do the same thing but with an *ensemble* of joint states of the MRF; these are known as "particle filtering" algorithms [3].

The main result that is presented in this paper is a new way of describing MCMC algorithms, in which the updating of the MRF joint state (i.e. the joint state of the network nodes) is decomposed into a set of more elementary operations, which are the creation and annihilation of network node states. In the simplest case, a single MCMC update changes the joint state of an MRF by modifying its state at a single node of the network, which can be decomposed into first annihilating the old node state then creating the new node state. Any MCMC algorithm can be composed out of a sequence of such creation and annihilation operations. Furthermore, the properties of the operators that enact these creation and annihilation operations are very familiar to physicists,

[*]Typeset in LaTeX on May 3, 2019.
This is archived at http://arxiv.org/abs/cs/0511027, 7 Nov 2005. This paper was submitted to J. Machine Learning Research on 25 January 2006. Paper No. 06-026. It was not accepted for publication, but it underpins several subsequently published papers. © Copyright QinetiQ Ltd 2005

because they are identical to the properties of the creation and annihilation operators that appear in a quantum field theory (QFT) of bosons [9]. This allows a lot of prexisting conceptual and computational machinery to be brought to bear upon the problem of describing MCMC algorithms. By drawing an analogy with multiparticle QFT states, the MRF framework can be consistently generalised so that each node of the network exists in a *multiply* occupied state, rather than a *singly* occupied state. There are also many other points of contact with QFT.

The generalisation of the MRF framework to multiply occupied node states allows contact to be made with a particular type of self-organising network (SON) theory known as the adaptive cluster expansion network (ACEnet) [6]. One of the aims of a SON is to discover for itself what network architecture to use to solve an information processing task, so it must be able to dynamically change its architecture. This requires splitting and merging of network nodes, and also the creation of appropriate links between them. In an MRF, if a node is split into two nodes there is no consistent way of assigning a pairwise state to the resulting pair of nodes, unless the preexisting single node had two (or more) states assigned to it in the first place. This is exactly what multiple occupancy in the generalised MRF framework provides, using creation and annihilation operators to manipulate these states. Thus the creation and annihilation operator approach allows MRF theory and SON theory can be cleanly unified.

The structure of this paper is as follows. In Section II the theory of MRFs is summarised, together with the details of MCMC algorithms for simulating MRFs. In Section III the main new contribution of this paper is presented, which is an operator implementation of the MCMC algorithm that generalises MRF theory to multiple occupancy states. Finally, in Section IV some simple applications are used to illustrate the use of this operator implementation, one of which is the demonstration that the equilibrium state of a particular type of multiply occupied MRF has the same properties as ACEnet.

II. MARKOV RANDOM FIELDS

The aim of this section is to review the MRF framework for building and manipulating the joint probability models that are used when doing Bayesian information processing. This includes some informal material in which multiple occupancy of node states is discussed before giving the more formal development later on in Section III.

Section II A introduces MRFs and the Hammersley-Clifford expansion of joint probabilities, and Section II B describes an MCMC algorithm for sampling the joint states of an MRF. Section II D introduces the concept of a multiple occupancy state which is essential for the generalisation of MRFs that is presented later in Section III. Finally, Section II C describes how MRFs can be used to do Bayesian inference.

A. Basic Markov Random Field Theory

MRFs are a flexible way of constructing joint probabilities based on the Hammersley-Clifford expansion (HCE), which is defined as [1]

$$\Pr(\boldsymbol{x}) = \frac{1}{Z}\prod_k \prod_c p_c^k(\boldsymbol{x}_c) \tag{2.1}$$

where $\boldsymbol{x}$ is the joint state $(x_1, x_2, \cdots, x_N)$ of an MRF with N nodes, k is the order of the term in the expansion (i.e. k is the number of components of $\boldsymbol{x}$ that the term depends on, which is thus a k-tuple), c labels the particular k-tuple (or k-clique) that the term depends on, $\boldsymbol{x}_c$ is the k-tuple (or clique state), $p_c^k(\boldsymbol{x}_c)$ is the probability factor (or clique factor) associated with $\boldsymbol{x}_c$, and Z is a normalisation factor to ensure that the total probability sums to unity as $\sum_{\boldsymbol{x}} \Pr(\boldsymbol{x}) = 1$, so Z is defined as

$$Z \equiv \sum_{\boldsymbol{x}} \prod_k \prod_c p_c^k(\boldsymbol{x}_c) \tag{2.2}$$

There are some minor technical issues to do with exactly how the states of the $\boldsymbol{x}_c$ are enumerated in the HCE to ensure that states are not double-counted, but these are not important here.

To compute the average $\langle \boldsymbol{S} \rangle$ of a statistic $\boldsymbol{S}(\boldsymbol{x})$ you need to evaluate the following

$$\begin{aligned}\langle \boldsymbol{S} \rangle &= \sum_{\boldsymbol{x}} \Pr(\boldsymbol{x})\, \boldsymbol{S}(\boldsymbol{x}) \\ &= \frac{\sum_{\boldsymbol{x}} \prod_k \prod_c p_c^k(\boldsymbol{x}_c)\, \boldsymbol{S}(\boldsymbol{x})}{\sum_{\boldsymbol{x}} \prod_k \prod_c p_c^k(\boldsymbol{x}_c)}\end{aligned} \tag{2.3}$$

where the probability factor $\Pr(\boldsymbol{x})$ appropriately weights the contribution of each $\boldsymbol{x}$ in the sum, so that overall the correct weighted average $\langle \boldsymbol{S} \rangle$ is computed. Despite the functional simplicity of the HCE expression for $\Pr(\boldsymbol{x})$, it is usually *not* possible to evaluate Equation 2.3 in closed-form, so numerical techniques must be used.

An intuitive feel for how Equation 2.3 can be evaluated can be obtained by noting that the *relative* probability of a pair of joint states $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is given by

$$\frac{\Pr(\boldsymbol{x}_1)}{\Pr(\boldsymbol{x}_2)} = \frac{\prod_k \prod_c p_c^k((\boldsymbol{x}_1)_c)}{\prod_k \prod_c p_c^k((\boldsymbol{x}_2)_c)} \tag{2.4}$$

where the normalising Z factor in Equation 2.1 cancels, and also any factors in common between the numerator and denominator of the ratio in Equation 2.4 will cancel. Thus, if the joint states $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ differ in only a few of their vector components, then any of the probability factors $p_c^k(\boldsymbol{x}_c)$ that do *not* depend on these differing components will cancel out, leaving a relatively simple expression for the ratio $\frac{\Pr(\boldsymbol{x}_1)}{\Pr(\boldsymbol{x}_2)}$. This cancellation is a key property of the functional form of the HCE in Equation 2.1. Once a simple expression for the relative probability $\frac{\Pr(\boldsymbol{x}_1)}{\Pr(\boldsymbol{x}_2)}$ of a pair of joint states $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is available, it can be used to define an MCMC algorithm (see Section II B) for hopping around between the various joint states $\boldsymbol{x}$, and which is designed to visit each joint state with a frequency that is propartional to $\Pr(\boldsymbol{x})$, as is required for computing a numerical estimate of $\langle \boldsymbol{S} \rangle$ in Equation 2.3.

B. Markov Chain Monte Carlo Algorithm

It is possible to construct an MCMC algorithm for hopping between joint states of an MRF that respects their relative probability of occurrence. It is *not* trivially obvious how to design a hopping algorithm with these properties, because one has to consider the net effect of *all* of the ways that one's proposed algorithm can hop in to and out of each state, and to check that this does indeed give rise to the correct joint $\Pr(\boldsymbol{x})$.

Consider a network of nodes whose joint state of its nodes splits into two parts $(\boldsymbol{x}, \boldsymbol{y})$ whose joint probability is $\Pr(\boldsymbol{x}, \boldsymbol{y})$. This joint probability can be split into two parts as

$$\Pr(\boldsymbol{x}, \boldsymbol{y}) = \Pr(\boldsymbol{x}|\boldsymbol{y})\ \Pr(\boldsymbol{y}) \tag{2.5}$$

where $\Pr(\boldsymbol{x}|\boldsymbol{y})$ and $\Pr(\boldsymbol{y})$ are obtained from $\Pr(\boldsymbol{x}, \boldsymbol{y})$ as $\Pr(\boldsymbol{x}|\boldsymbol{y}) \equiv \frac{\Pr(\boldsymbol{x},\boldsymbol{y})}{\sum_{\boldsymbol{x}} \Pr(\boldsymbol{x},\boldsymbol{y})}$ and $\Pr(\boldsymbol{y}) \equiv \sum_{\boldsymbol{x}} \Pr(\boldsymbol{x}, \boldsymbol{y})$. Now update the joint state using $(\boldsymbol{x}, \boldsymbol{y}) \xrightarrow{\Pr(\boldsymbol{x}'|\boldsymbol{y})} (\boldsymbol{x}', \boldsymbol{y})$ where $\boldsymbol{x}'$ is a sample that is drawn from $\Pr(\boldsymbol{x}'|\boldsymbol{y})$, where $\Pr(\boldsymbol{x}'|\boldsymbol{y})$ is a conditional probability that has the *same* dependence on its arguments as $\Pr(\boldsymbol{x}|\boldsymbol{y})$ above. The joint probability $\Pr(\boldsymbol{x}', \boldsymbol{y})$ of the updated joint state is then

$$\Pr(\boldsymbol{x}', \boldsymbol{y}) = \Pr(\boldsymbol{x}'|\boldsymbol{y})\ \Pr(\boldsymbol{y}) \tag{2.6}$$

Comparing Equation 2.5 with Equation 2.6 shows that the new joint probability $\Pr(\boldsymbol{x}', \boldsymbol{y})$ is the *same* function of its arguments as the old joint probability $\Pr(\boldsymbol{x}, \boldsymbol{y})$, by construction. This would *not* be the case if the sample $\boldsymbol{x}'$ was drawn from a $\Pr(\boldsymbol{x}'|\boldsymbol{y})$ that did *not* have the same dependence on its arguments as $\Pr(\boldsymbol{x}|\boldsymbol{y})$ above.

The above argument shows that if you have a network whose joint probability is $\Pr(\boldsymbol{x}, \boldsymbol{y})$, and assuming that the network starts in an initial joint state $(\boldsymbol{x}, \boldsymbol{y})$ that has joint probability $\Pr(\boldsymbol{x}, \boldsymbol{y})$, then updating the joint state using $(\boldsymbol{x}, \boldsymbol{y}) \xrightarrow{\Pr(\boldsymbol{x}'|\boldsymbol{y})} (\boldsymbol{x}', \boldsymbol{y})$ guarantees that the new joint state $(\boldsymbol{x}', \boldsymbol{y})$ has joint probability $\Pr(\boldsymbol{x}', \boldsymbol{y})$ (which has the same dependence on its arguments as $\Pr(\boldsymbol{x}, \boldsymbol{y})$). Thus the joint probability of the joint state of the network nodes maps to itself under the update prescription $(\boldsymbol{x}, \boldsymbol{y}) \xrightarrow{\Pr(\boldsymbol{x}'|\boldsymbol{y})} (\boldsymbol{x}', \boldsymbol{y})$.

Typically, a *sequence* of updates is applied, where the joint state of the network is split into two parts in *different* ways for successive updates, so that eventually all the nodes in the network are visited for updating. The overall effect is that updating causes the network to move around in the joint state space of its nodes, whilst guaranteeing that the joint probability of the network node states stays the same.

On the other hand, if the initial joint state $(\boldsymbol{x}, \boldsymbol{y})$ does *not* have joint probability $\Pr(\boldsymbol{x}, \boldsymbol{y})$, then $\Pr(\boldsymbol{x}', \boldsymbol{y})$ and $\Pr(\boldsymbol{x}, \boldsymbol{y})$ will *not* be the same functions of their arguments, so the joint probability will *change* as the updating scheme is applied. If a sequence of updates (using a variety of splittings of the network of nodes, as described above) is applied then this evolution can converge to a fixed point where the joint probability is *stationary* under updating. However, convergence to a unique fixed point is not actually guaranteed, because an inappropriate update prescription could be used that leads to non-ergodic behaviour where the whole joint state space is not explored, for instance. However, in practical problems with soft joint probabilities convergence usually occurs.

In an MRF the ratio of conditional probabilities $\frac{\Pr(\boldsymbol{x}'_1|\boldsymbol{y})}{\Pr(\boldsymbol{x}'_2|\boldsymbol{y})}$ that is used to generate the MCMC updates $(\boldsymbol{x}, \boldsymbol{y}) \xrightarrow{\Pr(\boldsymbol{x}'|\boldsymbol{y})} (\boldsymbol{x}', \boldsymbol{y})$ is given in Equation 2.4. If the joint states $\boldsymbol{x}'_1$ and $\boldsymbol{x}'_2$ differ in only a few of their vector components, then there is a lot of cancellation in $\frac{\Pr(\boldsymbol{x}'_1|\boldsymbol{y})}{\Pr(\boldsymbol{x}'_2|\boldsymbol{y})}$ so the fully simplified expression for $\frac{\Pr(\boldsymbol{x}'_1|\boldsymbol{y})}{\Pr(\boldsymbol{x}'_2|\boldsymbol{y})}$ is relatively simple. This is what makes MCMC algorithms so appropriate for MRF networks.

C. Inference Using an MRF

Image processing is an area where MRFs have proved to be particularly useful [4]. The starting point is to define an MRF model of the joint probability $\Pr(\boldsymbol{x})$ of the image pixels

$$\begin{aligned} \Pr(\boldsymbol{x}) &\equiv \sum_{\boldsymbol{y}} \Pr(\boldsymbol{x}, \boldsymbol{y}) \\ \Pr(\boldsymbol{x}, \boldsymbol{y}) &= \Pr(\boldsymbol{x}|\boldsymbol{y})\ \Pr(\boldsymbol{y}) \end{aligned} \tag{2.7}$$

where $\Pr(\boldsymbol{x})$ is expressed as the marginal probability of $\Pr(\boldsymbol{x}, \boldsymbol{y})$ after the hidden variables $\boldsymbol{y}$ have been averaged over, and both $\Pr(\boldsymbol{x}|\boldsymbol{y})$ and $\Pr(\boldsymbol{y})$ may be written as products of factors using the HCE in Equation 2.1. The hidden variables $\boldsymbol{y}$ are the unobserved causes that determine the values of the image pixels $\boldsymbol{x}$, and are thus the causal factors that are used to construct a generative model of the image. This generative model can be multi-layered with several levels of hidden variables.

To compute the probability of the joint state of the hidden variables $\boldsymbol{y}$ given an observation of the image pixel values $\boldsymbol{x}$ the posterior probability $\Pr(\boldsymbol{y}|\boldsymbol{x})$ must be used, which may be obtained using Bayes theorem as

$$\Pr(\boldsymbol{y}|\boldsymbol{x}) = \frac{\Pr(\boldsymbol{x}|\boldsymbol{y})\ \Pr(\boldsymbol{y})}{\sum_{\boldsymbol{y}} \Pr(\boldsymbol{x}|\boldsymbol{y})\ \Pr(\boldsymbol{y})} \tag{2.8}$$

An MCMC algorithm (see Section II B) can then be used to draw samples from $\Pr(\boldsymbol{y}|\boldsymbol{x})$. Note that successive samples produced by the MCMC algorithm are strongly correlated with each other because the MCMC algorithm has a finite memory time; this makes MCMC run times (for a given size of error bar) much longer than would be the case if the samples could be somehow independently drawn from $\Pr(\boldsymbol{y}|\boldsymbol{x})$.

Also, if $\Pr(\boldsymbol{y}|\boldsymbol{x})$ has a *single* well-defined peak of probability, then the MCMC algorithm can be used to locate this, usually with the assistance of a simulated annealing algorithm to "soften" $\Pr(\boldsymbol{y}|\boldsymbol{x})$ during the early stages of the algorithm, and then MCMC fluctuations about this peak can be observed in order to deduce the robustness of the solution.

Typically, in image processing applications there is a single overwhelmingly likely hidden variables interpretation of the image pixels (i.e. $\Pr(\boldsymbol{y}|\boldsymbol{x})$ has a *single* well-defined peak of probability). However, the above approach gracefully (and consistently) degrades when the interpretation is ambiguous (i.e. $\Pr(\boldsymbol{y}|\boldsymbol{x})$ does *not* have a single well-defined peak of probability). This graceful degradation in the face of ambiguity is one of the strengths of the Bayesian approach.

D. Multiply Occupied States

It is useful to develop a concrete way of visualising the hopping processes that underlie the MCMC algorithm described in Section II B. This is a prerequisite for the generalisation of MCMC algorithms that developed later in Section III.

The state $\boldsymbol{x}$ of an N-node MRF is $\boldsymbol{x} \equiv (x_1, x_2, \cdots, x_N)$, and for a given $\boldsymbol{x}$ each of its components x_i lives in *one* of an assumed finite number m of states that are available to x_i, where for simplicity we assume that all the x_i have the *same* number of states m. One way of representing each x_i is as an m-component vector $(0, 0, \cdots, 0, 1, 0, \cdots, 0, 0)$, where the "1" identifies which of the m states x_i happens to have. This representation is essentially a histogram with m bins, with a *single* sample occupying one of the bins. The whole state of the N component $\boldsymbol{x}$ vector is then represented by N such histograms, each with a *single* "1" placed in the appropriate bin to identify the state of *all* of the x_i for $i = 1, 2, \cdots, N$. Naturally, this use of histograms is an exceedingly wasteful coding of the state $\boldsymbol{x}$ because it consists mostly of "0" entries. However, it *does* allow the hopping operations that are generated by the MCMC algorithm to be represented directly as operations in which each "1" hops around between the bins of its histogram. More importantly, this representation of the MRF state is suitable for the generalisation in Section III where each histogram will have *multiple* samples occupying its bins (i.e. multiple states will be recorded at each MRF node). This is discussed in more detail below.

Figure 1 shows a Markov chain with 7 nodes (i.e. $N = 7$), each of which has 7 possible states (i.e. $m = 7$). The state space of each node is represented by one of the rectangles, the particular bin that is occupied by a sample is shown as a blob (the unoccupied bins are shown as dots), and the particular 2-clique interactions (see Equation 2.1) that are activated by the occupied node states are shown as bold lines.

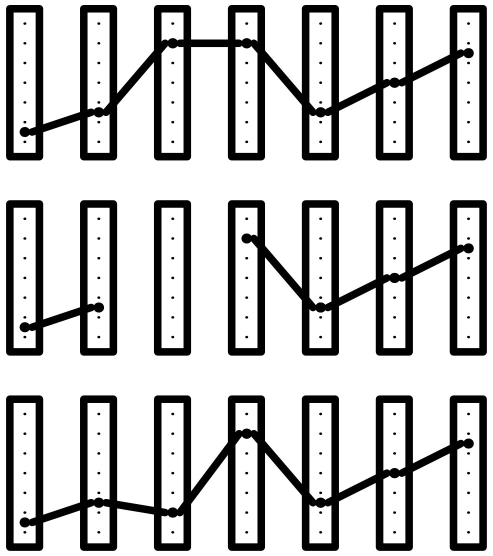


Figure 1: Steps of an MCMC update of a Markov chain with $N = 7$ and $m = 7$.

1. The top row of Figure 1 shows a random initial state of the Markov chain.

2. The middle row of Figure 1 shows that the sample in node 3 has been annihilated. This is the *first* step of an MCMC update, in which a node is chosen at random and its state is erased.

3. The bottom row of Figure 1 shows that a sample in node 3 has been created. This is the *second* step of an MCMC update, in which a sample is created in node 3 whose state was previously erased in step 2 above. The influence of the neighbouring nodes is used to probabilistically determine the state in which to create the sample, as described in Section II B.

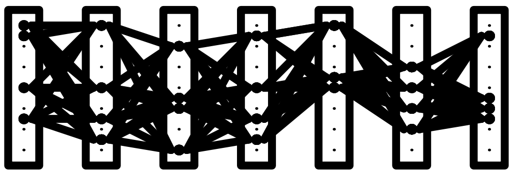


Figure 2: Multiply occupied Markov chain showing a random state.

The histogram representation allows generalisations of the MCMC algorithm in which each MRF node is occupied by more than one sample, when it is said to be multiply occupied. Figure 2 shows an example of this type of MRF state.

It is important not to confuse multiply occupied states with other uses of state space:

1. Histograms with more than one sample are *not* the same as ensembles of histograms each with one sample. This is because the former allow for the possibility that the MCMC algorithm can cause the samples to interact with each other, whereas the latter is a means of running multiple standard MCMC algorithms in parallel.

