Efficient Covariance Matrix Methods for Bayesian Gaussian Processes and Hopfield Neural Networks

Amos James Storkey

Supervisor: Dr. P. De Wilde

A thesis submitted in
partial fulfilment for the degree of
Doctor of Philosophy of the University of London
and the Diploma of Membership of Imperial College

Neural Systems Group
Department of Electrical Engineering
Imperial College

January 20, 1999

To Helen

Abstract

Covariance matrices are important in many areas of neural modelling. In Hopfield networks they are used to form the weight matrix which controls the autoassociative properties of the network.

In Gaussian processes, which have been shown to be the infinite neuron limit of many regularised feedforward neural networks, covariance matrices control the form of Bayesian prior distribution over function space.

This thesis examines interesting modifications to the standard covariance matrix methods to increase functionality or efficiency of these neural techniques.

Firstly the problem of adapting Gaussian process priors to perform regression on switching regimes is tackled. This involves the use of block covariance matrices and Gibbs sampling methods. Then the use of Toeplitz methods is proposed for Gaussian process regression where sampling positions can be chosen.

A comparison is made between Hopfield weight matrices, and sample covariances. This allows work on sample covariances to be used to estimate the eigenvalue structure of standard Hopfield weight matrices.

This structure of Hopfield weight matrices enables the development of efficient incremental and local learning techniques for Hopfield networks. A new Hopfield learning rule is introduced. It is shown both empirically and mathematically that such learning methods give a higher capacity than the usual Hebb rule. Furthermore it is shown that this rule is more able to deal with the storage of correlated patterns.

The basins of attraction of this learning rule are examined empirically. It is found that they are larger, rounder and more evenly distributed than those of the Hebb rule.

Lastly we show using iterated function sequence methods that the learning rule acts as a palimpsest (or forgetful) learning rule, and does not suffer from catastrophic forgetting.

Contents

List of Tables

List of Figures

Acknowledgements...

I would like to thank many people for their help and advice. I would especially like to mention Philippe De Wilde, Romain Valabregue, Helen Storkey, Richard Everson, Dirk Hussmier, Will Penny, Iead Rezek, Steve Roberts, Jon Rogers, Sreejith Das and Steve Hunt.

Chapter 1

Introduction

Research in neural networks has often tackled the problems of association and inference. In fact today what we might call neural networks is a diverse research area, covering topics from cellular neural modelling, through computational neuroscience, cognitive science, spin-glass type neural networks, hardware design, optimisation, functional approximation, computation, inference and statistics. With such coverage, some fragmentation of the field was an inevitability, but there has certainly been room for cross-pollination between different research areas.

This thesis will focus on the traditional problems of association and inference. It will be unusual in that it will look at two distinct areas: that of Hopfield networks (usually considered within a spin-glass statistical physics framework), and that of Bayesian statistical inference, specifically looking at the Gaussian process models which are limit cases of infinite neural networks [1]. The primary emphasis will be on developing the methods and theory of each area, although there will be some concepts which are useful in both areas.

Both Hopfield networks and Gaussian processes rely on interesting matrix structures to give them their strength and elegance. By extending or improving the structure of these matrices, significant improvements in the power of each technique can be made. Gaussian processes are defined in terms of their covariance matrices. Furthermore the matrices of Hopfield networks are related to sample covariances, and this useful relationship acts as a step towards developing new learning rules.

It is important to emphasise from the beginning that this thesis is theoretical and methodological in nature. It stems from an interest in developing methods in the subject area, grounding the methods in a theoretical framework, and examining the methods through use of theory and simulation. Experimental comparisons with other classes of methods will not form part of this thesis (in many cases these have been done elsewhere). Where particular experimental situations are used, it is primarily to illustrate the methods involved.

1.1 An outline of the thesis

This thesis consists of two parts. The first part deals with Gaussian process modelling. It tackles the efficient extension of Gaussian process techniques to non-Gaussian processes, especially to the area of switching processes. It also develops efficient methods for Gaussian process implementation for higher dimensional systems with regular sampling patterns.

The second part looks at extending the standard learning rules for Hopfield models. High capacity learning rules are developed which are both local and incremental.

Chapter one introduces the different concepts which will be important in the thesis. Inference problems and association problems are specified, and the basics of Bayesian inferential methods are developed.

Chapter two formalises the Bayesian approach, and introduces Gaussian processes methods.

Chapter three extends the Gaussian process techniques for use in non-Gaussian situations. Specifically, efficient methods are developed for modelling signals which are formed by switching between a number of separate independent signals.

In chapter four, Toeplitz techniques are used to provide efficient implementation of Gaussian process techniques in many dimensions, where the sampling is regular.

Chapter five starts the Hopfield model part. In chapter five, Hopfield networks are outlined, and the standard training algorithms are given. The importance of locality and incrementality is outlined. A new learning rule is introduced which has a higher capacity than the standard Hebb rule. This new learning rule is both local and incremental. The capacity of the learning rule is proved analytically to be (at least) $n/\sqrt{2\ln(n)}$ where n is the number of neurons in the network. The network is simulated, and the capacity relationship illustrated for finite network sizes. Various extensions of this rule are discussed.

Chapter six looks at the behaviour of this new learning rule when correlated patterns are to be stored in the network.

In chapter seven the basins of attraction of the Hopfield memories are analysed for this new learning rule. Direct attraction basins are examined for the pseudo-inverse, the Hebb rule and the new rule. Complete attraction basins are also examined for the new rule and the Hebb rule, and their size, structure, is compared.

Chapter eight shows how a slight variation of the learning rule produces a very high capacity palimpsest, or forgetful, learning algorithm. The palimpsest nature of the rule is illustrated using iterated function system theory.

In chapter nine the thesis is concluded.

1.2 Inference problems

What is meant by an inference problem? A first step is to give a number of examples to show how common inference is in everyday life.

- The milk. Mark gets up in the morning, and sleepily pulls the milk out of the fridge to pour onto his cereal. Is the milk off? He gives it a sniff, and makes an assessment.

He adjusts his future judgement criteria by the taste of his cereal. An inference problem.

- As he sits down to eat his breakfast he turns on the radio and catches the morning weather forecast. The forecasters have historical information from a finite number of weather stations. They try to infer what the weather will be for any person sitting in the area covered for times in the future. Another inference problem.

- He opens his mail. A letter has arrived from his aunt. He undertakes a complicated inference process in deciphering her exceedingly bad handwriting.

And so even before leaving the house in the morning, each person has performed many different inference tasks without thinking about it.

Inference is the process of using past experience or some limited data along with some prior belief about the structure of a problem to draw conclusions about situations not met before. Prior beliefs (or assumptions) are vital, because without prior assumptions there is no reason to assume any relationship whatsoever between the data received and the new situations, and so no inference can be made. An inference problem consists of

- A characteristic question
- Some information
- Some prior belief about the possible answers to the question, and about the relationship of the information to the answer to the question.

A characteristic question is a question about the nature of a system, in contrast to a decision question which is a question about an action to be taken.

A decision problem on the other hand consists of

- A decision question.
- Some information
- Some prior beliefs
- A measure of the utility of each decision given each system state

In other words a decision problem is an inference problem combined with some way of calculating benefits. A decision question has one answer: the best action to take.

There are a number of ways which an inference problem can be formulated. In this thesis a Bayesian approach will be used. There is much philosophical and practical debate about Bayesian methods which, while being of great importance, will not be covered in detail here. Instead a note of some benefits of this approach are given:

- The importance of prior information is recognised and made explicit through the use of probabilities.

- Other classical statistical formulations can be seen as special cases of the Bayesian approach.

- It gives a probability distribution over possible answers, rather than a single answer, thereby giving more information.

This approach is formalised in the following way. Consider a set of models or hypotheses $\{H_1, H_2, \ldots\}$ of a system (note here we have made the set countable, but this need not be the case). The system will include details of all the possible information which will later be received, as well as all the possible answers to the question asked. It is assumed that one of the models chosen is a true model. Each of the models is then assigned a prior probability $P(H_i)$ which measures the apriori chance of a model being true. Note that the choice of models is also assigning prior information: all possible models not considered are assigned zero probability.

The information, denoted by D for data, is then received. This is combined with the prior information using Bayes theorem to give the posterior probability of the model:

$$P(H_i|D) = \frac{P(D|H_i)P(H_i)}{P(D)}$$

where $P(D|H_i)$ is the *likelihood* of the model, and is calculated from the model itself. $P(H_i)$ is the prior of the model given above, and $P(D)$ is a normalising constant given by

$$P(D) = \sum_i P(D|H_i)P(H_i)$$

Obviously here, and in the future, for uncountable hypothesis sets the summation is replaced by the corresponding integral.

The posterior probability of the models can then be used to calculate the posterior probability of a given answer A to the characteristic question by

$$P(A|D) = \sum_i P(A, H_i|D) = \sum_i P(A|H_i)P(H_i|D)$$

where $P(A|H_i)$ is given by the model, and $P(H_i|D)$ is the posterior probability previously calculated. It is assumed that the answers to the characteristic question are conditionally independent of the data given some hypothesis. This would be the case, for example, if the hypothesis is presumed to be a as full an explanation of the data as is possible.

The answer the Bayesian methodology gives is, therefore, a probability distribution over the possible answers to the characteristic question. Clearly high probability (or probability density) answers are more likely than low probability ones. Here are some standard types of problems.

1.2.1 Classification problems

A classification problem consists of some 'independent variable' space X. Each element or point in X can be put into one of a number of classes. Sometimes the classification is probabilistic: $x \in X$ might be put into one class, other times it might go into another. Let the possible classes be denoted by a small countable set Y.

In order to model this system the model/prior $P(y \in Y|x \in X)$ is needed. The data consists of independent measurements $\{(x_i, y_i)\}$ drawn from the distribution over $X \times Y$. The characteristic question concerns the value of y given that $x = x_j$ is some new point in X which is drawn independent of the data.

1.2.2 Regression problems

The regression system consists of spaces X and Y, The model/prior information tends to represent a set of possible functions from X to Y, and often some smoothness conditions with respect to some underlying metrics of X and Y. It is expressed in the form $P(y \in Y|x \in X)$.

The data consists of independent measurements $\{(x_i, y_i)\}$ drawn from the distribution over $X \times Y$. The characteristic question concerns the value of y given that $x = x_j$, some new point in X which is drawn independent of the data.

1.2.3 Probability density estimation problems

The probability density estimation system consists of a space X, modelled by distributions of the form $P(x \in X)$. The data consists of independent measurements $\{x_i\}$ drawn from the true distribution over X. The characteristic question concerns the value of a new independent measurement x_j.

1.2.4 Hierarchy and hierarchical problems

A simple but important rule in Bayesian methods says that one should introduce prior distributions where one has prior information. If the prior information is hierarchical in nature, then hierarchy should be introduced into the models used. This is often done by introducing latent or hidden variables into the model. Often the form of the problem does not change. However sometimes for a hierarchical problem we are interested in the hierarchical variable. The characteristic question then concerns the value of the latent variables.

1.2.5 Clustering problems

A clustering problem is a simple example of a latent variable problem. This can be seen as a classification problem with missing classification values. The latent variables are used to represent the missing classification variable.

As a result we have a system consisting of a set $M \subset Z^+$ of possible sizes of the latent variable space L and the space X. The models are of the form $P(m \in M, l \in L|x \in X)$. The data is simply a measurement set $\{x_i\}$. The characteristic question asks how a given set of points should be labelled (taking into account the symmetry introduced by the arbitrary nature of the labels).

1.2.6 Mixture of densities

Mixture of density problems are similar to clustering problems. Such problems can be seen as probability density estimation problems where different points are drawn from different distributions. The latent variables are used to denote which distribution a point is drawn from.

Here we have a system consisting of a set M of possible sizes of the latent variable space L and the space X. The models are of the form $P(m \in M, l \in L, x \in X)$. Again the data is a measurement set $\{x_i\}$. The characteristic question might ask what the value of some new measurement $\{x_i\}$ will be, or the value of some new measurement drawn from a specific distribution, or the number of clusters, or some other related question.

1.3 Association problems

Association problems should correctly be viewed as a subclass of inference problems. However in the neural network community, association has less often been formulated in a Bayesian manner. Such a framework can be helpful. In its most general form association is a combination of clustering and mixture density estimation.

An association problem can be seen as a hidden variable problem. Different values of the latent variable generate different distributions in two separate spaces. Given a point in one space, we want a set of probabilities that each point in the other space was generated from the distribution controlled by the same latent variable.

Hence an association problem consists of a space X and a space Y, as well as a latent variable space L. It is modelled by conditional distributions $P(y \in Y|l \in L)$ and $P(x \in X|l \in L)$ and $P(l \in L)$. The data are sets of pairs, $\{(x_i, y_i)\}$. The latent variable $l \in L$ is used to model which of a set of distributions a point is drawn from. The characteristic question concerns the value of $y_j \in Y$ given a new point $x_j \in X$.

In some situations the distribution over Y given l is taken as deterministic.

1.4 Autoassociation problems

An autoassociation problem is another form of association problem. Here, however, the space Y is the same as X, and $P(y \in Y|l \in L)$ is apriori related to $P(x \in X|l \in L)$, and is usually some 'noise free' version of $P(x \in X|l \in L)$.

Again sometimes the distribution over Y given l is taken as deterministic.

Another type of autoassociation problem is a clustering problem, where instead of wanting to label the points, the probability of some point being in the same cluster as a given point is wanted.

1.5 Bayesian methods

We have mentioned Bayesian methods and shown how Bayes theorem can be used to perform inference. In this section, more details about Bayesian methods is given.

1.5.1 Bayesian methods and probability

The Bayesian statistical approach differs from classical statistics in believing that it is useful to assign probabilities to hypothesis as a measure of belief. It does this because of a recognition that all statistical methods rely on (usually implicit) prior information in performing inference. Bayesian's claim that it is more transparent, more informative, and more versatile to represent such prior information explicitly. Probabilities are the natural way of representing such belief, because then hypothesis space beliefs can be directly related to data space probabilities through the use of Bayes theorem.

In what way might Bayesian methods be more transparent, more informative and more versatile? More transparent because implicit assumptions are made explicit. Modern Bayesian statistics recognises the fact that there are basic assumptions which any statistical approach makes, which can not be substantiated from within statistics itself. The Bayesian prior embodies those assumptions.

Bayesian methods are more informative, because many of the techniques used in classical statistics, such as regularisation, have a natural Bayesian interpretation in terms of prior distributions, which often give a clearer understanding of what the methods assume. They are more versatile because instead of rather than just restricting the hypothesis space by the set of models chosen (effectively using zero-one priors) or by using null and test hypotheses, more and less likely hypotheses can be combined using probability weighting.

Let us now formalise the Bayesian statistical approach.

Firstly we have a set of random variables $X = \{X_i\}$ which may be dependent on one another. This set of random variables can be considered to be the system which is being studied. An inherent assumption in this is that the system is isolated (i.e. independent of any other variables) or more realistically that the effect of any other variables is understood to be incorporated into the probability distribution over these random variables.

Secondly we have a set of models $H = \{H_i\}$, or hypotheses. These models are in fact just possible joint probability distributions of the random variables. Each H_i attributes some functional form to the distribution $P(X)$.

Now in the third place the Bayesian method claims that H_i are themselves random variables, and can be assigned a probability distribution. These probability distributions are used to describe the apriori belief that a given model is the true model. They are called prior distributions.

The fundamental difference between the random variables H and the random variables X is that any X_i could, in theory, be observed, whereas the variables H can never be observed. Rather the variables H can be inferred from the observations of X.

Sometimes we might not be immediately sure about exactly what shape a prior distribution should take. It might be helpful to consider a set of possible prior distributions, as another group of models, and put what is called a hyperprior distribution over these new models. This can be continued in a hierarchical manner. At the end of the day there is no theoretical change introduced by this procedure because the original prior distribution can be directly inferred from these hyper-distributions (by performing a weighted sum over all the hyper-models). Thus one ends up with a simple non-hierarchical prior after all. However there are practical reasons (because of the intractable nature of the integrals which are often involved) why such structures can be usefully kept in hierarchical form.

Note also that hierarchy can be introduced into the structure of X. When this is done the higher order structures are known as hierarchical variables, latent variables or hidden variables. This is because although they tend not to be observed they do have some tangible meaning, and could reasonably be observed in some other (perhaps hypothetical) setting.

The fourth aspect of Bayesian modelling is the data. A dataset is the observation (or measurement) of a subset D of the random variables X. This provides us with new information, which is used to modify the prior beliefs. To do this we find probabilities conditioned on the data.

Lastly there are is a set of variables of interest Y, a subset of X/D. To infer the values of these variables (or in fact distributions over these values) the conditional probability $P(Y|D)$ needs to be calculated. This can be done in two steps. Firstly Bayes theorem can be used to calculate the posterior distribution over H

$$P(H_i|D) = \frac{P(D|H_i)P(H_i)}{P(D)}$$

where, as before, $P(D|H_i)$ is the *likelihood* of the model, and is calculated from the model itself. $P(H_i)$ is the prior of the model, and $P(D)$ is a normalising constant given by

$$P(D) = \sum_i P(D|H_i)P(H_i)$$

The second step involves using $P(Y|D) = \sum_i P(Y|H_i)P(H_i|D)$, summing (or integrating) over all the possible models. $P(Y|H_i)$ is the probability distribution over the variable Y which is given by model H_i.

We then have the required result, $P(Y|D)$, a distribution over the values of Y given we know the data D. Bayesian inference has been performed.

1.6 Neural networks

Initial research in what has become called neural networks was prompted by the wish to model the computational aspects of the working of the brain. Over the years some of the models produced have become quite formalised, and although some of them might no longer bear any immediate and direct relevance to brain modelling, they have inspired biologists, mathematicians and engineers alike.

The two models which concern us here are the Hopfield model and the feedforward model. The first of these is a fully connected feedback model which is used for autoassociation, and has interesting mathematical structures and properties.

1.6.1 The Hopfield network

The Hopfield network is simply a neural network with discrete units $x_i = \pm 1$, where all the units are bi-directionally connected to one another. The connections are symmetrically weighted with weight w_{ij} between neuron i and neuron j. The weights are adapted according to a simple learning rule which stores given patterns to be recalled later. It has been proposed as a model of the storage mechanism for memories in the brain [2], but on the whole is of greater interest because of analogies with interesting areas of physics (such as spin glasses) and the fact that its functionality can be assesses in a number of different mathematically tractable ways.

Interest in the Hopfield network developed along two lines. Hopfield initially proposed the Hopfield model as a content addressable memory [3]. Using a simple storage rule, the Hopfield network could be designed so that the network dynamics moved towards and into a fixed state which corresponded to the stored pattern. It is possible to create an *energy function* for the Hopfield network which has the property that the Hopfield dynamics always reduces the energy at each change. Because the energy function is bounded below, and because the Hopfield model has a finite number of states, the existence of this energy function guarantees convergence to a fixed state.

Hopfield and Tank later proposed the use of the Hopfield network for optimization problems [4]. There the optimization problem and constraints were written in terms of a quadratic form using Lagrange multipliers. This was used to build a Hopfield energy function whose minimization implied the optimization of the given problem. The dynamics of the Hopfield model were used to find the minimum. This use of the Hopfield model is very different from that which is of interest to us here. The Hopfield-Tank model suffered from a number of significant problems. Most notably, the resulting energy function tends to have many local minima which the Hopfield dynamics get stuck in. In fact this is the very same reason why the Hopfield model works as an associative memory.

There have been a number of attempts to build highly parallel hardware which might practically implement Hopfield model methods at high speed. For example, use of VLSI Hopfield models for character recognition can be found in [5]. Wafer scale hardware has also been developed [6]. The use of Hopfield networks for handwritten character recognition was illustated in [7].

1.6.2 Feedforward networks

Feedforward neural networks have been used primarily for classification and regression problems. Feedforward networks consist of a set of input neurons which a connected to a set of output neurons, usually indirectly through a one or more layers of hidden neurons. The neurons in the hidden layer usually have sigmoidal activation functions. By this it is meant that the neuron performs the function

$$y = \frac{1}{1 + \exp(-\alpha \sum_i w_i x_i)}$$

where α is some constant, x_i are the outputs form the neurons of the previous layer, w_i are the weights of the connections from those neurons to the current neuron, and y is the output of the current neuron.

It was shown that a sigmoidal neural network with one hidden layer could represent any function to arbitrary accuracy (by increasing the number of neurons in the hidden layer) [8, 9]. For this reason a feedforward neural network with one hidden layer is called a universal approximator. Because of this people became interested in using feedforward networks as function approximators, and more specifically as the set of model functions for regression/classification.

In order to perform classical regression using feedforward networks, some sort of maximum likelihood algorithm was needed. In neural network terms this led to error minimization between the target outputs of a training set and the actual outputs of the neural network. This could be performed by a simple adaptation of the chain rule of differentiation in order to propagate the effects of any error back through the network [10]. This 'backpropagation' algorithm became very popular.

One of the big problems with the use of backpropagation was that the flexibility of neural models allowed overfitting to occur: the noise about the true function was incorporated into the model fit, and as a result the model did not represent the original function well.

Various solutions to this problem were given (among them early stopping and regularisation). However one of the more comprehensive ways of tackling this and other issues came from the introduction of Bayesian techniques into the neural networks field. Instead of having a single network providing a deterministic regression function, and infinite number of networks could be used, with a distribution over these networks given in terms of the distribution over the weights of the network. These distributions would encode the prior beliefs about the functional form and the nature of the noise term. Techniques based around this method provided a principled backing for techniques such as regularisation in terms of prior distributions, as well as producing good results on standard regression/classification tests. Also the answers given were in terms of probability distributions, rather than single numbers, allowing some idea of the likely error in a given prediction.

1.7 Gaussian processes

In 1994 Radford Neal [1] noticed that as the number of hidden neurons in a neural network tended to infinity, a Bayesian neural network with Gaussian priors over the weights would tend to a Gaussian process. Although in that paper Neal was more interested in looking at types of neural network which were distinct from Gaussian processes, Williams [11] showed how Gaussian processes could be used as an exact Bayesian modelling tool, developing on earlier work by O'Hagan [12] and some work of geologists on what they named 'kriging' after D.G. Krige. The work of O'Hagan and others was extended in a number of ways, notably by allowing the adaptation or integration over hyperparameters. Gaussian process approches were also adapted for use in classification [13, 14]. Gaussian processes are also similar to other Kernel based methods, such as spline models [15] and autoregressive moving average (ARMA) models [16].

Since 1994 there has been notable interest in this area, with a host of different experimental and theoretical work. The thesis of Carl Rasmussen [17] compared a number of different regression and classification methods. Gaussian processes performed very well in these tests. Attempts to speed up Gaussian processes have been an important part of recent work [18], as has improving the functionality of the model, such as by introducing time dependent noise variances [19]. These last two factors, speed and functionality of Gaussian process methods, will be the focus of the first part of this thesis.