2. Histograms with more than one sample could be viewed as having a single "super"-state that recorded as a single state the entire contents of the histogram bins, which would disguise the fact that the histogram was actually constructed out of samples occupying the histogram bins. The higher level super-state description is mathematically equivalent to the lower-level description in terms of individual samples, but it does *not* allow the development of detailed MCMC algorithms. We prefer to view the higher level super-state description as an interpretation that is used *after* the lower level details have been worked out using the techniques that are presented in this paper.

In Figure 2 the histogram associated with each node contains more than one sample. Such multiple occupancy was *not* present in the basic MRF theory of Section II A, so the detailed form of the MCMC algorithm of Section II B must now be generalised. Multiple occupancy is explored in detail in Section III using creation and annihilation operator techniques to hop samples between histogram bins, which is achieved by annihilating a sample from one bin and creatng a sample in another bin, as illustrated in Figure 1.

When more than one sample per histogram is allowed then various new types of processing become possible:

1. The number of samples per histogram can be varied with time. This requires birth and death rules as well as migration (or hopping) rules for the histogram samples. In this case the creation and annihilation operators would be applied in ways that do *not* enforce conservation of the number of samples in each histogram, so annihilation without subsequent creation (and vice versa) are permitted operations. This is how "reversible jump" MCMC algorithms [5] might be implemented using creation and annihilation operators.

2. The samples can interact with each other in complicated ways to form "bound states", which would then behave like higher level "symbols" (i.e. sets of interacting histogram samples) that are constructed out of "sub-symbols" (i.e. the histogram samples themselves). This is illustrated in Figure 3, Figure 4 and Figure 5 below.

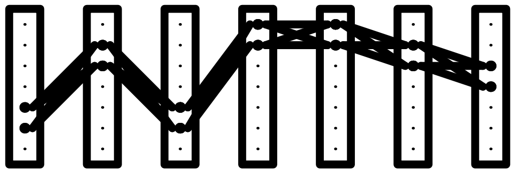


Figure 3: Multiply occupied Markov chain showing a tube-like joint state.

Figure 3 shows a multiple-sample version of Figure 1 that is more highly structured than the example shown in Figure 2. For illustrative purposes, the samples are now assumed to be in neighbouring states at each node rather than spread out at random; typically this would be the case for Markov chains whose properties are optimised to encode information in a topographically ordered way. The 2-cliques that then contribute typically form the tube-like joint state of activated 2-cliques shown in Figure 3.

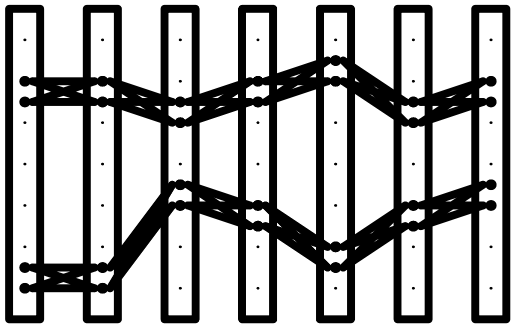


Figure 4: Multiply occupied Markov chain showing two parallel tube-like joint states.

Figure 4 shows another possibility that can arise with multiple sample occupancy, where the occupancy of each node splits into two separate clusters of samples, *and* where the probability factors associated with the 2-cliques is such that only node states that are both in the top half of the diagram are connected (and similarly for the bottom half of the diagram), so that there are no activated 2-cliques running between the top and bottom halves of the diagram (or at least the contribution of these is negligible). Effectively, this multiply occupied Markov chain has two completely independent Markov chains embedded within it, each of which has its own tube-like joint state of activated 2-cliques. This type of structure emerges in multiply occupied Markov chains that have a *limited* number of states available to each node of the chain, and which are optimised to encode information topographically (which ensures that the tube-like joint states are localised in the node state spaces).

This type of behaviour emerges when SON training methods are used, but it will not be discussed further in this paper.

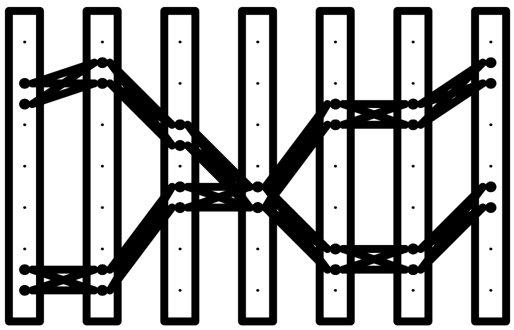


Figure 5: Multiply occupied Markov chain showing two parallel "tube" states bound together.

Figure 5 shows how Figure 4 can be modified if the two tube-like joint states have some node states in common, which binds the tubes together. An extreme version of this binding between tubes can occur if the situation is as shown in Figure 4, but *additionally* there are some weak interactions between the tubes.

III. OPERATOR IMPLEMENTATION OF MCMC ALGORITHMS

The aim of this section is to present a theoretical framework for expressing MCMC algorithms, which is based on operators that have very simple algebraic properties, but which is nevertheless sufficiently flexible that it allows a large class of MCMC-like algorithms to be represented.

Section III A gives some background material that motivates the use of MCMC algorithms as the primary means of building dynamical models for discrete networks. Section III B introduces creation and annihilation operators for manipulating samples in multiply occupied network nodes. Section III C uses these basic operators to construct a composite operator for generating MCMC updates, Finally, Section III D summarises a diagrammatic representation of MCMC algorithms.

A. Background

The aim here is to rewrite the MCMC algorithm for running an MRF (see Section II B) using operator algebra. This will allow the algorithm to be run in state spaces where the basic MCMC algorithm has not previously been used, and will thus generalise the algorithm. Throughout this section the emphasis is on using the MCMC algorithm as the *starting point* for deducing the properties of an MRF, so the MRF is viewed as corresponding to the equilibrium behaviour of a (stochastic) discrete-time dynamical system. Hitherto, the MCMC algorithm could be viewed as an artefact of a particular way of sampling from an MRF, but here it is viewed as the way in which the MRF actually behaves. This moves slightly away from the original motivation for using MRFs to model and manipulate joint probabilities for use in Bayesian calculations (see Section I), but this change of emphasis allows full advantage to taken of the flexibility of the MCMC approach, and in particular its generalisation to multiply occupied states.

This jump to using discrete-time dynamical systems as the starting point for building models allows a much larger class of behaviours to be explored, including ones that do *not* have a corresponding HCE representation of the equilibrium behaviour (i.e. as a simple product of probability factors, as in Equation 2.1), or do *not* have a steady state equilibrium behaviour at all (e.g. a limit *cycle* rather than a limit *point*, etc).

The MCMC approach models everything as part of a dynamical evolution process, where a static statistical model of the world is obtained by taking a snapshot of the evolution of the dynamical system. Those who insist on starting from a fixed graphical model based on the HCE (or a set of such models) might be disappointed that this is *not* the starting point that is used here. However, they should note that the underlying process that generates their graphical model in the first place is actually dynamical, and that their model merely describes the statistical properties through a time slice of this dynamical process; in other words, their model describes only a *marginal* distribution. For instance, an MRF image model does *not* attempt to model the history of the dynamical processes that cause the (hidden) objects to eventually give rise to the observed pixel values. Analogously, *all* MRFs derive from a hidden dynamical process.

The results presented in this section make use of creation and annihilation operator techniques to generate the hopping processes that underlie MCMC algorithms, which allows MCMC algorithms to be written using a very compact notation. These operator techniques will be familiar to physicists who use quantum field theory (QFT) [9], and for the convenience of physicists the notation used here is the same as is used in QFT. Generally, creation and annihilation operators can be used to generate birth and death processes (respectively), which thus increase and decrease the dimensionality of the state space (respectively), so this approach naturally lends itself to describing processes that correspond to "reversible jump" MCMC algorithms [5].

B. Creation and Annihilation Operators

In this section the mathematical development of the properties of creation and annihilation operators is deliberately presented in an informal way, by expressing it in terms of operations on the samples occupying histogram bins. This is to encourage a concrete and intuitive understanding of how these operators act on samples, rather than to merely think of them as objects that have particular algebraic properties. To a physicist who is familiar with the use of these techniques in QFT, the explanations will appear to be very long-winded and the derivations very cavalier, and to them we apologise.

1. *Multiply Occupied States*

The multiply occupied states described in Section II D can be manipulated by suitably defined creation and annihilation operators.

Multiply occupied states can viewed as hsistograms with multiple samples occupying the histogram bins. These histograms can be represented thus:

1. Empty histogram: $|0\rangle$. This represents the bins (an indeterminate number of them) of a histogram with no samples in any of the bins. The notation $|0\rangle$ has been chosen to correspond exactly to the "vacuum" state as used by physicists; it represents the background in which we will create and annihilate histogram samples (or particles).

2. Histogram with one sample in bin i: ${a_i}^\dagger\ |0\rangle$. The $|0\rangle$ represents the empty histogram (as defined above), and the creation operator ${a_i}^\dagger$ acting from the left represents the action of creating one sample in bin i of the empty histogram. The notation ${a_i}^\dagger$ has been chosen to correspond exactly to the operator for creating a particle in state i as used by physicists, and the notation ${a_i}^\dagger\ |0\rangle$ corresponds exactly to the notation for a single particle in state i. The use of the dagger notation $\dagger$ (i.e. adjoint operator) is chosen to make our notation compatible with that used in QFT [9], which will be discussed in more detail in Section III B 8.

3. Histogram with n_i samples in bin i : $\left({a_i}^\dagger\right)^{n_i}\ |0\rangle$. This is a multiply occupied histogram, which is obtained by operating on the empty histogram $|0\rangle$ multiple times with the creation operator ${a_i}^\dagger$.

4. Histogram with n_i samples in bin i (for $i = 1, 2, \cdots, m$): $\prod_{i=1}^{m}\left({a_i}^\dagger\right)^{n_i}\ |0\rangle$. This is a straightforward generalisation of the above, where creation operators are applied multiple times to all of the histogram bins.

The above representation of histogram states does *not* provide a means for freely manipulating them. In order to be able to do this it is necessary to be able to annihilate samples as well as create them as above.

2. *Creation and Annihilation Operators*

The annihilation operations discussed below may be achieved by using the annihilation operator a_i which is the adjoint of the creation operator ${a_i}^\dagger$. See the discussion on adjoint operators in Section III B 8 for more details on why the creation operator ${a_i}^\dagger$ and annihilation operator a_i are adjoints of each other. Note that in the description immediately below the behaviour of ${a_i}^\dagger$ and a_i corresponds to our intuitive notion of how these operators should behave, rather than formally derived from their algebraic properties which are presented later on in Section III B 3.

Annihilating a sample from an empty histogram erases the state space itself. This simply *defines* what happens when you try to remove a sample from an already empty histogram, which is very useful for cleaning up algebraic expressions involving a_i and $|0\rangle$. In effect, this defines the "vacuum" $|0\rangle$ as the reference state for determining the occupancy of each histogram bin.

$$a_i\ |0\rangle = 0 \tag{3.1}$$

which can be represented for a 4-bin histogram for any i as

$$(0,0,0,0) \xrightarrow{a_i} 0 \tag{3.2}$$

Annihilating a sample from a 1-sample histogram leaves an empty histogram. This definition is the common-sense notion of what should happen when you create a sample in a histogram bin, then annihilate it again. Thus

$$a_i\, {a_i}^\dagger\ |0\rangle = |0\rangle \tag{3.3}$$

which can be represented for a 4-bin histogram and for $i = 3$ as

$$(0,0,0,0) \xrightarrow{{a_i}^\dagger} (0,0,1,0) \xrightarrow{a_i} (0,0,0,0) \tag{3.4}$$

Annihilating the *wrong* sample (i.e. $j \neq i$) from a 1-sample histogram erases the state space itself. This is a generalisation of Equation 3.1 in which the histogram already contains one sample, but it is in a *different* bin from the one from which we are trying to remove a sample.

$$a_j\, {a_i}^\dagger\ |0\rangle = 0 \qquad j \neq i \tag{3.5}$$

which can be represented for a 4-bin histogram and for $i = 3$ and $j \neq i$ as

$$(0,0,0,0) \xrightarrow{{a_i}^\dagger} (0,0,1,0) \xrightarrow{a_j} 0 \tag{3.6}$$

Equation 3.4 and Equation 3.6 can now be combined to give (the illustration shows the $i = 3$ case)

$$(0,0,0,0) \xrightarrow{{a_i}^\dagger} (0,0,1,0) \xrightarrow{a_j} \begin{array}{ll} (0,0,0,0) & \qquad j = i \\ 0 & \qquad j \neq i \end{array} \tag{3.7}$$

If the location of the occupied bin is unknown, yet you want to be certain that you annihilate the sample, then you have to attempt to annihilate a sample from every one of the histogram bins. This combines the properties of both Equation 3.3 and Equation 3.5. Note that $|0\rangle$ (the empty histogram) is different from 0 (no histogram at all, i.e. not even an empty one).

$$\begin{aligned}\left(\sum_{j=1}^{m} a_j\right) a_i{}^{\dagger}\,|0\rangle &= a_1 a_i{}^{\dagger}\,|0\rangle + a_2 a_i{}^{\dagger}\,|0\rangle + \cdots + a_i a_i{}^{\dagger}\,|0\rangle + \cdots + a_m a_i{}^{\dagger}\,|0\rangle \\ &= 0 + 0 + \cdots + 0 + |0\rangle + 0 + \cdots + 0 \\ &= |0\rangle \end{aligned} \tag{3.8}$$

which can be represented for a 4-bin histogram and for $i = 3$ as

$$(0,0,0,0) \xrightarrow{a_i{}^{\dagger}} (0,0,1,0) \xrightarrow{\sum_{j=1}^{m} a_j} (0,0,0,0) \tag{3.9}$$

Annihilating a sample from a 2-sample histogram (samples in *different* bins, i.e. $i_1 \neq i_2$) leaves two 1-sample histograms. This is a generalisation of Equation 3.8 in which the histogram starts with *two* samples (known to be in different bins) rather than *one* sample.

$$\begin{aligned}\left(\sum_{j=1}^{m} a_j\right) a_{i_1}{}^{\dagger} a_{i_2}{}^{\dagger}\,|0\rangle &= a_1\, a_{i_1}{}^{\dagger} a_{i_2}{}^{\dagger}\,|0\rangle + \cdots + a_{i_i}\, a_{i_1}{}^{\dagger} a_{i_2}{}^{\dagger}\,|0\rangle + \cdots\cdots + a_{i_2}\, a_{i_1}{}^{\dagger} a_{i_2}{}^{\dagger}\,|0\rangle + \cdots + a_m\, a_{i_1}{}^{\dagger} a_{i_2}{}^{\dagger}\,|0\rangle \\ &= 0 + \cdots + 0 + a_{i_2}{}^{\dagger}\,|0\rangle + 0 + \cdots + 0 + a_{i_1}{}^{\dagger}\,|0\rangle + 0 + \cdots + 0 \\ &= a_{i_1}{}^{\dagger}\,|0\rangle + a_{i_2}{}^{\dagger}\,|0\rangle \end{aligned} \tag{3.10}$$

which can be represented for a 4-bin histogram and for $(i_1, i_2) = (1, 3)$ as

$$(0,0,0,0) \xrightarrow{a_{i_1}{}^{\dagger}} (1,0,0,0) \xrightarrow{a_{i_2}{}^{\dagger}} (1,0,1,0) \xrightarrow{\sum_{j=1}^{m} a_j} \begin{matrix}(1,0,0,0)\\ + \\ (0,0,1,0)\end{matrix} \tag{3.11}$$

Annihilating a sample from a 2-sample histogram (samples in the *same* bin, i.e. $i_1 = i_2 = i$) leaves two copies of the same 1-sample histogram (because either of the two samples can be annihilated to leave one sample). This is a variation of Equation 3.10, and it is the *first* example of attempting to annihilate a sample from a bin that has more than one sample in it. The number of ways of annihilating a sample from a multiply occupied bin is equal to the number of samples in the bin.

$$\begin{aligned}\left(\sum_{j=1}^{m} a_j\right) \left(a_i{}^{\dagger}\right)^2 |0\rangle &= a_1 \left(a_i{}^{\dagger}\right)^2 |0\rangle + a_2 \left(a_i{}^{\dagger}\right)^2 |0\rangle + \cdots + a_i \left(a_i{}^{\dagger}\right)^2 |0\rangle + \cdots + a_m \left(a_i{}^{\dagger}\right)^2 |0\rangle \\ &= 0 + 0 + \cdots + 0 + 2a_i{}^{\dagger}\,|0\rangle + 0 + \cdots + 0 \\ &= 2a_i{}^{\dagger}\,|0\rangle \end{aligned} \tag{3.12}$$

which can be represented for a 4-bin histogram and for $i_1 = 1$ as

$$(0,0,0,0) \xrightarrow{a_{i_1}{}^{\dagger}} (1,0,0,0) \xrightarrow{a_{i_1}{}^{\dagger}} (2,0,0,0) \xrightarrow{\sum_{j=1}^{m} a_j} \begin{matrix}(1,0,0,0)\\ + \\ (1,0,0,0)\end{matrix} \tag{3.13}$$

3. *Creation and Annihilation Operator Commutation Relations*

Now that some of the required properties of creation and annihilation operators have been established, we are in a position to guess what their general algebraic properties should be, so that we can do arbitrarily complicated operator manipulations on states of arbitrary occupany.

All of the above behaviour of creation and annihilation operators (apart from $a_i|0\rangle = 0$ in Equation 3.1) can be summarised in the following commutation relations

$$\begin{aligned} a_i\, {a_j}^\dagger - {a_j}^\dagger\, a_i &= \delta_{i,j} \\ a_i\, a_j - a_j\, a_i &= 0 \\ {a_i}^\dagger\, {a_j}^\dagger - {a_j}^\dagger\, {a_i}^\dagger &= 0 \end{aligned} \tag{3.14}$$

where $\delta_{i,j}$ is a Kronecker delta ($\delta_{i,j} = 1$ if $i = j$, and $\delta_{i,j} = 0$ if $i \neq j$). These commutation relations are usually written in shorthand notation as

$$\begin{aligned} \left[a_i,\, {a_j}^\dagger\right] &= \delta_{i,j} \\ \left[a_i,\, a_j\right] &= 0 \\ \left[{a_i}^\dagger,\, {a_j}^\dagger\right] &= 0 \end{aligned} \tag{3.15}$$

The $[a_i,\, a_j] = 0$ and $[{a_i}^\dagger,\, {a_j}^\dagger] = 0$ commutation relations follow from the fact that a sequence consisting solely of annihilation operators (or solely of creation operators) has the same effect whatever the order in which the operators appear in the sequence. However, this order independence property vanishes when the sequence contains interleaved creation and annihilation operators, as will be explained below.

The $[a_i,\, {a_j}^\dagger] = \delta_{i,j}$ commutation relation may be illustrated for a 4-bin *empty* histogram and for $j = 3$ as

$$\begin{array}{ccccccl} (0,0,0,0) & \xrightarrow{{a_j}^\dagger} & (0,0,1,0) & \xrightarrow{a_i} & \begin{array}{c}(0,0,0,0)\\ 0\end{array} & \begin{array}{c} i=j \\ i\neq j\end{array} \\ (0,0,0,0) & \xrightarrow{a_i} & 0 & \xrightarrow{{a_j}^\dagger} & 0 & \end{array} \tag{3.16}$$

and for the general histogram as

$$\begin{array}{ccccccl} (n_1,n_2,\cdots) & \xrightarrow{{a_j}^\dagger} & (n_1,n_2,\cdots,n_j+1,\cdots) & \xrightarrow{a_i} & \begin{array}{l}(n_i+1)\,(n_1,n_2,\cdots)\\ n_i\,(n_1,\cdots,n_i-1,\cdots,n_j+1,\cdots)\end{array} & \begin{array}{c} i=j \\ i\neq j\end{array} \\ (n_1,n_2,\cdots) & \xrightarrow{a_i} & n_i\,(n_1,\cdots,n_i-1,\cdots) & \xrightarrow{{a_j}^\dagger} & \begin{array}{l}n_i\,(n_1,n_2,\cdots)\\ n_i\,(n_1,\cdots,n_i-1,\cdots,n_j+1,\cdots)\end{array} & \begin{array}{c} i=j \\ i\neq j\end{array} \end{array} \tag{3.17}$$

and by taking the difference of the $a_i\, {a_j}^\dagger$ (i.e. the first line in Equation 3.17 above) and the ${a_j}^\dagger\, a_i$ (i.e. the second line in Equation 3.17 above) results above the commutator relation $[a_i,\, {a_j}^\dagger] = \delta_{i,j}$ is correctly verified. The key result is the $i = j$ case in Equation 3.17 which has a factor $n_i + 1$ in the $a_i\, {a_j}^\dagger$ case and a factor n_i in the ${a_j}^\dagger\, a_i$ case, which arises because the number of ways of annihilating a sample is equal to the number of samples in the histogram bin which the annihilation operator acts upon, and this number is *one greater* in the case where a creation operator got to act on the bin *before* the annihilation operator got its chance to act on the same bin.

Note that the commutation relation in Equation 3.15 *extends* the properties of the creation and annihilation operators independently of the states that they act upon, so that the operators now have specific effects on histograms with multiple samples in multiple bins; these extended properties were not specified in the development up as far as Equation 3.13. Thus the particular choice of commutation relation in Equation 3.15 defines a specific set of combinatoric factors for how one can select samples for creation and annihilation, which are described above and which have intuitively reasonable properties.

The above properties of the creation and annihilation operators have been justified by appealing to simple operations on the samples in histogram bins, which leads automatically these operators having the same combinatoric properties as the creation and annihilation operators that are used in a QFT of bosons [9].

4. *Commutation Relations Generalise MCMC Algorithms*

In Section III B 3 a set of commutation relations was defined based on the required properties of the creation and annihilation operators in a variety of simple cases that were discussed in Section III B 2. However, these commutation relations do more than just summarise these special cases, they extend the use of creation and annihilation operators to *all* situations, including cases where the histogram bins are occcupied by an arbitrary number of samples. Thus these commutation relations provide an algebraically simple route to generalisation of MCMC algorithms. No doubt there are other generalisations of the standard MCMC algorithm, but none of them will have the algebraic simplicity of the properties defined in Section III B 3.

For instance, consider the multiply occupied state $({a_1}^\dagger)^{n_1} \cdots ({a_m}^\dagger)^{n_m}\, |0\rangle$. As in QFT [9], the creation operators can be used to construct a Fock space of states with all possible occupancies, and this Fock space can be explored by applying creation and annihilation operators. This type of exploration corresponds to what is done in reversible jump MCMC algorithms [5], where the scope of MCMC updates is extended so that they sample from various models, in additional to the sampling within a single model that usually occurs.

It can be seen that the effect of $\sum_{j=1}^{m} a_j$ is to count the number of samples in each histogram bin (i.e. the number of ways of annihilating a sample from a bin is equal to the number of samples in the bin), and to also annihilate one of the samples from each bin, as is shown in Equation 3.18.

$$\begin{aligned}\left(\sum_{j=1}^{m} a_j\right) \left({a_1}^\dagger\right)^{n_1} \left({a_2}^\dagger\right)^{n_2} \cdots \left({a_m}^\dagger\right)^{n_m} |0\rangle \;=\; & n_1 \left({a_1}^\dagger\right)^{n_1-1} \left({a_2}^\dagger\right)^{n_2} \cdots \left({a_m}^\dagger\right)^{n_m} |0\rangle \\ & + n_2 \left({a_1}^\dagger\right)^{n_1} \left({a_2}^\dagger\right)^{n_2-1} \cdots \left({a_m}^\dagger\right)^{n_m} |0\rangle \\ & \vdots \\ & + n_m \left({a_1}^\dagger\right)^{n_1} \left({a_2}^\dagger\right)^{n_2} \cdots \left({a_m}^\dagger\right)^{n_m-1} |0\rangle \end{aligned} \tag{3.18}$$

The above deficit of one sample after the application of $\sum_{j=1}^{m} a_j$ can be rectified by altering the operator as $\sum_{j=1}^{m} a_j \longrightarrow \sum_{j=1}^{m} {a_j}^\dagger a_j$, because the inclusion of ${a_j}^\dagger$ to the left of a_j ensures that a sample will be created in bin j to make up for the one that a_j annihilated. Note that there is only *one* way of creating a sample in a bin, but there are as many ways of annihilating a sample as there are samples in the bin.

The result in Equation 3.18 can be summarised as follows for $n \geq 1$ (note that the r.h.s. is 0 for $n = 0$)

$$a_i \left({a_j}^\dagger\right)^n |0\rangle = n\, \delta_{i,j} \left({a_j}^\dagger\right)^{n-1} |0\rangle \tag{3.19}$$

which can be represented for a 4-bin histogram and for $j = 3$ as

$$(0,0,0,0) \xrightarrow{\left({a_j}^\dagger\right)^n} (0,0,n,0) \xrightarrow{a_i} \begin{array}{ll} n\,(0,0,n-1,0) & i = j \\ 0 & i \neq j \end{array} \tag{3.20}$$

This result may be used in general to move annihilation operators to the right of all creation operators. The result in Equation 3.19 is easily proved by using $[a_i,\, {a_j}^\dagger] = \delta_{i,j}$ to progressively move a_i to the right through one ${a_j}^\dagger$ at a time, and then using $a_i\,|0\rangle = 0$ to discard any terms that contain $a_i\,|0\rangle$.

5. *Doing Calculations with Creation and Annihilation Operators*

Using explicit notation (e.g. $(0,0,0,0) \xrightarrow{{a_i}^\dagger} (0,0,1,0)$) for what the creation and annihilation operators are doing to the samples in the histogram bins is very tedious in cases that are not much more complicated than the ones discussed above. The purpose of introducing creation and annihilation operators is to replace the manipulation of histogram samples by algebraic manipulations based on the properties $a_i\,|0\rangle = 0$ and $[a_i,\, {a_j}^\dagger] = \delta_{i,j}$, which also has the desirable side effect that the calculations can be completely automated by using symbolic algebra techniques. In general, explicit notation should be needed only to verify what is being done to the samples in the histograms, and to check that this corresponds to what was intended.

From a theoretical point of view the commutation relations in Equation 3.15 are an algebraic way of doing the book-keeping to keep track of how creation and annihilation operators construct and modify histogram states depending on the order in which the operators are applied. The $[a_i,\, {a_j}^\dagger] = \delta_{i,j}$ commutation relation can be written in the form $a_i\, {a_j}^\dagger = {a_j}^\dagger\, a_i + \delta_{i,j}$, which can then used to replace $a_i\, {a_j}^\dagger$ by ${a_j}^\dagger\, a_i + \delta_{i,j}$, which effectively moves the annihilation operator to the right (giving the ${a_j}^\dagger\, a_i$ term) whilst picking up a commutator (the $\delta_{i,j}$ term) as a side effect. This says that annihilation after creation (i.e. $a_i\, {a_j}^\dagger$) is the same as annihilation before creation (i.e. ${a_j}^\dagger\, a_i$), except for when the operators are applied to the same bin, which triggers the appearance of the $\delta_{i,j}$ term for reasons discussed above.

As a manual exercise, it can be verified that operators with the above properties (i.e. $a_i\,|0\rangle = 0$ and $[a_i,\, {a_j}^\dagger] = \delta_{i,j}$) correctly annihilate a sample from a 2-sample histogram (samples in any bins); this generalises Equation 3.12 to the case where the bins are *not* assumed to be the same. The strategy in this derivation (and in all other derivations using creation and annihilation operators) is to move the annihilation operators to the right of all the creation operators (using $a_i\, {a_j}^\dagger = {a_j}^\dagger\, a_i + \delta_{i,j}$), thus generating a sum of terms of the form $(a^\dagger\, a^\dagger\, a^\dagger\, a^\dagger \cdots)(a\, a\, a\, a \cdots)\,|0\rangle$, and wherever there is a non-zero number of annihilation operators acting on $|0\rangle$ the term may be removed (using $a_i\,|0\rangle = 0$). This leaves a sum of terms that contain only creation operators acting on $|0\rangle$.

The detailed derivation of the effect of applying $\sum_{j=1}^{m} a_j$ to ${a_{i_1}}^\dagger {a_{i_2}}^\dagger |0\rangle$ is as follows

$$\begin{aligned}
\left(\sum_{j=1}^{m} a_j\right) {a_{i_1}}^\dagger {a_{i_2}}^\dagger |0\rangle &= \left(\sum_{j=1}^{m} a_j {a_{i_1}}^\dagger\right) {a_{i_2}}^\dagger |0\rangle \\
&= \sum_{j=1}^{m} \left({a_{i_1}}^\dagger a_j + \delta_{i_1,j}\right) {a_{i_2}}^\dagger |0\rangle \\
&= \sum_{j=1}^{m} \left({a_{i_1}}^\dagger (a_j {a_{i_2}}^\dagger) + \delta_{i_1,j} {a_{i_2}}^\dagger\right) |0\rangle \\
&= \sum_{j=1}^{m} \left({a_{i_1}}^\dagger ({a_{i_2}}^\dagger a_j + \delta_{i_2,j}) + \delta_{i_1,j} {a_{i_2}}^\dagger\right) |0\rangle \\
&= \sum_{j=1}^{m} \left({a_{i_1}}^\dagger {a_{i_2}}^\dagger a_j + \delta_{i_2,j} {a_{i_1}}^\dagger + \delta_{i_1,j} {a_{i_2}}^\dagger\right) |0\rangle \\
&= \sum_{j=1}^{m} \left({a_{i_1}}^\dagger {a_{i_2}}^\dagger (a_j |0\rangle) + \delta_{i_2,j} ({a_{i_1}}^\dagger |0\rangle) + \delta_{i_1,j} ({a_{i_2}}^\dagger |0\rangle)\right) \\
&= {a_{i_1}}^\dagger |0\rangle + {a_{i_2}}^\dagger |0\rangle
\end{aligned} \quad (3.21)$$

After this sort of manipulation has been done a few times it is not necessary to write down all of the intermediate steps as above, because the manipulations have a very simple form where each annihilation operator a_i is moved freely to the right, except that whenever it passes through a corresponding creation operator ${a_j}^\dagger$ an additional term is created (i.e. the $\delta_{i,j}$ commutator term). In more complicated cases it is more convenient to replace manual manipulations with symbolic manipulations.

6. Number Operator

The above results (e.g. see Equation 3.18) allow the definition of a *number operator* $\mathcal{N}$ that counts the total number of samples in the histogram. Thus

$$\mathcal{N} \equiv \sum_{i=1}^{m} {a_i}^\dagger a_i \quad (3.22)$$

This gives

$$\mathcal{N} \left({a_1}^\dagger\right)^{n_1} \left({a_2}^\dagger\right)^{n_2} \cdots \left({a_m}^\dagger\right)^{n_m} |0\rangle = (n_1 + n_2 + \cdots + n_m) \left({a_1}^\dagger\right)^{n_1} \left({a_2}^\dagger\right)^{n_2} \cdots \left({a_m}^\dagger\right)^{n_m} |0\rangle \quad (3.23)$$

where the *total* number of histogram samples $n \equiv n_1 + n_2 + \cdots + n_m$ is the quantity that is measured by applying $\mathcal{N}$. For instance, $\mathcal{N} \left({a_j}^\dagger\right)^{n_j} |0\rangle$ can be represented for a 4-bin histogram and for $j = 3$ as

$$(0,0,0,0) \xrightarrow{\left({a_j}^\dagger\right)^{n_j}} (0,0,n_j,0) \xrightarrow{\mathcal{N}} n_j \ (0,0,n_j,0) \quad (3.24)$$

The structure of $\mathcal{N}$ in Equation 3.22 makes it clear how to define the number operator $\mathcal{N}_i$ for bin i of the histogram, so that $\mathcal{N} = \sum_{i=1}^{m} \mathcal{N}_i$ where $\mathcal{N}_i$ is defined as

$$\mathcal{N}_i \equiv {a_i}^\dagger a_i \quad (3.25)$$

and $\mathcal{N}_i \left({a_j}^\dagger\right)^{n_j} |0\rangle$ may be represented for a 4-bin histogram and for $j = 3$ as

$$(0,0,0,0) \xrightarrow{\left({a_j}^\dagger\right)^{n_j}} (0,0,n_j,0) \xrightarrow{\mathcal{N}_j} \begin{matrix} n_j \ (0,0,n_j,0) & i = j \\ 0 & i \neq j \end{matrix} \quad (3.26)$$

7. Orthogonality and Completeness

The states constructed using the creation operators described above are orthogonal and complete. Consider the general histogram state $\left(a_1{}^\dagger\right)^{n_1}\left(a_2{}^\dagger\right)^{n_2}\cdots\left(a_m{}^\dagger\right)^{n_m}|0\rangle$ and attempt to annihilate its samples. The strategy of the proof will be to demonstrate that there is a *unique* set of annihilation operators that you have to use in order to recover the empty histogram state $|0\rangle$.

Apply a single annihilation operator a_1 (using Equation 3.19 to move it to the right)

$$a_1\left(a_1{}^\dagger\right)^{n_1}\left(a_2{}^\dagger\right)^{n_2}\cdots\left(a_m{}^\dagger\right)^{n_m}|0\rangle = n_1\left(a_1{}^\dagger\right)^{n_1-1}\left(a_2{}^\dagger\right)^{n_2}\cdots\left(a_m{}^\dagger\right)^{n_m}|0\rangle \tag{3.27}$$

Now apply the same annihilation operator n_1-1 more times to eventually obtain

$$\left(a_1\right)^{n_1}\left(a_1{}^\dagger\right)^{n_1}\left(a_2{}^\dagger\right)^{n_2}\cdots\left(a_m{}^\dagger\right)^{n_m}|0\rangle = n_1!\left(a_2{}^\dagger\right)^{n_2}\cdots\left(a_m{}^\dagger\right)^{n_m}|0\rangle \tag{3.28}$$

Repeat this pattern of annihilation successively for bins $2,3,\cdots,m$ of the histogram to obtain

$$\left(a_m\right)^{n_m}\cdots\left(a_2\right)^{n_2}\left(a_1\right)^{n_1}\left(a_1{}^\dagger\right)^{n_1}\left(a_2{}^\dagger\right)^{n_2}\cdots\left(a_m{}^\dagger\right)^{n_m}|0\rangle = n_1!\,n_2!\cdots n_m!\,|0\rangle \tag{3.29}$$

where the resulting state is (proportional to) the empty histogram $|0\rangle$.

Thus we recover the empty histogram by applying exactly those annihilation operators to the histogram that correspond to the creation operators that we used to construct the histogram in the first place. The fact that the empty histogram can be recovered *only* by applying the *same* set $(n_1,n_2,\cdots,n_m)$ of annihilation operators as creation operators means that the states are *orthogonal*, and the fact that *all* possible states are constructable using the appropriate set $(n_1,n_2,\cdots,n_m)$ of creation operators means that the states are *complete*.

The constant of proportionality $n_1!\,n_2!\cdots n_m!$ is the number of ways in which the annihilation operators can annihilate the histogram samples, which corresponds to the total number of ways of permuting the samples within the histogram bins (but *not* permuting between bins). If this permutation factor is not required then the states could be defined as $\frac{1}{\sqrt{n_1!\,n_2!\cdots n_m!}}\left(a_1{}^\dagger\right)^{n_1}\left(a_2{}^\dagger\right)^{n_2}\cdots\left(a_m{}^\dagger\right)^{n_m}|0\rangle$, and a similar normalisation factor $\frac{1}{\sqrt{n_1!\,n_2!\cdots n_m!}}$ should be included with the annihilation operators when this whole state is to be annihilated. It is a matter of tast whether the normalisation factor *is* included along with the state, or whether it is *not* included but is then subsequently divided out from the results of calculations.

8. *States and Adjoint States*

The above results on orthogonality and completeness can be written more rigorously by introducing the *adjoint* state. Intuitively, the adjoint state is obtained by time-reversing everything, so that instead of making operators act to the right (with operators that act later being placed further to the left), the operators in an adjoint state act to the left (with operators that act earlier being placed further to the right). Note that between these two viewpoints the time order of operator action corresponds to the order in which the operators appear in the "operator product". Also note that a creation operator acting to the right (i.e. create a sample as time increases, as in $a_i{}^\dagger|0\rangle$) behaves in the same way as an annihilation operator acting to the left (i.e. annihilate a sample as time decreases, as in $\langle 0|\,a_i{}^\dagger=0$). In this case $a_i{}^\dagger|0\rangle$ says (reading from right to left) that there is an empty histogram in the distant past which later has a sample created in bin i, whereas $\langle 0|\,a_i{}^\dagger$ says (reading from left to right) that there is an empty histogram in the distant future which earlier has a sample annihilated from bin i to give 0 (i.e. $\langle 0|\,a_i{}^\dagger=0$).

Introduce a notation for a histogram with occupancies $(n_1,n_2,\cdots,n_m)$

$$\Theta_{n_1,n_2,\cdots,n_m}\equiv\left(a_1{}^\dagger\right)^{n_1}\left(a_2{}^\dagger\right)^{n_2}\cdots\left(a_m{}^\dagger\right)^{n_m}|0\rangle \tag{3.30}$$

and its adjoint state for creating a histogram with occupancies $(n_1,n_2,\cdots,n_m)$, *but* done in the reversed time sense where there is an empty histogram in the far future, which is then populated as we move backwards in time

$$\Theta^\dagger{}_{n_1,n_2,\cdots,n_m}=\langle 0|\left(a_m\right)^{n_m}\cdots\left(a_2\right)^{n_2}\left(a_1\right)^{n_1} \tag{3.31}$$

The orthogonality property can then be stated as

$$\Theta^\dagger{}_{\nu_1,\nu_2,\cdots,\nu_m}\,\Theta_{n_1,n_2,\cdots,n_m}=\delta_{n_1,\nu_1}\,\delta_{n_2,\nu_2}\cdots\delta_{n_m,\nu_m}\,n_1!\,n_2!\cdots n_m! \tag{3.32}$$

where the result in Equation 3.29 is used, and where $\langle 0|\,|0\rangle\equiv 1$ is defined. The completeness property then corresponds to the following resolution of the identity operator

$$\sum_{n_1,n_2,\cdots,n_m} \frac{1}{n_1!\,n_2!\,\cdots\,n_m!}\,\Theta_{n_1,n_2,\cdots,n_m}\,{\Theta^\dagger}_{n_1,n_2,\cdots,n_m} = 1 \tag{3.33}$$

where the states that this operator acts upon are assumed to be constructed in the same way as $\Theta_{n_1,n_2,\cdots,n_m}$ (i.e. using creation operators).

9. *Summary of Useful Results*

1. Creation operator for bin i: ${a_i}^\dagger$. When applied to a histogram state this creates one sample in bin i.

2. Annihilation operator for bin i: a_i. When applied to a histogram state this annihilates one sample from bin i in as many ways (i.e. n_i) as there are samples already in bin i. The result is n_i copies of the histogram state with one sample annihilated from bin i. This includes the special case $n_i = 0$ where the histogram is annihilated altogether to give 0.

3. Annihilation operator for all bins: $\sum_{i=1}^m a_i$. This produces a generalisation of what a_i alone does. For each i ($i = 1, 2, \cdots, m$) the result is n_i copies of the histogram state with one sample annihilated from bin i, which gives a total of $\sum_{i=1}^m n_i$ histograms. This operator is useful for preparing a histogram for an MCMC update because it removes a sample at random from the histogram (i.e. it prepares $\sum_{i=1}^m n_i$ copies of the histogram in each of which a different sample has been annihilated).

4. Annihilate an empty histogram: $a_i\,|0\rangle = 0$. This defines the "vacuum" state as a reference state for determining the occupancy of each histogram bin. This definition is very useful for removing terms that do not contribute to the overall histogram state.

5. Creation/annihilation commutator: $[a_i, {a_j}^\dagger] = \delta_{i,j}$. This summarises the basic interaction between the creation and annihilation operators. It is mainly used in the form $a_i\,{a_j}^\dagger = {a_j}^\dagger\,a_i + \delta_{i,j}$ to move annihilation operators to the right of creation operators, which eventually brings the annihilation operators so that they act directly on $|0\rangle$, where they can be removed (using $a_i\,|0\rangle = 0$).

6. Annihilation/annihilation and creation/creation commutators: $[a_i, a_j] = 0$ and $[{a_i}^\dagger, {a_j}^\dagger] = 0$. These summarise the fact that a sequence consisting solely of annihilation operations (or solely of creation operations) has the same effect whatever the order in which the operators appear in the sequence.

7. Moving an annihilation operator to the right: $a_i\,({a_j}^\dagger)^n\,|0\rangle = n\,\delta_{i,j}\,({a_j}^\dagger)^{n-1}\,|0\rangle$: This is the basic result that is used to remove annihilation operators from expressions. The a_i is moved progressively to the right through the ${a_j}^\dagger$ (using $a_i\,{a_j}^\dagger = {a_j}^\dagger\,a_i + \delta_{i,j}$) until it reaches the $|0\rangle$, where it is discarded (using $a_i\,|0\rangle = 0$).

8. Number operator for bin i: $\mathcal{N}_i = {a_i}^\dagger\,a_i$. This annihilates then creates a sample in bin i. Because there are n_i ways of annihilating a sample but only 1 way of creating a sample, the net effect is to count the number n_i of samples in bin i.