Gaussian processes provide an interesting extension to neural network techniques in that many of the necessary integrals can be done tractably. Secondly the prior is automatically incorporated into the functional form, and is tunable using the covariance function of the process. Exactly how this is the case is the subject of the next chapter.

Chapter 2

Bayesian Approaches and Gaussian Processes

In this chapter we formally introduce Gaussian process techniques and compare them with Bayesian neural network tools. In order to do this neural networks are introduced first. Then the Bayesian approach to neural networks is outlined, and the relationship between such neural networks and Gaussian processes is given. The benefits that Gaussian processes techniques have over neural methods are examined.

2.1 Neural network methods

A feedforward neural network is a layered network of neurons, where the outputs from an input layer of neurons feeds into a hidden layer, and then possibly into further hidden layers until eventually the connections lead to an output layer from which the result can be obtained.

This can be formalised in the following way. Given a set of inputs x_i^0, each neuron performs the following function

$$x_i^r = f_r(\sum_j w_{ij}^r x_j^{r-1})$$

where r denotes the layer in which the neuron is situated, x_i^r is the output of the ith neuron in the rth layer, w_{ij}^r is the weight of the connection from neuron j in the $(r-1)$th layer to neuron i in the rth layer. The activation function of the neurons in layer r is given by f_r. Probably the most common form of neural network has one hidden layer, with the hidden layer activation function given by a sigmoidal function

$$f_1(h) = \frac{1}{1+\exp(-h)}$$

and a linear output layer activation function. Such a neural network can approximate any continuously differentiable function to arbitrary accuracy [9].

2.2 Using neural networks as a functional form for regression

A feedforward network takes a set of inputs and transforms them to produce a set of outputs. In other words a neural network is simply a function, with the weights as the function parameters, and the network defining the functional form. Hence a neural network can be used in exactly the same way as any other function. Specifically it is a possible model for use in nonlinear regression.

In order to use a neural network for regression, three things are needed. Clearly there is a need for data. Secondly there is a need to choose a specific structure of neural network. Thirdly some way of adapting the weights of the network to get a best fit function is needed.

Suppose the data has been collected, and the structure of the network has been decided (The number of input neurons and output neurons is determined by the data. The number of hidden neurons determines the desired flexibility of the functions). Then for classical regression, the likelihood of the neural network weights needs to be maximised. To do this, let the set of network weights be represented by θ, and the dataset be denoted $\{(\mathbf{x}^k, \mathbf{t}^k)\ k = 1 \dots n\}$ where $\mathbf{x}$ is the vector of inputs (or independent variables), and $\mathbf{t}$ is the vector of targets (or observed variables). Then the log likelihood is given by

$$\sum_{k=1}^{n} \ln p(\mathbf{t}^k | \mathbf{x}^k, \theta)$$

Often it is assumed that the neural function is corrupted by some Gaussian noise. In this case the negative log likelihood of the model is given by the sum of square error

$$\sum_{k=1}^{n} (\mathbf{y}^k - \mathbf{t}^k)^2$$

where $\mathbf{y}^k$ is the output of the neural network on presentation of $\mathbf{x}^k$ at the input. Maximising the likelihood is equivalent to minimizing the negative log likelihood. This can be done using a gradient descent approach, by finding the derivative of the sum of square error function, and adapting the parameters θ in order to move in the direction of the gradient. Because of the layered nature of the network, the chain rule of differentiation needs to be used in order to calculate the required change in the weights of the connections into the hidden layer neurons. Use of the chain rule in this way leads to the backpropagation algorithm for adapting the weights of a network.

The backpropagation algorithm has been put to much use. However one significant problem with the method as it stands above is that it causes what has become known as overfitting. A neural network can represent very complicated functions. Maximum likelihood methods such as error backpropagation effectively attempt to fit the training data as best as possible, but as a result can also include the additive noise as part of the function.

There are many approaches to tackling this issue, but most are not of direct concern here. What is of immediate interest is the development of Bayesian techniques for neural networks. Bayesian techniques solve the problem of overfitting by putting a probability distribution over the possible neural networks and over the noise. A very complicated function will have a lower probability, and so is less likely. However very high noise levels are also less likely, and so there is a payoff between minimizing the predicted noise, and minimizing the complexity of the neural network function.

2.3 The Bayesian framework

The Bayesian framework is different from the classical in that it treats all parameters as random variables. As they are random variables, they must have probability distributions. These probability distributions are used to represent the degree of belief about the possible values of the parameters.

So in the case of a neural network, the parameters must now have an associated probability distribution, called a prior distribution. It is often the case that this prior distribution represents the belief that smoother, less complicated functions are more likely. Also a prior is put over the noise function. This might be that it is a Gaussian white noise, which gives and independent identically distributed Gaussian noise at every data point.

Suppose then a data sample is obtained. Now the probability distributions need to be updated to account for the new information. This involves calculating the probability distributions given the data. Bayes theorem is the way of calculating this.

$$P(\theta|data) = \frac{P(data|\theta)P(\theta)}{P(data)}$$

This takes the prior probability distribution, and multiplies it with the likelihood to get the posterior distribution (after normalisation with $P(data)$).

After this, predictions can be made:

$$P(prediction|data) = \int_\theta d\theta P(prediction|\theta)P(\theta|data)$$

Hence in the Bayesian framework, no maximum is calculated. Instead all the possible values of the parameters are integrated over, weighted by the posterior probability of the parameter.

This integration (and that involved in calculating $P(\theta)$) is the difficult bit of Bayesian inference, because the integral is often not analytic. Most work on Bayesian methods has concentrated on two issues: determining good priors for the problems in question, and finding good and fast methods or approximations for doing the necessary integrations. These are the topics of the next two section.

2.4 Suitable priors for neural networks

The need for priors over the parameters of a neural network has been stated, but the form such parameter priors might take has not been discussed. The simplest situation is when smoother neural network functions are considered to have higher probability.

One way of implementing this prior is to note that if the weights of a neural network are small, then each neuron will be performing in the more linear region of the sigmoidal activation function (the part around zero). If each neuron is linear, then the whole neural network will be linear, and so the resulting function will be a flat hyperplane. So if neurons are acting 'more linearly', the function would be expected to be smoother.

To encapsulate the need for each weight to be small, a Gaussian prior can be put on the weights of the neural network. A large weight would then have a smaller prior probability. The result of this is that a smoother function is preferred to a more complicated one. Other choices of prior are possible, but they are usually of a similar form. For example a Laplacian prior is discussed in [20].

2.4.1 Hyperparameters

Sometimes it is helpful to use hyperparameters in defining the prior distribution of a parameter. These hyperparameters determine the shape of the distribution. For example, if a Gaussian prior over the parameters is used, then the variance of that Gaussian is a hyperparameter. Sometimes a hyperparameter might be considered to be known exactly. For example the magnitude of the noise might be known. If this is not the case, then once again a distribution must be defined for this hyperparameter. This distribution is called a hyperprior, and will also need to be integrated over to obtain the relevant prediction.

2.5 Practical methods for Bayesian neural network implementation

The integrals involved in Bayesian inference are often intractable. This is certainly the case for standard neural network priors. Because of this, approximation techniques need to be used in order to estimate the value of the relevant integrals. A number of different approaches have been developed to tackle this problem.

2.5.1 Maximum probability methods

This approach is an adapted maximum likelihood approach. It approximates the distribution over the parameters by a delta function at the most probable parameter value. This is effectively maximum likelihood with a weighting term given by the prior probability of the parameters. In the neural network case above this becomes a minimum square error problem with a weight decay regularisation term. In other words the network weights should be chosen to minimize

$$\sum_{k=1}^{n}(\mathbf{y}^k - \mathbf{t}^k)^2 + \alpha \sum_{ir}(w_{ij}^r)^2$$

This method tends to be a poor approximation to the Bayesian integral, because it completely disregards all other possible parameter values except for the most probable value.

2.5.2 Gaussian approximations

One way of improving the maximum probability approximation is to represent the posterior by a Gaussian distribution centred on the maximum posterior value [21]. The covariance matrix of the Gaussian is obtained from the Hessian of the error function.

2.5.3 Variational Methods

Even if the posterior is reasonably approximated by a Gaussian, the Hessian at the maximum posterior value need not provide a good estimate for the covariance. Another method for approximating the posterior with a Gaussian distribution involves trying to minimize some measure of discrepancy between the true distribution and the approximating one. One common measure that is used is the Kullback-Liebler (KL) divergence between the approximating distribution and the true one. These methods have a number of advantages over the method outlined in section 2.5.2. First, the variance of the Gaussian does not only depend on derivative information from the maximum posterior point. Rather it takes into account the overall shape of the distribution. Second, if the KL divergence is used, a lower bound to the true log likelihood of the model can be obtained. Third, variational methods can also be used to fit non Gaussian distributions.

2.5.4 Monte-Carlo methods

Monte Carlo Markov chain (MCMC) methods are a general tool for approximating non tractable integrals of the form (2.3). Details will not be given here, except to say that the method involves setting up a Markov chain which has the required posterior distribution as its limit distribution. A large sample from this limit distribution can be used to approximate the posterior and calculate the integral. MCMC suffers from the fact that the limit distribution can never be reached, and a big enough sample can never be taken. As a result MCMC is an approximation to the posterior. If the sampled distribution is close to the limit distribution, and the sample is representative of this distribution, then MCMC can give a good approximation. Unfortunately it is not possible to be entirely sure how close to the limit distribution the chain is, and what a suitable sample size should be. For example, even in theoretically ergodic systems, two high posterior regions of the state space might be seperated by very low probability regions. Hence a Markov chain might take a very long time to get from one high probability region to another. For example, this is the case for the ferromagnet below its critical temperature [22].

The good thing about MCMC is that it does not rely on one specific shape of distribution. Hence MCMC can be used in situations where no other method is appropriate.

2.5.5 The hyperparameters

Approximating the integral over the hyperparameters can be done in similar ways. The evidence framework [21] can be used, which chooses the hyperparameters that maximise the posterior probability of the model (given the data). This stems from the assumption that the posterior distribution is sharply peaked about this maximum value. Whether or not the evidence framework gives good approximation has been hotly debated. For details of this debate see [23] and [24].

MCMC approaches can also be used, and are often combined with Monte carlo methods over the parameters. Then the Markov chain will converge to the joint equilibrium distribution of the parameters and hyperparameters. [25].

In fact, in some situations, there is a tractable integration over the hyperparameters with respect to the prior distribution. Then the hyperparameters can be integrated out, and an approximate technique can be used over the parameter space using the new prior distributions [23].

The benefits and disadvantages of each of these methods has been discussed in a number of places. Bishop [26, chapter 10] gives a good overview. In fact the appropriateness of each method will depend on the nature of the form of the posterior, and will therefore be situation dependent.

2.6 The problems

Some of the problems of using Bayesian methods with neural networks have already been outlined: the necessary integrals are not analytic, and not necessarily straightforward to approximate well. There are a number of other things which can cause difficulties. One problem is that neural networks have numerous symmetries; there are different network structures which represent exactly the same function. Furthermore many of the prior distributions used do not respect those particular symmetries.

Although there are good arguments for weight decay priors, the implications of those priors in terms of the functional form cannot be properly assessed. Because the Bayesian method requires that one starts with a suitable prior, and then implements it, this can be a problem. There is often little way of knowing how well a neural network 'prior' actually represents the prior beliefs because these priors are not transparent. This also goes for the problem of setting distributions over the hyperparameters.

Gaussian process methods, which will be introduced next, resolve these particular problems. Interestingly the resurgence of interest in Gaussian process techniques has come from within the neural network community.

2.7 The Gaussian process approach

In 1994, Radford Neal produced a paper looking at the issue of neural networks with an infinite number of hidden neurons [1]. Within a Bayesian framework, there is not a problem with choosing very complicated functional forms, because the prior distribution prevents overfitting. Hence a neural network can have as many hidden neurons as anyone wants. In fact there is no reason why the case of an infinite number of hidden neurons cannot be considered. In this paper he showed that a Gaussian prior for the hidden to output weights results in a Gaussian process prior over function space. The nature of the covariance function of the Gaussian process was determined by the activation function and the prior on the input to hidden weights.

Since then, others have shown that a whole host of networks tend to Gaussian processes in the infinite limit. For example, Gibbs and Mackay consider a radial basis function network in [18].

2.7.1 Introduction to Gaussian processes

It will be worthwhile at this point to introduce the use of Gaussian process priors. To do this a standard regression problem is considered. We have a set of data points consisting of the value of some input (independent) variables, and the corresponding value of an output (or dependent) variable. These data points come from some function corrupted by white Gaussian noise. From this we want to infer the value of the function at some new set of points.

Let $\{\mathbf{x}_i\}$ denote the set of input space data-points $\{\mathbf{x}_i\}$ $i = 1, 2, \ldots, n$ combined with the points $\{\mathbf{x}_i\}$ $i = n+1, 2, \ldots, m$ at which predictions will be made. A superscript D (for DATA) denotes an m-vector truncated to the elements $i = 1, 2, \ldots, n$, and a superscript P (for PREDICTION) denotes an m-vector truncated to the elements $i = n+1, \ldots, m$.

An unknown function $f(\mathbf{x})$ generates datum f_i at point $\mathbf{x}_i$, corrupted by zero mean Gaussian noise η_i, with variance σ^2. Let

$$y_i = \begin{cases} f_i + \eta_i & i = 1, 2, \ldots, n \\ f_i & i = n+1, \ldots, m \end{cases}$$

So y_i combines the information we have (noisy data) with the information we want (noiseless predictions).

The issue we have discussed up to now regards what form the set of modelling functions, and the prior over those functions should take. One could take a linear function form, a polynomial function form, or maybe a neural network function form. In this latter case a Gaussian prior over the weights could be used. Instead of all of these, here we will use a Gaussian process form.

Because a Gaussian process is a random process, it combines the prior and functional form into one. The probability distributions are an inherent part of the functional definition. Furthermore there is no parametric form, and resulting prior probability for any Gaussian process sample function. Instead Gaussian processes are defined in terms of the resulting distribution over a set of sample points.

This, however, is very useful for Bayesian regression, because a set of sample points is precisely what we have.

Definition 1 *A random process (or random field) $y(\mathbf{x})$ is called a Gaussian process (or Gaussian random field) if for any finite dimensional vector* $\mathbf{y} = (y(\mathbf{x}_1), y(\mathbf{x}_2), \ldots, y(\mathbf{x}_n))$

has the distribution

$$P(\mathbf{y}) = \frac{1}{Z} \exp\left(-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^T C^{-1}(\mathbf{y}-\boldsymbol{\mu})\right)$$

where Z is the normalisation constant, $\boldsymbol{\mu}$ is some mean vector with $\mu_k(\mathbf{x}_k)$, and C is some covariance matrix with $C_{ij} = C(\mathbf{x}_i, \mathbf{x}_j)$. $C(\mathbf{x}, \mathbf{x}')$ is called the covariance function of the Gaussian process. The covariance function is stationary if $C(\mathbf{x}, \mathbf{x}') = C(\mathbf{x} - \mathbf{x}')$. In other words the covariance between two points is dependent only on the vector difference between those points. Note that a covariance function must generate a positive semidefinite symmetric covariance matrix.

From this definition, a Gaussian process prior implies that the prior probability over the set of sample points of our function implies

$$P(\mathbf{f}|H) = \frac{1}{Z'} \exp\left(-\frac{1}{2}(\mathbf{f}-\boldsymbol{\mu})^T C^{-1}(\mathbf{f}-\boldsymbol{\mu})\right)$$

where $\boldsymbol{\mu} = \boldsymbol{\mu}(H)$ is the mean vector, $C = C(H)$ is a covariance matrix and $Z'(H)$ is the normalisation constant. H stands for any set of hyperparameters, which, for now, are assumed to be known. The covariance matrix is generated from the covariance function of the Gaussian process.

In fact the modeller has complete freedom over the choice of the covariance function. This gives Gaussian process techniques some significant power, because a whole host of functional forms can be obtained through different types of covariance functions. These vary from smooth covariances to fractional Brownian fields, and fractional Gaussian noises. They can incorporate periodicity or locality. Much of the Gaussian process work in this thesis will be involved in looking at the power and resulting functionality of different forms of covariance functions.

The covariance function of the Gaussian process should be chosen to best represent the prior information. A common choice is

$$C(\mathbf{x}_i, \mathbf{x}_j; H) = \theta_1 \exp\left(-\frac{1}{2}\sum_{l=1}^{L} \frac{(x_i^{(l)} - x_j^{(l)})^2}{r_l^2}\right) + \theta_2$$

where L is the dimension of the input space, and l counts through each dimension. This covariance function, sometimes called a Gaussian decay covariance, or a exponential negative distance covariance, produces functions which are infinitely differentiable.

There are some important things to note about this formulation. Firstly the prior distribution is defined directly in function space. Hence Gaussian processes do not suffer from the problem of neural networks, where simple priors over weights gave complicated priors over function space, which were not easy to determine. Secondly, because of the way the priors are defined, it is more likely that the actual prior knowledge can be converted into a suitable prior distribution. This is because the parameters of the covariance function are easily interpretable in terms of scaling parameters, and it is this covariance which determines the prior. With neural networks on the other hand, the hyperparameters are not as transparently interpretable.

2.7.1.1 Performing Bayesian inference

Having defined the prior distribution in a suitable way, all that is left is to use Bayes theorem to calculate the posterior distribution over the prediction points.

Now $\mathbf{f}$ and $\boldsymbol{\eta}$ are independent Gaussian random variables. Sums of independent Gaussians variables are also Gaussian, and have a mean and covariance equal to the sum of the means and covariances (respectively) of the original independent variables. Hence $\mathbf{y}$ has a prior

$$P(\mathbf{y}|H) = \frac{1}{Z}\exp\left(-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^T Q^{-1}(\mathbf{y}-\boldsymbol{\mu})\right) \tag{2.1}$$

where

$$Q_{ij} = \begin{cases} C_{ij} + \delta_{ij}\sigma^2 & i,j \leq n \\ C_{ij} & \text{otherwise} \end{cases}$$

In order to continue further it is helpful to partition Q into the form

$$\begin{pmatrix} Q^{DD} & Q^{DP} \\ Q^{PD} & Q^{PP} \end{pmatrix}$$

where Q^{DD} is $n \times n$ and Q^{PP} is $(m-n) \times (m-n)$. Note that $Q^{PD} = (Q^{DP})^T$ by the symmetric nature of the covariance function.

Suppose we have now received data at points $\mathbf{x}^D$ given by $\mathbf{y}^D = \mathbf{y}^*$. We now want to know the posterior distribution over the value of the function $\mathbf{y}^P$ at the predictive points $\mathbf{x}^P$. This is given by

$$P(\mathbf{y}^P|\mathbf{y}^D = \mathbf{y}^*, H) = \frac{P(\mathbf{y}^P, \mathbf{y}^D = \mathbf{y}^*|H)}{P(\mathbf{y}^D = \mathbf{y}^*|H)}$$

$$= \frac{Z^D}{Z}\exp\left(-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^T Q^{-1}(\mathbf{y}-\boldsymbol{\mu}) - \frac{1}{2}(\mathbf{y}^D - \boldsymbol{\mu}^D)^T (Q^{DD})^{-1}(\mathbf{y}^D - \boldsymbol{\mu}^D)\right)$$

where $\mathbf{y} = (\mathbf{y}^D, \mathbf{y}^P)$ with $\mathbf{y}^D = \mathbf{y}^*$.

In order to simplify this, the partitioned inverse equations are needed (this is why the matrix Q was partitioned earlier):

Theorem 1

$$\begin{pmatrix} A & B \\ B^T & C \end{pmatrix}^{-1} = \begin{pmatrix} P & R \\ R^T & U \end{pmatrix}$$

with

$$\begin{aligned} P &= A^{-1} + A^{-1}B(C - B^T A^{-1} B)^{-1} B^T A^{-1} = (A - BC^{-1}B^T)^{-1} && (2.2) \\ U &= C^{-1} + C^{-1}B^T(A - BC^{-1}B^T)^{-1}BC^{-1} = (C - B^T A^{-1} B)^{-1} && (2.3) \\ R &= -A^{-1}BU = -P^T BC^{-1} && (2.4) \end{aligned}$$

and the alternatives for P and U are obtained directly from the Bartlett-Sherman-Morrison-Woodbury formula:

$$A^{-1}B(C - B^T A^{-1} B)^{-1} B^T A^{-1} = (A - BC^{-1}B^T)^{-1}$$

For further details see [27].

By using the partitioned inverse equations, we can find the posterior distribution we want:

$$P(\mathbf{y}^P|\mathbf{y}^D = \mathbf{y}^*, H) = \frac{1}{Z^P}\exp\left(-\frac{1}{2}(\mathbf{y}^P - \hat{\mathbf{y}})^T S^{-1}(\mathbf{y}^P - \hat{\mathbf{y}})\right) \tag{2.5}$$

where $S = (Q^{PP} - Q^{PD}(Q^{DD})^{-1}Q^{DP})$ and $\hat{\mathbf{y}} = Q^{PD}(Q^{DD})^{-1}\mathbf{y}^D + \boldsymbol{\mu}$. Hence $\hat{y}_i$ is the mean prediction at point $\mathbf{x}_{n+i}$, and $\sqrt{S_{ii}}$ is the corresponding standard deviation.

And so we have found the prediction points and error bars for any points we might choose. The great benefit of Gaussian processes is that the integrals involved in calculating the posterior distributions can be evaluated exactly. This means that the accuracy of Gaussian processes is guaranteed. Furthermore because the methods used only involve inverting the covariance of the data points, the method can be used efficiently for any number of predictive points. The main catch with Gaussian processes is a scaling one. Inversion of an $n \times n$ matrix is an $o(n^3)$ process. Hence for large datasets, Gaussian process techniques do not scale very well. The payoff is that a Gaussian process is able to capture more complicated functions as the datasize increases, whereas a neural network with a fixed number of hidden neurons, say, has its functionality fixed by the network size.

There are ways of improving the scaling of Gaussian processes. Gibbs and Mackay [18] showed that approximate methods could be used to obtain a matrix inverse in $o(n^2)$ steps. Parts of this thesis will be looking at situations in which the inversion of the covariance function can be speeded up.

2.7.1.2 Hyperparameters

So far it has been assumed that the hyperparameters of the Gaussian process are known. This is more likely with Gaussian processes than with neural networks, because the hyperparameters are interpretable scaling factors. Even so, some uncertainty about the hyperparameters would often be expected. In these cases the distribution over the hyperparameters needs to be integrated out. Once again this is intractable, and so the approximate methods introduced earlier will have to be used at this higher level of inference.

2.8 Moving forward

We have seen in this introduction how Gaussian process models have tackled some of the problems associated with Bayesian inference with neural networks. In the next two chapters, the emphasis will be on ways of extending the repertoire of situations appropriate for the use of Gaussian process techniques, as well as finding ways of speeding up the process of inversion.

Chapter 3

Non-Gaussian Process Methods

3.1 Introduction

[1]Gaussian processes were introduced in the previous chapter as a multi-purpose modelling tool, well suited to the Bayesian approach. However in their simplest form there are a number of problems they are not very well suited to. In this chapter we introduce a number of these, and show how simple and fast modifications of the basic Gaussian process techniques can significantly extend the modelling power of the method.

3.2 Outline of Gaussian process approaches

In the previous chapter the underlying theory of Gaussian process modelling was introduced. In this chapter we will investigate the use of different types of covariances, as well as ways of combining Gaussian processes to model non-Gaussian situations.

The notation of the previous chapter is used here. Let $\{\mathbf{x}_i\}$ denote the set of input space data-points $\{\mathbf{x}_i\}$ $i = 1, 2, \ldots, n$ combined with the points $\{\mathbf{x}_i\}$ $i = n+1, 2, \ldots, m$ at which predictions will be made. A superscript D (for DATA) denotes an m-vector truncated to the elements $i = 1, 2, \ldots, n$, and a superscript P (for PREDICTION) denotes an m-vector truncated to the elements $i = n + 1, \ldots, m$.

An unknown function $f(\mathbf{x})$ generates datum f_i at point $\mathbf{x}_i$, corrupted by zero mean Gaussian noise η_i, with variance σ^2. As before we define $\mathbf{y}$ by

$$y_i = \begin{cases} f_i + \eta_i & i = 1, 2, \ldots, n \\ f_i & i = n+1, \ldots, m \end{cases}$$

The covariance Q is once again defined by

$$Q_{ij} = \begin{cases} C_{ij} + \delta_{ij}\sigma^2 & i, j \leq n \\ C_{ij} & \text{otherwise} \end{cases}$$

[1]A part of this chapter has been published in [28]

be partitioned into the form

$$\begin{pmatrix} Q^{DD} & Q^{DP} \\ Q^{PD} & Q^{PP} \end{pmatrix}$$

Assuming a Gaussian process prior over the functions f and receiving data $\mathbf{y}^D = \mathbf{y}^*$ at points $\mathbf{x}^D$, we obtain predictions 2.5

$$P(\mathbf{y}^P|\mathbf{y}^D = \mathbf{y}^*, H) = \frac{1}{Z^P}\exp\left(-\frac{1}{2}(\mathbf{y}^P - \hat{\mathbf{y}})^T S^{-1}(\mathbf{y}^P - \hat{\mathbf{y}})\right)$$

where $S = (Q^{PP} - Q^{PD}(Q^{DD})^{-1}Q^{DP})$ and $\hat{\mathbf{y}} = Q^{PD}(Q^{DD})^{-1}\mathbf{y}^D + \boldsymbol{\mu}$ by the partitioned inverse equations. Hence $\hat{y}_i$ is the mean prediction at point $\mathbf{x}_{n+i}$, and $\sqrt{S_{ii}}$ is the corresponding standard deviation.

3.3 Problems with Gaussian processes

As they stand in the previous section, Gaussian processes suffer from a number of problems. These include:

- Stationarity
- Gaussianity
- Lack of any hierarchical structure

In this chapter we take a look at these issues and try to develop a framework for using Gaussian processes to tackle some standard problems which do not have a basic stationary Gaussian and non-hierarchical structure. We will start by assuming that any hyperparameters are fixed and known. The problem of inferring or integrating out the hyperparameters will be dealt with later.

3.3.0.3 Gibbs sampling

Throughout this chapter, references will be made to Gibbs sampling techniques. These methods will generally be used to approximate the integral over a latent variable space.

A standard technique for approximating integrals involves using Monte-Carlo Markov chain (MCMC) techniques. Suppose we have the integral

$$\int dF(x)r(x)$$

Then MCMC involves generating a sample from the distribution $F(x)$, (say the set $\{x_i\}$ for $i = 1, 2, \ldots, s$), and using the sum

$$\sum_{i=1}^{s} r(x_i)$$

to approximate the relevant integral. The main problem lies in the fact that it is rare that a sample can be generated directly from the distribution F. Instead a Markov chain is set up which has F as its equilibrium distribution. The sample can then be taken from the Markov chain after it has settled down to something near its equilibrium.

In many situations the number of possible configurations of latent variables grows exponentially with the size of the training set. Hence, for any reasonable size of problem, it is impossible to test all possible values. Furthermore direct sampling from the posterior distribution of the latent variables is usually impossible.

On the other hand it is usually straightforward to sample from the posterior distribution of one of the latent variables, given the value of the other latent variables. This is the basis of Gibbs sampling techniques. In the applications used here, the sample space will be countably finite. The following argument is a description of Gibbs sampling specifically applicable to such finite systems. Gibbs sampling is also applicable in other (non-finite) systems but the following reasoning would be a little different.

Gibbs sampling works by splitting the set of latent variables L into a number of distinct parts L_k say for $k = 1, 2, \ldots, K$ where $L = \bigcup_k L_k$. Often each k will represent an individual latent variable. After initializing the latent variables at some sensible values, each set L_k is then chosen in turn and a sample is generated from the posterior distribution of the latent variables L_k given the values of all the rest of the latent variables. This is repeated many times. The process is a Markov chain, because at each stage the new state is dependent only on the previous. Furthermore, given that the state space is finite and assuming that all the transition probabilities are nonzero (a sufficient condition, but one that can be relaxed) it is an irreducible ergodic Markov chain (all states can be reached in finite time from all other states, it has a finite state space and is aperiodic), and therefore has a unique equilibrium distribution, the Gibbs distribution

$$P_G(x) = \frac{1}{Z} \exp(-H(x))$$

where $H(x)$ is some Energy function. Now all we need to be sure of is that this unique equilibrium distribution of the Markov chain is in fact the posterior distribution of the latent variables. To show this we use *detailed balance*. Detailed balance occurs for distribution P_D and transition probabilities T if for any two states A and B

$$P_D(A)T(A \to B) = P_D(B)T(B \to A)$$

The reason that detailed balance is useful is that detailed balance implies the invariance of the distribution P_D under a Markov chain with transition probabilities T. In other words if we find some P_D for which our transition probabilities T satisfy detailed balance, then that P_D is an invariant distribution under the Markov chain.

Suppose then we choose T to be the Gibbs sampler transition probabilities. Then we find that detailed balance holds if $P_D = P$, the true posterior distribution of the latent variables. Hence the true posterior is an invariant distribution of the Markov chain.

Finally, ergodicity implies that the equilibrium Gibbs distribution is the unique invariant distribution. Hence the true posterior distribution of the latent variables must be the unique equilibrium distribution of the Markov chain, and we can safely say that the

Markov chain must converge to the required posterior distribution.

For further details on Gibbs sampling and sampling methods in general, the reader should refer to [25, 29].

3.4 Non-Gaussian noise

One of the simplest problems that might be faced is one where the prior noise model is not Gaussian. For example the data may contain points which are outliers, or the correct distribution might be skew. Non-Gaussian noise cannot be directly incorporated into the Gaussian process framework because it destroys the Gaussian structure of the final process. This not only means that the process cannot be described in the simple form (2.1) but also removes the analytic nature of the marginalisation and conditioning upon which these methods rely.

Radford Neal [30] shows how a t-distributed noise prior can indirectly be incorporated into the Gaussian process framework. This distribution would be more suitable than a Gaussian for data where outliers are expected. He notes that the t-distribution is one of the class of probability-weighted fixed-mean Gaussian distributions [31, section 28.3]. This means it can be expressed in the form

$$P(x) = \int dW(\sigma) \frac{1}{Z(\sigma)} \exp\left(-\frac{(x-\mu)^2}{2\sigma}\right)$$

where the weighting Kernel W is a distribution.

Therefore a t-distributed variable can be thought of as a normally distributed variable whose variance is drawn from the weighting kernel distribution W. This also gives an intuitive grasp of why it acts as a distribution for outliers: sometimes the noise value comes from a Gaussian distribution with a larger variance than the average. For the t-distribution W is an inverse gamma distribution [30].

This suggests that in order to incorporate the t-distributed noise into the Gaussian process framework we need only to introduce a latent variable for each point. This latent variable is used to indicate what variance the Gaussian noise has at that particular point. Given the values of the latent variables, our system is now fully Gaussian again, and all the standard Gaussian process techniques can be used. All that is left is to integrate over the latent variable space with respect to the weighting kernel. Because the kernel is a distribution, this integration can be approximated by Gibbs sampling over the latent variable space. This Gibbs sample will give a mixture of Gaussian processes, each corresponding to a different set of latent variable values.

We need to update Q^{DD} each time a sample is made. This corresponds to changing one diagonal element Q^{DD}_{ii} of Q^{DD}. Suppose that the diagonal element changes from Q^{DD}_{ii} to $Q^{DD}_{ii} + \Delta$. Then using the Bartlett-Sherman-Morrison-Woodbury formula we find that

$$(Q^{DD})^{-1}_{new} = (Q^{DD})^{-1}_{old} - \frac{\mathbf{q}_i \mathbf{q}_i^T}{\Delta - (Q^{DD})^{-1}_{ss}}$$

where $(\mathbf{q}_i)_j = (Q^{DD})^{-1}_{ij}$. Hence the update cost is $o(n^2)$ for each i, making a whole Gibbs pass $o(n^3)$

Of course this method is not only appropriate for t-distributions or other weighted fixed mean distributions. A more general set of weighted Gaussians can be used:

$$P(x) = \int dW(\sigma,\mu) \frac{1}{Z(\sigma,\mu)} \exp\left(-\frac{(x-\mu)^2}{2\sigma}\right)$$

This allows the mean to be varied as well as the variance. A much greater class of distributions can be expressed as general weighted Gaussians. This includes skew distributions, and bimodal ones. For example a bimodal Gaussian might be used where there is digital interference on an analogue device. The same methods can be used for all these distributions, although the mean term of the Gaussian process will need to be included in the framework.

3.4.1 Digital noise

The use of weighted Gaussians was tested on the case where some standard signal was corrupted by noise from a digital device. Suppose we have a signal where the noise source comes partly from a digital device. In other words the signal is corrupted by another additive signal which takes one of two values $\pm l$ say at each point. This second signal is changing rapidly. Then we can model this with a Gaussian process where the noise term consists of a digital shift combined with a Gaussian noise. Hence we get a noise model with a bimodal distribution. In order to get this into the usual Gaussian process framework we sample from the two possible states of the digital noise, and set the mean of the Gaussian process to account for this. The task can be broken down in this way.

- Initialize the latent variables s_i for $i = 1, 2, \ldots, n$) which represent the state of the digital noise at each point (either $+1$ or -1 say).
- REPEAT
  - Choose i randomly without replacement from set $\{1, 2, \ldots, n\}$.
  - Set the Gaussian process means to be the digital noise level l times the state at each point.
  - Set up the covariance matrix using only the Gaussian noise part, and not the digital noise.
  - Use standard Gaussian process methods to calculate P=P(data|digital states, everything else).
  - dL = change in lnP if s_i were to be set to be equal to $-s_i$.
  - Choose a random number R from Uniform$(0,1)$. If $R > 1/(1+\exp(-dL))$ then $s_i = -s_i$.
  - If the system appears to be near equilibrium, accumulate the mean predictor, and the square mean predictor at each prediction point, given by the Gaussian process.

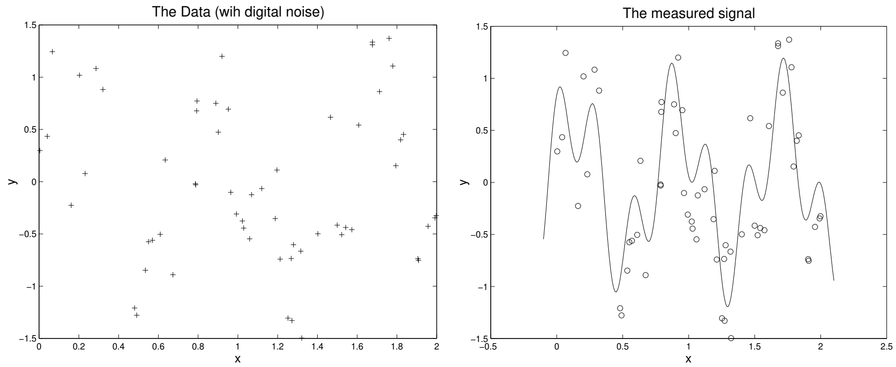


Figure 3.1: Regression with digital noise: data and signal

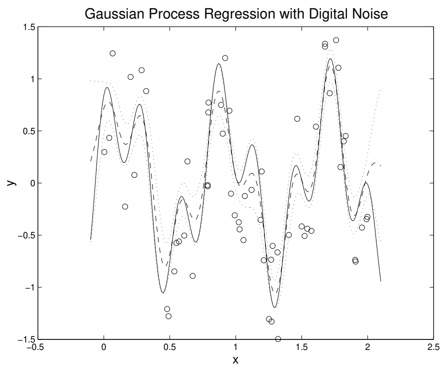


Figure 3.2: Process predictor for signals with digital noise (solid=true, dashed=GP mean predictor, dots=error bars)

- UNTIL the sample size is large enough.
- Calculate the average mean predictor $\mu(x)$ and the average mean square predictor $M(x)$.
- The mean prediction at a point x is given by $\mu(x)$, the variance by $M(x) - \mu(x)^2 + \sigma(x)^2$, where $\sigma(x)^2$ is the variance of the Gaussian process predictor (it is independent of the state of the digital noise.)

Computationally, as the data size increases, this method scales the same way as a simple Gaussian process.

The following graphs show this method tested on a toy problem. The dataset consisted of sixty points sampled from the given function, corrupted by both Gaussian noise of standard deviation 0.1 and digital noise of magnitude ± 0.4. Figure 3.4.1 shows the data used and the actual source signal, and the final figure 3.4.1 shows the mean and error bars of the Gaussian process predictor. The hyperparameter values used to generate the figure were $r_l = 0.1$, $\theta_1 = 0.8$, $\theta_2 = 0$, $\sigma = 0.1$, with the digital noise value given by ± 0.4.

3.5 Non-Gaussian and non-stationary process functions

There are a number of ways of generalizing the problem to that of non-Gaussian process functions. Here we will focus on those which involve using Gaussian processes to modify a second non-Gaussian distribution. Many of the models we discuss here will also produce non-stationary processes for suitable choices of the relevant functions.

We have already noted that it is not necessary for a Gaussian process model to have a zero mean. However the introduction of a mean term corresponds to a known mean prior, and hence this term can simply be subtracted from the target data to get a corresponding set of data which can properly be modelled by a zero mean process. Note that any non-zero mean process is non-stationary, although can easily be transformed into a stationary process by this method.

A different situation arises if we believe the data consists of a Gaussian process variation on top of (or added to) another independent random process. Such a system can be modelled with a Gaussian process where the prior mean is not fixed, but is itself a random variable. Then we cannot simply subtract out the mean, but instead end up with a different process prior, which is non-Gaussian for most situations. Many interesting modelling processes can be formed using these methods. For example there might be some economic process which would generally be expected to follow the level of the overall economy, but with some delayed effects. This could be modelled with a Gaussian process whose mean is a function of the relevant economic index I with some time delay d built in.

$$\mu(t) = I(t - d)$$

Then given a suitable prior over the parameter d, the mean vector can be chosen by sampling from d. As changing the mean vector has no effect on the covariance matrix of the Gaussian process model, each Gibbs sample is inexpensive, and this method adds little computational time on to a standard Gaussian process approach.

Another straightforward way of dealing with certain types of non-Gaussian distribution can be gained by passing the dependent variable space through a non-linear transformation before modelling using a Gaussian process. This is valid for the standard transformations, such as with logarithmically scaled variables, but might also be used in situations where the expected functions have, say flatter/sharper maxima or minima than those produced by a standard Gaussian process. Such methods are trivial for standard Gaussian processes, as the non-linear transformation can be made once and for all before the modelling session. However in situations where the Gaussian process mean is being obtained through sampling, such transformations will need to be done for each sample. This can create some powerful non-Gaussian modelling tools. This can be extended further as different nonlinear transformations can be used for different points in space.

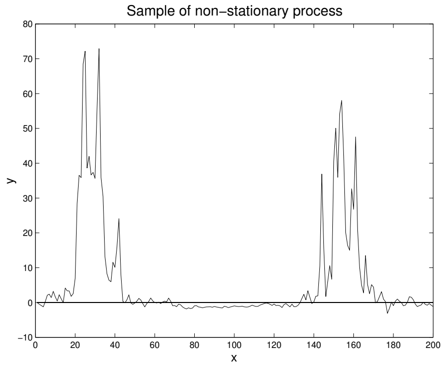


Figure 3.3: Process sample for a non-stationary problem

3.5.1 Illustration

In order to illustrate this the following Gaussian process is set up: Suppose the function is expected to have some cyclic variation, which we could choose to model with a sinusoidal mean μ, choosing suitable priors over the frequency, phase and amplitude. At the peaks of the cyclic variation the signal is expected to have sharp peaks, while at the bottom, a squarer form is expected. To achieve this the Gaussian process output is mapped through the function

$$f(y) = a\mathrm{sgn}(y)(|ry/w|^r)$$

where a, w are scaling parameters and $r(\mu)$ is another parameter (note that the redundancy helps to allow good priors to be set on each of these parameters). We set $r = \alpha^{\beta\mu}$ for hyperparameters α, β, and put priors over a and w.

So what might such a process look like? To show this we illustrate samples of this function in figure 3.5.1. Here all the hyperparameters are sampled, and the signal is sampled from the resulting Gaussian process.

Because such a process is created by sampling without modifying the covariance matrix of the original Gaussian process, it is computationally cheap to use.

3.6 Non-stationary and hierarchical distributions

The introduction of non-stationarity into the Gaussian process framework can be straightforward. We have seen that sometimes the non-stationarity is simply an additive mean. Not all non-stationarities can be treated that way though: it might be that given the means, the covariance function still changes from one part of the space to another. Modelling these types of processes requires a different approach.

Sometimes we might know apriori the non-stationary covariance structure. If this is the case then the Gaussian process treatment can be used without modification. We simply have a position dependent covariance function, and calculate the covariance matrix between two points from their absolute rather than relative positions.

This situation is rarely the case though. Often we might know exactly the nature of

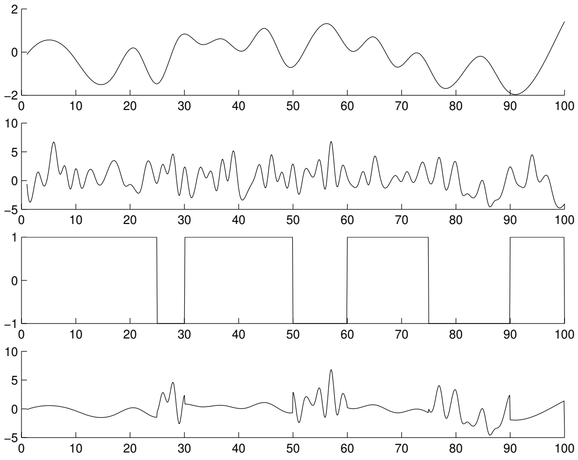


Figure 3.4: Example source graphs, and final signal in switching problem

the non-stationarity. In these situations we can follow a similar procedure to that taken in the previous section. We can draw up a model representing the different possible non-stationary covariance structures. Then we can sample from these models to choose a single model, from which an exact covariance function can be calculated. When a number of these samples are collected the final integration over all possible models of the non-stationarity can be approximated by the relevant sample sum.

There is a catch to this method though. Unlike in all the previous situations, each sample corresponds to a distinctly different covariance matrix. As a result a matrix inverse has to be calculated for each sample, at a much greater computational expense. This can become a significant problem if it takes some time to explore the sample space and to settle to an equilibrium distribution.

The following section will look in greater detail at one such non-stationary problem of great importance, and show ways of making the sampling procedure more efficient.

3.6.1 Determining different signal regimes: Gaussian process mixtures

Very often the data under study has not been generated from a stationary process. A common example of this is where a number of different signal sources are present, and the observable signal is created by switching between these different regimes.

In this problem, it is believed that there are a fixed number of source signals. It is on the whole assumed that these signals are independent. The measured signal, however, only has information from any one of the signals at any one time. In other words at any point the measured signal can be seen to be created by a signal chosen from the set of sources by some discrete random process. Figure 3.6.1 illustrates this. The top two graphs are two independent sources. The third graph illustrates the switching process. The final graph gives the final signal made up of parts of each of the two sources.

Hence the resulting signal is made up of different regimes, each of which gives the state of one of the source signals.

Here this situation is modelled with a mixture of Gaussian processes in the way that has been described in the previous section. Latent variables represent which of the current regimes generated a sample datum. Then different hyperparameters or covariance structures can be used to represent the characteristics of the different regimes.

The great benefit of Gaussian processes is that the covariance matrix structure can represent many different signal structures and textures, from smooth curves to random fractal textures.

As it is not known which regime is generating the signal at any point, and the structure of the signals is unknown, these variables/parameters are given prior distributions which should be integrated over.

Suppose then we have a dataset of size n with data at points x_i, $i = 1, 2, \ldots, n$ given by values y_i respectively. We believe this data to be generated by a system switching between two different smooth signals.

Let s be some switching process, which for now we assume to be binary. Let $s_k = s(x_k)$ denote the regime which generated datum k. Then we can form the Gaussian process prior

$$P(\mathbf{y}|H) = \sum_{\mathbf{s} \in B(n)} p_{\mathbf{s}} \Phi_{\mathbf{s}}(\mathbf{y}; H)$$

where $B(n)$ is the set of binary vectors of length n and $p_{\mathbf{s}}$ is the prior over the latent variables s which will be defined later. Φ is a Gaussian kernel of the form

$$\Phi_{\mathbf{s}} = \frac{1}{Z_{\mathbf{s}}} \exp\left(-\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu}(H))^T Q_{\mathbf{s}}(H)^{-1}(\mathbf{y} - \boldsymbol{\mu}(H))\right)$$

with $\boldsymbol{\mu}(H) = (\mu_{s_1}, \mu_{s_2}, \ldots, \mu_{s_m})^T$ where $\mu_1, \mu_2 \in H$. The covariance function $Q_{\mathbf{s}}$ is given by

$$(Q_{\mathbf{s}})_{ij} = \begin{cases} \theta_{1s} \exp\left(-\frac{1}{2}\sum_{l=1}^{L} \frac{(x_i^{(l)} - x_j^{(l)})^2}{r_{ls}^2}\right) + \theta_{2s} + \delta_{ij}\sigma^2 & \text{for } s = s_i = s_j \\ 0 & \text{otherwise} \end{cases}$$

which says that within a given regime we have the usual smooth functions, but there is no correlation between the points in different regimes. Note that we are free to choose whatever covariance functions we want for the within-regime part of the above definition. We might want to have different hyperparameters for the two regimes. We might want to have completely different types of covariance functions. Whatever we choose, the positive semidefinite nature of the resulting covariance matrices is ensured, because the matrix partitions into two parts. After suitable rearrangement of the columns and rows, we get a matrix of the form

$$\begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix}$$

We know both A and B are positive semidefinite because they are covariance matrices for a standard Gaussian process. Hence the resulting matrix is also positive semidefinite.