9. Total number operator for all bins: $\mathcal{N} = \sum_{i=1}^m {a_i}^\dagger\,a_i$. This counts the total number of samples in the histogram. This follows directly from $\mathcal{N}_i = {a_i}^\dagger\,a_i$ above.

10. State and adjoint state: $\Theta_{n_1,n_2,\cdots,n_m} = ({a_1}^\dagger)^{n_1}\,({a_2}^\dagger)^{n_2}\,\cdots\,({a_m}^\dagger)^{n_m}\,|0\rangle$ and ${\Theta^\dagger}_{n_1,n_2,\cdots,n_m} = \langle 0|\,(a_m)^{n_m}\,\cdots\,(a_2)^{n_2}\,(a_1)^{n_1}$ (respectively). The adjoint state can be applied to the left of a state and the annihilation operators then moved to the right using $a_i\,({a_j}^\dagger)^n\,|0\rangle = n\,\delta_{i,j}\,({a_j}^\dagger)^{n-1}\,|0\rangle$ to demonstrate orthogonality (assuming $\langle 0|\,|0\rangle \equiv 1$). The adjoint of $a_i\,|0\rangle = 0$ implies $\langle 0|\,{a_i}^\dagger = 0$.

11. Orthogonality: ${\Theta^\dagger}_{\nu_1,\nu_2,\cdots,\nu_m}\,\Theta_{n_1,n_2,\cdots,n_m} = \delta_{n_1,\nu_1}\,\delta_{n_2,\nu_2}\,\cdots\,\delta_{n_m,\nu_m}\,n_1!\,n_2!\,\cdots\,n_m!$. Here $\langle 0|\,|0\rangle \equiv 1$ is assumed by definition.

12. Completeness: All states $\Theta_{n_1,n_2,\cdots,n_m}$ are constructable by using the appropriate set $(n_1, n_2, \cdots, n_m)$ of creation operators.

10. *Multiple MRF Nodes*

The above results are for a *single* MRF node. When there are multiple nodes, each MRF node has it own set of creation and annihilation operators, which have all of the properties described above. Operators for different nodes commute with each other because they act on different state spaces, so the generalised form of Equation 3.15 is

$$\begin{aligned} \left[a_i^s, a_j^{t\dagger}\right] &= \delta_{i,j}\delta_{s,t} \\ \left[a_i^s, a_j^t\right] &= 0 \\ \left[a_i^{s\dagger}, a_j^{t\dagger}\right] &= 0 \end{aligned} \tag{3.34}$$

where s and t are node indices. There are analogous generalisations of all the results in Section III B 9.

C. MCMC Update Operator

In Section II D it was shown how the state of an N-node MRF can be represented as a set of N histograms each of which contains one sample in one of the histogram bins, and how MCMC updates of the MRF can be represented as hopping operations where each sample hops around between the bins of its histogram. The aim now is to use the creation and annihilation operators defined in Section III B to implement these MCMC hopping operations.

The MCMC update operator $\mathcal{H}$ can be constructed in several easy steps, in which each MCMC hopping operation is broken down into annihilation followed by subsequent creation of a sample.

1. Annihilate a sample (see the middle row of Figure 1). Apply $\sum_{j=1}^{m} a_j$ to the histogram state to annihilate one sample from each bin, which prepares $\sum_{i=1}^{m} n_i$ copies of the histogram in each of which a different sample has been annihilated. The output of this operation is thus a linear combination of histogram states, where each state is weighted by the same factor of unity (i.e. all states are equally likely). This linear combination of $\sum_{i=1}^{m} n_i$ terms (of which only m are distinct) represents the ensemble of all the possible outcomes of annihilating one sample.

2. Create a sample (see the bottom row of Figure 1). Apply $\sum_{i=1}^{m} p_i \, {a_i}^\dagger$ to each histogram state in the ensemble generated above, which prepares m copies of the histogram in each of which a different sample has been created, and weight each of these m histogram states so that where the sample is created in bin i the state is weighted by a factor p_i. If the p_i satisfy $p_i \geq 0$ and $\sum_{i=1}^{m} p_i = 1$ then p_i can be interpreted as the probability of creating a sample in bin i. Actually, the normalisation condition $\sum_{i=1}^{m} p_i = 1$ can be omitted because the *relative* size of the p_i is all that is required. The output of this operation is thus a linear combination of histogram states, where each state is weighted by the appropriate probability factor p_i corresponding to the bin i in which a sample has just been created. This linear combination of m terms represents the ensemble of all the possible outcomes of creating one sample in one of the bins of a histogram.

Concatenate these two operators to define the MCMC update operator $\mathcal{H}$

$$\mathcal{H} \equiv \sum_{i=1}^{m} p_i \, {a_i}^\dagger \sum_{j=1}^{m} a_j \qquad (3.35)$$

where the action of $\sum_{j=1}^{m} a_j$ produces $\sum_{i=1}^{m} n_i$ histograms, then the action of $\sum_{i=1}^{m} p_i \, {a_i}^\dagger$ on *each* of these $\sum_{i=1}^{m} n_i$ histograms produces m histograms. Finally, all of these histograms should be regrouped so that multiple copies of identical histograms are represented as a single copy with an appropriate weighting factor.

The weighting factor that is applied to the state (as used here) represents probability itself rather than probability amplitude (as used in the corresponding QFT). However, if a QFT is "Wick rotated" to become a Euclidean QFT then it is equivalent to quantum statistical mechanics [9], where the state is a probability-weighted mixture of states. So the approach discussed in this paper has a mathematical structure that is similar to the Euclidean version of a QFT of bosons.

The pieces $p_i \, {a_i}^\dagger \, a_j$ of the MCMC update operator may be represented diagrammatically as

$$p_i \left(\begin{tikzcd} j \arrow[r, "a_j"] & \cdot \arrow[r, "{a_i}^\dagger"] & i \\ & \text{source} \arrow[u, Rightarrow] & \end{tikzcd} \right)$$

where state j comes in from the left and is annihilated by a_j, and a new state i is created by ${a_i}^\dagger$ which then goes out to the right, and the probability of this transition occurring is p_i which depends only on the output state (so it is memoryless), which is in turn generated by a source (e.g. MRF neighbours, external source, etc). The whole MCMC update operator $\mathcal{H}$ is the sum of this diagram over states i and j.

This result can be generalised to an MRF with N nodes (with node s having m_s states)

$$\mathcal{H} \longrightarrow \sum_{s=1}^{N} \sum_{i=1}^{m_s} p_i^s \, a_i^{s\dagger} \sum_{j=1}^{m_s} a_j^s \qquad (3.36)$$

which can be written using the transition operator $\mathcal{T}_{i,j}^s \equiv a_i^{s\dagger} \, a_j^s$ that hops a sample from bin j to bin i at node s.

$$\mathcal{H} = \sum_{s=1}^{N} \sum_{i,j=1}^{m_s} p_i^s \, \mathcal{T}_{i,j}^s \qquad (3.37)$$

In practice the creation probability p_i^s depends (via a product of clique factors, as described in the discussion on the HCE in Section II A) on the states of the other nodes in the MRF. This probability can be computed by applying an appropriately designed operator to the MRF node states. Thus use the number operator for bin k at node t (which is $\mathcal{N}_k^t \equiv a_k^{t\dagger} \, a_k^t$) weighted by $p_{i,k}^{s,t}$ to determine the 2-clique contribution (i.e. pairwise interactions between nodes of the MRF) for creation in bin i at node s due to bin k at node t being occupied. This operator expression is appropriate for *any* number of samples in bin k at node t, because the number operator $\mathcal{N}_k^t$ automatically determines the number of samples as needed, and then uses this number to weight any clique factor that involves this node.

This use of sample number to weight clique factors is consistent because it guarantees that a *single* sample at each node (i.e. standard HCE) is physically equivalent to the situation where each of these samples is cut into a number of equal-sized sub-samples, because the additional factors then generated by the number operator applied to these sub-samples are exactly cancelled by the additional factors then generated by the fact that interactions between *sub*-samples are proportionally weaker than interactions between samples.

This allows p_i^s to be replaced by an operator $\mathcal{P}_i^s$, which can be used to construct a p_i^s based on whatever samples it finds in the histograms in the neighbourhood $C(s)$ of node s of the MRF.

$$p_i^s \longrightarrow \mathcal{P}_i^s \equiv \prod_{t \in C(s)} \sum_{k=1}^{m_t} p_{i,k}^{s,t} \mathcal{N}_k^t \qquad (3.38)$$

This result should be compared with the product form of the HCE in Equation 2.1, where the $\prod_{t\in C(s)}(\cdots)$ in Equation 3.38 corresponds to the $\prod_c(\cdots)$ in Equation 2.1, and the sum over operators $\sum_{k=1}^{m_t}(\cdots)$ in Equation 3.38 is needed to cover all the possibilities that might appear in the $(\cdots)$ inside $\prod_c(\cdots)$ in Equation 2.1. More generally for 3-cliques the operator $\mathcal{P}_i^s$ is given by

$$p_i^s \longrightarrow \mathcal{P}_i^s \equiv \prod_{t_1,t_2 \in C(s)} \sum_{k_1=1}^{m_{t_1}} \sum_{k_2=1}^{m_{t_2}} p_{i,k_1,k_2}^{s,t_1,t_2} \mathcal{N}_{k_1}^{t_1} \mathcal{N}_{k_2}^{t_2} \qquad (3.39)$$

which may be straightforwardly generalised to higher order cliques.

Inserting the operator-valued version of p_i^s into Equation 3.37, the MCMC update operator $\mathcal{H}$ becomes (using 2-cliques only)

$$\mathcal{H} \longrightarrow \sum_{s=1}^{N} \sum_{i,j=1}^{m_s} \mathcal{T}_{i,j}^s \left(\prod_{t\in C(s)} \sum_{k=1}^{m_t} p_{i,k}^{s,t} \mathcal{N}_k^t \right) \qquad (3.40)$$

with analogous expressions for higher order cliques. This operator-valued object $\mathcal{H}$ can be applied to *any* MRF state, whether it is a conventional *single* sample per node state, or has *multiple* samples per node. This is the key advantage of using operators, because they are effectively general procedures (e.g. algorithms) that can be applied to any state that is constructed using creation operators. The algebra of the creation and annihilation operators provides a unified framework for handing all of these possibilities consistently.

The functional form used in Equation 3.40 is enforced by backward compatibility with the MCMC update operator for an MRF shown in Equation 3.36, where the factor p_i^s is a product of clique factors that intersect with node s (i.e. for 2-cliques only, it is generated by the $\prod_{t\in C(s)} \sum_{k=1}^{m_t} p_{i,k}^{s,t} \mathcal{N}_k^t$ factor in Equation 3.40). However, the framework developed here allows for any functional form built out of creation and annihilation operators, so a very large class of update operators $\mathcal{H}$ can be constructed such as:

1. The operator that generates the product of clique factors $\prod_{t\in C(s)} \sum_{k=1}^{m_t} p_{i,k}^{s,t} \mathcal{N}_k^t$ can be replaced by some other functional form, such as a non-linear sigmoid squashing function $\sigma\left(\sum_{t\in C(s)} \sum_{k=1}^{m_t} p_{i,k}^{s,t} \mathcal{N}_k^t\right)$, as is typically done in "neural network" implementations of recurrent networks. One possible way of viewing the relationship between this non-linear sigmoidal version and the clique product can be obtained by perturbatively expanding the sigmoid to obtain various powers of its argument $\sum_{t\in C(s)} \sum_{k=1}^{m_t} p_{i,k}^{s,t} \mathcal{N}_k^t$, which includes terms that look like the original clique product $\prod_{t\in C(s)} \sum_{k=1}^{m_t} p_{i,k}^{s,t} \mathcal{N}_k^t$, plus other higher order terms.

2. The hopping operator $\mathcal{T}_{i,j}^s = a_i^{s\dagger} a_j^s$ can be replaced by some other functional form, such as one that increases (i.e. birth) or decreases (i.e. death) the number of samples, which may be used to allow the update operator $\mathcal{H}$ to explore histogram states with various occupancies. Note that if this part of the overall update operator $\mathcal{H}$ is used *alone* as the update operator (i.e. without the clique factor piece above), then it can be used to generate the prior behaviour that the histogram state has before any interactions with other histograms are included.

The effect of the creation and annihilation operators can be viewed in terms of elementary operations on histograms (as described in Section III B), and their operator algebra can be used to do calculations in which $\mathcal{H}$ is applied to multiply occupied states to generate MCMC updates. It is also possible to use symbolic algebra to do these operator manipulations automatically. In general, the effect of the MCMC update operator $\mathcal{H}$ on a set of histogram states can be represented as a type of Feynman diagram, in which each vertex represents a product of operators acting on an incoming state to produce an outgoing state (if any), and a (weighted) sum of such diagrams represents the corresponding (weighted) sum of products of operators (note that here the weights are probabilities rather than probability amplitudes).

Note that the MCMC update operator $\mathcal{H}$ in Equation 3.40 is number-conserving in the sense that its transition operator $\mathcal{T}_{i,j}^s \equiv a_i^{s\dagger} a_j^s$ causes samples to hop from bin j to bin i at node s, without gain or loss of the total number of samples at node s. Formally, this property may be written as $[\mathcal{H}, \mathcal{N}^s] = 0$ where $\mathcal{N}^s \equiv \sum_{i=1}^{m_s} \mathcal{N}_i^s$ is the *total* number operator at node s. This result can be seen intuitively because it may be written as $\mathcal{H}\mathcal{N}^s = \mathcal{N}^s \mathcal{H}$, which states that when you measure the total number of samples at node s then do an MCMC update, you get the *same* result as when you do an MCMC update then measure the total number of samples at node s, so there must be number conservation.

The steps in the derivation of the number conservation property $[\mathcal{H}, \mathcal{N}^u] = 0$ are as follows

$$
\begin{aligned}
[\mathcal{H}, \mathcal{N}^u] &= \sum_{s=1}^{N} \sum_{i,j=1}^{m_s} \left[\mathcal{T}^s_{i,j} \left(\prod_{t \in C(s)} \sum_{k=1}^{m_t} p^{s,t}_{i,k} \mathcal{N}^t_k \right), \mathcal{N}^u \right] \\
&= \sum_{s=1}^{N} \sum_{i,j=1}^{m_s} \left(\mathcal{T}^s_{i,j} \left(\prod_{t \in C(s)} \sum_{k=1}^{m_t} p^{s,t}_{i,k} \mathcal{N}^t_k \right) \mathcal{N}^u - \mathcal{N}^u \mathcal{T}^s_{i,j} \left(\prod_{t \in C(s)} \sum_{k=1}^{m_t} p^{s,t}_{i,k} \mathcal{N}^t_k \right) \right) \\
&= \sum_{s=1}^{N} \sum_{i,j=1}^{m_s} \left(\mathcal{T}^s_{i,j} \left(\prod_{t \in C(s)} \sum_{k=1}^{m_t} p^{s,t}_{i,k} \mathcal{N}^t_k \right) \mathcal{N}^u - \mathcal{T}^s_{i,j} \mathcal{N}^u \left(\prod_{t \in C(s)} \sum_{k=1}^{m_t} p^{s,t}_{i,k} \mathcal{N}^t_k \right) \right) \\
&= \sum_{s=1}^{N} \sum_{i,j=1}^{m_s} \left(\mathcal{T}^s_{i,j} \left(\prod_{t \in C(s)} \sum_{k=1}^{m_t} p^{s,t}_{i,k} \mathcal{N}^t_k \right) \mathcal{N}^u - \mathcal{T}^s_{i,j} \left(\prod_{t \in C(s)} \sum_{k=1}^{m_t} p^{s,t}_{i,k} \mathcal{N}^t_k \right) \mathcal{N}^u \right) \\
&= 0
\end{aligned}
\tag{3.41}
$$

using $[\mathcal{N}^u, \mathcal{T}^s_{i,j}] = 0$ ($\mathcal{T}^s_{i,j}$ causes hopping at node s but conserves total number at node s, *and* also trivially conserves total number at all other nodes) to make the replacement $\mathcal{N}^u \mathcal{T}^s_{i,j} \longrightarrow \mathcal{T}^s_{i,j} \mathcal{N}^u$, and $[\mathcal{N}^u, \mathcal{N}^t_k] = 0$ (number operators always commute) to make the replacement $\mathcal{N}^u \left(\prod_{t \in C(s)} \sum_{k=1}^{m_t} p^{s,t}_{i,k} \mathcal{N}^t_k \right) \longrightarrow \left(\prod_{t \in C(s)} \sum_{k=1}^{m_t} p^{s,t}_{i,k} \mathcal{N}^t_k \right) \mathcal{N}^u$. Note that the fact that $[\mathcal{N}^u, \mathcal{T}^s_{i,j}] = 0$ and $[\mathcal{N}^u, \mathcal{N}^t_k] = 0$ are simple to derive from the basic creation/annihilation content of the various operators.

The overall effect of using creation and annihilation operators is to formalise the act of manipulating samples in histograms, so that these manipulations are now represented algebraically. One *could* avoid the use of this algebraic approach (especially when each histogram has only a single sample, as in a standard MRF), but as the manipulations become more complicated (e.g. subtle interdependencies between histograms) it is better to do them by using this algebraic approach.

D. Diagrammatic Representation of MCMC Algorithms

A sequence of MCMC updates (e.g. see Section II B) in which $\boldsymbol{x}$ and $\boldsymbol{y}$ are *alternately* updated by sampling from $\Pr(\boldsymbol{x}, \boldsymbol{y})$ is illustrated below where each arrow represents a dependency. The graph structure shows that the updates are memoryless. For instance, $\boldsymbol{x}_2$ depends on $\boldsymbol{y}_1$ via $\Pr(\boldsymbol{x}_2|\boldsymbol{y}_1)$, but it does *not* depend on $\boldsymbol{x}_1$.

$$
\begin{tikzcd}
\boldsymbol{x}_1 & \boldsymbol{x}_2 \arrow[r] \arrow[ddr] & \boldsymbol{x}_2 & \boldsymbol{x}_3 \arrow[r] \arrow[ddr] & \boldsymbol{x}_3 \cdots \\
 & & & & \cdots \\
\boldsymbol{y}_1 \arrow[r] \arrow[uur] & \boldsymbol{y}_1 & \boldsymbol{y}_2 \arrow[r] \arrow[uur] & \boldsymbol{y}_2 & \boldsymbol{y}_3 \cdots
\end{tikzcd}
$$

$\Pr(\boldsymbol{x}_2|\boldsymbol{y}_1)$ $\Pr(\boldsymbol{y}_2|\boldsymbol{x}_2)$ $\Pr(\boldsymbol{x}_3|\boldsymbol{y}_2)$ $\Pr(\boldsymbol{y}_3|\boldsymbol{x}_3)$

The above diagram can be skeletonised by omitting all inessential labelling in order to emphasis the information flow, in which case the result looks like this

$$
\begin{tikzcd}
\cdot & \cdot \arrow[r] \arrow[dr] & \cdot & \cdot \arrow[r] \arrow[dr] & \cdot \cdots \\
 & & & & \cdots \\
\end{tikzcd}
$$

$$
\begin{tikzcd}
\cdot & \cdot \arrow[r] \arrow[dr] & \cdot & \cdot \arrow[r] \arrow[dr] & \cdot \cdots \\
\cdot \arrow[r] \arrow[ur] & \cdot & \cdot \arrow[r] \arrow[ur] & \cdot & \cdot \cdots
\end{tikzcd}
$$

If this skeletonisation is used to draw an information flow diagram for a sequence of MCMC updates of a 4 node Markov chain, then a typical result looks like the diagram below.

$$\begin{tikzcd}
& \cdot \arrow[r] \arrow[ddr] & \cdot \arrow[r] & \cdot \arrow[r] & \cdot & \cdot \arrow[r] & \cdot \arrow[r] & \cdot \arrow[r] \arrow[ddr] & \cdot \arrow[r] & \cdot \arrow[r] & \cdot \arrow[r] & \cdot & \cdots \\
\pm 1 \\
& \cdot & \cdot \arrow[r] \arrow[ddr] & \cdot \arrow[r] & \cdot \arrow[r] \arrow[uur] & \cdot & \cdot & \cdot & \cdot \arrow[r] & \cdot \arrow[r] \arrow[ddr] & \cdot \arrow[r] & \cdot & \cdots \\
\pm 2 \\
& \cdot \arrow[r] & \cdot & \cdot & \cdot \arrow[r] & \cdot \arrow[r] \arrow[uur] & \cdot \arrow[r] \arrow[uur] & \cdot \arrow[r] & \cdot & \cdot & \cdot & \cdot & \cdots \\
\pm 3 \\
& \cdot \arrow[r, "+1"'] & \cdot \arrow[r, "+2"'] & \cdot \arrow[r, "-3"'] \arrow[uur] & \cdot \arrow[r, "-1"'] & \cdot \arrow[r, "-2"'] & \cdot \arrow[r, "-2"'] & \cdot \arrow[r, "+1"'] & \cdot \arrow[r, "-3"'] \arrow[uur] & \cdot \arrow[r, "+2"'] & \cdot \arrow[r, "-3"'] \arrow[uur] & \cdot & \cdots
\end{tikzcd}$$

For illustrative purposes the Markov chain is drawn in the up-down direction in the diagram, with the horizontal direction being used for the discrete time steps that are generated by the MCMC update procedure. The $\pm n$ notation at the left hand side shows the labelling convention that is used for the update that occurs at each time step, where $+n$ indicates an interaction between a node and its right hand neighbour (right is "down" in the diagram), and $-n$ is the analogous notation for the left hand neighbour. The $\pm n$ notation along the bottom of the diagram shows the actual update interaction that occurs at each time step. The particular sequence of MCMC updates that is represented in the diagram above is unimportant because it is random.