It is not enough to have a probability distribution defined by the Gaussian process. A new random process has been introduced, the switching process, and this needs to be given a suitable prior definition. Once again we are at complete liberty as to what we choose. However for a one dimensional system a Poisson process would be a simple and sensible prior process to choose. By this we mean that the switching points are determined by a Poisson process.

In one dimension we can (and do) assume that the switching regime is given by a Poisson process with rate given by λ. Suppose the order $k = 1, 2, \ldots, n$ of the data points is arranged so that x_k is increasing. Then the probability $P(s(x_k) = s(x_{k-1}|s(x_{k-1}))$ is given by the probability of an even number of switches between the two points x_{k-1} and x_k:

$$\cosh(\lambda(x_k - x_{k-1}))\exp(-\lambda(x_k - x_{k-1}))$$

Hence $P(s(x_k) \neq s(x_{k-1})|s(x_{k-1})) = \sinh(\lambda(t_k - t_{k-1}))\exp(-\lambda(t_k - t_{k-1}))$ where λ is a hyperparameter. The complete prior over s can then be written

$$\begin{aligned} P(s|H, \mathbf{x}) &= P(s(x_1)) \prod_{k=2}^{n} P(s(x_k)|s(x_{k-1})) \\ &= \frac{1}{2} \prod_{k=2}^{n} [cosh(\lambda(x_k - x_{k-1}))\exp(-\lambda(x_k - x_{k-1}))]^{I(s_k = s_{k-1})} \\ &\quad [sinh(\lambda(x_k - x_{k-1}))\exp(-\lambda(x_k - x_{k-1}))]^{I(s_k \neq s_{k-1})} && (3.1) \\ & && (3.2) \end{aligned}$$

where $x_1 < x_2 < \ldots < x_n$. Of course other switching priors can be used. For example in higher dimensions a Gaussian process classification prior could be used.

All the priors are now defined, and the problem can be passed through the usual Gaussian process machinery. The first stage involves a tractable Gaussian marginalisation. The second stage involves an intractable integration over the latent variables. (Note that here we assume fixed hyperparameters.) The required integration over the latent variables is not analytic, and so we need to resort to some form of approximation. We will once again use Gibbs sampling methods.

3.6.1.1 The posterior distribution of the latent variables

The Gaussian process which has been defined is given by

$$P(\mathbf{y}^D|\mathbf{s}, H) = \frac{1}{Z}\exp\left(-\frac{1}{2}(\mathbf{y}^D - \boldsymbol{\mu}^D)^T(Q^{DD})^{-1}(\mathbf{y}^D - \boldsymbol{\mu}^D)\right)$$

with $P(\mathbf{s}|H)$ given by the relevant prior. Hence we can find the posterior distribution of the latent variables by using Bayes theorem

$$P(\mathbf{s}|H, \mathbf{y}^D) = \frac{P(\mathbf{y}^D|\mathbf{s}, H)P(\mathbf{s}|H)}{P(\mathbf{y}^D|H)}$$

We have already noted that we cannot sum over all $\mathbf{s}$, so we need to find a way to choose a set of $\mathbf{s}$ which somehow represent the distribution well. Let us look at the distribution of one latent variable s given the rest $\mathbf{s}'$. Without loss of generality (we can just relabel the points) we assume that s is the last latent variable in the list. We also call the corresponding output value y, with the rest denoted by $\mathbf{y}'$. Note that $\mathbf{y}'$ is conditionally independent of s given $\mathbf{s}'$. We can now write the covariance matrix Q^{DD} as

$$\begin{pmatrix} A & \mathbf{b} \\ \mathbf{b}^T & c \end{pmatrix}$$

where A is independent of the value of s, $\mathbf{b}$ is a vector and c is a scalar. Note that Q^{DD}_{ij} depends only on s_i and s_j, and $\boldsymbol{\mu}_i$ depends only on s_i. Then we have

$$P(s|\mathbf{s}', H, \mathbf{y}^D) = \frac{P(\mathbf{y}^D|s, \mathbf{s}', H)P(s|\mathbf{s}', H)}{P(y^D|\mathbf{s}', H)}$$

Now $P(s|\mathbf{s}', H)$ comes simply from the prior distribution of the latent variables. That leaves

$$P(y, \mathbf{y}'|s, \mathbf{s}', H) = P(y|s, \mathbf{y}', \mathbf{s}', H)P(\mathbf{y}'|s, \mathbf{s}', H)$$

but $\mathbf{y}'$ is independent of s (the dependence comes from the conditioning on y), and so $P(\mathbf{y}'|s, \mathbf{s}', H) = P(\mathbf{y}'|\mathbf{s}', H)$ implying

$$P(s|\mathbf{s}', H, \mathbf{y}^D) = \frac{P(y|s, \mathbf{y}', \mathbf{s}', H)P(\mathbf{y}'|s, \mathbf{s}', H)P(s|s', H)}{P(y^D|\mathbf{s}', H)}$$

Now $P(y|s, \mathbf{y}', \mathbf{s}', H)$ is just the predictive distribution over y, and is given by (2.5)

$$P(y|s, \mathbf{y}', \mathbf{s}', H) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu(s))^2}{2\sigma(s)}\right)$$

with $\mu(s) = \mathbf{b}^T(s)A^{-1}\mathbf{y}'$ and $\sigma(s) = c - \mathbf{b}^T(s)A^{-1}\mathbf{b}(s)$.

Putting all this together we have

$$P(s|\mathbf{y}^D, \mathbf{s}', H) \propto \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu(s))^2}{2\sigma(s)}\right) P(s|\mathbf{s}', H)$$

So sampling is simply a matter of choosing one of the two values of s according to the two probabilities given above.

Most of the operations involved here cost only $o(n^2)$ floating point operations. The exception is the inverse A^{-1} which is $o(n^3)$. We now show how this can be reduced using information already obtained.

In order to produce the Markov chain, the system is initialised with a certain choice of latent variables, and the inverse covariance matrix $(Q^{DD})^{-1}$ is calculated. In fact, from this point on, no more matrices inversions need be done. The reason why this is the case is now explained. Suppose latent variable s is chosen for update as before. Then the matrix A^{-1} can be calculated from $(Q^{DD})^{-1}$ using the partitioned inverse equations. These state that $(Q^{DD})^{-1}$ can be written

$$(Q^{DD})^{-1} = \begin{pmatrix} P & \mathbf{r} \\ \mathbf{r}^T & \sigma \end{pmatrix}$$

where $\sigma = (c - \mathbf{b}^T A^{-1}\mathbf{b})^{-1}$, $\mathbf{r} = -\sigma A^{-1}\mathbf{b}$ and $P = A^{-1} + \mathbf{r}\mathbf{r}^T/\sigma$.

Hence $A^{-1} = P - \mathbf{r}\mathbf{r}^T/\sigma$. This step only involves an outer product and hence involves only $o(n^2)$ floating point operations, reducing the whole sampling process to $o(n^2)$ per sample.

Once the sample has been made further work may need to be done. If s does not change then we already have $(Q^{DD})^{-1}$ and so can stop there. However if s changes value, we need to calculate $(Q^{DD})^{-1}$ again for the new latent variables. Once again we cannot afford to do this directly. However we can calculate $(Q^{DD})^{-1}$ from A^{-1}: The new $\sigma(s)$ and $\mathbf{r}(s)/\sigma(s)$ have already been calculated to perform the sampling. It remains to calculate $P(s) = A^{-1} + \mathbf{r}(s)\mathbf{r}(s)^T/\sigma$, another $o(n^2)$ operation. We now rebuild the new $(Q^{DD})^{-1}$ using $\sigma(s), P(s)$ and $\mathbf{r}(s)$, and can proceed to the next sample.

In conclusion we can do the whole step in $o(n^2)$ operations, through the use of the partitioned-inverse equations. Hence a complete Gibbs run can be completed in $o(n^3)$ floating point operations.

3.6.2 Hyperparameters: integrating-out or maximizing?

Up to now we have assumed that the hyperparameters of the Gaussian process are known. In most situations these scaling parameters are unlikely to be known exactly. In these situations the correct Bayesian procedure is to write down a prior distribution over the hyperparameters, and then integrate out the hyperparameters with respect to their posterior distribution. Once again though, such integrations are intractable.

Calculating the inverse covariance for a Gaussian process is computationally intensive. Therefore approximating the integration over hyperparameters using Monte-Carlo Markov chain approaches can be very slow for large data sizes. One possible approach is to sample from the posterior over the latent variables and the use a maximum posterior value for the hyperparameters. This is a form of generalised expectation maximisation algorithm [32], where a sample distribution over the latent indicator variables is used to approximate the true distribution.

The steps of the algorithm are:

- Choose suitable values of the hyperparameters, H.
- Choose suitable starting values for the latent variables $\mathbf{s}$.
- Gibbs sample the latent variables to get a sample from $P(\mathbf{s}|H, \mathbf{y}^D)$: $s_1, s_2, \ldots, s_G$
- Use this sample as an approximating distribution for the posterior over the latent variables: $Q(\mathbf{s}|H) = \frac{1}{G}\sum_{g=1}^{G}\delta(s - s_g)$

- Now the log posterior of the hyperparameters can be bounded below using

$$\ln P(H|\mathbf{y}^D) \geq -\sum_{\{\mathbf{s}\}} Q(\mathbf{s}|H)\ln Q(\mathbf{s}|H) + \sum_{\{\mathbf{s}\}} Q(\mathbf{s}|H)(\ln P(H|\mathbf{s},\mathbf{y}^D) + \ln P(\mathbf{s}|\mathbf{y}^D)), \tag{3.3}$$

- Move towards the maximum of this lower bound to the log posterior value of the hyperparameters, given the data. This involves choosing H which maximises

$$\frac{1}{G}\sum_{g=1}^{G} \ln P(H|\mathbf{s}_g, data)$$

  where we have substituted for Q in (3.3) and ignored terms independent of H.

- Repeat these steps until suitably near convergence.

This process depends on gradient information being available in order to move towards the maximum. This can be obtained directly from the Gaussian process formalism. Given the latent variables s, the derivative of the log likelihood $\ln L_s = \ln P(\mathbf{s}|H, data)$ with respect to some hyperparameter θ is given by

$$\frac{\delta(\ln L_s)}{\delta\theta} = -\frac{1}{2}\text{trace}\left((Q_s^{DD})^{-1}\frac{\delta Q_s^{DD}}{\delta\theta}\right) + \frac{1}{2}(\mathbf{y}^D - \boldsymbol{\mu}^D)^T(Q_s^{DD})^{-1}\frac{\delta Q_s^{DD}}{\delta\theta}(Q_s^{DD})^{-1}(\mathbf{y}^D - \boldsymbol{\mu}^D)$$

This can be added to the derivative of the log prior, and summed over the sample $\{s_i\}$ to get the gradient we need.

In this algorithm, the convergence proofs used for the strict expectation maximization (EM) algorithm are not appropriate, because we have replaced the true distribution with a sample. However if that sample is reasonably representative, the algorithm is likely to move nearer the maximum likelihood value. It might be worthwhile to increase the sample size as the algorithm gets closer to its end point. This approach bears a certain similarity to variational approaches to the EM algorithm (see [33] for a variational method applied to a factorial hidden Markov model of switching systems). Instead of picking the best match between an approximating distribution and the true distribution, a sample is used as the approximating distribution.

One result of this is that a looser stopping criteria will be needed to account for the fact that there will be some random fluctuations about the maximum likelihood point, due to the different latent variable samples. One benefit is that the algorithm is less likely to get stuck in a local minima: the fluctuations can act as a form of annealing.

3.6.3 Assessing the validity of the sampling regime

We have shown that it is possible to use a Gibbs sampling method, where each step of the Gibbs sampler takes only $o(n^2)$ floating point operations. However we have not stopped to discuss the validity of the Markov chain which this method produces. Just because there is a theoretical equilibrium distribution does not necessarily mean that we can get near that distribution or sample from that distribution in any sensible time. In this section we discuss the properties of the posterior latent space, and see how the sampling regime might fare in different circumstances.

There are two main problems to be considered when assessing the power of a MCMC method. The first of these involves the possibility of undirected random walk behaviour: if the choice of possible new points tends not to direct the chain to plausibly better regions of the space then points with a higher posterior probability might not be found for a long time. The second problem comes from that of local maxima: if a local maximum is surrounded by areas of low probability, then the Markov chain can get stuck in that local maximum, and not properly sample from the whole space in a reasonable time.

These types of problems will occur in our system under certain circumstances. The reason for this is simple: for any given value of the latent variables, there are only n other states (the hypercube neighbour states) that are tested for possible transitions. There are situations where we would expect these neighbour states to have much lower probabilities. The main reason for this stems from the prior distribution over the latent variables.

On the whole, the prior distribution over the latent variables will encapsulate the concept that the value of latent variables for neighbouring points will be positively correlated. (This is the case when a Poisson process prior is used, for example). In situations where the correlation between neighbouring points is high, the majority of the transitions available to the Markov chain will fail. This is because given a reasonable current state, the test state will often involve misaligning a latent variable with those of neighbouring points. This state has low prior probability, and therefore is more likely to have lower posterior probability.

For example, consider a system with $n = 25$ training points. Suppose the value of the latent variables at the current state of the Markov chain is given by

$$(11111000000000111111000000)$$

Then one of the possible test transitions is

$$(11111000001000111111000000)$$

but this will have a much lower prior probability because the new '1' has two '0' neighbours. From this it is clear that the majority of possible transitions are unlikely.

If on the other hand the prior on the latent variable space does not strongly correlate adjacent latent variables, the simple Gibbs sampling structure we have outlined can work well.

3.6.4 Better sampling methods

In those situations where the prior distribution produces correlations between latent variables belonging to adjacent points, better sampling methods need to be developed. One way of making improvements is to update latent variables in bigger blocks. If the blocks are chosen to be blocks of adjacent latent variables, then the correlation between variables in the block and those outside will become more of an 'edge effect'. This will have less of a constraining effect on the variables in the block. We show how this can be done to help solve this problem while at the same time improving the efficiency of the techniques!

Suppose now that one step of the sampling process involves updating a block of k consecutive latent variables. (Note that in some situations we might have to make k variable in order to maintain ergodicity.) Then we can use the same techniques as before, but using the more general form of the partitioned inverse equations. Working through the maths again, we denote the block of latent variables to be updated by $\mathbf{s}^U$ (again, without loss of generality, we assume they are the last k). The remaining latent variables by $\mathbf{s}'$. Likewise $\mathbf{y}^D$ is split into $\mathbf{y}^U$ and $\mathbf{y}'$, and $\boldsymbol{\mu}^D$ is split into $\boldsymbol{\mu}^U$ and $\boldsymbol{\mu}'$. Consider changing $\mathbf{s}^U$ from $\mathbf{s}^U_{old}$ to $\mathbf{s}^U_{new}$. Then

$$\ln P(\mathbf{s}^D_{new}|H, \mathbf{y}^D) - \ln P(\mathbf{s}^D_{old}|H, \mathbf{y}^D) = \ln P(\mathbf{s}^U_{new}|\mathbf{s}', H, \mathbf{y}^D) - \ln P(\mathbf{s}^U_{old}|s', H, \mathbf{y}^D) \quad (3.4)$$

and

$$\ln P(\mathbf{s}^U|\mathbf{s}', \mathbf{y}^D, H) = -\frac{1}{2}(\mathbf{y}^U - \boldsymbol{\mu})^T U(\mathbf{y}^U - \boldsymbol{\mu}) + \frac{1}{2}\ln\det U + \ln P(\mathbf{s}^U|\mathbf{s}')$$

where we have Q^{DD} and $(Q^{DD})^{-1}$ written as

$$\begin{pmatrix} A & B \\ B^T & C \end{pmatrix} \qquad \begin{pmatrix} P & R \\ R^T & U \end{pmatrix}$$

respectively. The partitioned inverse equations give us $U^{-1} = (C(\mathbf{s}^U) - B^T(\mathbf{s}^U)A^{-1}B(\mathbf{s}^U))$ and $\boldsymbol{\mu} = B^T(\mathbf{s}^U)A^{-1}\mathbf{y}' + \mu^U$ (see (2.4)).

Starting with $\mathbf{s}^U = \mathbf{s}^U_{old}$, and knowing Q^{DD}_{old} and $(Q^{DD}_{old})^{-1}$, we can efficiently calculate A^{-1} from the previous covariance matrix using the partitioned inverse equations. It is given by

$$A^{-1} = P_{old} - R_{old}U^{-1}_{old}R^T_{old}$$

In all this the *old* subscript refers to the fact that the matrices are calculated for $\mathbf{s}^U = \mathbf{s}^U_{old}$. This calculation takes two steps, one of order k^3, and the other of order $k^(n-k)^2$. The matrix A, and hence the matrix A^{-1} does not change when $\mathbf{s}^U$ is updated.

Knowing A^{-1} we now want to calculate posteriors for all values of $\mathbf{s}^U$. In fact we only need the differences in the log posteriors (3.4). To do this we need values for $\ln P(\mathbf{s}^U_{new}|\mathbf{s}')$ (which falls out of the prior definition), U_{new}, $\boldsymbol{\mu}_{new}$ and $\ln\det U_{new}$. We have

$$U^{-1}_{new} = (C(\mathbf{s}^U_{new}) - B^T(\mathbf{s}^U_{new})A^{-1}B(\mathbf{s}^U_{new}))$$

and

$$\boldsymbol{\mu}_{new} = B^T(\mathbf{s}^U_{new})A^{-1}\mathbf{y}'$$

For each of the 2^k values of s^U we need to calculate the new U - $o(k(n-k)^2)$, invert U - $o(k^3)$, obtain $\det U$ in order to get the normalisation factor (can fall out of inversion calculations), and perform various other low cost calculations. For k of any size, all this will be costly.

These naive costs can be significantly reduced. B has k different columns denoted $\mathbf{b}_i$, each of which takes two values $\mathbf{b}_i^+$ and $\mathbf{b}_i^-$ according to the corresponding component of $\mathbf{s}^U$. So $(\mathbf{b}_i^{\pm})^T A^{-1} \mathbf{b}_j^{\pm}$ can be calculated in $o(k(n-k))^2$ steps. Then for any s^U the value of $B^T A^{-1} B$ can then be read from this table at a cost of $\mathrm{o}(k^2)$. In fact, because of the method which will be used to calculate U, we will find that only one component at a time need be changed, and so this cost is reduced to $\mathrm{o}(1)$.

Now consider the problem of inverting U. There is no reason that U^{-1} need be calculated from scratch for each possible U. In fact after the first inversion, it is straightforward to step through all possible 2^k values of $\mathbf{s}^U$ by only changing one component from the value of $\mathbf{s}^U$ just visited (a hypercubic graph has hamiltonian paths). This corresponds to changing a single row and column of U, and hence U can be inverted at each step (again using the partitioned inverse equations) in $\mathrm{o}(k^2)$.

There is also the issue of $\det U$. This can also be calculated in $\mathrm{o}(n^2)$ steps:

Proposition 1 *Suppose that we need to calculate* $\det U^*$ *where* U^* *is obtained from* U *by changing only one row and the equivalent column (without loss of generality we presume it to be the first row and column). Suppose also that* U^{-1}, $\det U$ *and* $(U^*)^{-1}$ *are all known. We use the notation*

$$U = \begin{pmatrix} v & \mathbf{v}^T \\ \mathbf{v} & V \end{pmatrix} \quad U^* = \begin{pmatrix} v^* & (\mathbf{v}^*)^T \\ \mathbf{v}^* & V \end{pmatrix} \quad U^{-1} = \begin{pmatrix} w & \mathbf{w}^T \\ \mathbf{w} & W \end{pmatrix} \quad (U^*)^{-1} = \begin{pmatrix} w^* & (\mathbf{w}^*)^T \\ \mathbf{w}^* & W^* \end{pmatrix}$$

Then

$$\det(U^*) = \det(U)(v^* + (\mathbf{v}^*)^T \mathbf{w}^*/w^*)/(v + \mathbf{v}^T \mathbf{w}/w)$$

Proof Now $U^{-1}U = I$, and hence $w\mathbf{v}^T + \mathbf{w}^T V = 0$. Because we can add multiples of other rows to any row of a matrix without affecting the determinant, we use this to find

$$|U| = \begin{vmatrix} v & \mathbf{v}^T \\ \mathbf{v} & V \end{vmatrix} = \begin{vmatrix} v + \frac{\mathbf{v}^T\mathbf{w}}{w} & \mathbf{v}^T \\ 0 & V \end{vmatrix} = \begin{vmatrix} v + \frac{\mathbf{v}^T\mathbf{w}}{w} & 0 \\ 0 & V \end{vmatrix}$$

by adding to the first column $\mathbf{w}^T/w$ times the other columns, and then adding to the first row $\mathbf{w}/w$ times the other rows.

Repeating the same procedure for $|U^*|$ we obtain

$$|U^*| = \begin{vmatrix} v^* & (\mathbf{v}^*)^T \\ \mathbf{v}^* & V \end{vmatrix} = \begin{vmatrix} v^* + \frac{(\mathbf{v}^*)^T\mathbf{w}^*}{w^*} & (\mathbf{v}^*)^T \\ 0 & V \end{vmatrix} = \begin{vmatrix} v^* + \frac{(\mathbf{v}^*)^T\mathbf{w}^*}{w^*} & 0 \\ 0 & V \end{vmatrix}$$

So

$$|V| = \frac{|U|}{v + (\mathbf{v}^T\mathbf{w})/w} \text{ and } |V| = \frac{|U^*|}{v^* + ((\mathbf{v}^*)^T\mathbf{w}^*)/w^*}$$

Putting these two equations for $|V|$ together we get the required result. □

The methods for calculating $B^T A^{-1}\mathbf{y}'$ are the same: $A^{-1}\mathbf{y}'$ is independent of $\mathbf{s}^U$, and can be calculated once off. B^T can again be split up into rows as above. Hence all the necessary calculations can be done once off at a cost of $\mathrm{o}(k(n-k)^2)$.

In summary then, there is a once off cost of $o(k(n-k)^2)$, followed by a cost of $o(k^2)$ for each of the 2^k possible values of $\mathbf{s}^U$.

As a result we can improve on the unit-size-block model which we used earlier. By that scheme, sampling k latent variables would cost $o(kn^2)$ floating point operations. If we choose k so that $2^k k \approx n^2$ (this matches the cost of calculating all the probabilities to the cost of calculating A^{-1}) then we find that this new scheme costs $o(k(n-k)^2)$ floating point operations. This is effectively the same order of magnitude, but in the process we choose between 2^k possible states instead of $k+1$ states, thereby increasing the coverage of the Markov chain. For $n = 100$ this would mean choosing about $k = 10$, giving 1024 times greater coverage, without any increase in the order of magnitude of the computational cost.