There are 6 separate basic diagrams that are used to build the above diagram which are shown in the diagram below. Usually a randomly selected sequence of these diagrams forms the MCMC algorithm, but other choices are possible.

$$\begin{tikzcd}
\cdot \arrow[r] \arrow[dr] & \cdot & \cdot & \cdot & \cdot \arrow[r] & \cdot & \cdot \arrow[r] & \cdot & \cdot \arrow[r] & \cdot & \cdot \arrow[r] & \cdot \\
\cdot & \cdot & \cdot \arrow[r] \arrow[ur] & \cdot & \cdot \arrow[r] \arrow[dr] & \cdot & \cdot & \cdot & \cdot \arrow[r] & \cdot & \cdot \arrow[r] & \cdot \\
\cdot \arrow[r] & \cdot & \cdot \arrow[r] & \cdot & \cdot & \cdot & \cdot \arrow[r] \arrow[ur] & \cdot & \cdot \arrow[dr] & \cdot & \cdot & \cdot \\
\cdot \arrow[r, "+1"'] & \cdot & \cdot \arrow[r, "-1"'] & \cdot & \cdot \arrow[r, "+2"'] & \cdot & \cdot \arrow[r, "-2"'] & \cdot & \cdot \arrow[r, "+3"'] & \cdot & \cdot \arrow[r, "-3"'] \arrow[ur] & \cdot
\end{tikzcd}$$

The skeletonised structure of the diagrams can now be simplified further to make it look more symmetrical as shown in the diagram below, where the pieces of the above diagrams are drawn individually in more symmetrical fashion.

$$\begin{tikzcd}
\cdot \arrow[r] & \cdot & \Longrightarrow & {} \arrow[r] & \cdot \arrow[r] & {}
\end{tikzcd}$$

$$\begin{tikzcd}
\cdot \arrow[r] \arrow[ddr] & \cdot & & {} \arrow[r] & \cdot \arrow[r] \arrow[dd] & {} \\
& & \Longrightarrow & & & \\
\cdot & \cdot & & {} \arrow[r] & \cdot \arrow[r] & {}
\end{tikzcd}$$

$$\begin{tikzcd}
\cdot \arrow[r] & \cdot & & {} \arrow[r] & \cdot \arrow[r] & {} \\
& & \Longrightarrow & & & \\
\cdot \arrow[uur] & \cdot & & {} \arrow[r] & \cdot \arrow[r] \arrow[uu] & {}
\end{tikzcd}$$

This reduces the description of the MCMC algorithm to a set of basic diagrams in which the state of a node evolves freely (i.e. $\longrightarrow \cdot \longrightarrow$) or is involved in an interaction (i.e. $\underset{\downarrow}{\longrightarrow \cdot \longrightarrow}$ and $\underset{\uparrow}{\longrightarrow \cdot \longrightarrow}$). These diagrams allow for the possibility that a node has a "memory" of its previous state (i.e. an arrow comes in from the left), so the MCMC diagrams above are a special case in which this memory is discarded.

These diagrams can be used to represent higher order MCMC algorithms which amalgamate the effect of several basic MCMC updates. Thus, start by defining an MCMC update operator $\mathcal{H}$. For a *pair* of MRF nodes this is illustrated in Equation 3.42, which is of the form $\mathcal{H} \equiv \mathcal{I} + \mathcal{H}_1 + \mathcal{H}_2$. The $\mathcal{I}$ is the "identity" which corresponds to no update occurring, and the $\mathcal{H}_1$ and $\mathcal{H}_2$ pieces correspond to updates that occur on one or the other of the two nodes, respectively.

$$\mathcal{H} \equiv \begin{pmatrix} \longrightarrow & \cdot & \longrightarrow \\ & & \\ \longrightarrow & \cdot & \longrightarrow \end{pmatrix} + \begin{pmatrix} \longrightarrow & \cdot & \longrightarrow \\ & \downarrow & \\ \longrightarrow & \cdot & \longrightarrow \end{pmatrix} + \begin{pmatrix} \longrightarrow & \cdot & \longrightarrow \\ & \uparrow & \\ \longrightarrow & \cdot & \longrightarrow \end{pmatrix} \tag{3.42}$$

Multiple MC updates may then be generated by iterating $\mathcal{H}$ to create powers of $\mathcal{H}$. For instance, $\mathcal{H}^2$ may be derived as by expanding out $\{\mathcal{I} + \mathcal{H}_1 + \mathcal{H}_2\}^2$ and collecting together similar terms, as shown in Equation 3.43 and Equation 3.44.

$$\mathcal{H}^2 = A_0 + A_1 + A_2 \tag{3.43}$$

where

$$\begin{aligned} A_0 &\equiv \begin{pmatrix} \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \\ & & & & \\ \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \end{pmatrix} \\ A_1 &\equiv \begin{pmatrix} \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \\ & \downarrow & & & \\ \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \end{pmatrix} + \begin{pmatrix} \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \\ & & & \downarrow & \\ \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \end{pmatrix} + \begin{pmatrix} \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \\ & \uparrow & & & \\ \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \end{pmatrix} + \begin{pmatrix} \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \\ & & & \uparrow & \\ \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \end{pmatrix} \\ A_2 &\equiv \begin{pmatrix} \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \\ & \downarrow & & \downarrow & \\ \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \end{pmatrix} + \begin{pmatrix} \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \\ & \uparrow & & \uparrow & \\ \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \end{pmatrix} + \begin{pmatrix} \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \\ & \downarrow & & \uparrow & \\ \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \end{pmatrix} + \begin{pmatrix} \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \\ & \uparrow & & \downarrow & \\ \longrightarrow & \cdot & \longrightarrow & \cdot & \longrightarrow \end{pmatrix} \end{aligned} \tag{3.44}$$

The result in Equation 3.43 and Equation 3.44 may be simplified to Equation 3.45 and Equation 3.46 (using $\mathcal{I}^2 = \mathcal{I}$ and $\mathcal{I}\,\mathcal{H}_i = \mathcal{H}_i\,\mathcal{I} = \mathcal{H}_i$).

$$\mathcal{H}^2 = B_0 + B_1 + A_2 \tag{3.45}$$

where

$$\begin{aligned} B_0 &\equiv \begin{pmatrix} \longrightarrow & \cdot & \longrightarrow \\ & & \\ \longrightarrow & \cdot & \longrightarrow \end{pmatrix} \\ B_1 &\equiv 2\begin{pmatrix} \longrightarrow & \cdot & \longrightarrow \\ & \downarrow & \\ \longrightarrow & \cdot & \longrightarrow \end{pmatrix} + 2\begin{pmatrix} \longrightarrow & \cdot & \longrightarrow \\ & \uparrow & \\ \longrightarrow & \cdot & \longrightarrow \end{pmatrix} \end{aligned} \tag{3.46}$$

In the diagrammatic expression for $\mathcal{H}^2$ in Equation 3.46 the first row represents *no* interaction, the second row *one* interaction, and the third row *two* interactions. Note that the order in which the interactions occur is important (i.e. $\mathcal{H}_1\,\mathcal{H}_2 \neq \mathcal{H}_2\,\mathcal{H}_1$ in general) so the diagrams in the third row *cannot* be combined. On the other hand $\mathcal{I}\,\mathcal{H}_i = \mathcal{H}_i\,\mathcal{I} = \mathcal{H}_i$ so the diagrams in the second row *can* be combined.

These diagrams are actually Feynman diagrams, which describe operator expressions in an visually appealing way. In this case they show how the various operations invoked by the pieces of the MCMC update operator $\mathcal{H}$ fit together in various ways to generate the diagrammatic representation of the higher order MCMC update operator $\mathcal{H}^2$. This example is simple enough that the results are obvious, but the diagrammatic technique generalises to arbitrarily complicated cases.

IV. APPLICATIONS OF THE MCMC UPDATE OPERATOR

The aim of this section is to show some simple practical uses of the operator approach that is described in Section III. No attempt will be made to do extensive computations, because these will be presented in future papers in this "discrete network dynamics" series of papers.

Section IV A illustrates how the MCMC update operator correctly generates MCMC updates for histograms that are each occupied by a single sample, thus ensuring backwards compatibility between the operator approach and the standard MCMC algorithm for sampling MRFs. Section IV B generalises this to the case of multiply occupied states, and derives the equilibrium state of a single node MRF which has the same properties as ACEnet [6].

A. Update of Single-Sample States

As a check on the result for $\mathcal{H}$ in Equation 3.40 verify that the application of $\mathcal{H}$ to a standard MRF state (i.e. *one* sample per node) leads to the expected standard form of the MCMC update.

In a standard MRF only a single bin i_u is occupied at each node u. For an N-node MRF this defines a *pure* state $\Psi(i_1, i_2, \cdots, i_N)$ that has the form

$$\Psi(i_1, i_2, \cdots, i_N) \equiv \left(\prod_{u=1}^{N} a_{i_u}^{u\dagger}\right)|0\rangle \tag{4.1}$$

The first operator to consider in Equation 3.40 is the number operator $\mathcal{N}_k^t$ (for measuring how many samples are in bin k at node t). When $\mathcal{N}_k^t$ is applied to $\Psi(i_1, i_2, \cdots, i_N)$ it gives

$$\begin{aligned}\mathcal{N}_k^t\,\Psi(i_1, i_2, \cdots, i_N) &= \mathcal{N}_k^t \left(\prod_{u=1}^{N} a_{i_u}^{u\dagger}\right)|0\rangle \\ &= \delta_{i_t,k}\,\Psi(i_1, i_2, \cdots, i_N) \quad (4.2)\end{aligned}$$

so the number $\delta_{i_t,k}$ is 1 if the bin at node t being examined (i.e. k) matches the bin in which the sample at node t is to be found (i.e. i_t), and is 0 otherwise.

Insert this result into the $\prod_{t\in C(s)}\sum_{k=1}^{m_t} p_{i,k}^{s,t}\,\mathcal{N}_k^t$ part of $\mathcal{H}$ in Equation 3.40 to obtain the following simplification

$$\begin{aligned}\left(\prod_{t\in C(s)}\sum_{k=1}^{m_t} p_{i,k}^{s,t}\,\mathcal{N}_k^t\right)\Psi(i_1, i_2, \cdots, i_N) &= \left(\prod_{t\in C(s)}\sum_{k=1}^{m_t} p_{i,k}^{s,t}\,\mathcal{N}_k^t\right)\left(\prod_{u=1}^{N} a_{i_u}^{u\dagger}\right)|0\rangle \\ &= \left(\prod_{t\in C(s)}\sum_{k=1}^{m_t} p_{i,k}^{s,t}\,\delta_{i_t,k}\right)\Psi(i_1, i_2, \cdots, i_N) \\ &= \left(\prod_{t\in C(s)} p_{i,i_t}^{s,t}\right)\Psi(i_1, i_2, \cdots, i_N) \qquad (4.3)\end{aligned}$$

which is equal to $\Psi(i_1, i_2, \cdots, i_N)$ weighted by the product of the 2-clique factors that involve node s. This result correctly computes the 2-clique influence of the neighbours of node s that is expected in a standard MCMC algorithm.

$\mathcal{H}$ in Equation 3.40 also involves the transition operator $\mathcal{T}_{i,j}^s$. Apply $\mathcal{T}_{i,j}^s$ to $\Psi(i_1, i_2, \cdots, i_N)$ to obtain

$$\begin{aligned}\mathcal{T}_{i,j}^s\,\Psi(i_1, i_2, \cdots, i_N) &= \mathcal{T}_{i,j}^s\left(\prod_{u=1}^{N} a_{i_u}^{u\dagger}\right)|0\rangle \\ &= a_i^{s\dagger}\,a_j^s\,a_{i_1}^{1\dagger}\,a_{i_2}^{2\dagger}\cdots a_{i_s}^{s\dagger}\cdots a_{i_N}^{N\dagger}\,|0\rangle \\ &= a_i^{s\dagger}\,a_{i_1}^{1\dagger}\,a_{i_2}^{2\dagger}\cdots\left(a_{i_s}^{s\dagger}\,a_j^s + \delta_{i_s,j}\right)\cdots a_{i_N}^{N\dagger}\,|0\rangle \\ &= a_i^{s\dagger}\,a_{i_1}^{1\dagger}\,a_{i_2}^{2\dagger}\cdots a_{i_s}^{s\dagger}\cdots a_{i_N}^{N\dagger}\,a_j^s\,|0\rangle + \delta_{i_s,j}\,a_{i_1}^{1\dagger}\,a_{i_2}^{2\dagger}\cdots a_i^{s\dagger}\cdots a_{i_N}^{N\dagger}\,|0\rangle \\ &= \delta_{i_s,j}\,\Psi(i_1, i_2, \cdots, i_{s-1}, i, i_{s+1}, \cdots, i_N) \qquad (4.4)\end{aligned}$$

where the annihilation operator a_j^s is moved to the right, picking up a non-zero commutator only when it moves past the creation operator $a_{i_s}^{s\dagger}$ (i.e. both the creation and the annihilation are at the *same* node so they do *not* commute if $i_s = j$), and finally meets the empty state $|0\rangle$ which it annihilates. This result is equal to $\Psi(i_1, i_2, \cdots, i_{s-1}, i, i_{s+1}, \cdots, i_N)$ weighted by a factor $\delta_{i_s,j}$, which corresponds to a new pure state in which the sample at node s has hopped to bin i, weighted by 1 if the sample at node s started off in bin j, and 0 otherwise. This is exactly the behaviour that is expected of the transition operator $\mathcal{T}_{i,j}^s$.

Finally, inserting the results in Equation 4.3 and Equation 4.4 into $\mathcal{H}$ in Equation 3.40 gives

$$\begin{aligned}\mathcal{H}\,\Psi(i_1, i_2, \cdots, i_N) &= \sum_{s=1}^{N}\sum_{i,j=1}^{m_s}\mathcal{T}_{i,j}^s\left(\prod_{t\in C(s)}\sum_{k=1}^{m_t} p_{i,k}^{s,t}\,\mathcal{N}_k^t\right)\Psi(i_1, i_2, \cdots, i_N) \\ &= \sum_{s=1}^{N}\sum_{i,j=1}^{m_s}\delta_{i_s,j}\left(\prod_{t\in C(s)} p_{i,i_t}^{s,t}\right)\Psi(i_1, i_2, \cdots, i_{s-1}, i, i_{s+1}, \cdots, i_N) \\ &= \sum_{s=1}^{N}\sum_{i=1}^{m_s}\left(\prod_{t\in C(s)} p_{i,i_t}^{s,t}\right)\Psi(i_1, i_2, \cdots, i_{s-1}, i, i_{s+1}, \cdots, i_N) \qquad (4.5)\end{aligned}$$

The action of $\mathcal{H}$ on the *pure* state $\Psi(i_1, i_2, \cdots, i_N)$ produces a weighted sum of states (or *mixed* state), because the effect of $\mathcal{H}$ at each node s is to *simultaneously* create m_s states $\Psi(i_1, i_2, \cdots, i_{s-1}, i, i_{s+1}, \cdots, i_N)$ (for $i = 1, 2, \cdots, m_s$), each of which has its own probability factor $\prod_{t \in C(s)} p_{i,i_t}^{s,t}$ (i.e. product of 2-clique factors), which is a total of $m_1 m_2 \cdots m_N$ states with their corresponding probability factors. Note that this ensemble of histograms should be regrouped so that multiple copies of identical histograms are represented as a single copy with an appropriate weighting factor. Thus $\mathcal{H}\,\Psi(i_1, i_2, \cdots, i_N)$ is *precisely* the ensemble of states from which the standard MCMC update algorithm draws its updated state.

This verifies that the update operator $\mathcal{H}$ generates the correct behaviour when only a *single* bin i_u is occupied at each node u, as is the case in standard MCMC simulations of MRFs. Similarly, higher order cliques produce the same consistency between what the update operator $\mathcal{H}$ generates and what the standard MCMC algorithm generates, so the assumed operator form of $\mathcal{H}$ is backwardly compatible with MCMC simulations of standard MRFs with a single sample per node.

Standard MCMC algorithms randomly select a *single* state from the above ensemble of states generated by the action of the update operator $\mathcal{H}$; the probability of a particular state being selected is given by the probability factor that weights that state in the ensemble. More sophisticated MCMC algorithms, known as particle filtering algorithms [3], select *several* states from the ensemble which allows several alternative updates to be simultaneously followed, which allows the probability over alternatives to be represented in a sampled form. However, all of these approaches fit into the same theoretical framework where the update operator $\mathcal{H}$ generates the *full* ensemble of alternatives.

Note that pure states and mixed states are related to doubly distributional population codes [7]. Thus a pure state specifies a single joint state of the MRF nodes, whereas a mixed state specifies a range of alternative joint states of the MRF nodes. The operator algebra presented in this paper provides a complete and consistent framework for using MCMC algorithms to manipulate these pure and mixed MRF states, or equivalently the corresponding doubly distributional population codes.

B. Equilibrium Multi-Sample State

The aim of this section is to demonstrate in detail that the MCMC update operator $\mathcal{H} \equiv \sum_{i=1}^{m} p_i\, a_i{}^\dagger \sum_{j=1}^{m} a_j$ has an equilibrium state which has the same properties as ACEnet [6].

In Section IV A the application of $\mathcal{H}$ to a pure state $\Psi(i_1, i_2, \cdots, i_N)$ converts it into a mixed state (see Equation 4.5). The aim now is to derive the equilibrium mixed state that self-consistently maps to itself under the action of $\mathcal{H}$. This would correspond to a mixed state that contains exactly the right mixture of pure states to balance the hopping rates generated by $\mathcal{H}$. In physics this is known as the *detailed balance* condition. When there is a *single* sample per node this equilibrium mixed state corresponds to the equilibrium ensemble that the standard MCMC update algorithm seeks to generate.

It is *not* possible in general to analytically derive this equilibrium mixed state; if it were then MCMC algorithms would not be needed. This intractability arises because the clique factors cause the samples at neighbouring nodes (i.e. nodes in the same clique) to interact with each other, which leads to the development of *indirect* long-range correlations between nodes by cascading together multiple *direct* short-range interactions (i.e. paths of influence are built out of interlinked clique factors). The summation over all possible paths via which the nodes can interact indirectly with each other is *not* analytically tractable, except in simple cases such as when the nodes interact along a 1-dimensional chain (or any acyclic graph of interactions). More interesting cases, such as 2-dimensional sheets of node interactions, are *not* analytically tractable in general (although there are special cases that are exceptions, such as the 2-dimensional Ising model).

One case which *can* be solved analytically is the case of an MRF with a *single* node that interacts with a fixed external source. In effect, this is an N-node MRF in which $N-1$ of the nodes are frozen, and their influence on the single remaining (unfrozen) node is represented by the external source. This case is interesting because it is the model that is used in the simplest version (i.e. single coding layer) of ACEnet [6]; it is therefore prudent to use the operator methods developed in this paper to verify that the MCMC equilibrium state corresponds to the behaviour that is observed in ACEnet.

The state space of a multiply occupied 1-node MRF is an n-sample histogram. The aim now is to derive the equilibrium state of an n-sample histogram under the action of repeated MCMC samplings generated by $\mathcal{H} = \sum_{i=1}^{m} p_i\, a_i{}^\dagger \sum_{j=1}^{m} a_j$ (see Equation 3.35), where the probabilities p_i are derived from a fixed external source. The equilibrium mixed state Ψ must satisfy the self-consistent bound state equation

$$\left(\sum_{j=1}^{m} p_j\, a_j{}^\dagger\right)\left(\sum_{i=1}^{m} a_i\right)\Psi = \lambda\Psi \tag{4.6}$$

where λ is an eigenvalue. In other words the MCMC update operator must map the equilibrium state into a multiple of itself, as is expected of an equilibrium state. Because correct normalisation of the state and of the MCMC update operator have not been imposed (to avoid lots of distracting normalisation factors appearing in the mathematics), the eigenvalue is not the expected $\lambda = 1$, but nevertheless the value of λ may be readily interpreted (see after Equation 4.15).