This is a great improvement, and makes a big difference in many circumstances. However for very large systems the size of k given above will be relatively small. There are things which might improve things further. The first way is by noting that we have not yet used any prior information we might have to guide our computation. Rather than search through all 2^k possible settings of $\mathbf{y}^U$, why not choose a larger k, but only search through those that are more likely. In doing this we need to be careful to maintaining both ergodicity and detailed balance. Ergodicity will be ensured by varying k. Sometimes, when k is small, all 2^k possible values will be tested. Hence any state can be visited from any other state.

When k is large, on the other hand, only those sequences with a higher prior probability are tested. Because the prior distribution tends to correlate latent variables corresponding to nearby points, it would seem sensible to test latent variable sets with fewer changepoints rather than those with a larger number of changepoints. Limiting the number of changepoints tested for is tempting, but unfortunately this will remove detailed balance.

Instead what we can do is limit the number of new changepoints added. When a changepoint is set at latent variable i say, then all the latent variables from position i onwards change state. If one of the new changepoints is put on top of an old changepoint, it annihilates it (in this two state situation).

Say we have a block of size k, and choose up to c extra changepoints. By limiting to at most c changepoints, that gives a total of $2\sum_{r=0}^{c}({}^{(k-1)}C_r) = o(k^c)$ values of $\mathbf{s}^U$ to be tested. (Of course we also include the current state as the no extra changepoints case). If we choose c by $k^{c+1} = n^2$, then again we will not change the order of magnitude of any computations. For $n = 100$ again, we can now increase k to 20 sometimes, but only test values of $\mathbf{s}^U$ with at most 2 extra changepoints.

The addition of extra changepoints is a symmetric operation. The probability of one state being tested for when in another state is the same as the probability of the other state being tested for when in that one state. This ensures that detailed balance is observed.

In conclusion then, these techniques produce methods by which the space can be sampled from much more extensively, with a complete Gibbs run still only taking $o(n^3)$ flops. This ensures that the convergence and sampling rates are much faster. When utilising prior information in the way described, getting stuck becomes less common, because now the sampling method provides a larger number of high prior probability steps at each iteration.

3.6.5 More general methods

These last methods, flipping groups of latent variables by introducing changepoints, is limited to one dimensional two state systems. For more general systems it would be interesting to assess to what extent Swendsen Wang [34] approaches might be adaptable to this system. Such methods are applicable to lattice spin systems with non-binary spins. This is a possible line of future work.

3.7 Example

These methods were tested on a toy problem. Here two functions are generated from two unit magnitude sine waves of different frequency, with noise levels of 0.1. The two functions are mixed together by regularly switching from one to another. A random sample of 70 points was taken.

This particular form was chosen because it tests these methods quite thoroughly. Detecting regimes with different means is, for example, a relatively simple task. Detecting regimes with different smoothness properties or length scales is somewhat harder.

The function of interest is illustrated in figure 3.5 was used. It is made up of the two separate sin waves in the way that is illustrated in the figure. No knowledge was presumed about the functional form, or periodicity of the data.

We used the approaches discussed to model this. Uninformative hyperpriors were used. The simple unit block sized update process worked well on this process. The Markov chain appeared to have converged after a hundred Gibbs passes, sometimes less, and a relatively small sample size of 50 points was used. The two regimes were often fully detected, or other reasonable segmentations were chosen. The graphs in figure 3.5 give the predictive mean, and error bars for the two signals. The true signals are given as solid lines. Running in Matlab, the process was not very fast, with each full sample taking about 2 minutes to collect.

3.8 Further examples

One step on from the simple toy example above, the methods were tested on another simple problem. The data is the spectral density of a sound, consisting of a descending note followed by a broad frequency 'splat' sound. The sound comes from the matlab 'falling egg' demo, obtained through the command *xpsound*. It is good, because there are two constituent parts to the spectrum. The data comes from a number of frequency sensors, which give the 120 point dataset illustrated in the top left of figure 3.6. The full log spectrum is given in the top right of the same figure.

Once again the two regimes were generally separated. The same sampling procedure was used. The bottom graphs of figure 3.6 give the resulting predictors for the two regimes.

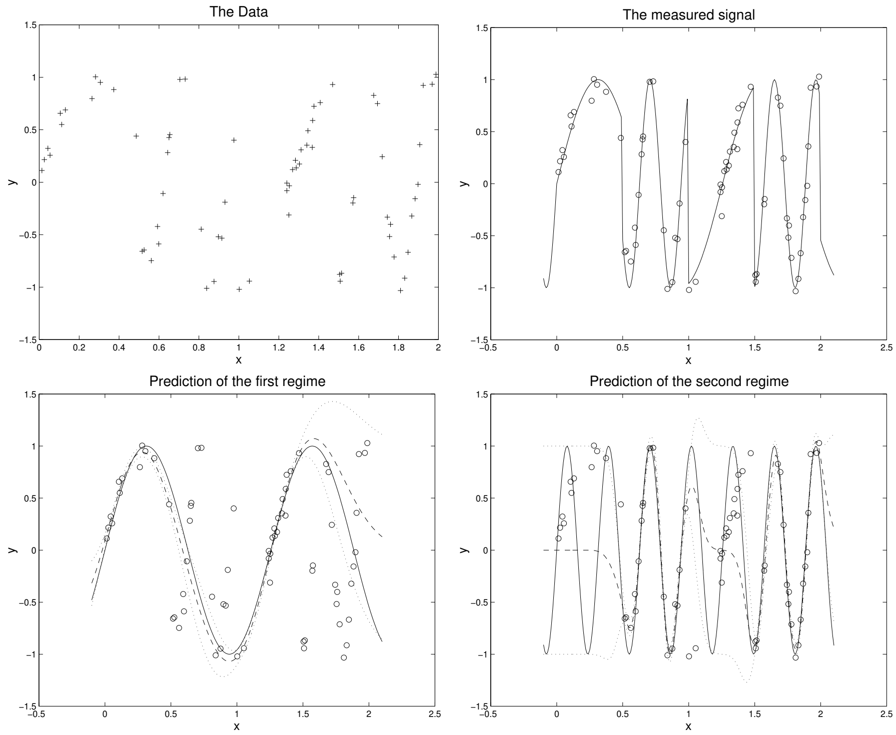


Figure 3.5: A test problem. Predictors (dashed) and error bars (dotted) for the two different regimes (solid)

The figure 3.7 gives the segmentation produced by one high posterior sample produced by the Markov chain.

3.8.1 Problems

These tests have also highlighted some problems with the approach. In cases where either one set of hyperparameters is highly inappropriate, or where both sets of hyperparameters are too similar to one another, the choice of latent variables all set to one regime may dominate the Markov chain completely. This means that little or no update information is available for the other regime. Hence the results are somewhat unfavourably dependent on initialisation. In some cases initially sampling from an annealed distribution [1] helped to get things started.

3.9 Extensions: higher dimensional systems, and larger numbers of regimes

These methods can be extended to higher dimensions. In these situations a Poisson process prior over the switching is no longer any use. However it is possible to use a Gaussian process classification prior [35] over the latent variables. It is perfectly possible to have a larger number of regimes by introducing a dimension for each possible regime, and using a logistic choice model to obtain the probability of a given regime.

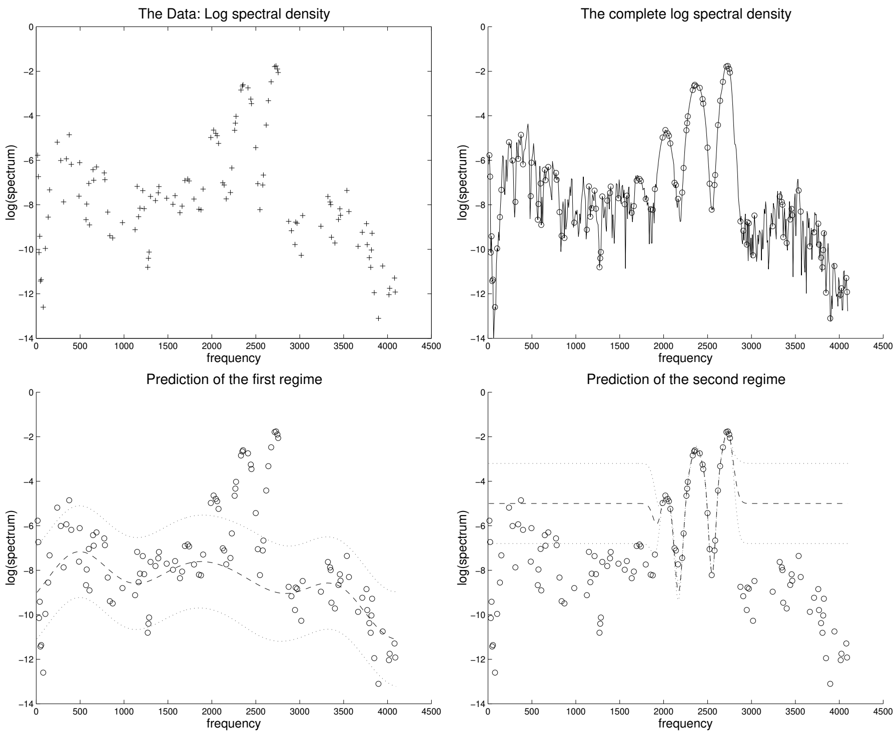


Figure 3.6: The falling egg test problem. Predictors (dashed) and error bars (dotted) for the two generating functions (solid)

3.9.1 Outline of results

In this chapter a method for modelling Gaussian processes with digital noise is examined. This is a simple extension of the use of weighted Gaussians, used in [30], to the non-zero mean case. Some simple examples are given about how to extend the use of Gaussian processes to cope with known non-stationarity.

Methods for using Gaussian process techniques are developed for signals made by switching between different signal sources. The methods involve the use of Gibbs sampling techniques. This would normally be very slow. However the use of the partitioned inverse equations enables each Gibbs step to be calculated in $o(n^2)$ rather than $o(n^3)$ flops. Techniques for updating blocks of latent variables speed things up, and improve the coverage of the sampling routines. Furthermore block sampling such as this also reduces the problem of highly correlated latent variable states causing the sampler to remain in local minima. These methods are tested on toy problems, with reasonable results.

The approaches developed here are different from other approaches to switching systems. Some other notable approaches include the work of Matthew Hodgson and Peter Green on ion channel models (using simulated data) [36], and Zoubin Ghahramami's work on switching state space models [33]. The principle difference is that Gaussian processes allow irregularly spaced samples. On the other hand, it is not a state spaced based approach, and hence does not perform state space prediction. Rather this is more a regression model, where there might be hidden, but dependent discrete states which affect the results.

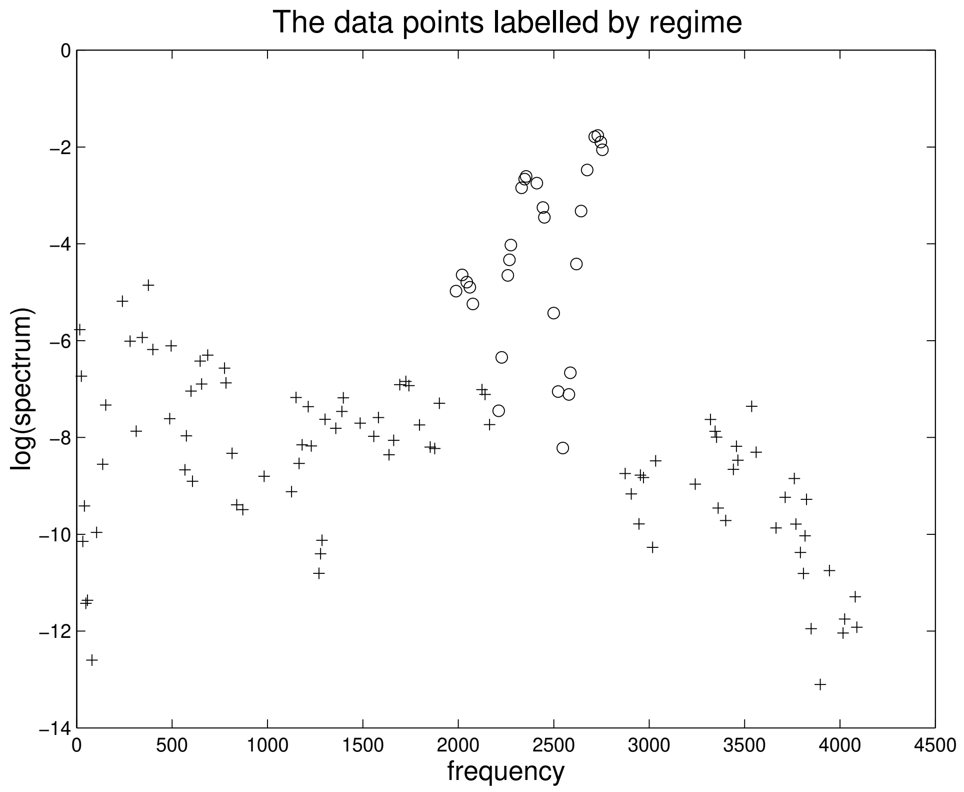


Figure 3.7: The falling egg test problem. A high posterior segmentation of the data points.

When developed further for multidimensional systems, a typical application area would be regression on measurements from mechanical or material systems whose behaviour might be dependent on some particular phase state or production source.

3.9.2 Conclusions

Gaussian processes can represent many types of functions because of the versatility of the covariance structure. Furthermore, by allowing transformations of the output space to be varied according to a Gibbs sampled latent variable this functionality can be increased.

Using hierarchical structure not only enriches Gaussian process modelling tools; more importantly it allows prior information to be naturally represented. In the model developed above, priors over the latent variable space enable switching systems to be accurately modelled. Furthermore, use of the partitioned inverse equations allows an efficient approach to Gibbs sampling the latent variables. These methods can be further improved by sampling blocks of latent variables at once, and by utilising prior information in the process of choosing sampling points.

Here we have demonstrated situations where different regimes are accurately recognised and modelled, while at the same time prior information about the signal form is properly represented. These methods could be extended to higher dimensions, for example to recognise different textures in two dimensional data.

Chapter 4

Toeplitz Covariance Matrices in Gaussian Processes

4.1 Introduction

The problems associated with inversion of large covariance matrices have been tackled by Gibbs and Mackay in [18]. There they used the methods of Skilling [37] to approximate the inversion in $o(n^2)$ flops. Here this problem is tackled in a different, but less general, way. However the resulting inversion is faster, and can be done exactly.

For many real life problems, there is some control over how data is collected, and this data often takes a regular form. For example data can be collected at regular time intervals or at points on a grid (e.g. video pictures). Often this structure can be used to ensure covariance matrices have a specific form. If this form happens to be Toeplitz, then the covariance inversion can be performed exactly in $o(n^2)$ flops. Here it is shown that for certain input space topologies, collection methods can be proposed which enable the covariance matrix to take a Toeplitz form. Furthermore the collection methods proposed are ones which are often naturally used. It is also shown how these methods can be used with other dataspace topologies not immediately amenable to this approach.

4.2 Gaussian processes for regression

We will use the exactly same notation as that of the previous chapters, with y given by a Gaussian process.

$$P(\mathbf{y}|H) = \frac{1}{Z}\exp\left(-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^T Q^{-1}(\mathbf{y}-\boldsymbol{\mu})\right)$$

with the covariance Q is partitioned into the form

$$\begin{pmatrix} Q^{DD} & Q^{DP} \\ Q^{PD} & Q^{PP} \end{pmatrix}$$

as before. The final predictive distribution is given by

$$P(\mathbf{y}^P|\mathbf{y}^D = \mathbf{y}^*, H) = \frac{1}{Z^P} \exp\left(-\frac{1}{2}(\mathbf{y}^P - \hat{\mathbf{y}})^T S^{-1} (\mathbf{y}^P - \hat{\mathbf{y}})\right)$$

where $S = (Q^{PP} - Q^{PD}(Q^{DD})^{-1}Q^{DP})$ and $\hat{\mathbf{y}} = Q^{PD}(Q^{DD})^{-1}\mathbf{y}^D + \boldsymbol{\mu}$

4.3 Covariance functions

Once again the starting point for the choice of covariance function is the squared exponential. Given hyperparameters θ_1, θ_2, r_l, C is then given by

$$C(\mathbf{x}_i, \mathbf{x}_j; H) = \theta_1 \exp\left(-\frac{1}{2} d(\mathbf{x}_i, \mathbf{x}_j)^2\right) + \theta_2 \tag{4.1}$$

where d is the metric of the underlying data space, in this case a Euclidean metric. This corresponds to saying that the closer points are in input space, the more correlated their function values will be, and that the function is smooth. In this chapter we will examine ways of modifying the covariance function to give us extra speed benefits.

4.4 Problems with Gaussian processes

One of the biggest difficulties with Gaussian Processes is inverting the covariance matrices. These matrices have n^2 elements where n is the sample size. Furthermore general matrix inversion is an $o(n^3)$ process. Hence for large sample sizes, the use of Gaussian processes becomes very slow.

Here fast inversion methods are introduced for a certain class of applications, namely those where there is some control over the data selection procedure, and where regular sampling structures are natural (e.g. time series, images, grid detectors). In these circumstances, the benefits of Toeplitz matrices can be used.

4.5 Toeplitz matrices

A Toeplitz matrix is a matrix A_{ij} of the form $A_{ij} = A_{i+1,j+1}$ for all i, j. In other words it is constant along all diagonals.

Toeplitz matrices can be inverted in $13n^2/4$ floating point operations using Trench's algorithm [38], [39, p199]. There are other approximate schemes which will solve Toeplitz systems in o$(n \ln n)$ flops: see [40, 41].

4.6 Data collection and regularity

In many situations the sample points $\mathbf{x}_k$ are chosen. Furthermore it is often easiest to take measurements in a regular manner.

The simplest such case is a time series. Time series are measurements of a given variable at regular time intervals. It is well known that the covariance matrix of a stationary time series is Toeplitz. Therefore the application of Gaussian Process methods to time series is a lot less computationally intensive than for other less regular signals.

However for the higher dimensional systems Toeplitz covariance matrices do not occur so simply. The natural two dimensional extension to the time series regularity is the grid. If data is received at points of the plane which form a grid, then we have a regular structure, but do not get a Toeplitz covariance matrix for any ordering of the data points.

4.7 Suitable topologies for Toeplitz covariance matrices

In this section, we examine what topologies and data structures might be amenable for generating Toeplitz covariance matrices, and give examples of practical applications.

It has already been noted that it is possible to generate Toeplitz covariance matrices on the real line using 4.1, given regularly spaced readings. Given a metric space and a form of covariance matrix, we call an ordered set of points which generate a Toeplitz covariance matrix a *Toeplitz ordering*. We call the unordered set of such points a *Toeplitz set*.

We call the covariance function *metric preserving* if each element is an invertible function of the distance between the relevant points. If the covariance function is of this form, then a Toeplitz ordering is independent of the actual choice of covariance function.

A set A induces an ϵ-cover of a space if the set of ϵ-balls around each point of A is a cover of the space.

We say that a family of Toeplitz sets provides a cover for a space if for all $\epsilon > 0$, there is a Toeplitz set in the family which induces an ϵ cover of the space. We use the shorthand 'Toeplitz covering' for such a family.

If we have a Toeplitz covering, then we can usefully sample the space in a way which will generate a Toeplitz covariance matrix.

4.8 Two dimensional systems: problems

When we go to two dimensions we have a problem:

Proposition 2 *In $(R^2, Euclidean)$ with a metric preserving covariance function we cannot generate a family of Toeplitz orderings which provide a cover for the space.*

Proof We generate a Toeplitz ordering. Choose x_0 and x_1 to be any two points in R^2. Then x_3 must be such that $d(x_3, x_2) = d(x_2, x_1)$. In other words x_3 must lie on a circle of radius $d(x_2, x_1)$ about x_2. Choose x_3 accordingly. Now x_4 must be on a circle of radius $d(x_3, x_2)$ about x_3 and also on a circle of $d(x_3, x_1)$ about x_2. There are only two such points. From x_5 on, each point is fully constrained by the previous choices. The result is either all the points lying on a circle in R^2 (corresponding to one choice of x_4) or $(x_1, x_3, \ldots)$ lying on one straight line and $(x_2, x_4, \ldots)$ lying on another (for the other choice of x_4). Hence all possible Toeplitz orderings lie on lines or circles of R^2, implying that there is some finite ϵ for which no Toeplitz ordering provides an ϵ cover (there is always some point in the space which is further than ϵ away from the line or circle on which the Toeplitz ordering lies). $\Box$

In fact covariance matrices on a grid based system take a Block-Toeplitz Toeplitz-Block (BTTB) structure. There is no equivalent of the Trench algorithm for BTTB systems, although some improvements can be made using conjugate gradient methods.

4.9 The surface of a cylinder

On the surface of a cylinder, $(R \times S, Euclidean)$, where S denotes the unit circle, things are slightly different. Toeplitz orderings will cover this space.

In order to show this, define a set of points x_n of a spiral on $R \times S$ parametrically by $(h, \theta) = (\alpha n, 2\pi\beta n)$ for $n = 1, 2, 3, \ldots$. Then it is straightforward to see that this is a Toeplitz ordering in this space. We have $d^2(x_n, x_m) = \alpha^2(n-m)^2 + 2 - 2\cos(2\pi\beta(n-m))$ which is a function of $n-m$, and therefore generates Toeplitz metric-preserving covariance matrices.

If we choose a set of orderings corresponding to different α and β, then we can see that this set covers the space, because the smaller α and β get the denser the points get.

Hence we have a useful space within which it is possible to generate Toeplitz covariance matrices.

4.9.1 An example

The above method can be used in any two dimensional predictive situation where one of the independent variables is periodic in nature. This might be months of the year, direction of view etc. Take for example the case of a company trying to estimate the supply-demand curve for a seasonal good. Then (neglecting complications due to Easter/weekends) we have the point in the year, and the price as the independent variables, and sales per hour, say, as the dependent variable. The point in the year is clearly a cyclic variable. Some smoothness between the demand at nearby times, and similar prices is to be expected. Hence a Euclidean metric on the cylinder is a reasonable choice of space.

Note that it might be the case that prior information suggests a smoother structure in some parts of the space than in others. This can implemented straightforwardly by transforming the cylinder space in a topologically conserving way, so that points are chosen in the transformed space (the space relating to the real situation) such that the ordering in the cylinder space is Toeplitz.

4.10 Using Toeplitz matrices in other topologies

It has already been shown that Toeplitz orderings do not usefully occur in R^2. However there are ways of getting around this. A section of the plane is similar to a section of the surface of a cylinder for a large cylinder. As all practical problems are finite in nature, this approximation can be useful.

We consider two situations when these methods can be useful. Firstly when there are known boundary conditions. Secondly in the case of what I will call 'foveal' prediction without boundary conditions, ie situations where predictions are only required in a central region (the outer area effectively giving the boundary conditions).

Unfortunately these methods cannot be used with the standard covariance functions used earlier. We will need to look at a new class of covariance functions which will be useful for this purpose.

4.10.1 Truncated covariance functions

The problem with the standard types of covariance functions used in Gaussian processes is that they tend to have an infinite range. This means that the value of the process at one point is affected (albeit only a little) by points which are a huge distance away. There is another class of covariance matrices which do not have this property. Here these have been called truncated covariances (or sometimes covariances with finite support), because the covariance between two points is zero if they are greater than a certain threshold distance apart.

The first thing to note that truncated covariance functions cannot simply be produced by cutting the tails of another type of covariance function: this would not generate positive semi-definite covariance matrices. Instead we have to go back to square one.

The sum of square distance covariance function is formed by convolving a Gaussian kernel with itself:

$$C(x,y) = \int \exp(-\frac{(x-r)^2}{2\sigma}) \exp(-\frac{(y-r)^2}{2\sigma})$$

We can generate covariances functions (which are positive semidefinite) by convolving any kernel with itself, and so we can choose some more suitable kernels. Mackay [42] mentions the use of a top hat kernel ($K(x,r) = 1$ if $|x-r| < 1$, zero elsewhere) to generate the covariance

$$C(x,y) = \begin{cases} 1 - |x-y| \text{for } |x-y| < 1 \\ 0 \text{elsewhere} \end{cases}$$

Clearly this covariance function is of the type we are interested in. It is zero outside the region $|x-y| < 1$. But it does not really satisfy all the requirements. It produces Gaussian processes which are far from smooth. This might be useful in some situations, but in general smoothness priors are more common. It would be good to find some covariance function with the same truncation properties, but which is also smooth and usable in higher dimensions.

There are a number of possibilities, but the one which will be introduced here is interesting because it is also generated from bell shaped kernels, and looks very similar to the sum of square distance covariance. However it is also a truncated covariance. Consider the kernel

$$K(x) = \begin{cases} 1 + \cos(x) & -\pi < x < \pi \\ 0 & \text{otherwise} \end{cases}$$

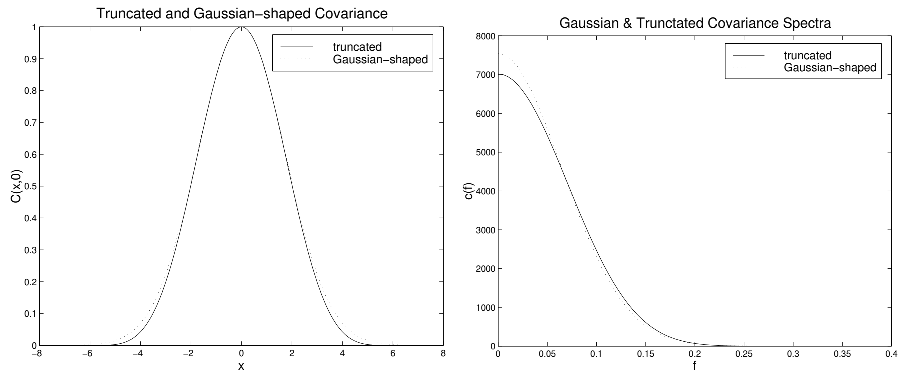


Figure 4.1: Comparing a Truncated covariance with a squared exponential

Then, when this is convolved with itself, we get the covariance function

$$C(x,y) = \begin{cases} [(2\pi - |d|)(1 + \cos|d|/2) + \frac{3}{2}\sin|d|]/(3\pi) & -2\pi < x < 2\pi \\ 0 & otherwise \end{cases}$$

where d is the Euclidean distance $d(x,y)$ between x and y This satisfies the required properties. It is zero outside $\pm 2\pi$.

Figure 4.10.1 illustrates this covariance function, and its spectrum, and compares it with the standard squared-exponential (Gaussian shaped) covariance function.

For higher dimensions we can define

$$C(\mathbf{x},\mathbf{y}) = C(x_1,y_1)C(x_2,y_2)\ldots C(x_D,y_D)$$

Of course not all situations are suited to covariance functions which are zero outside a certain region. However some of those with larger distance correlations can be expresses as

$$C_2(x,y) = C(x,y) + \alpha$$

where C is a truncated covariance. All of the benefits of truncated covariances are still available in these situations. This is because α generates a rank 1 contribution to the resulting covariance matrix. Then the Bartlett-Sherman-Morrison-Woodbury formula can be used to express the inverse of C_2 in terms of that of C.

4.10.2 Benefits of truncated covariances

In the next sections we will see how truncated covariances are useful in generating Toeplitz covariance matrices. However they have many other benefits. Firstly they speed up calculations. Because truncated covariances have zeros where other covariances might have very small values, many of the multiplications involved in matrix inversion and calculation of the predictive mean and variances become trivial multiplications by zero. Although not significantly affecting the nature of the results, the end result is a much faster implementation of Gaussian process methods.

Secondly the use of truncated covariances increases the accuracy of the calculations. Some of the problems encountered in the use of Gaussian process methods stem from the fact that the predictor mean and variances are calculated from large matrix-vector products. The result is a cumulative rounding error which can sometimes be as large as the predictor itself. Once again truncated covariances put zeros where other covariances would put non-zero values. As multiplications by zero are error free, these do not contribute to the cumulative error.

Having dealt with truncated covariances in a more general setting, it would be good to focus more on the topic at hand: how can truncated covariances help generate Toeplitz covariance matrices in situations where they are not normally found?

In most modelling situations, we are dealing with bounded spaces. Consider for example a rectangular region of (R^2, Euclidean). This region can be mapped onto the surface of a large cylinder ($R \times S$, Euclidean). We know that in the limit of the cylinder becoming infinitely wide, the metric properties of the rectangular region are preserved. For large but finite cylinders, the metric properties will be slightly but insignificantly distorted.

Now consider what we gain through the use of truncated covariances. The contribution to the value of the Gaussian process at a given point comes only from those points in the immediate neighbourhood. Hence any distortion of the metric at large distances is immaterial to the results. Therefore the region of interest can be mapped onto the whole cylinder, by 'wrapping it around'. Because the covariance is of truncated form, there is little distortion of the covariance between any two points in the space: the larger, more distorted, distances do not contribute to the covariance.

Let us consider three possible cases. Firstly there is the possibility that we are only interested in the foveal prediction, i.e. we want to infer things about the central region, not about the borders. In this situation we can map the region around the whole cylinder. Having done this, we can then choose a set of measurement points around the cylinder which produces a Toeplitz covariance matrix (see section 4.9).

Case two is the situation where we have global boundary conditions. (We have dealt with the case of periodic boundary conditions already.) In this situation we map the rectangular region onto the cylinder, leaving a strip the width of the 'truncation distance'. Then points on one side of the strip are unaffected by points on the other. The strip (and top and bottom) can be filled with the boundary values. This method is only an approximation, because the boundary values are only specified at the set of measurement points.

Case three is the situation where we want good prediction throughout the sample region. In this case we also leave a strip of width equal to the truncation distance. The sample points are taken from the whole cylinder in the usual Toeplitz manner. However in this situation, we do not have information for a small number of values along the strip. Hence the covariance matrix for all the points except those along the strip must be formed by using the partitioned inverse equations.

Suppose the strip width is k, and the sample region is n_ω. Then the fast Toeplitz inverse costs $o((n_w+k)^2 n_h^2)$, and the partitioned inverse step costs at most $o(n_h^3 k(n_w+k)^2)$. This speeds up the matrix inverse for k as large as $n_w/3$.

4.11 An example and more specific details

Let us take the first case: the situation in which we are only interested in foveal prediction. Here we map the whole region around the surface of a cylinder. Now we can choose the set of sampling points already mentioned in section 4.9: define a set of points x_n of a spiral on $R \times S$ parametrically by $(h, \theta) = (\alpha n, 2\pi\beta n)$ for $n = 1, 2, 3, \ldots$. We can map these points back onto the original rectangle to get the required sample points in the original space. We build the covariance matrix and invert it using Toeplitz techniques.

The set of sample points which this method produces is not exactly a grid, but without much loss of accuracy we can use a grid based system. Rather than transforming the rectangle onto the cylinder surface by mapping the vertical component to the vertical component of the cylinder, and the horizontal component to the horizontal component of the cylinder, we can skew the rectangle slightly, so that after mapping onto the cylinder the end of the first line meets the beginning of the second. Then the grid system in the original space becomes a Toeplitz set in the new space.

4.11.1 Example

The following example illustrates this point. The top-left picture of figure 4.11.1 illustrates a two dimensional surface. The top-right gives a grid based sample of that surface (a 12 by 12 grid, giving a total of 144 sample points). When Gaussian measurement noise of standard deviation 0.4 is added we get the middle left picture.

Toeplitz Gaussian process methods were used with this noisy data to try to reconstruct the original, using a prior based on the covariance given in (4.10.1). The final result is given in the middle-right graph. In this particular example, the foveal approach is used, which we would expect to give poor results around the edges of the surface. The error and error bars are given in the bottom two graphs. Note the poor prediction, and the unrepresentative error bars given for the far right of the surface (the result of using the foveal techniques).

4.12 Dealing with the hyperparameters

All of the work done up to this point has assumed that the hyperparameters are already known. If this is not the case then a number of problems will be encountered. Firstly, the truncation distance will vary as the hyperparameters change. Secondly, in the case of foveal prediction, or when using boundary approximations, the distribution of the data given the model cannot be accurately calculated because the boundary effects then matter. They will bias the hyperparameters. However it must be said that in practice, these boundary affects often appear negligible compared with the affect of the rest of the space.

Here we will look at the theory in the third case, where Bayesian inference is done properly. The methods can be applied to the other situations with the understanding that

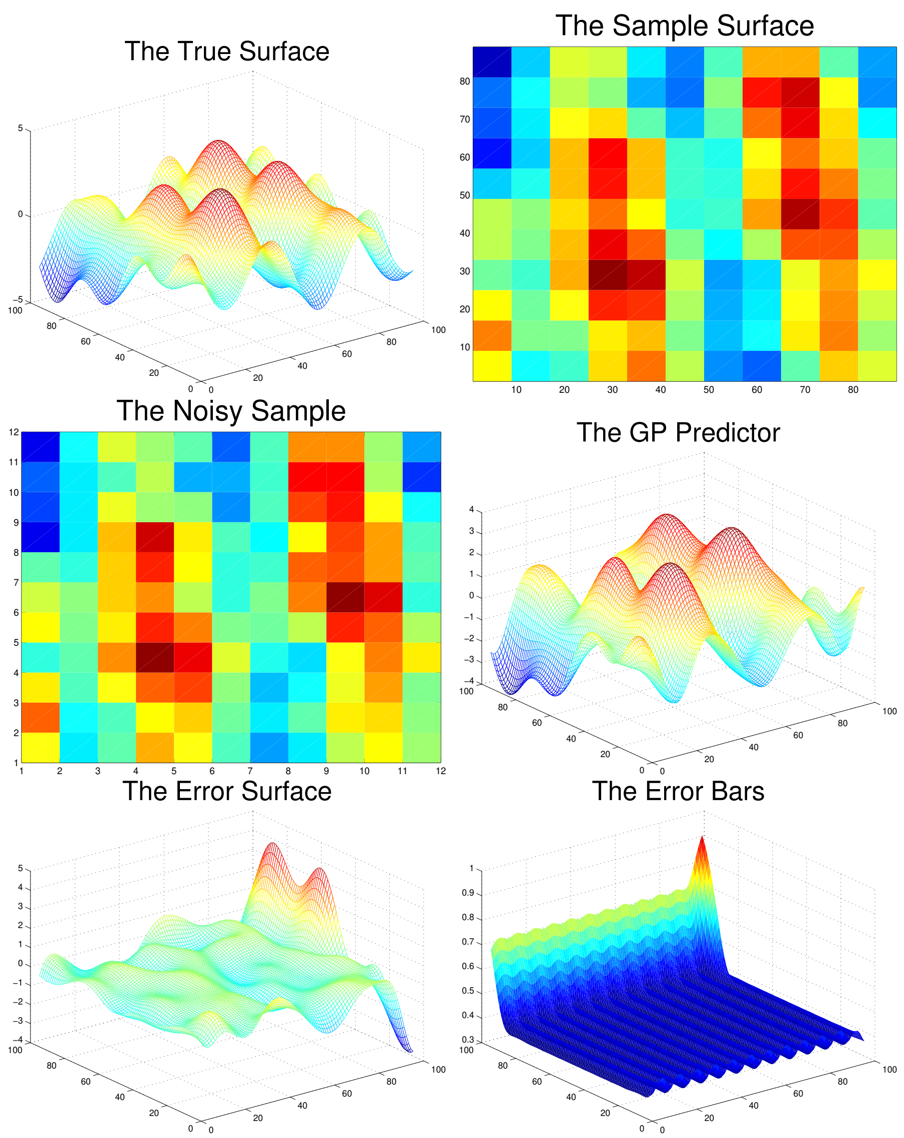


Figure 4.2: An example of using Toeplitz Gaussian process methods to model two dimensional systems

the posterior distribution will not be exact.

The first way of dealing with hyperparameters involves using a prior on the width hyperparameters which is zero when larger than a certain width l_{1max}. Then the width k of the 'missing strip' can be set to the truncation width of the covariance function when $l_1 = l_{1max}$, and the usual process of maximising or integrating out the hyperparameters can be followed.

This is a rather inefficient way of doing things though. The high posterior values of the hyperparameters are unlikely to be around this maximum position. Instead what we can do is set k to be equal to the truncation width of covariance given the current hyperparameter. In this situation the mapping of the rectangular space to the cylinder will change as the hyperparameter width changes. Because the inverse covariance must be

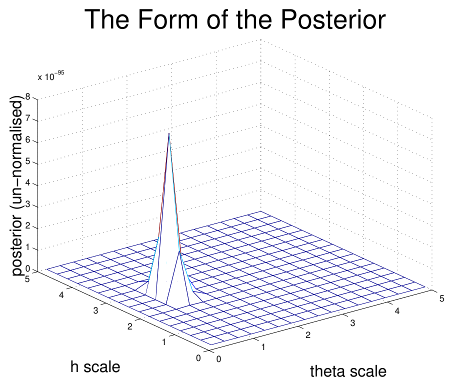


Figure 4.3: The posterior distribution of the the two x-length scales.

recalculated each time the hyperparameters change, this will not cause any extra overhead. When the hyperparameters correspond to larger scaling factors, then the region will be mapped to a larger cylinder. This is not a problem; it just means that the mapping onto the cylinder needs to be incorporated into the model. In fact the effect will be beneficial in ensuring that there is a similarly low level of distortion from the flat space covariance for all hyperparameter values.

4.12.1 Example

To show that these methods can also be used in situations when the model is used for foveal prediction, we plot the posterior distribution of the hyperparameter space for the example above in figure 4.12.1. Here It can be seen that the peak in the distribution is found for the values $1, 2.5$ for the scaling hyperparemers in the h (height) direction and the θ (circumfrential) direction respectively. These are in fact the values used to plot figure 4.11.1. There is no noticeable contribution from the edge effects of the wrapping.

4.13 Higher dimensional spaces

All these methods can also be used in higher dimensional spaces in the same ways. Toeplitz orderings can be found on spaces $R \times S^s$ for all s. For example in four dimensions we choose the points

$$(h, \theta, \phi, \rho) = (\alpha n, 2\pi\beta n, 2\pi\gamma n, 2\pi\delta n)$$

for some $\alpha, \beta, \gamma, \delta$.

In three dimensions this can be visualised as a spiral winding round a torus, while the third component moves steadily along a straight line.

4.14 Quantifying the speed and accuracy

The speed and accuracy of this approach can be understood by comparing these techniques with standard methods using the squared exponential function. Even at the small sample sizes used here the speed of Toeplitz inversion (using the Trench algorithm) is about a factor of 30 times faster (6074000 flops against 187000 flops).

The accuracy of the method was tested by comparing the results of the Toeplitz approach with truncated covariances with results obtained through the usual Gaussian process approach with squared exponential covariances. To do this, the root average squared differences were compared. The average was taken over 30 different trials and over 490 test points arranged in a grid, and not in a small border around the edge of the region. The 30 different trials used the same function, but different noise samples.

It was found that the differences between the two methods was significantly lower than the differences between the squared exponential and the true function. For high noise levels (signal to noise ratio of 2) we get the root average squared difference between the two methods to be 0.12, and the root average squared errors for the usual and Toeplitz methods to be 0.33 and 0.36 respectively. At lower noise levels (signal to noise of 10) we get differences (listed in the same order) of 0.035, 0.076, and 0.085. Hence the Toeplitz approach could be considered to be a reasonable approach, with errors for this problem slightly, but not significantly, worse than the usual squared exponential method. Here the hyperparemeters were optimised independently for each method.

4.15 Conclusions

The use of truncated covariances with Gaussian processes can be used to represent smoothness priors, where infinite distance correlations are considered unlikely. Furthermore they are also the starting point for a number of efficient modelling methods. Not only do they automatically increase speed and reduce inaccuracies by increasing the number of zero multiplications, but they also enable the use of Toeplitz matrix inversion techniques with grid-based systems. The key to this approach is to provide low distortion mappings from spaces in which Toeplitz coverings cannot be found to ones where they can.

For grid based systems, Fourier domain approaches will be fast, and would generally be more appropriate than the standard Gaussian process approach. However it seems that the Toeplitz methods used here are of comparable speed to Fourier methods. The combination of Toeplitz inverses as fast as $o(n \ln n)$ and the extra speed gained from the more sparse nature of truncated covariance matrices contribute to this. The Toeplitz method is a different approach, and has different advantages. Once the inverse is calculated, it can be used to predict at any points which might be chosen. It allows missing data to be dealt with properly: the marginalisation integral can be done exactly and efficiently. On the whole this method should be seen as an alternative perspective on the solution to these types of problems.

The use of truncated covariances has further potential. The speed increases they might give to the general use of Gaussian processes has not been investigated. This might be a fruitful line for future investigation.

Chapter 5

A New Incremental and Local Learning Rule for Hopfield Networks

5.1 Introduction

[1]The remainder of this thesis will be looking at a different sort of area from those which have been discussed up to now. This chapter will introduce the Hopfield network, and the problems of association and autoassociation. Association involves the process of building structures which implicitly encode some link between patterns which are presented. Autoassociation can be thought of in a complementary way; autoassociators are content addressable memories. Some original stored pattern can be retrieved from such a memory, not by knowing the address at which it was stored, but rather by knowing some limited (perhaps slightly erroneous) part of the content. In this context we would expect the original stored pattern to be retrieved.

The Hopfield network is what we call an attractor neural network. It consists of a number of nodes (or neurons) whose binary state dynamically depends on the weighted input it receives from the other neurons. When analysed mathematically, the dynamics of the state of all the network nodes converges to a fixed point attractor (hence the term attractor network). Because there are many such attractors, the Hopfield model can be used as an autoassociator: if the fixed points correspond to the stored patterns, then some new pattern which is presented will be in the attraction basin of some fixed point. If the presented pattern is close enough to the stored pattern corresponding to the fixed point, then after convergence the network state will correspond to the original pattern.

Attractor neural networks such as the Hopfield network can be trained by a number of algorithms. These algorithms adapt the weights from one neuron to another in a way to make the presented patterns fixed points of the network. These algorithms have a number of characteristics. The characteristics we will be focusing on here are locality,

[1]A part of this chapter has been published in [43]

incrementality and capacity. A local learning rule is one where the update depends only on information available from the neurons either side of the connection. An incremental rule is one which allows a new pattern to be added without having to know all the previous patterns. These properties are desirable because they make the Hopfield network learning rule updatable and parallel. The capacity of a learning rule is simply some measure of how many patterns can be stored (and retrieved) in a network as the number of neurons increases. Higher capacities allow greater storage levels, and are therefore advantageous.

The simplest, and most common, of the Hopfield learning methods is the Hebb rule which is local and incremental. This means that the patterns to be stored can be presented one at a time (in one pass), and each weight can update itself, without reference to the updates which the other weights are making. However it has been shown that the Hebbian network has a capacity (by one particular measure) of $n/(2\ln n)$ where n is the total number of neurons [44]. This capacity can be increased to n by use of the pseudo-inverse rule [45], but this rule is neither local nor incremental. At best the pseudo-inverse weight matrix can be generated by local (not strictly local) and non-incremental limit processes [46]. The question addressed in this chapter is, 'Can the capacity of the Hebbian rule be increased without losing locality or incrementality?'

Here a new algorithm is proposed. This algorithm is local. It is immediate: the learning process is one shot rather than a limit process. It is also incremental. In addition it has a capacity significantly higher than that of the Hebbian method: $n/\sqrt{2\ln n}$

In this thesis the new learning method is introduced. The relationship between the new rule, the Hebbian rule and the pseudo-inverse is given; the capacity of the learning algorithm is calculated rigorously, using a simple probabilistic approach. Complicated large deviation calculations are avoided by a judicious use of an exponential upper bound. Simulations are provided to show that this calculation does indeed provide a good measure of the capacity for finite network sizes.

5.1.1 Attractor neural networks

Attractor neural networks such as Hopfield networks are used as content addressable memories. These networks retrieve stored patterns from an example presented to the network. This example is an incomplete or noisy version of one of the stored patterns.

The Hopfield network considered in this thesis consists of binary (1,-1) neurons connected together by weighted links. Each neuron represents one bit of a pattern. The weights of the links are determined by a learning rule, and only learning rules producing symmetric weight matrices are considered. The output of the neuron is given by the sign of the weighted sum of the inputs from the other neurons, and these neurons are updated asynchronously. Because the weight matrix is symmetric, the state of the network tends to a fixed point in state space [3]. The aim of the learning rule is to ensure that the stored patterns are fixed points of the network dynamics.

To use the network, the weights are set up by the learning rule. An input pattern is presented to the network by initializing (or 'clamping') the neurons into the state determined by the pattern. The 'clamping' is then released: in other words, the neurons are now free to change state. The network is repeatedly updated using the Hopfield update rule, choosing one neuron at a time in a random manner. The network dynamics take the system state to one of the fixed points. On the whole this fixed point will be the stored pattern closest (in Hamming distance say) to the input pattern.

5.1.2 The learning rule

The learning rule is used to define the weights of the network. Incremental and local properties of the learning rule are very important in attractor neural networks.

If the learning process can modify an old network configuration to store a new pattern, without needing to refer to any of the previously learnt patterns, then an algorithm is called incremental. Incrementality makes the Hopfield network adaptive. Locality, on the other hand deals with the whether or not the learning rule can be seen as a set of independent learning processes for each connection. Locality provides a natural parallelism to the architecture, because each connection can be seen as a separate processor in its own right. An incremental system provides a straightforward adaptability: New attractors can be introduced without needing to know the information from which old ones were generated. The Hebbian learning rule is both local and incremental, and has been studied extensively. Many researchers have focussed their attention on assessing the capacity of the Hebbian rule, and so capacity is now considered.