The mixed state Ψ can be expanded as a weighted mixture of pure states thus

$$\Psi = \sum_{n_1,n_2,\cdots,n_m} \psi(n_1,n_2,\cdots,n_m) \prod_{k=1}^{m} \left({a_k}^\dagger\right)^{n_k} \; |0\rangle \quad (4.7)$$

where $\left({a_k}^\dagger\right)^{n_k} \; |0\rangle$ is (up to a normalising constant) a histogram with n_k samples in bin k, $\prod_{k=1}^{m} \left({a_k}^\dagger\right)^{n_k} \; |0\rangle$ is (up to a normalising constant) a histogram with occupancy $(n_1,n_2,\cdots,n_m)$, $\psi(n_1,n_2,\cdots,n_m)$ is the probability (up to a normalising constant) of this histogram occurring, and $\sum_{n_1,n_2,\cdots,n_m}(\cdots)$ is a mixture of such histograms. Note that it is not necessary to introduce the normalising constants explicitly because all we are trying to do is to demonstrate that Ψ is a solution of Equation 4.6.

First of all, force the *total* number of samples to be constrained. In physicists' terminology, the case with a fixed number of samples is a *canonical* ensemble, rather than a *grand canonical* ensemble in which the total number of samples would be allowed to vary. Thus write Ψ as

$$\Psi = \sum_{n_1,n_2,\cdots,n_m} \delta_{n,n_1+n_2+\cdots+n_m}\, \psi(n_1,n_2,\cdots,n_m) \prod_{k=1}^{m} \left({a_k}^\dagger\right)^{n_k} \; |0\rangle \tag{4.8}$$

where the Kronecker delta $\delta_{n,n_1+n_2+\cdots+n_m}$ ensures that only terms in $\sum_{n_1,n_2,\cdots,n_m}(\cdots)$ that satisfy the condition $n = n_1+n_2+\cdots+n_m$ can contribute.

Now find the state Ψ that satisfies the consistency condition in Equation 4.6. First substitute Equation 4.8 into the left hand side of Equation 4.6 to obtain

$$\sum_{n_1,n_2,\cdots,n_m} \delta_{n,n_1+n_2+\cdots+n_m}\, \psi(n_1,n_2,\cdots,n_m) \left(\sum_{j=1}^{m} p_j\, {a_j}^\dagger\right) \left(\sum_{i=1}^{m} a_i\right) \prod_{k=1}^{m} \left({a_k}^\dagger\right)^{n_k} \; |0\rangle \tag{4.9}$$

Now use that $a_i \left({a_j}^\dagger\right)^n \; |0\rangle \;=\; n\, \delta_{i,j} \left({a_j}^\dagger\right)^{n-1} \; |0\rangle$ to move all of the annihilation operators to the right in the $\left(\sum_{j=1}^{m} p_j\, {a_j}^\dagger\right)\left(\sum_{i=1}^{m} a_i\right) \prod_{k=1}^{m} \left({a_k}^\dagger\right)^{n_k} \; |0\rangle$ part of the expression in Equation 4.9 to obtain the following simplification

$$\begin{aligned} (\cdots)\; |0\rangle \;&=\; \left(\sum_{j=1}^{m} p_j\, {a_j}^\dagger\right) \sum_{i=1}^{m} n_i \left({a_1}^\dagger\right)^{n_1} \cdots \left({a_i}^\dagger\right)^{n_i-1} \cdots \left({a_m}^\dagger\right)^{n_m} \; |0\rangle & (4.10)\\ &=\; \sum_{j=1}^{m} p_j \left(n_j \left({a_1}^\dagger\right)^{n_1} \cdots \left({a_m}^\dagger\right)^{n_m} \; |0\rangle + \sum_{\substack{i=1\\ i\neq j}}^{m} n_i \left({a_1}^\dagger\right)^{n_1} \cdots \left({a_i}^\dagger\right)^{n_i-1} \cdots \left({a_j}^\dagger\right)^{n_j+1} \cdots \left({a_m}^\dagger\right)^{n_m} \; |0\rangle \right) \end{aligned}$$

where the cases $i = j$ (annihilation and creation within a single bin) and $i \neq j$ (annihilation in one bin and creation in another bin, i.e. hopping) have to be considered separately.

The contribution for a given final state j (but summing over the initial state i) can be represented diagrammatically as follows

$$p_j\, n_j \begin{pmatrix} i\,(=j) \xrightarrow{a_i} & \begin{array}{c} . \\ \Uparrow \\ \text{source} \end{array} & \xrightarrow{{a_j}^\dagger} j \end{pmatrix} + p_j \sum_{\substack{i=1\\ i\neq j}}^{m} n_i \begin{pmatrix} i\,(\neq j) & & \\ & \overset{a_i}{\searrow} & \\ & & \begin{array}{c} . \\ \Uparrow \\ \text{source} \end{array} \xrightarrow{{a_j}^\dagger} j \end{pmatrix}$$

which is a sum of contributions of the form

$$p_j\, n_i \begin{pmatrix} i & & \\ & \overset{a_i}{\searrow} & \\ & & \begin{array}{c} . \\ \Uparrow \\ \text{source} \end{array} \xrightarrow{{a_j}^\dagger} j \end{pmatrix}$$

where the overall factor of n_i comes from the fact that the annihilation operator a_i has n_i samples to choose from in the initial state.

The coefficients of corresponding contributions to the left hand side and right hand side of the equilibrium condition in Equation 4.6 can now be matched up. Note that this matching of coefficients is allowed because the set of states $\prod_{k=1}^{m} \left(a_k{}^\dagger\right)^{n_k} |0\rangle$ is orthogonal and complete (see Section III B 7). This leads to the following consistency equation that interrelates the $\psi(n_1, n_2, \cdots, n_m)$.

$$\sum_{j=1}^{m} p_j \left(n_j\, \psi(n_1, n_2, \cdots, n_m) + \sum_{\substack{i=1 \\ i \neq j}}^{m} (n_i + 1)\, \psi(n_1, \cdots, n_i + 1, \cdots, n_j - 1, \cdots, n_m) \right) = \lambda\, \psi(n_1, n_2, \cdots, n_m) \qquad (4.11)$$

Now define a trial solution to this equation (where $n = n_1 + n_2 + \cdots + n_m$)

$$\psi(n_1, n_2, \cdots, n_m) = \frac{n!}{n_1!\, n_2! \cdots n_m!}\, p_1{}^{n_1}\, p_2{}^{n_2} \cdots p_m{}^{n_m} \qquad (4.12)$$

This trial solution corresponds to placing n samples at random into the histogram, using sampling probabilities $(p_1, p_2, \cdots, p_m)$ for each of the m bins. The probability factor $p_1{}^{n_1}\, p_2{}^{n_2} \cdots p_m{}^{n_m}$ is the probability of each possible way of placing n samples (taking account of the order in which the samples are placed), and the multinomial factor $\frac{n!}{n_1!\, n_2! \cdots n_m!}$ is the number of possible orderings of samples that leave the histogram unchanged (i.e. permute *within* bins but not *between* bins). It is reasonable to expect this to be the solution because the effect of $\mathcal{H}$ (i.e. $\sum_{i=1}^{m} p_i\, a_i{}^\dagger \sum_{j=1}^{m} a_j$) is to randomly annihilate a sample from the histogram, and then to create it again with probability p_i in bin i (which is a *memoryless* operation), so the $\psi(n_1, n_2, \cdots, n_m)$ given in Equation 4.12 should be an equilibrium solution for updates generated by $\mathcal{H}$.

Substitute this trial solution into the consistency equation Equation 4.11 to obtain

$$\sum_{j=1}^{m} p_j \left(\begin{array}{l} n_j\, \frac{n!}{n_1! \cdots n_m!}\, p_1{}^{n_1} \cdots p_m{}^{n_m} \\ + \sum_{\substack{i=1 \\ i \neq j}}^{m} (n_i + 1)\, \frac{n!}{n_1! \cdots (n_i+1)! \cdots (n_j-1)! \cdots n_m!}\, p_1{}^{n_1} \cdots p_i{}^{n_i+1} \cdots p_j{}^{n_j-1} \cdots p_m{}^{n_m} \end{array} \right) = \lambda\, \frac{n!}{n_1! \cdots n_m!}\, p_1{}^{n_1} \cdots p_m{}^{n_m} \qquad (4.13)$$

Cancel the factorials and the probability factors.

$$\sum_{j=1}^{m} p_j \left(n_j + \sum_{\substack{i=1 \\ i \neq j}}^{m} n_j\, \frac{p_i}{p_j} \right) = \lambda \qquad (4.14)$$

Solve this equation for the eigenvalue λ, and use that $\sum_{i=1}^{m} p_i = 1$ and $\sum_{j=1}^{m} n_j = n$ to simplify the result.

$$\begin{aligned} \lambda &= \sum_{j=1}^{m} p_j\, n_j + \sum_{j=1}^{m} \sum_{\substack{i=1 \\ i \neq j}}^{m} p_i\, n_j \\ &= \sum_{j=1}^{m} p_j\, n_j + \left(\sum_{i,j=1}^{m} p_i\, n_j - \sum_{j=1}^{m} p_j\, n_j \right) \\ &= \left(\sum_{i=1}^{m} p_i \right) \left(\sum_{j=1}^{m} n_j \right) \\ &= n \end{aligned} \qquad (4.15)$$

Thus $\lambda = n$ which is the (fixed) total number of samples in the histogram. The source of this factor is $\mathcal{H} \equiv \sum_{i=1}^{m} p_i\, a_i{}^\dagger \sum_{j=1}^{m} a_j$, where each annihilation operator a_j has n_j to choose from in the initial state, so the sum of annihilation operators $\sum_{j=1}^{m} a_j$ generates $\sum_{j=1}^{m} n_j = n$ separate contributions. The fact that λ is a constant means that the consistency equation (i.e. Equation 4.11) has an eigenvalue λ that is *independent* of the choice of $(n_1, n_2, \cdots, n_m)$, which means that the update operator $\mathcal{H}$ has the *same* effect on each pure state component of the equilibrium state Ψ (as is required in order for Ψ to satisfy Equation 4.6).

The result in Equation 4.15 verifies that the trial solution proposed in Equation 4.12 is correct, and that the equilibrium histogram state corresponds to placing n samples at random into the histogram using sampling probabilities $(p_1, p_2, \cdots, p_m)$ for each of the m bins.

Summarise these results:

1. Basic MCMC update operator: $\mathcal{H} = (\sum_{j=1}^{m} p_j\, a_j{}^\dagger)(\sum_{i=1}^{m} a_i)$

2. General state (fixed n): $\Psi = \sum_{n_1,n_2,\cdots,n_m} \delta_{n,n_1+n_2+\cdots+n_m}\, \psi(n_1,n_2,\cdots,n_m) \prod_{k=1}^{m} \left(a_k{}^{\dagger}\right)^{n_k} |0\rangle$
3. Equilibrium condition: $(\sum_{j=1}^{m} p_j\, a_j{}^{\dagger})(\sum_{i=1}^{m} a_i)\,\Psi = \lambda\,\Psi$
4. Equilibrium state: $\psi(n_1,n_2,\cdots,n_m) = \frac{n!}{n_1!\,n_2!\cdots n_m!}\, p_1{}^{n_1}\, p_2{}^{n_2}\cdots p_m{}^{n_m}$ with $\lambda = n$

The equilibrium state is a *mixture* of pure states, where each pure state is weighted by the probability of its occurrence. In this approach the state Ψ of the system corresponds to the *entire* probability-weighted ensemble of alternative histograms. In effect, these histograms mix with each other under the updating action of the fixed external source that causes the samples in the bins of each histogram to hop from bin to bin, whilst conserving the total number of samples in the histogram (i.e. there is migration of samples but no birth or death of samples). The equilibrium condition ensures that the mixing that occurs due to the hopping of samples has no net effect on the probability-weighted ensemble of alternative histograms.

This completes the demonstration that the simplest (i.e. a single node) multiple occupancy MRF has the same properties as ACEnet [6], which is *defined* as having an equilibrium state that is generated by the random (but probability-weighted) placement of n samples into a set of histogram bins. Also, larger SONs can be built out of multiple linked ACEnet modules, and these correspond to MRFs with a larger number of nodes. This unification of MRFs and SONs is possible because both approaches can be viewed as implementing algorithms for manipulating samples in histogram bins, and all such algorithms can be expressed by using the algebra of creation and annihilation operators. A key advantage of this MRF/SON unification is that the techniques that are used to train SONs (i.e. to discover structure in data) can now be used to train MRFs, which allows the MRF graph structure (i.e. nodes and connections) to adapt itself so that it is better matched to the data it is trying to model.

The MCMC updating of MRFs whose nodes are occupied by multiple samples potentially leads to lots of interesting properties. The derivation above shows how a single node MRF behaves under the influence of a *fixed* external source, but more interesting behaviour occurs when either the MRF has a single node but the external source is *variable*, or if the MRF has multiple interacting nodes so that each node sees the variable state of the other nodes. This last case is especially interesting in MRFs that are trained as SONs, because it leads to behaviours in which the samples that occupy the nodes act collectively, and thus cause the joint node states to behave like extended symbols (see Section II D for some diagrams that illustrate this point in more detail).

V. CONCLUSIONS

The work described in this paper assumes that Markov random field models are used to implement Bayesian inference. The key contribution of this paper is an implementation using creation and annihilation operators of MCMC algorithms for simulating MRFs. This theoretical framework has a similar structure to that used in quantum field theories of bosons in physics [9]. An equilibrium solution of the MCMC update operator is derived which is shown to be equivalent to the equilibrium behaviour of the adaptive cluster expansion network (ACEnet) [6], which is a type of self-organising network that computes using discrete-valued quantities.

This point of contact between MRF theory and SON behaviour allows the theories of these two fields to be unified. Although MRFs and SONs are superficially *different* (MRFs have one sample per node, whereas ACEnet SONs have multiple samples per node), the underlying operators that are used to manipulate them are the *same.* MRF theory could benefit from this unification by being able to make use of SONs to build MRF networks in a data-driven way. SON theory could benefit from this unification by being able to make full use of the rich theoretical theory of MRFs.

It is very convenient that MRFs and SONs are unified within a QFT framework, because such theories are used extensively by physicists to describe the interaction of particles, and many techniques have been developed to compute results using such theories. We have found that it is very easy to transfer knowledge from QFT to the unified MRF/SON framework presented in this paper. Also, the diagrammatic notation (i.e. Feynman diagrams) makes it much easier to understand what MCMC algorithms are actually doing, without becoming submerged in large amounts of theory.

Future papers in this “discrete network dynamics” series of papers will focus in detail on the consequences of implementing MCMC algorithms using update operators built out of creation and annihilation operators.

VI. ACKNOWLEDGEMENTS

This research presented in this paper was supported by the United Kingdom’s MoD Corporate Research Programme.

[1] J Besag, *Spatial interaction and the statistical analysis of lattice systems*, Journal of the Royal Statistical Society: Series B **36** (1974), no. 2, 192–236.

[2] R T Cox, *Probability, frequency and reasonable expectation*, American Journal of Physics **14** (1946), no. 1, 1–13.

[3] A Doucet, S Godsill, and C Andrieu, *On sequential Monte*

Carlo sampling methods for Bayesian filtering, Statistics and Computing **10** (2000), no. 3, 197–208.

[4] S Geman and D Geman, *Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images*, IEEE Transactions on Pattern Analysis and Machine Intelligence **6** (1984), no. 6, 721–741.

[5] P J Green, *Reversible jump Markov chain Monte Carlo computation and Bayesian model determination*, Biometrika **82** (1995), no. 4, 711–732.

[6] S P Luttrell, *A discrete firing event analysis of the adaptive cluster expansion network*, Network: Computation in Neural Systems **7** (1996), no. 2, 285–290.

[7] M Sahani and P Dayan, *Doubly distributional population codes: simultaneous representation of uncertainty and multiplicity*, Neural Computation **15** (2003), no. 10, 2255–2279.

[8] P Smyth, *Belief networks, hidden Markov models, and Markov random fields: a unifying view*, Pattern Recognition Letters **18** (1998), no. 11-13, 1261–1268.

[9] A Zee, *Quantum field theory in a nutshell*, Princeton University Press, 2003.

Guided Tour of Publications [*]

Stephen Luttrell

This is an informal walk-through the research context of all of the papers that I wrote between 1981 and 2007. Nearly all of the papers are single-authored and interconnected so this document is self-contained.

1981-1982: PhD

Papers Luttrell and Wada [1], Luttrell et al. [2], Luttrell and Wada [3] (written in collaboration with others) and PhD dissertation Luttrell [4] apply quantum chromodynamics to modelling the quark/gluon structure of hadrons seen in deep inelastic scattering experiments. Various higher order effects that modify lower order perturbative results are studied both phenomenologically and theoretically. The operator algebra techniques used in this PhD research unexpectedly turned out to be very useful for the description of Markov chain Monte Carlo methods, as described in my publications from 2005 onwards.

1984-1989: Bayesian Super-Resolution and Mutual Information (scattered field)

Report Luttrell and Oliver [5] describes how linear least squares error reconstruction using an appropriately weighted reconstruction space can be used to introduce prior knowledge to enhance the resolution of coherent images. The material in this report was not published because it was quickly overtaken by the full Bayesian treatment described below.

Paper Luttrell [6] describes how Bayes' theorem may be used to derive the solution to linear inverse problems under the assumption of Gaussian probability density functions (PDF). For suitable Bayesian priors this leads to super-resolution where details on a scale shorter than the Rayleigh resolution length become visible.

Book chapter Luttrell and Oliver [7] and report Luttrell [8] present an introduction to the use of prior knowledge in the analysis of SAR images. This was patented in Luttrell and Oliver [9].

Papers Luttrell [10, 11] introduce the principle of mutual information maximisation as a way of optimising the extraction of information from data.

Paper Luttrell and Oliver [12] is on clutter and targets in SAR images. My contribution showed how to use a generalisation of the super-resolution techniques originally developed in Luttrell [6] to analyse images of targets, and showed how to use mutual information (as in Luttrell [10, 11]) to interpret the information processing in terms of information channels. A brief description of this work is in Luttrell and Oliver [13].

Report Luttrell [14] describes an analysis of images of point targets in images produced by the Royal Signals and Radar Establishment (RSRE) SAR. The purpose of this was to accurately calibrate the point spread function (PSF) of the RSRE SAR so that its images could be robustly super-resolved. However, this analysis led me to the conclusion that for the RSRE SAR the PSF was itself a function of the data; in other words the system response was *non-linear*. This totally invalidated the super-resolution theory developed thus far, which assumed that the system response was *linear*, which then triggered a shift of my research away from super-resolution. A side effect of this non-linearity was that the higher moments of the image data were biased away from their naïve values, and thus should *not* have been used in their uncorrected form to deduce the statistics of the underlying clutter model.

Report Luttrell [15] discusses the intimate connection between super-resolution and the analysis of phase information in SAR data.

[*]Typeset in LaTeX on May 6, 2019.
This guided tour was originally written for my curriculum vitae.

Paper Delves et al. [16] and Conference papers Pryde et al. [17, 18] describes the results of collaborative work on a parallel processing implementation of these super-resolution techniques for analysing synthetic aperture radar (SAR) images.

Book chapter Luttrell [19] (delayed for several years in publication) reviews the ideas behind the use of information channels, drawing together ideas from mutual information, super-resolution, and Bayesian inference.

1985-1988: Markov Random Fields for Clutter and Texture Modelling

Report Luttrell [20] is the first in a long series of publications on the use of Markov random fields (MRF) to build probability density function (PDF) models, which describes of how to train MRF models using a generalisation of the Boltzmann machine learning algorithm that I called the Gibbs Machine. Various generalisations were discussed such as the idea of enhancing an MRF model by using these learning techniques to attach a "brain graft" to the MRF model to patch it up wherever it did not give a good enough approximation to the data.

Paper Luttrell [21] describes how to do Monte Carlo sampling of arbitrarily complicated MRFs by bit-flipping operations that could be easily implemented in hardware.

Report Luttrell [22] uses elementary PDF methods to measure radar sensitivity in a way that is invariant w.r.t. the receiver law.

Paper Luttrell [23] presents a careful analysis of the use of MRF models for texture modelling, using arguments based on sufficient statistics to understand how the texture information is spread out amongst the various measured statistics. This is conceptually similar to the use of information channels discussed in Luttrell and Oliver [12]. Contact with image processing techniques is made via the grey level co-occurrence matrix method and the WISARD n-tuple processing network, both of which are special cases of this MRF analysis.