Capacity is a measure of the maximal storage of an attractor neural network. Capacity is important for a network, because a higher capacity makes more efficient use of the processors. Two main types of capacity are commonly considered. These are often called absolute capacity and relative (or spin glass) capacity. This distinction stems more from the theoretical methods used to calculate capacity than anything else. Absolute capacity results tend to be obtained through what are know as 'signal to noise' approaches [44]. Other (relative) capacities tend to be obtained through replica theoretic spin glass methods [47]. The main difference is that absolute capacity deals with perfect retrieval, whereas spin glass approaches presume some small but significant level of error in retrieval of each memory.

Here only absolute capacity is considered. The absolute capacity of the Hebbian rule is $n/(2\ln n)$ where n is the total number of neurons. This chapter introduces a new learning rule which keeps both the local and incremental properties of the Hebbian rule, but increases the absolute capacity to $n/\sqrt{2\ln n}$.

5.1.3 Local and strictly local

A distinction is made between strictly local and local learning algorithms. A *strictly local* algorithm, such as the Hebb rule, adapts its synaptic weight using only information about the activity of the two neighbouring neurons. That is, the update of the weight between neurons i and j is given by

$$\Delta w_{ij} = f(x_i, x_j)$$

where f is some function, and x_i is the state (either $+1$ or -1) of the ith neuron.

A *local* procedure, on the other hand may also use information from the inputs to those two neurons. Usually this is in the form of the sum of the weighted inputs (what we call the local field). In other words the weight update will take the form

$$\Delta w_{ij} = f(x_i, x_j, \{(w_{ik}x_k), (w_{jk}x_k) | k = 1, \ldots, n\})$$

and in fact this will usually be

$$\Delta w_{ij} = f(x_i, x_j, \sum_k w_{ik}x_k, \sum_k w_{jk}x_k)$$

where w_{ik} is the weight between neurons i and k before any update is made, and Δw_{ij} is the change in w_{ij} due to the update.

The important thing is that the update of a synaptic weight only requires information available to the neurons either side of the synapse. A non-local procedure requires further information on the activity of other neurons. The term *loosely local* is used to imply local but not strictly local.

This distinction is made for two reasons. Firstly, although in a parallel processing neural system both loosely local and strictly local systems allow immediate learning, in a sequential computing system the loosely local learning rule requires slightly more processing than the strictly local. This is because the local fields need to be calculated.

Secondly it could be argued that strictly local learning rules are biologically plausible whereas loosely local ones are not. The synapse never receives information about the input field to the presynaptic neuron. Although this might be fair, these physiological details are not yet entirely decided. Given that attractor neural networks are such an abstraction from real neural systems, this distinction should be seen as less important than the previous one.

5.1.4 History

Hopfield networks were first introduced over a decade ago. Since then much work has gone into assessing the properties of such networks, and also improving their functionality. Over that time, a number of problems with the use of such networks as content addressable memories have been documented. In addition certain properties of the networks have been recognised as important. In this chapter a new learning rule is proposed to maintain or improve the most desired properties of the Hopfield network, while beginning to tackle some of the associated problems. Most importantly, the learning algorithm keeps the local and incremental properties associated with the rule while increasing the capacity of the network.

As early as 1943, McCulloch and Pitts [48] came up with a simplified model of a neuron. They showed that such neurons could be used to represent any binary logical operation.

Hebb, in 1949, suggested that the synapses between neurons were modified in response to associated activity of the pre- and postsynaptic neurons. He believed that such modifications could begin to account for learning. Various formalisations of this Hebb rule have been used extensively in neural networks [3]. More recently experimental evidence has shown that Hebb was, at least to some degree, correct in his postulation [49].

One outcome of the application of the Hebb rule to neural network was the development of models of associative memory. Cragg and Temperley [50] noted the similarity between models of Ising spin systems and neural networks, and considered whether some of the physical concepts so important to such systems (temperature, energy) could have any use in models of biological networks. It was Little [51] who discovered that a stochastic version of the abstract neuron could be used to introduce noise into a neural network in such a way that it corresponded to the temperature in a spin system.

It was not until 1982, when Hopfield [3] showed how these neural networks could be used practically, that these types of neural networks found popular appeal. He showed how this form of neural network can act as an associative memory by introducing the patterns as attractors of the dynamical system which the network defined. Such memories were content addressable, unlike computer memories which required an memory address for retrieval.

Hopfield made early studies on the capacity of such networks through the use of simulations, but it was 1987 when Amit, Gutfreud and Sompolinsky [47] solved the Hopfield model using the replica method to get an exact capacity value. They calculated the (relative) capacity of the network to be about $0.14n$ where n is the number of neurons. The network can retrieve patterns well if fewer than $0.14n$ patterns are stored, but the retrieval quality deteriorates rapidly for storage levels above $0.14n$.

However, the Amit et al. paper had made two assumptions. The first of these involved the use of the mean field theory and replica method. This might be physically sensible, but is not mathematically justifiable. The second involved assuming that good but imperfect retrieval doesn't matter. McEliece, Posner, Rodemich and Venkatesh [44] did not make these assumptions, and they rigorously calculated the capacity of the network to be $n/2\ln n$. If the storage levels are below this amount, perfect recall can be guaranteed in the limit of an infinite size network. In fact this capacity value works very well for quite small networks.

These capacity calculations were made for the Hopfield network with the Hebb rule. They were made with the assumption that the patterns to be stored were independent. If patterns are correlated, then the capacity results drop significantly. In trying to tackle this problem, Personnaz et al [52, 53] introduced the pseudo-inverse learning rule, which Kanter and Sompolinsky [45] refined. This rule effectively makes the stored patterns eigenvectors of the weight matrix, with eigenvalues of 1. It has a capacity of n however correlated the patterns are.

The pseudo inverse rule is non-local, non-incremental and involves calculating an inverse, and so it lacks much of the simplicity of the Hebb rule. Amit et al [47] looked at the special case of biased patterns, and introduced threshold terms to remedy the reduction in storage capacity. This was modified by Fontanari and Theumann [54] so that the knowledge of the correlation did not need to be known before learning occurred.

5.2 The Hopfield model

5.2.1 The Hopfield model

The Hopfield network has a fully connected structure, and a symmetric connectivity. There are no connections from a neuron to itself. The neurons can be in two states, +1 or -1. The state of neuron i is denoted x_i. The update rule of a neuron is given by

$$x_i(t+1) = \operatorname{sgn}\left(\sum_{j=1, j\neq i}^{n} w_{ij} x_j(t)\right) \tag{5.1}$$

where W is the weight matrix with elements w_{ij}, and t is a discrete time parameter. The network state at time t is given by the vector $\mathbf{x}(t) = x_i(t)$ and $\mathbf{x}(0)$ is equal to the input pattern vector presented to the network. Note we could just set $w_{ii} = 0$. However sometimes it is useful to allow w_{ii} to be nonzero when recursive learning rules are used.

The neurons in the network are updated in discrete time in a random asynchronous manner. This means that one neuron is chosen at random, its new state is calculated depending on its inputs from the other states. All other neurons keep their old state. The process is repeated until it settles to an equilibrium point.

The network state must end up at a fixed point, because an energy function can be defined for the network [3]. Any change in the network state must decrease the energy of the network, and because the energy is bounded below and the number of states in the network is finite, the network state must stop changing at some point in time.

5.2.2 Learning rules for Hopfield networks

The two main learning algorithms are the Hebb rule and the pseudo-inverse rules. They are given below.

5.2.2.1 The Hebb rule

The Hebb rule is an abstraction of Hebb's original conjecture about the behaviour of biological synapses during learning. Hebb suggested that synapses are strengthened if there is correlated activity between the pre- and postsynaptic neurons. For the Hebb rule here, correlated firing patterns increase the weight, uncorrelated patterns decrease it.

Definition 2 (Hebbian learning rule) *The Hebbian learning rule is given by*

$$w_{ij}^0 = 0 \ \forall i, j$$

$$w_{ij}^{\mu} = w_{ij}^{\mu-1} + \frac{1}{n}\xi_i^{\mu}\xi_j^{\mu}$$

where W^{μ} is the weight matrix after the μth pattern $\boldsymbol{\xi}^{\mu}$ has been presented. The size of the network (that is the number of nodes) is denoted by n. The weight matrix W^{μ} has elements w_{ij}^{μ} with $1 \leq i, j \leq n$, and the pattern $\boldsymbol{\xi}^{\mu}$ has elements ξ_i^{μ} with $1 \leq i \leq n$.

The Hebb rule has been written out in this form to make its incremental and local properties evident.

5.2.2.2 The pseudo-inverse

The pseudo-inverse method was developed to cope with the problems introduced when correlated patterns are stored in a Hopfield network. However it is also useful in situations where patterns are uncorrelated. The rule is neither local nor incremental, and involves significantly more computation than the Hebb rule, because a matrix inverse needs to be calculated.

Definition 3 (Pseudo-inverse learning rule) *The pseudo inverse weight matrix for the batch storage of m patterns is given by*

$$w_{ij}^{m} = \frac{1}{n}\sum_{\mu=1}^{m}\sum_{\nu=1}^{m}\xi_i^{\mu}(C^{-1})^{\mu\nu}\xi_j^{\nu} \tag{5.2}$$

where $C^{\mu\nu} = \frac{1}{n}\sum_{k=1}^{n}\xi_k^{\mu}\xi_k^{\nu}$, n is the size of the network and m is the number of patterns to be stored. As before W^m is the weight matrix after all m patterns $\boldsymbol{\xi}^{\mu}$ $(1 \le \mu \le m)$ have been presented. In this form the pseudo-inverse rule is neither incremental nor local. Note that $(C^{-1})^{\mu\nu}$ is the $(\mu\nu)$th element of the inverse of the C matrix.

5.2.3 The new learning rule

The new learning rule, introduced in this chapter, is defined as follows.

Definition 4 (The new learning rule) *The weight matrix of an attractor neural network is said to follow the new learning rule if it obeys*

$$w_{ij}^{0} = 0 \ \forall i,j \in \{1,2,...,n\}$$

$$w_{ij}^{\mu} = w_{ij}^{\mu-1} + \frac{1}{n-1}(\xi_i^{\mu}\xi_j^{\mu} - \delta_{ij}) - \frac{1}{n-1}\xi_i^{\mu}h_{ji}^{\mu} - \frac{1}{n-1}h_{ij}^{\mu}\xi_j^{\mu} \tag{5.3}$$

where

$$h_{ij}^{\mu} = \sum_{k=1,k\neq j}^{n} w_{ik}^{\mu-1}\xi_k^{\mu}$$

is a form of local field at neuron i, and ξ^{μ} is the new pattern to be learnt. It is local because information from other neurons is modified by the relevant synaptic strength before being used in the learning rule.

The above rule can be expressed in matrix form:

$$W^{0} = 0 \ \forall i,j \in \{1,2,...,n\}$$

$$A^{\mu} = (\xi^{\mu}(\xi^{\mu})^{T} - I)/(n-1)$$

$$W^{\mu} = W^{\mu-1} + A^{\mu} - A^{\mu}W^{\mu-1} - W^{\mu-1}A^{\mu} \tag{5.4}$$

Note that the rule is taken in this form for ease of analysis. Particularly the division by $(n-1)$ is chosen to simplify future calculations. Practically the more common factor $(1/n)$ seems to work equally as well. Many other forms of the same rule could be used. They all have similar properties, although we will see in chapter 8, that certain forms are useful in other situations.

For example the simpler

$$w_{ij}^{0} = 0 \ \forall i, j \in \{1, 2, ..., n\}$$

$$w_{ij}^{\mu} = w_{ij}^{\mu-1} + \frac{1}{n-1}\xi_i^{\mu}\xi_j^{\mu} - \frac{1}{n-1}\xi_i^{\mu}h_j^{\mu} - \frac{1}{n-1}h_i^{\mu}\xi_j^{\mu} \tag{5.5}$$

where

$$h_i^{\mu} = \sum_{k=1}^{n} w_{ik}^{\mu-1}\xi_k^{\mu}$$

also works very well, as simulations will show.

The form

$$h_{ij}^{\mu} = \sum_{k=1, k\neq i,j}^{n} w_{ik}^{\mu-1}\xi_k^{\mu}$$

appears from simulations to have marginally higher capacity. In fact in later chapters, these simpler forms will be used.

As mentioned earlier, it is perfectly possible to use the form

$$w_{ij}^{0} = 0 \ \forall i, j \in \{1, 2, ..., n\}$$

$$w_{ij}^{\mu} = w_{ij}^{\mu-1} + \frac{1}{n}\xi_i^{\mu}\xi_j^{\mu} - \frac{1}{n}\xi_i^{\mu}h_j^{\mu} - \frac{1}{n}h_i^{\mu}\xi_j^{\mu} \tag{5.6}$$

where

$$h_i^{\mu} = \sum_{k=1}^{n} w_{ik}^{\mu-1}\xi_k^{\mu}$$

rather than that of (5.5) without any detrimental change.

It can be seen that this new rule has a Hebbian component $(\xi_i^{\mu}\xi_j^{\mu} - \delta_{ij})$ and a component which uses the local fields to the neurons i and j. Note that the subtraction of the δ_{ij} term just corresponds to the subtraction of the diagonal. In the Hebb rule this would make no difference, as the diagonal plays no part in the Hopfield update rule (5.1).

5.2.3.1 Some intuitive comparisons

If the simpler form (5.5) of this rule is expanded out for a finite number of memories m, then looking only to order $1/(n-1)^2$ gives

$$w_{ij}^{m} = \frac{1}{n-1}\sum_{\mu=1}^{m}\xi_i^{\mu}\xi_j^{\mu} - \frac{1}{(n-1)^2}\sum_{\mu=1}^{m}\sum_{\nu=1,\nu\neq\mu}^{m}\sum_{k=1}^{n}\xi_i^{\mu}\xi_k^{\mu}\xi_k^{\nu}\xi_j^{\nu} \tag{5.7}$$

This can be seen by induction: it is true for $m = 1$, and assuming it to be true for $m-1$, we have (using 5.5) that w_{ij}^m is given by w_{ij}^{m-1} plus

$$\frac{1}{(n-1)}\xi_i^m\xi_j^m + \frac{1}{(n-1)^2}\xi_i^m\left(\sum_{\mu=1}^{m-1}\sum_{k=1}^{n}\xi_k^m\xi_k^\mu\xi_j^\mu\right) + \frac{1}{(n-1)^2}\left(\sum_{\mu=1}^{m-1}\sum_{k=1}^{n}\xi_i^\mu\xi_k^\mu\xi_k^m\right)\xi_j^m + o(n^{-3})$$

which simplifies to the form of the RHS of (5.7).

This can be written as

$$w_{ij}^m = \frac{2}{(n-1)}\sum_{\mu=1}^{m}\xi_i^\mu\xi_j^\mu - \frac{1}{(n-1)}\sum_{\mu=1}^{m}\sum_{\nu=1}^{m}\xi_i^\mu C^{\mu\nu}\xi_j^\nu \tag{5.8}$$

where $C^{\mu\nu} = \frac{1}{n-1}\sum_{k=1}^{n}\xi_k^\mu\xi_k^\nu$. Note that the $(2/(n-1))$, comes from considering the $\mu = \nu$ terms, and using $\sum_k \xi_k^\mu\xi_k^\mu = n$.

Likewise the other form of the rule (5.6) can be written as

$$w_{ij}^m = \frac{2}{n}\sum_{\mu=1}^{m}\xi_i^\mu\xi_j^\mu - \frac{1}{n}\sum_{\mu=1}^{m}\sum_{\nu=1}^{m}\xi_i^\mu C^{\mu\nu}\xi_j^\nu \tag{5.9}$$

where $C = \frac{1}{n}\sum_{k=1}^{n}\xi_k^\mu\xi_k^\nu$.

How does this help to increase the capacity over that of the Hebb rule? A memory ξ^μ is a fixed point if and only if

$$\sum_{j=1,j\neq i}^{n}\xi_i^\mu w_{ij}^m\xi_j^\mu > 0 \;\forall i \tag{5.10}$$

Looking again at the form (5.5), and at W^m of order $1/(n-1)^2$, by substituting (5.7) into (5.10) we get the following requirement for memory ξ^μ to be a fixed point

$$\sum_{j=1,j\neq i}^{n}\frac{1}{n-1}\sum_{\nu=1}^{m}\xi_i^\mu\xi_i^\nu\xi_j^\nu\xi_j^\mu - \frac{1}{(n-1)^2}\sum_{\rho=1}^{m}\sum_{\nu=1,\nu\neq\rho}^{m}\sum_{k=1}^{n}\xi_i^\mu\xi_i^\rho\xi_k^\rho\xi_k^\nu\xi_j^\nu\xi_j^\mu > 0$$

(for all i). Summing over j when $\nu = \mu$ we get

$$1 - \frac{1}{(n-1)^2}\sum_{\nu=1\nu\neq\mu}^{m}\sum_{\rho=1,\rho\neq\nu}^{m}\sum_{k=1}^{n}\sum_{j=1,j\neq i}^{n}\xi_i^\nu\xi_k^\nu\xi_k^\rho\xi_j^\rho\xi_j^\mu > 0 \tag{5.11}$$

for all i.

This can be seen in signal to noise terms. The signal term produces the unit contribution to the left hand side of (5.11) and the rest of the left hand side of the expression is the noise term. The introduction of the local fields into this learning rule removes the order $1/(n-1)$ noise terms: they cancel with the $\rho = \mu$ part of the order $1/(n-1)^2$ noise terms. In section 5.4 more detailed signal to noise analysis will provide an evaluation of the absolute capacity of the new learning rule.

5.2.4 Hopfield learning rules and sample covariance matrices

The learning rules above are related to sample covariances matrices. This relationship will allow us to use an interesting result [55] to see how all these rules are related to one another.

Definition 5 *Given some zero mean multivariate distribution of dimension D, and some multivariate sample from that distribution $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^n$, the sample covariance is given by*

$$C_{ij} = \frac{1}{n} \sum_{\mu=1}^{n} (x_i)^\mu (x_j)^\mu$$

where i, j index each dimension.

We can see that this form is exactly that of the Hebb rule, and in another way, is the form of the inverted matrix in the pseudo-inverse rule. The main difference is that sample covariances are usually used when the sample size is much larger than the dimension size. Here we expect the sample size to be of the order of the number of dimensions.

This matrix will, in fact, be used in a way which is quite different from standard uses of sample covariances. However interesting results for such matrices will be directly applicable.

Siverstein [55] reviewed an interesting result, giving the asymptotic distribution of the eigenvalues of a sample covariance matrix.

Theorem 2 *Suppose we have X_n a matrix whose rows are the samples from a multivariate distribution. Let X_n have dimension $(n \times s)$: s samples from an n dimensional distribution. Let C_n be given by $C_n = (1/s) X_n X_n^T$. In other words C_n is a sample covariance matrix. Now let X_n have zero mean, unit variance, i.i.d. elements. Let $s = s(n)$ with $n/s \to y \geq 0$. Let $F_n(\lambda)$ be the cumulative distribution function of the eigenvalues of C_n, and $b_\pm = (1 \pm \sqrt{y})^2$.*

If $\exists \delta > 0$ s.t. $E|x_{ii}|^{2+\delta} < \infty$ then for every $\lambda \in R$, $F_n(\lambda) \to F_y(\lambda)$ a.s. as $n \to \infty$ where for $0 < y \leq 1$

$$F'_y(\lambda) = f_y(\lambda) = \begin{cases} \frac{1}{2\pi\lambda y} \sqrt{(\lambda - b_-)(b_+ - \lambda)} & \text{if } b_- < \lambda < b_+ \\ 0 & \text{otherwise} \end{cases}$$

and for $1 < y \leq \infty$

$$F_y(\lambda) = \left(1 - \frac{1}{y}\right) I^{\lambda}_{[0,\infty]} + \frac{1}{y} \int_{b_-}^{\lambda} f(t)dt$$

where the indicator function I^a_S is 1 if $a \in S$ and zero otherwise.

For proofs see [56, 57, 58]. Simulations show that this result holds well for modestly (even $n = 10$) sized matrices [59]. This result is interesting because it does not depend on the distribution of the elements (except in that they should be zero mean, unit variance)

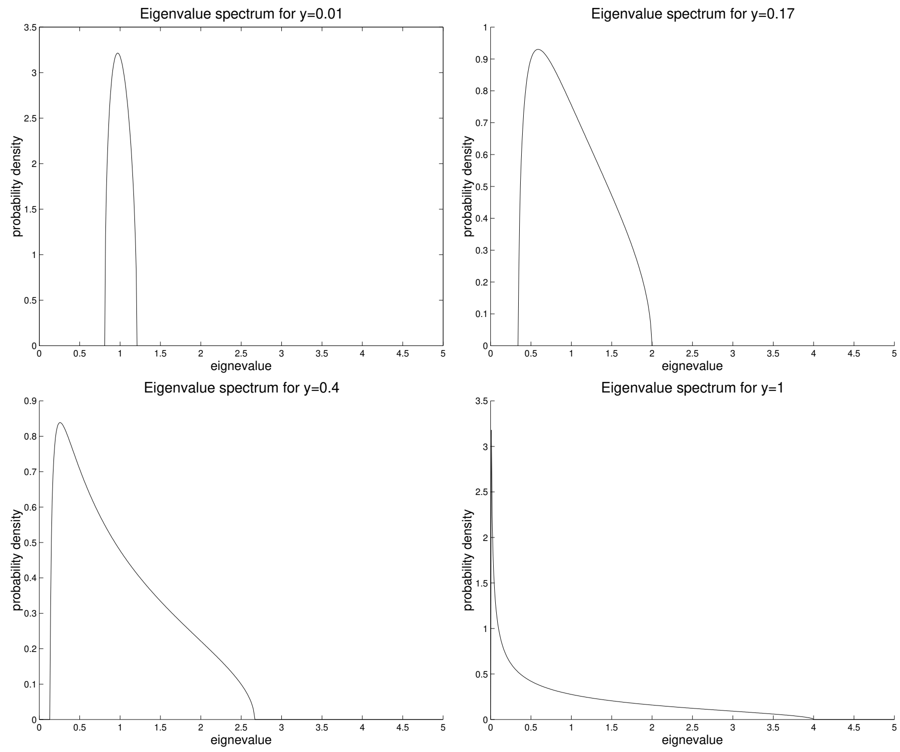


Figure 5.1: The distribution of eigenvalues of the covariance in the pseudo-inverse rule

Now consider the matrix C in the pseudo-inverse rule definition 5.2. This is of the form

$$C^{\mu\nu} = \frac{1}{n}\sum_{i=1}^{n} \xi_i^{\mu}\xi_i^{\nu}$$

where C is an $m \times m$ matrix and the elements ξ_i^{μ} are i.i.d. mean zero, unit variance. Furthermore we know that $m < n$. We can use Silverstein's result to find the distribution of the eigenvalues of C. For $m << n$ the eigenvalues cluster tightly around 1. For larger values of m this distribution spreads out until at $m = n$ the distribution changes form. Figure 5.1 illustrates the distributional form for various values of $y = m/n$

Knowledge of this distribution will help us to relate the different learning rules to one another.