Conference papers Luttrell [24, 25, 26, 27] develop various aspects of the use of MRFs for texture and clutter modelling.

Paper Luttrell [28] shows how the maximum entropy method can be used to pick good statistics to measure in textured images (as introduced in Luttrell [23]), so that an efficient MRF texture model can be built. An efficient algorithm (which does *not* require the fixing of Lagrange multipliers, unlike other approaches) is given for deciding what new statistics to add to the MRF model by comparing *synthetic* textures generated using the current MRF model with the *real* texture to be modelled. As a corollary, a generalisation of the original Boltzmann machine learning algorithm is given for arbitrary MRFs.

Conference paper Luttrell [29] compares and contrasts the MRF approach (as in Luttrell [28]) and a proposed hierarchical "cluster-decomposition" approach to modelling PDFs. The generalised form of the Boltzmann machine learning algorithm is presented in detail. The advantages of the cluster decomposition approach (particularly that it does *not* use Monte Carlo simulations) are explained.

Report Luttrell [30] analyses the optimisation of hidden Markov models (HMM) using Gibbs Machine methods.

Conference paper Luttrell [31] summarises the MRF approach in the context of modelling texture in radar images.

1989-1991: Bayesian Super-Resolution (scatterers themselves)

Paper Luttrell [32] extends the earlier Bayesian super-resolution method (see Luttrell [6]) by pointing out the fact that the inverse problem to solve should be the reconstruction of the object *scatterers* that produce the object field, and *not* the reconstruction of the object *field* itself. In report Luttrell [33] and paper Luttrell [34] this work is greatly extended by using the expectation-maximisation (EM) method to derive an iterative object reconstruction algorithm from first principles. This allows both super-resolution and auto-focusing to be handled within the same framework. This work was reviewed in Luttrell [35]. Report Luttrell [36] shows how the iterated EM method can be applied to reconstructing the object scatterers under the assumption that they produce a K-distributed object field.

1988-1989: Hierarchical Self-Organising Maps

Conference paper Luttrell [37] describes a hierarchical generalisation of the standard Kohonen self-organising map (SOM) network, and introduces the idea of the "growing grid" method of training SOMs. Conference paper Luttrell [38]) and paper Luttrell [39] train a tree of linked SOMs to encode SAR images. In paper Luttrell [40] this approach is described in more detail.

1986-1990: Digital Receiver

Report Luttrell and Pritchard [41] and paper Luttrell and Pritchard [42] describe the analysis, design and testing of a finite state machine for demodulating signals. My contribution was the analysis. This is essentially an early type of software radio receiver. This was patented in Luttrell and Pritchard [43].

1989-1992: Distortion Minimising Encoders (single SOM)

Conference papers Luttrell [44, 45] and paper Luttrell [46] present a novel derivation of SOM networks, based on a generalisation of the standard Linde-Buzo-Gray (LBG) vector quantisation algorithm to account for distortion on the communication channel, which approximates the standard Kohonen SOM algorithm as a special case (the distribution of distortions corresponds to the topographic neighbourhood function). The biggest advantage of the approach used is that it is based on minimisation of an objective function (unlike Kohonen's SOM), which thus allows many properties to be theoretically derived. The density of code vectors in this type of SOM (for 1-dimensional data) is derived in report Luttrell [47] and paper Luttrell [48], where the density is shown to be the *same* as in a standard vector quantiser (unlike the standard Kohonen SOM which leads to a *different* density which inconveniently depends on the choice of topographic neighbourhood function). A generalisation of this result to data of arbitrary dimensionality was given in report Luttrell [49]. Report Luttrell [50] shows how to use a vector quantiser to encode data under the assumption that its noise is K-distributed.

1988-1994: Lateral Mutual Information Maximising Hierarchical Encoders

Reports Luttrell [51, 52] and in the arXiv papers Luttrell [53, 54] show how optimisation of the hierarchical "cluster decomposition" introduced in conference paper Luttrell [29] can be used to detect anomalies in textured images. Conference paper Luttrell [55] presents a detailed analysis of this optimisation process, where the objective function used is the relative entropy between the model PDF and the true PDF, which is equivalent to the *sum* of the (lateral) mutual informations between various nodes in each layer of the network. This approach has been patented Luttrell [56].

1990-1992: Distortion Minimising Encoders (coarse-grain parallel SOMs)

Report Luttrell [57] and conference paper Luttrell [58] use a communication channel model of SOMs (see Luttrell [46]) to analyse networks of connected SOMs, and this led to paper Luttrell [59]. The model used is one in which two communication channels mutually interfere thus causing distortion that is correlated between the channels. This distortion defines the topographic neighbourhood functions that are used in the SOMs (that are the channel encoders) in the first place. The optimisation of the SOMs is influenced by mutual interactions between their outputs, hence the description of this type of training as "self-supervised".

1988-1994: Some Consolidation

Report Luttrell [60] and conference paper Luttrell [61] analyse adaptive n-tuple networks (e.g. WISARD) by using a relative entropy objective function to optimise a PDF model. Both supervised and unsupervised methods of training such networks emerge naturally from this analysis.

Report Luttrell [62] and conference paper Luttrell [63] review the Bayesian approach to training and using neural networks. Various models are discussed including the "adaptive cluster expansion" model originally introduced in conference paper Luttrell [29].

Conference paper Luttrell [64] presents a rigorous Bayesian analysis of the "adaptive cluster expansion" model.

1992-1994: Partitioned Mixture Distributions

Report Luttrell [65] and conference papers [66, 67] describe a generalisation (the *partitioned* mixture distribution, or PMD) of the standard mixture distribution used to model PDFs, and this led to paper Luttrell [68]. In image processing (the statistical properties of) each correlation area of an image can be modelled with a mixture distribution, and it would be convenient if all of these models could be collected together in a single translation invariant architecture. A nice solution to this problem is to build each mixture distribution using components selected from a common pool of mixture components, where each correlation area has a complete repertoire of components needed for its analysis. Mixture distributions that see overlapping areas of image have a lot of overlap in the components they use. A PMD can be trained using a generalisation of the EM method for training standard mixture distributions. Trained PMDs have many of the properties that are observed in the low-level visual cortex, such as a complete repertoire of processing machinery for each local patch of image.

1990-1994: Folded Markov Chains

Report Luttrell [69] introduces the folded Markov chain (FMC) technique, and paper Luttrell [70] uses it to give a Bayesian analysis of the SOM model originally described in Luttrell [46]. In an FMC information is passed along a Markov chain, and then Bayes' theorem is used to pass (virtually) the information back along the chain in the opposite direction until it reaches the start again. This final reconstruction of the input to the chain is compared with the original input, and the transitions in the Markov chain are adjusted to minimise the Euclidean reconstruction distortion (on average). The tight constraints that Bayes' theorem imposes on the various probabilities make it possible to derive a training algorithm that reduces to the algorithm in Luttrell [46] when the encoder is *deterministic*. This FMC analysis unifies all my earlier work on networks of SOMs, and by allowing *probabilistic* encoders to be handled in the same framework, it sets the scene for future work.

1993-1995: Some Consolidation

Reports Luttrell [71, 72, 73] describe various ways in which mixture distributions can be used in the analysis of images.

Conference paper Luttrell [74] and report Luttrell [75] shows the application of SOM techniques to the analysis of data (e.g. range profiles of radar returns from ships) that lies on a manifold with a circular topology. Prior knowledge of the topology is used to define an appropriate topographic neighbourhood function for use in the SOM.

1994-1997: Unification of FMC and PMD

I celebrate "FMCPMD Day" on 17th June each year – year zero was in 1994 – in recognition of the day that I finally realised that the correct objective function for optimising a PMD should focus on minimising the reconstruction distortion rather than optimising the PDF. More generally, distortion-minimisation rather than PDF-optimisation is the correct (and much simpler) way forward, and the reason it took me so long to realise this was because the mathematics of PDF-optimisation is so much prettier than the mathematics of distortion-minimisation.

Book chapter Luttrell [76] (delayed for several years in publication), reports Luttrell [77, 78, 79] and arXiv paper Luttrell [80] combine the FMC approach (introduced in Luttrell [70]) with the PMD approach (introduced in Luttrell [68]) to create an encoder structure that is appropriate for the analysis of images, where the form of the PMD posterior probability allows the encoder to break into local pieces that each encode a local patch of the image. This is then applied to the analysis of images derived from multiple sources. In the case of *pairs* of images the network learns a structure that closely resembles the dominance stripes and orientation maps observed in the visual cortex. The important point is that you can obtain these "biological" results as a consequence of the general properties of encoders – to emphasise this the ancronym VICON (VIsual COrtex Network) was coined in Luttrell [79, 80].

Report Luttrell [81] and conference papers [82, 83] present a detailed analysis of some of the properties of PMDs (introduced in Luttrell [68]). A dynamical PMD (which has the same relationship to a static PMD that a hidden Markov model has to a mixture distribution) is analysed in detail, and is successfully applied to the problem of tracking a weak target in clutter. A first order perturbation analysis reveals the low-order properties of PMDs and allows their behaviour to be related to various neural networks.

arXiv paper Luttrell [84], report Luttrell [85], and conference paper Luttrell [86] apply the FMC approach (introduced in Luttrell [70]) to optimising encoders that output several statistically independent (given the input) codes. Various analytic optimum solutions are obtained. However, these solutions are *less* useful than those that are obtained if the optimisation is constrained in various ways, which is the focus of all subsequent papers in this area.

1997: "State of the Nation"

Report Luttrell and Webber [87] summarises all publications produced during the period 1994-1997.

1997: Isaac Newton Neural Networks Programme

Report Luttrell [88] and arXiv paper Luttrell [89] describes the work I did at the Neural Networks research programme at the Isaac Newton Institute. I analysed the optimisation of an objective function depending on the *joint* PDF of the state of a multi-layer network, and showed that this unified all of my work on the optimisation of hierachical encoders.

arXiv paper [90] summarises various properties of this method of optimising a network.

1994-1997: Distortion Minimising Encoders (fine-grain parallel)

Book chapters Luttrell [91, 92] and paper Luttrell [93] give the generalisation of the FMC approach of Luttrell [70] to the case where multiple codes are output by a *probabilistic* encoder for each of its input vectors. Up to this point in my work the encoders have each output a *single* code, and thus each acts as a winner-take-all network. The advantage of allowing the possibility of *multiple* codes is that the encoded information can be split across more than one channel. This is conceptually similar to the use of information channels discussed in Luttrell and Oliver [12]. Further, if these codes are chosen stochastically, then by optimising the FMC (minimum Euclidean distortion) the network can decide for itself how many information channels it needs to use. This freedom allows the network to optimise its own architecture. In Luttrell [93] a simple application of these ideas is analysed in detail, where various (supposedly) optimal choices of information channel are compared with each other, and the conditions under which each type of solution is optimal are established. An important cross-disciplinary result is that the above extension to *multiple* codes allows contact to be made with neural network models based on discrete firing events, where the recent history of which neurons have fired corresponds to the set of codes that are currently active.

Paper Luttrell [94] applies the multiple sampling ideas of Luttrell [93] to the hierarchical vector quantisation network Luttrell [40]. Although the hierarchical network of Luttrell [40] has its structure manually defined, rather than learnt via optimisation (as *could* be done using Luttrell [93]), the earlier ideas in [40] automatically fit into the framework of the later theory in Luttrell [93].

Change in Publication Style

From here on there are currently no further journal papers, because delays in publication caused by the refereeing process had become excessive. However, the research continues to be published in conference papers and technical reports.

1998: Some Consolidation

Report Luttrell [95] describes various ways of training self-organising encoder networks by modelling their output as the posterior probability over samples (or "firing events") drawn from a stochastic code book. The different training schemes correspond to different conditions under which the samples are drawn.

1998-1999: Distortion Minimising Encoders (analytic optimisation)

Conference paper Luttrell [96] shows how *analytic* optimisation of a stochastic encoder (an FMC with multiple output codes) leads to optimal encoders being described by *piecewise linear* posterior probabilities. This emerges because the reconstruction is a *linear* superposition of contributions, and the objective function is a *Euclidean* distortion. This piecewise property also holds in networks of linked stochastic encoders. This piecewise linearity will prove to be very useful for deriving analytic solutions to various problems.

arXiv paper Luttrell [97], book chapter Luttrell [98], and report Luttrell [99] use the piecewise linear property of optimal stochastic encoders (see Luttrell [96]) to derive optimal encoding schemes for circular and toroidal manifolds. The results derived in this paper were obtained analytically by using *Mathematica* to manipulate the various cumbersome piecewise linear expressions leading to compact results. The circle and the torus are simple curved manifolds that serve as models for the more complicated curved manifolds that arise in signal and image processing. For the toroidal manifold *two* types of optimal encoder emerge depending on the size of the encoder and the number of stochastic codes that it is allowed to output. On the one hand a *joint* encoder is optimal if there are more than a certain minimum number of codes to choose from; this corresponds to an encoder that has so many resources that it can simply partition the torus into localised code cells. On the other hand a *factorial* encoder emerges if the number of codes to choose from is severely reduced *and* the number of codes it is allowed to output is sufficiently large; this corresponds to an encoder that is starved of resources but which is allowed to have several trials at outputting a code, and which therefore chooses to use anisotropic code cells to slice the torus into localised code cells defined by the regions of intersection of *pairs* of anisotropic code cells. The automatic emergence of factorial coding is a crucial property in a network that aims to discover *for itself* what information channels to use when processing data.

1999: Some Consolidation

Reports [100, 101] motivate the use of self-organising networks for discovering and extracting information in complicated data, with an emphasis on their potential application to data fusion. Report Luttrell [102] is a user's guide to stochastic encoders, and contains many simple examples of their use. arXiv paper Luttrell [103] is a comprehensive summary of the theory and many numerical simulations of stochastic encoders, and arXiv paper Luttrell [104] contains a small subset of these results.

2000: "State of the Nation 2"

Report [105] summarises all publications produced during the period 1997-2000.

2000-2003: Distortion Minimising Encoders (jammer suppression)

Conference paper Luttrell [106] applies a stochastic encoder to the problem of separating a signal from a jammer. Because the jammer is much stronger than the signal the optimisation concentrates on encoding the jammer alone. This allows the trained encoder to separate the jammer and signal subspaces (in this case these are *curved* subspaces), which allows the signal to be cleanly detected. This material is also covered in report Luttrell [107], conference papers Luttrell [108, 109], arXiv papers (extended versions of the preceding conference papers) Luttrell [110, 111], and book chapters [112, 113].

2001: Distortion Minimising Encoders (classifier fusion)

Conference paper Luttrell [114] and workshop paper [115] demonstrate some of the key properties of stochastic encoders. One of these is their ability to learn factorial codes in which a code book breaks into separate smaller code books, each of which encodes only one part of the input. Another is the natural way in which supervision can be introduced to steer the code books towards coding their inputs in particular ways. This is all presented in the context of multiple classifier fusion, where the processing is split across multiple information channels and then fused together again at the end.

2001-2004: Internal Reports (part 1)

Report Luttrell [116] describes how to use a stochastic encoder to suppress sea clutter in coherent images, which allows targets that are masked by the clutter to be revealed. The approach used is essentially the same as for separating a signal from a jammer (as in conference paper Luttrell [106]).

Report Luttrell [117] compares and contrasts the use of encoders and the use of PDFs for modelling (and suppressing) sea clutter in images.

Reports [118, 119] summarise all of my research results on self-organising networks.

Reports Luttrell [120, 121] describe various interesting and useful properties of the self-organising network known as ACEnet (Adaptive Cluster Expansion network).

Reports Luttrell [122, 123, 124] describe how to apply self-organising networks to the problem of combining the outputs of multiple classifiers.

2003: Isaac Newton Neural Networks Programme (published, at last!)

Conference paper Luttrell [125] introduces a new way of understanding networks of linked stochastic encoders. These ideas were originally described in report Luttrell [88]. Rather than modelling each encoder as an FMC (see Luttrell [70] and its generalisation to multiple codes in Luttrell [93]) and building up the overall network objective function as a *sum* of local contributions, the *whole* network of linked encoders is deemed to have a *single* objective function which compares the joint PDF of the whole network under two different modelling assumptions. For a chain-like network the two PDF models are built from conditional probabilities acting in opposite directions along the chain, and it is similar to Dayan and Hinton's Helmholtz machine *except that* the aim here is to optimise the *joint* network probability rather than only the *marginal* input probability. This approach is then shown to reduce to the FMC approach wherever that had been used earlier. An application of these ideas to training a network on hierarchically correlated data is presented, and the encoders (the bottom-up PDF models) automatically split into smaller encoders in precisely the way that is expected (i.e. process the most uncorrelated pieces separately, then progressively fuse the results until only the most correlated piece is left). This approach is thus capable of making *structural* changes to the network as it is trained, where it creates information channels for doing the lower-level processing, and then progressively fuses these channels to do the higher-level processing.

2003-2004: Distortion Minimising Encoders (multiple correlation scales)

arXiv paper Luttrell [126] describes the use of self-organising stochastic encoders for learning the structure of data manifolds. The main aim of this paper was to demonstrate the automatic separation of correlated components in the data, and their subsequent fusion whilst preserving only their dominant degree(s) of freedom, as was described in Luttrell [125]. This is a key paper that opens up a very fruitful area of research.

2005: Discrete Network Dynamics

arXiv paper [127] describes the use of operator algebra techniques to reformulate MCMC algorithms, so that the algorithms can be manipulated by purely algebraic methods, which would have applications in adaptive networks. This is a key paper that opens up a very fruitful area of research.

2005-2007: Internal Reports (part 2)

Report Luttrell [128] comprehensively reviews MRFs and SONs from the perspective of adaptive filters and recurrent networks, and introduces an operator algebra to describe the associated Markov chain Monte Carlo (MCMC) algorithms.

Reports Luttrell [129, 130] apply the operator algebra methods of [127, 128] to recurrent networks, dealing with both the small-occupancy (i.e. dominated by quantisation effects) and large-occupancy (i.e. quantisation effects negligible) cases.

Report Luttrell [131] describes a proposed application of recurrent SONs to the problem of language identification.

Report Luttrell [132] introduces symbolic algebra techniques for manipulating the operator algebra of MCMC algorithms.

Report Luttrell [133] presents a *Mathematica* implementation of recurrent SONs.

The appendix of report [134] presents some ideas on how to use operator algebra methods and self-organising networks to learn the structure of networks.

[1] S P Luttrell and S Wada. Heavy quark production in electromagnetic deep inelastic scattering and qcd analysis of isosinglet moments. *Nuclear Physics B*, 182(3):381–396, 1981.

[2] S P Luttrell, S Wada, and B R Webber. The Wilson coefficient functions of four quark operators and the four quark process in deep inelastic scattering. *Nuclear Physics B*, 188(2):219–232, 1981.

[3] S P Luttrell and S Wada. The current product expansion of the two quark process at the twist four level. *Nuclear Physics B*, 197(2):290–306, 1982.

[4] S P Luttrell. *Power law corrections to hadronic structure functions*. PhD thesis, Cambridge University, 1982.

[5] S P Luttrell and C J Oliver. Resolution enhancement in coherent images by the introduction of prior knowledge. Memorandum 3697, Royal Signals and Radar Establishment, Malvern, 1984.

[6] S P Luttrell. Prior knowledge and object reconstruction using the best linear estimate technique. *Optica Acta*, 32(6): 703–716, 1985.

[7] S P Luttrell and C J Oliver. *RSRE Research Review*, chapter Prior knowledge in synthetic aperture radar (SAR) processing, pages 73–77. RSRE, 1985.

[8] S P Luttrell. A super-resolution model for synthetic aperture radar. Memorandum 3785, Royal Signals and Radar Establishment, Malvern, 1985.

[9] S P Luttrell and C J Oliver. Super resolution imaging system. Patent GB 2173663 B, 1985.

[10] S P Luttrell. A new method of sample optimisation. *Optica Acta*, 32(3):255–257, 1985.

[11] S P Luttrell. The use of transinformation in the design of data sampling schemes for inverse problems. *Inverse Problems*, 1(3):199–218, 1985.

[12] S P Luttrell and C J Oliver. Prior knowledge in synthetic aperture radar processing. *Journal of Physics D: Applied Physics*, 19(3):333–356, 1986.

[13] S P Luttrell and C J Oliver. The role of prior knowledge in coherent image processing. *Philosophical Transactions of the Royal Society A*, 324(1579):397–407, 1988.

[14] S P Luttrell. The complex point spread function of the RSRE SAR. Memorandum 4079, Royal Signals and Radar Establishment, Malvern, 1987.

[15] S P Luttrell. The relationship between super-resolution and phase imaging of SAR data. Research Note BS1/41, Royal Signals and Radar Establishment, Malvern, 1987.

[16] L M Delves, G Pryde, and S P Luttrell. A super-resolution algorithm for SAR images. *Inverse Problems*, 4(3):681–703, 1988.

[17] G C Pryde, L M Delves, and S P Luttrell. The performance of a parallel super-resolution algorithm for synthetic aperture radar images. In *Proceedings of IMA conference on mathematics in signal processing*, pages 739–747, Oxford, 1988. University Press.

[18] G C Pryde, L M Delves, and S P Luttrell. A comparative study of the AMT DAP and of Transputer array architectures for the super-resolution of synthetic aperture radar images. In *Proceedings of conference on parallel processing*, pages 340–350, Cambridge, 1988. University Press.

[19] S P Luttrell. *Mathematics in remote sensing*, chapter Inference theory, pages 205–222. The Institute of Mathematics and its Applications Conference Series. Clarendon Press, 1989.

[20] S P Luttrell. The implications of Boltzmann-type machines for SAR data processing: a preliminary survey. Memorandum 3815, Royal Signals and Radar Establishment, Malvern, 1985.

[21] S P Luttrell. An optimisation of the Metropolis algorithm for multibit Markov random fields. *Inverse Problems*, 2(2): L15–L17, 1986.

[22] S P Luttrell. A proposal for a new method of calibrating radar sensitivity. Research Note BS1/37, Royal Signals and Radar Establishment, Malvern, 1987.

[23] S P Luttrell. The use of Markov random field models to derive sampling schemes for inverse texture problems. *Inverse Problems*, 3(2):289–300, 1987.

[24] S P Luttrell. Markov random fields: a strategy for clutter modelling. In *Proceedings of AGARD conference on scattering and propagation in random media*, pages 7.1–7.8, Rome, 1987. AGARD.

[25] S P Luttrell. The use of Markov random field models in sampling scheme design. In *Proceedings of SPIE international symposium on inverse problems in optics*, New York, 1987. SPIE.

[26] S P Luttrell. *Radar 87*, chapter Designing Markov random field structures for clutter modelling, pages 222–226. IEE, London, 1987.

[27] S P Luttrell. Markov random field image models of targets and clutter. In *Proceedings of US/UK workshop on parallel processing*, pages 771–793, Shrivenham, 1987.

[28] S P Luttrell. A maximum entropy approach to sampling function design. *Inverse Problems*, 4(3):829–841, 1988.

[29] S P Luttrell. *Maximum entropy and Bayesian methods*, chapter The use of Bayesian and entropic methods in neural network theory, pages 363–370. Kluwer, Dordrecht, 1989.

[30] S P Luttrell. The Gibbs Machine applied to hidden Markov model problems. Part 1: Basic theory. Research Note SP4/99, Royal Signals and Radar Establishment, Malvern, 1989.

[31] S P Luttrell. An adaptive Bayesian network for texture modelling. In *Proceedings of the IEE seminar on texture analysis in radar and sonar*, pages 6.1–6.10, London, 1993. IEE.

[32] S P Luttrell. The inverse cross section problem for complex data. *Inverse Problems*, 5(1):35–50, 1989.

[33] S P Luttrell. A Bayesian derivation of an iterative autofocus/super-resolution algorithm. Memorandum 4337, Royal Signals and Radar Establishment, Malvern, 1989.

[34] S P Luttrell. A Bayesian derivation of an iterative autofocus/super-resolution algorithm. *Inverse Problems*, 6(6):975–996, 1990.

[35] S P Luttrell. The theory of Bayesian super-resolution of coherent images: a review. *International Journal of Remote Sensing*, 12(2):303–314, 1991.

[36] S P Luttrell. An EM approach to iterative Bayesian super-resolution with applications to K-distributed data. Research Note SP4/102, Royal Signals and Radar Establishment, Malvern, 1989.

[37] S P Luttrell. Self organising multilayer topographic mappings. In *Proceedings of IEEE conference on neural networks*, pages 93–100, San Diego, 1988. IEEE.

[38] S P Luttrell. Image compression using a neural network. In *Proceedings of IGARSS conference on remote sensing*, pages 1231–1238, Edinburgh, 1988. ESA.

[39] S P Luttrell. Image compression using a multilayer neural network. *Pattern Recognition Letters*, 10(1):1–7, 1989.

[40] S P Luttrell. Hierarchical vector quantisation. *Proceedings of the IEE I*, 136(6):405–413, 1989.

[41] S P Luttrell and J A S Pritchard. A compact digital communications system - Part 1: rapid carrier acquisition. Memorandum 3925, Royal Signals and Radar Establishment, Malvern, 1986.

[42] S P Luttrell and J A S Pritchard. Rapid acquisition of low signal-to-noise carriers. *IEE Proceedings I*, 136(1):100–108, 1989.

[43] S P Luttrell and J A Pritchard. Signal transforming device. Patent GB 2190221 B, 1987.

[44] S P Luttrell. Hierarchical self-organising networks. In *Proceedings of IEE Conference on Artificial Neural Networks*, pages 2–6, London, 1989. IEE.

[45] S P Luttrell. Self-organisation: a derivation from first principles of a class of learning algorithms. In *Proceedings of IJCNN international joint conference on neural networks*, pages 495–498, Washington DC, 1989. IEEE.

[46] S P Luttrell. Derivation of a class of training algorithms. *IEEE Transactions on Neural Networks*, 1(2):229–232, 1990.

[47] S P Luttrell. Asymptotic code vector density in topographic vector quantisers. Memorandum 4392, Royal Signals and Radar Establishment, 1990.

[48] S P Luttrell. Code vector density in topographic mappings: scalar case. *IEEE Transactions on Neural Networks*, 2(4): 427–436, 1991.

[49] S P Luttrell. Code vector density in topographic mappings. Technical Report 4669, Defence Research Agency, Malvern, 1992.

[50] S P Luttrell. Vector quantisation of K-distributed data. Research Note SP4/110, Royal Signals and Radar Establishment, Malvern, 1990.

[51] S P Luttrell. Bayesian inference on a tree. Research Note SP4/109, Royal Signals and Radar Establishment, Malvern, 1990.

[52] S P Luttrell. A trainable texture anomaly detector using the adaptive cluster expansion (ACE) method. Memorandum 4437, Royal Signals and Radar Establishment, Malvern, 1990.

[53] S P Luttrell. Adaptive Cluster Expansion (ACE): a hierarchical Bayesian network. 1991. URL `http://arxiv.org/abs/cs/0410020`.

[54] S P Luttrell. Adaptive Cluster Expansion (ACE): a multilayer network for estimating probability density functions. 1991. URL `http://arxiv.org/abs/1012.3656`.

[55] S P Luttrell. A hierarchical network for clutter and texture modelling. In *Proceedings of SPIE conference on adaptive signal processing*, pages 518–528, San Diego, 1991. SPIE.

[56] S P Luttrell. Anomaly detector. Patent GB 2253697 B, 1992.

[57] S P Luttrell. Self-supervision in multilayer adaptive networks. Memorandum 4467, Royal Signals and Radar Establishment, Malvern, 1991.

[58] S P Luttrell. Self-supervised training of hierarchical vector quantisers. In *Proceedings of IEE conference on artificial neural networks*, pages 5–9, Bournemouth, 1991. IEE.

[59] S P Luttrell. Self-supervised adaptive networks. *Proceedings of the IEE F*, 139(6):371–377, 1992.

[60] S P Luttrell. A maximum entropy interpretation of random access memory network computations. Research Note SP4/137, Royal Signals and Radar Establishment, Malvern, 1990.

[61] S P Luttrell. Gibbs distribution theory of adaptive n-tuple networks. In I Aleksander and J Taylor, editors, *Proceedings of IEE conference on artificial neural networks*, pages 313–316, Brighton, 1992. IEE, North-Holland.

[62] S P Luttrell. Error measures in adaptive networks. Research Note SP4/111, Royal Signals and Radar Establishment, Malvern, 1990.

[63] S P Luttrell. Adaptive Bayesian networks. In *Proceedings of SPIE conference on adaptive signal processing*, pages 140–151, Orlando, 1992. SPIE.

[64] S P Luttrell. *Maximum entropy and Bayesian methods*, chapter The cluster expansion: a hierarchical density model, pages 269–278. Kluwer, Dordrecht, 1995.

[65] S P Luttrell. Partitioned mixture distributions: an introduction. Memorandum 4671, Defence Research Agency, 1992.

[66] S P Luttrell. An adaptive Bayesian network for low-level image processing. In *Proceedings of IEE conference on artificial neural networks*, pages 61–65, Brighton, 1993. IEE.

[67] S P Luttrell. *Maximum entropy and Bayesian methods*, chapter The partitioned mixture distribution: multiple overlapping density models, pages 279–286. Kluwer, Dordrecht, 1995.

[68] S P Luttrell. Partitioned mixture distribution: an adaptive Bayesian network for low-level image processing. *IEE Proceedings on Vision, Image, and Signal Processing*, 141(4):251–260, 1994.

[69] S P Luttrell. The Markov chain theory of vector quantisers. Technical Report 4742, Defence Research Agency, Malvern, 1993.

[70] S P Luttrell. A Bayesian analysis of self-organising maps. *Neural Computation*, 6(5):767–794, 1994.

[71] S P Luttrell. The detection of linear anomalies in images. Research Note CSE1/232, Defence Research Agency, Malvern, 1992.

[72] S P Luttrell. The application of mixture distributions to the detection of linear anomalies in cluttered images: a preliminary survey. Technical Report 4787, Defence Research Agency, Malvern, 1993.

[73] S P Luttrell. The use of mixture distributions to model the density of speckled data. Technical Report DRA/CIS/CBC3/TR94002, Defence Research Agency, 1994.

[74] S P Luttrell. Using self-organising maps to classify radar range profiles. In *Proceedings of IEE conference on artificial neural networks*, pages 335–340, Cambridge, 1995. IEE.

[75] S P Luttrell. Range profile classification using topographic mappings. Technical Report DRA/CIS(SE1)/651/11/RP/2, Defence Research Agency, Malvern, 1995.

[76] S P Luttrell. *Information theory and the brain*, chapter The emergence of dominance stripes and orientation maps in a network of firing neurons, pages 101–121. Cambridge University Press, 2000.

[77] S P Luttrell. A self-organising network for processing data from multiple sensors. Technical Report DRA/CIS(SE1)/651/11/RP/1, Defence Research Agency, Malvern, 1995.

[78] S P Luttrell. A componential self-organising neural network. Technical Report DRA/CIS(SE1)/651/11/RP/3, Defence Research Agency, Malvern, 1995.

[79] S P Luttrell. A self-organising visual cortex network (VICON) for processing data from two imaging sensors. Technical Report DRA/CIS(SE1)/651/FUN/STIT/RP/3, Defence Research Agency, Malvern, 1996.

[80] S P Luttrell. The development of dominance stripes and orientation maps in a self-organising visual cortex network (VICON). 1996. URL `http://arxiv.org/abs/1012.3724`.

[81] S P Luttrell. Dynamical partitioned mixture distributions: an introduction. Technical Report DRA/CIS(SE1)/651/11/RP/4, Defence Research Agency, Malvern, 1995.

[82] S P Luttrell. Partitioned mixture distributions: the dynamical case. In *Proceedings of IEE international conference on artificial neural networks*, pages 59–63, Cambridge, 1997. IEE.

[83] S P Luttrell. Partitioned mixture distributions: a first order perturbation analysis. In *Proceedings of IEE international conference on artificial neural networks*, pages 280–284, Cambridge, 1997. IEE.

[84] S P Luttrell. A self-organising neural network for processing data from multiple sensors. 1995. URL `http://arxiv.org/abs/1012.4173`.

[85] S P Luttrell. Optimal posterior probabilities for self-organised neural networks. Technical Report DRA/CIS(SE1)/651/FUN/STIT/RP/1, Defence Research Agency, Malvern, 1996.

[86] S P Luttrell. Optimal response functions in a network of discretely firing neurons. In *Proceedings of IEE conference on artificial neural networks*, pages 216–220, Cambridge, 1997. IEE.

[87] S P Luttrell and C J S Webber. Spatio-temporal inference theory: final report (1 April 1994 - 31 March 1997). Technical Report DRA/CIS(SE1)/651/FUN/STIT/RP/4, Defence Research Agency, Malvern, 1997.

[88] S P Luttrell. A unified theory of density models and auto-encoders. Technical Report DERA/CIS/CIS5/TR97303 (also Isaac Newton Institute for Mathematical Sciences Preprint NI97039-NNM), Defence Evaluation and Research Agency, Malvern, 1997.

[89] S P Luttrell. Modelling the probability density of Markov sources. 1998. URL `http://arxiv.org/abs/cs/0607019`.

[90] S P Luttrell. Some theoretical properties of a network of discretely firing neurons. 1998. URL `http://arxiv.org/abs/1505.00444`.

[91] S P Luttrell. *Handbook of neural computation*, chapter Designing analysable networks, pages B5.3:1–B5.3:8. IOP and OUP, Oxford, 1995.

[92] S P Luttrell. *Mathematics of neural networks: models, algorithms and applications*, chapter A theory of self-organising neural networks, pages 240–244. Kluwer, Boston, 1997.

[93] S P Luttrell. Self-organisation of multiple winner-take-all neural networks. *Connection Science*, 9(1):11–30, 1997.

[94] S P Luttrell. A discrete firing event analysis of the adaptive cluster expansion network. *Network: Computation in Neural Systems*, 7(2):285–290, 1996.

[95] S P Luttrell. A Bayesian analysis of complementary approaches to training self-organising stochastic networks. Technical Report DERA/CIS/CIS5/TR97165, Defence Evaluation and Research Agency, Malvern, 1998.

[96] S P Luttrell. An adaptive network for encoding data using piecewise linear functions. In *Proceedings of international confererence on artificial neural networks*, pages 198–203, Edinburgh, 1999. IEE.

[97] S P Luttrell. Self-organised factorial encoding of a toroidal manifold. 1998. URL `http://arxiv.org/abs/cs/0410036`.

[98] S P Luttrell. *Combining artificial neural nets: Ensemble and modular multi-net systems*, chapter Self-organised modular neural networks for encoding data, pages 235–263. Perspectives in Neural Computing. Springer-Verlag, London, 1999.

[99] S P Luttrell. Encoding data from a curved manifold. Technical Report DERA/S&P/SPI/TR98116, Defence Evaluation and Research Agency, Malvern, 1999.

[100] S P Luttrell and C J S Webber. Using self-organising neural networks to decompose complex tasks into simpler sub-tasks. Technical Report DERA/S&P/SPI/TR980086, Defence Evaluation and Research Agency, Malvern, 1998.

[101] S P Luttrell and C J S Webber. Using self-organising neural networks to discover structure in data. Technical Report DERA/S&P/SPI/TR990564, Defence Evaluation and Research Agency, Malvern, 1999.

[102] S P Luttrell. A user's guide to stochastic encoder/decoders. Technical Report DERA/S&P/SPI/TR990290, Defence Evaluation and Research Agency, Malvern, 1999.

[103] S P Luttrell. Self-organising stochastic encoders. 1999. URL `http://arxiv.org/abs/1012.4126`.

[104] S P Luttrell. Stochastic vector quantisers. 2000. URL `http://arxiv.org/abs/1012.3705`.

[105] S P Luttrell. Spatio-temporal inference theory: final report (1 April 1997 - 31 March 2000). Technical Report DERA/CIS(SE1)/651/FUN/STIT/RP/5 [also DERA/S&P/SPI/CR000118], DERA, Malvern, 2000.

[106] S P Luttrell. Self-organised discovery of structure in signal manifolds. In *Proceedings of DERA workshop on sensor array signal processing*, number DERA/S&E/SPI/TR000349, pages 88–93, Malvern, 2000. DERA.

[107] S P Luttrell. Invariance discovery by vector quantisation of noisy data. Technical Report DERA/S&P/SPI/TR000070, Defence Evaluation and Research Agency, Malvern, 2000.

[108] S P Luttrell. Using stochastic encoders to discover structure in data. In *Digest of the IMA international conference on mathematics in signal processing*, Warwick, 2000. IMA.

[109] S P Luttrell. Invariant stochastic encoders. In *Digest of the IMA international conference on mathematics in signal processing*, Warwick, 2000. IMA.

[110] S P Luttrell. Using stochastic encoders to discover structure in data. 2000. URL `http://arxiv.org/abs/cs/0408049`.

[111] S P Luttrell. Invariant stochastic encoders. 2000. URL `http://arxiv.org/abs/cs/0408050`.

[112] S P Luttrell. *Advances in self-organising maps*, chapter Adaptive subspace encoders using stochastic vector quantisers, pages 102–109. Springer, 2001.

[113] S P Luttrell. *Mathematics in Signal Processing*, volume 5, chapter Using stochastic vector quantisers to characterise signal and noise subspaces, pages 193–204. Oxford University Press, 2002.

[114] S P Luttrell. *Multiple classifier systems*, chapter A self-organising approach to multiple classifier fusion, pages 319–328. Lecture notes in computer science. Springer-Verlag, London, 2001.

[115] S P Luttrell. Adaptive sensor fusion using stochastic vector quantisers. In *Proceedings of the DERA/IEE workshop on intelligent sensor processing*, pages 2/1–2/6, Birmingham, 2001. DERA and IEE.

[116] S P Luttrell. A self-organising network for sea clutter suppression. Technical Report DERA/S&E/SPI/TR000513, Defence Evaluation and Research Agency, Malvern, 2001.

[117] S P Luttrell. A comparison of self-organising network and probability modelling approaches to sea clutter suppression. Technical Report DERA/S&E/SPI/TR010477, Defence Evaluation and Research Agency, Malvern, 2001.

[118] S P Luttrell. Self-organising information processing: research position. Technical Report QinetiQ/S&E/SIP/CR011926, QinetiQ, Malvern, 2001.

[119] S P Luttrell. Self-organising information processing: research position update. Technical Report QinetiQ/S&E/APC/CR041282, QinetiQ, Malvern, 2004.

[120] S P Luttrell. Multi-layer self-organising networks using ACEnet. Technical Report QinetiQ/S&E/APC/CR021071, QinetiQ, Malvern, 2002.

[121] S P Luttrell. Multi-layer self-organising networks using ACEnet: Part 2. Technical Report QinetiQ/S&E/APC/CR021835, QinetiQ, Malvern, 2002.

[122] S P Luttrell. A self-organising approach to optimally combining classifiers. Technical Report QinetiQ/S&E/SIP/CR011805, QinetiQ, Malvern, 2002.

[123] S P Luttrell. A self-organising approach to optimally combining classifiers: Part 2. Technical Report QinetiQ/S&E/SIP/CR022003, QinetiQ, Malvern, 2003.

[124] S P Luttrell. A self-organising approach to optimally combining classifiers: Part 3. Technical Report QinetiQ/S&E/SIP/CR040126, QinetiQ, Malvern, 2004.

[125] S P Luttrell. A Markov chain approach to multiple classifier fusion. In T Windeatt and F Roli, editors, *Proceedings of international workshop on multiple classifier fusion*, pages 217–226, London, 2003. Springer-Verlag.

[126] S P Luttrell. Using self-organising mappings to learn the structure of data manifolds. 2003. URL `http://arxiv.org/abs/cs/0406017`.

[127] S P Luttrell. Discrete network dynamics. Part 1: operator theory. 2005. URL `http://arxiv.org/abs/cs/0511027`.

[128] S P Luttrell. Review of recurrent networks. Technical Report QinetiQ/SPS/CSIP/TR050877, QinetiQ, Malvern, 2005.

[129] S P Luttrell. Recurrent self-organising networks: large occupancy case. Technical Report QinetiQ/SPS/CSIP/TR057273, QinetiQ, Malvern, 2006.

[130] S P Luttrell. Recurrent self-organising networks: small occupancy case. Technical Report QinetiQ/SPS/CSIP/TR057352, QinetiQ, Malvern, 2006.

[131] S P Luttrell. Proposed use of recurrent self-organising networks for language identification. Technical Report QinetiQ/D&TS/SS/TWP057427, QinetiQ, Malvern, 2006.

[132] S P Luttrell. Application of symbolic algebra techniques to Bayesian information processing. Technical Report QinetiQ/D&TS/SS/CR0602193, QinetiQ, Malvern, 2006.

[133] S P Luttrell. Recurrent self-organising networks: a software implementation. Technical Report QinetiQ/SPS/CSIP/TR0703548, QinetiQ, Malvern, 2007.

[134] S P Luttrell. Proposed use of self-organising networks for identifying communities of interest. Technical Report QinetiQ/D&TS/SS/TR0612427 (draft appendix, omitted in final version), QinetiQ, Malvern, 2007.