5.2.5 Relationship with pseudo-inverse and Hebbian rules

Returning now to the Hopfield network, let us make the standard assumption that each element of each pattern takes values ± 1 with equal probability (and hence each element is zero mean, unit variance), and is independent of all other pattern elements. This enables the result above to be used for the Hopfield network.

The inverse of a matrix can be calculated through the use of a Taylor expansion:

$$C^{-1} = (I - (I - C))^{-1} = I + (I - C) + (I - C)^2 + \ldots$$

This will converge to C^{-1} if $(I-C)$ has all eigenvalues with magnitude less than one. Equivalently C must have eigenvalues between 0 and 2. Thankfully because of the Silverstein result given above we know that to be true for the $C = C^{\mu\nu}$ used in the pseudo-inverse rule if the number of patterns is significantly smaller than the pattern size ($m < 0.17n$). This can be read from the top left graph of figure 5.1.

So using this in the pseudo-inverse (5.2), we can get a first order approximation of

$$w_{ij}^m = \frac{2}{n}\sum_{\mu=1}^{m}\xi_i^\mu\xi_j^\mu - \frac{1}{n}\sum_{\mu=1}^{m}\sum_{\nu=1}^{m}\xi_i^\mu C^{\mu\nu}\xi_j^\nu$$

which is the same as the equivalent order expansion (5.9) of one form of the new learning rule.

The Hebbian rule is the zeroth order expansion of the pseudo-inverse rule. The first order expansion itself is non-incremental, but the new learning rule provides an incremental method whose first order expansion is equivalent to that of the pseudo inverse.

The speed of conversion of the Taylor expansion is dependent on how close the eigenvalues of $(I-C)$ are to zero. Hence for low memory loadings, where the eigenvalues of C are more tightly distributed about 1, the new learning rule is a better approximation to the pseudo-inverse.

5.2.6 Expanding the new learning rule

The weight matrix of the new learning rule is defined in terms of previous weight matrices, and as such this is a recursive definition. The definition of the new learning rule (5.3) can be expanded out to

$$\begin{aligned} w_{ij}^m \quad = \quad & \sum_{r=1}^{m}\frac{(-1)^{r-1}}{(n-1)^r}\sum_{\mu_r=1}^{m}\left(\prod_{s=1}^{r-1}\sum_{\mu_s=1}^{m}\sum_{k_s=1}^{n}\right)R(\mu_1,\ldots,\mu_r) \\ & \times T(i,k_1,...,k_{n-1},j)\xi_i^{\mu_1}\left(\prod_{t=1}^{r-1}\xi_{k_t}^{\mu_t}\xi_{k_t}^{\mu_{t+1}}\right)\xi_j^{\mu_r} \end{aligned} \tag{5.12}$$

where the $(\prod_{a=1}^{n}\sum_{z_a}\sum_{y_a})$ notation is used to represent $\sum_{z_1}\sum_{y_1}\sum_{z_2}\sum_{y_2}\cdots\sum_{y_n}\sum_{z_n}$, for simplicity. (This can be seen as a product of summation operators). Function R is given by

$$R(a_1,a_2,\ldots,a_r) = \begin{cases} 1 & \text{if } \exists p\in\{1,2,\ldots,r\} \text{ s.t. } a_1>\ldots>a_p<a_{p+1}<\ldots<a_r \\ & \text{and } a_i\neq a_j\ \forall i\neq j \\ 0 & \text{otherwise} \end{cases}$$

and T is the function

$$T(a_1,a_2,\ldots,a_r) = \begin{cases} 1 & \text{if } a_i\neq a_{i+1} \text{ for all } 1\le i<r \\ 0 & \text{otherwise} \end{cases}$$

There is an easy way of thinking about the indicator function R. Suppose we have r cards numbered with values $a_1, \ldots, a_r$. Sort these cards into descending order with the lowest valued card on top. Place the cards on the table one at a time from the top of the pack. Each card must be placed adjacent to and either on the far right or far left of the cards already on the table. If at the end of doing this you could have obtained a row of cards which read $a_1, a_2, \ldots, a_r$ from left to right, then $R(a_1, \ldots, a_r) = 1$. Otherwise $R = 0$.

To see the truth of (5.12), we outline an inductive argument similar to that used earlier. Equation (5.12) is true for $m = 1$. Assuming it to be true for $(m-1)$, we look at the form of w_{ij}^m given by (5.3). We write it out again here for convenience:

$$w_{ij}^m = w_{ij}^{m-1} + \frac{1}{n-1}(\xi_i^m \xi_j^m - \delta_{ij}) - \frac{1}{n-1} \sum_{k=1, k\neq i}^{n} \xi_i^m \xi_k^m w_{kj}^{m-1} - \frac{1}{n-1} \sum_{k=1, k\neq j}^{n} w_{ik}^{m-1} \xi_k^m \xi_j^m$$

All the terms of the expansion of W^m which do not contain any ξ^m factors come from the w_{ij}^{m-1} part of (5.3), and correspond to the terms on the RHS of (5.12) without any ξ^m factors. Hence we only have to account for the part of the expansion containing ξ^m factors. The first of these, $\frac{1}{(n-1)}\xi_i^m \xi_j^m$, is straightforward. It corresponds to the $r = 1, \mu_1 = m$ term of (5.12). The final two parts of the definition of w_{ij}^m contribute in the following way. They modify w_{ij}^{m-1} by effectively putting two ξ^m factors on either the beginning of the terms in the expansion or on the end of the terms in the expansion (hence the condition R), multiplying by (-1/n), and summing over the two new indices. This provides the remaining terms of (5.12). The T constraint comes from $k \neq i$ and $k \neq j$ parts of the definition above.

5.2.6.1 Pattern stability

In order to store a pattern in a Hopfield network, the pattern must be a fixed point of the Hopfield dynamics. In other words ξ^ν must be such that $\sum_{j=1, j\neq i}^{n} \xi_i^\nu w_{ij}^m \xi_j^\nu > 0 \ \forall i$. This leads to the requirement

$$\sum_{j=1, j\neq i}^{n} \xi_i^\nu w_{ij}^m \xi_j^\nu = 1 + N_i^\nu > 0$$

where

$$\begin{aligned} N_i^\nu \quad &= -w_{ii} + \sum_{r=1}^{m} \frac{(-1)^{r-1}}{(n-1)^r} \sum_{j=1}^{n} \sum_{\mu_r=1, \mu_r\neq\nu}^{m} \left(\prod_{s=1}^{r-1} \sum_{\mu_s=1}^{m} \sum_{k_s=1}^{n} \right) S(\mu_1, \mu_2, \ldots, \mu_r; \nu) \\ &\times T(i, k_1, \ldots, k_{n-1}, j) \xi_i^\nu \xi_i^{\mu_1} \left(\prod_{t=1}^{r-1} \xi_{k_t}^{\mu_t} \xi_{k_t}^{\mu_{t+1}} \right) \xi_j^{\mu_r} \xi_j^\nu \end{aligned} \tag{5.13}$$

with

$$S(a_1, a_2, \ldots, a_r; b) = R(a_1, a_2, \ldots, a_r) - R(a_1, a_2, \ldots, a_r, b)$$

where we have used the fact that $\frac{1}{n-1}\sum_{j\neq k_{r-1}} \xi_j^{\mu_r} \xi_j^\nu = 1$ for $\mu_r = \nu$ The N_i^ν is called the noise at neuron i for the recall of memory ν. This is because the signal term is given by the 1 (obtained from $(1/(n-1)) \sum_{j\neq i} \xi_i^\nu \xi_j^\nu \xi_j^\nu = 1$), and the noise term is required to be low enough in magnitude so that $(1+N_i^\mu)$ does not go below zero. If $N_i^\nu < -1$ then the pattern ξ^μ is unstable at neuron i and is therefore recalled incorrectly.

The diagonal term $j = i$ is removed by subtracting w_{ii}. This is not expanded for ease of future analysis.

5.2.6.2 Use of random patterns

The aim is to find the absolute capacity of the network given this new learning rule. In order to make any estimate of capacity something must be known about the patterns that are to be stored. Here the patterns are assumed to be random in the usual way [44]. The ξ_k^μ are taken to be random variables with $\xi_k^\mu = \pm 1$ and $P(\xi_k^\mu = +1) = \frac{1}{2}$. The variables are assumed to be mutually independent for all (k, μ).

Definition 6 (Absolute Capacity) *The absolute capacity of a network is given by the asymptotic relationship between the number of patterns and the number of neurons. It describes how the maximum number of patterns stored increases as the number of neurons increases, given that in the limit $n \to \infty$ the probability that all the patterns are stable is 1.*

In reality we are interested in finding good lower bounds to the capacity. Suppose the dependence of m on n is written $m(n)$ and is nondecreasing. Consider first the noise at the single neuron i for the recall of memory ν. This is given by N_i^ν as defined above. The N_i^ν consists of the sum of many random variables. These random variables are products of independent random variables ξ_k^μ, but are not themselves independent.

5.3 The capacity: signal to noise method

5.3.1 A heuristic approach

Though the noise (5.13) term might look very complicated, it is in fact just a sum of many ± 1 variables. Furthermore it can be shown that

$$E|N_i^\nu|^2 < \frac{m^2}{2n^2(1-\sqrt{m/n})^2}, \; E(N_i^\mu) = 0$$

asymptotically (See (5.32), although this second moment case is in fact much simpler).

At this point we make the assumption that the noise term is Gaussian for large n and m, being a sum of many variables, which, although not independent, have a somewhat 'sparse' dependence. Note that really the noise is not Gaussian, although it might tend to a Gaussian distribution. In fact for this method to work, it must tend to a Gaussian distribution much faster than the variance decreases. This is a form of large deviations problem. (The next section looks at a way of tackling this issue properly). In this case the probability that a particular bit of a particular pattern is unstable is given by

$$P(N_i^\nu < -1) = \frac{1}{2} + \frac{1}{2}\mathrm{erf}\left(-\frac{1}{\sqrt{2}\sigma}\right) \to \frac{\sigma}{\sqrt{2\pi}}\exp\left(-\frac{1}{2\sigma^2}\right) \tag{5.14}$$

as $\sigma \to 0$. Here $\sigma^2 = E|N_i^\nu|^2 < m^2/[2n^2(1-\sqrt{m/n})^2]$.

Now if we choose $m(n)$ such that

$$\sigma^{-2} = \frac{2n^2\left(1-\sqrt{\frac{m(n)}{n}}\right)^2}{m(n)^2} = 2\ln n \tag{5.15}$$

then we find from (5.14) that

$$P(N_i^\nu < -1) < p_{mn} < \frac{1}{n^2\sqrt{8\pi \ln n}}$$

and so $mnP(N_i^\nu < -1) < mnp_{mn} \to 0$.

Finally, because this is true for all $m \times n$ values of i, ν, the simple lemma

Lemma 1 *Let A_{nk} be an event. Let $A_n = \bigcup_{k=1}^n A_{nk}$. Then*

$$\lim P(A_n) = 0 \text{ if } \limsup_{n\to\infty} \sum_{k=1}^n P(A_{nk}) \to 0$$

shows that with $m(n) = n/(\sqrt{2\ln n})$, the probability that any of the bits of any of the patterns are unstable tends to zero as the network gets larger, and hence the network is within capacity. Calculating m from (5.15) to the highest order, we find the capacity is at least $n/(\sqrt{2\ln n})$.

5.4 A rigorous analysis

The above method is heuristic. It makes strong assumptions on the distribution of the noise term. When we analyse the noise term, we see it is far from Gaussian, and we have no reason to make assumptions about any convergence rates.

From simulations, the structure of the noise N_i^1 is illustrated (figure 5.2). It certainly does not take the simple zero mean binomial form that the noise term for the Hebb rule takes [44]

This figure is obtained by taking a sample of twenty thousand sets of 15 random patterns of length 100 (each bit being independently chosen, and ± 1 with equal probability). These sets of patterns were used to calculate the noise term for the first bit of the first pattern using (5.13). Figure 5.2 is a histogram of the resulting twenty thousand noise values.

In this section we take the time to prove, without assumption, that the capacity is indeed $n/\sqrt{2\ln n}$.

5.4.1 Outline of proof

Because the proof is somewhat long-winded, an outline of the approach will be given here.

Consider the event that the noise term outweighs the signal for some neuron and some pattern. We wish to show that the probability of this event tends to zero as $n \to \infty$ for a

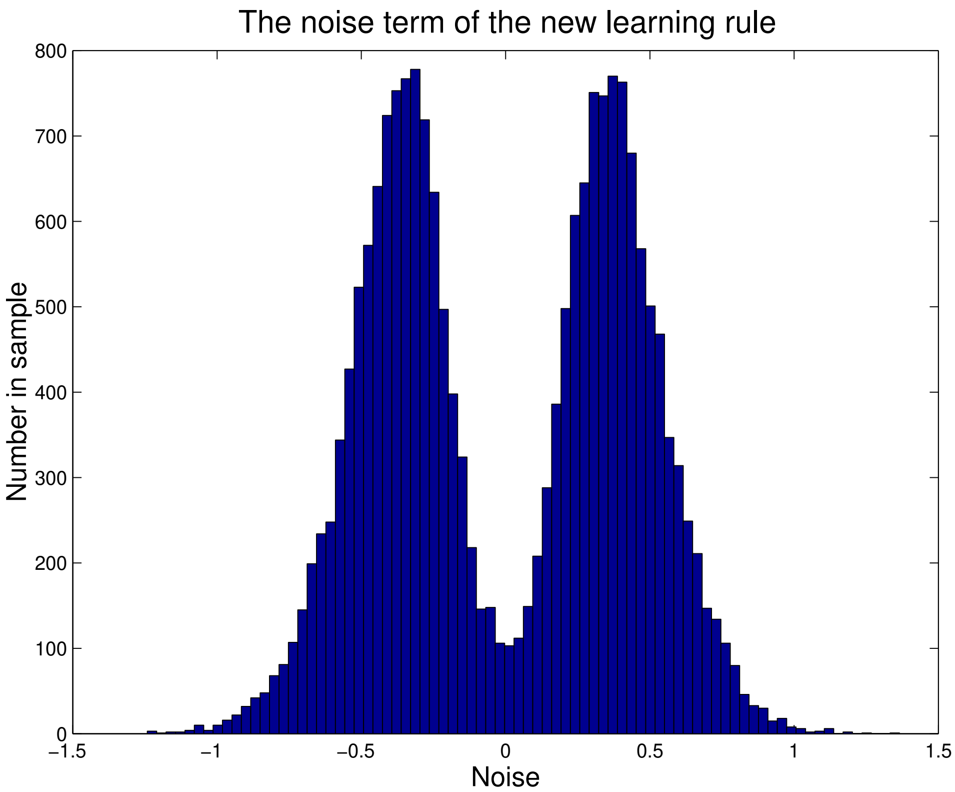


Figure 5.2: Sample from noise term for a single bit of the first stored pattern (new rule) n=100 m=15

memory storage level of $n/\sqrt{2\ln n}$.

The first step involves reducing this to a problem on one neuron and one pattern. To do this the Borrel-Cantelli lemmas are used, because these are ideally suited to situations such as this where limit arguments are required.

Now we want to make no assumptions about the distribution of the noise term. Hence a general method is needed for assessing the probability that the noise will be too large. For this reason the Chebyshev or Markov inequality is used to bound the probability by a moment expansion. The exact form of Markov inequality needs to be chosen carefully so that it is not too loose, in which case the moment expansion will not converge. At the same time it needs to be tight enough for the bound to decay to zero suitably fast as the network size tends to infinity.

At this stage the problem has been transformed into one of calculating moments. If the network were trained using the Hebb rule, this would be unproblematic because one would have an expectation of a product of independent elements. The result would follow simply (see the analysis in [60, 61]). In general however this move could be undesirable: if we do not know the distribution, finding all the moments is unlikely to be any easier. However in this case, after the Minkowski inequality is used, the moments can be expressed in terms of sums of the expectations of products of ξ_k^μ terms. Because these terms are mutually independent, and take the values $+1$ and -1 with equal probability, the expectation of any of these products is zero unless the product contains an even number of ξ_k^μ terms with each upper and lower index, in which case the expectation is one.

So the original problem has become a combinatoric problem, involving the counting of products which are pairings; in other words products with an even number of ξ_k^μ terms with the same upper and lower index. Again this combinatorial problem is hard, but it is possible to calculate an upper bound to the number of combinations which is suitably tight to prove the original proposition.

5.4.2 Reducing to a bounding problem

Firstly we reintroduce and prove the simple lemma given above (a variant of a Borrel-Cantelli lemma).

Lemma 2 *Let A_{nk} be an event. Let $A_n = \bigcup_{k=1}^{K(n)} A_{nk}$ for some K a function of n. Then*

$$\lim P(A_n) = 0 \text{ if } \limsup_{n\to\infty} \sum_{k=1}^{K(n)} P(A_{nk}) \to 0$$

Proof

$$P(\bigcup_{k=1}^{K(n)} A_{nk}) \leq \sum_{k=1}^{K(n)} P(A_{nk})$$

so

$$\limsup_{n\to\infty} P(A_n) \leq \limsup_{n\to\infty} \sum_{k=1}^{K(n)} P(A_{nk}) = 0 \Rightarrow \lim_{n\to\infty} P(A_n) = 0$$

□

So, suppose $P(N_i^\nu < -1) < p_{mn}$ for all i, ν, then the probability of all stored patterns being stable tends to one as $n \to \infty$ if

$$\limsup_{n\to\infty} m(n) n p_{m(n)n} \to 0$$

5.4.3 Using the Markov inequality

The following is an extension of the Markov or Chebyshev inequality.

Lemma 3 (Overbound)

$$P(|N| \geq 1) \leq \frac{E[e_n(uN^2)]}{e_n(u)}$$

where $e_n(x)$ is the truncation of the exponential power series

$$e_n(x) = \sum_{d=1}^{D(n)} \frac{x^d}{d!}$$

for some $D(n)$ non decreasing.

Proof Let A be the event $\{|N| \geq 1\}$ and I_A be the indicator function of A. Then $|N| \geq I_A$ and hence $e_n(uN^2) \geq e_n(u) I_A$ as e_n is non-decreasing on $(0, \infty)$. Take expectations to obtain the result. □

So suppose $u_n = 2\ln n$ and $D(n) = [4\ln n]$, where we use the square brackets to imply the least integer greater, then

$$\mathrm{e}_n(u_n) = \exp(u_n) - \sum_{d=D(n)+1}^{\infty} \frac{(2\ln n)^d}{d!}$$

but

$$\begin{aligned} 0 < \sum_{d=D(n)+1}^{\infty} \frac{(2\ln n)^d}{d!} &< \frac{(2\ln n)^{D(n)}}{D(n)!} \sum_{d=0}^{\infty} \frac{(2\ln n)^d}{D(n)^d} \\ &= \frac{(2\ln n)^{D(n)}}{D(n)!} \left(\frac{1}{1 - \frac{2\ln n}{D(n)}} \right) \\ &\leq \frac{6(2\ln n)^{D(n)}}{D(n)!} \text{ because } D(n) = [4\ln n] \\ &\to 0 \text{ as } n \to \infty \end{aligned}$$

To see this one can take logs and see that the log tends to $-\infty$.

So $\mathrm{e}_n(u_n)$ is asymptotically equal to $\exp(u_n) = n^2$. Let

$$P(N_i^\nu(n) \leq -1) < P(|N_i^\nu(n)| \geq 1) \leq \frac{E[\mathrm{e}_n\{u_n(N_i^\mu(n))^2\}]}{\mathrm{e}_n(u_n)} \stackrel{\text{def}}{=} p_{mn}$$

Now if $E[\mathrm{e}_n\{u_n(N_i^\mu(n))^2\}]$ is bounded as $n \to \infty$ then $mnp_{mn} \to 0$ as long as $m/n \to 0$. Then by lemma 2 above, the probability of all memories being stable will tend to one.

Henceforth we take $u_n = 2\ln n$ and $D(n) = [4\ln n]$, as above. Then the problem of calculating the capacity is reduced to that of finding an $m(n)$ such that $m(n)/n \to 0$ and such that $E[\mathrm{e}_n\{u_n(N_i^\mu(n))^2\}]$ is bounded as $n \to \infty$.

5.4.4 Calculating the moments

The problem now involves looking at $E[\mathrm{e}_n\{u_n(N_i^\mu(n))^2\}]$. To begin to calculate this, the expectation is expanded into a form which involves calculation of the moments of the noise.

Now

$$\begin{aligned} E[\mathrm{e}_n\{u_n(N_i^\mu(n))^2\}] &= E\left(\sum_{d=1}^{D(n)} \frac{(u_n)^d (N_i^\mu(n))^{2d}}{d!} \right) \\ &= \sum_{d=1}^{D(n)} \frac{(u_n)^d}{d!} E|N_i^\nu(n)|^{2d} \end{aligned} \tag{5.16}$$

Furthermore

$$E^{\frac{1}{2d}}|N_i^\nu(n)|^{2d} \leq E^{\frac{1}{2d}}|w_{ii}|^{2d} + \sum_{r=1}^{m(n)} E^{\frac{1}{2d}}|N_{ir}^\nu(n)|^{2d} \tag{5.17}$$

using the Minkowski inequality with $N_i^\mu(n)$ given by (5.13) and with

$$N_{ir}^\nu = \frac{1}{(n-1)^r}\sum_{j=1}^{n}\sum_{\mu_r=1,\mu_r\neq\nu}^{m}\left(\prod_{s=1}^{r-1}\sum_{\mu_s=1}^{m}\sum_{k_s=1}^{n}\right)S(\mu_1,\mu_2,\ldots,\mu_r;\nu)$$
$$T(i,k_1,\ldots,k_{n-1},j)\xi_i^\nu\xi_i^{\mu_1}\left(\prod_{t=1}^{r-1}\xi_{k_t}^{\mu_t}\xi_{k_t}^{\mu_{t+1}}\right)\xi_j^{\mu_r}\xi_j^\nu \tag{5.18}$$

5.4.5 Case one: $r = 1$

Firstly we consider the value of $E|N_{i1}^\nu|^{2d}$. The term N_{i1}^ν contains the term $S(\mu_1;\nu)$ which is equal to $R(\mu_1) - R(\mu_1,\nu)$. But $R(\mu_1) = 1$, and $R(\mu_1,\nu) = 1$ for $\mu_1 \neq \nu$.

The sum in (5.18) is over all $\mu \neq \nu$, and so for all elements in the sum we get $S = 0$. Hence the whole sum is zero, and

$$E|N_{i1}^\nu|^{2d} = 0 \tag{5.19}$$

5.4.6 Case two: $r > 1$

Now we go on to look at $E|N_{ir}^\nu|^{2d}$. We expand out the $2d$ power, and rewrite $E|N_{ir}^\nu|^{2d}$ in the form

$$E|N_{ir}^\nu|^{2d} = \frac{1}{(n-1)^{2dr}}\sum_{\omega\in\Omega}S(\omega)T(\omega)E(G_\omega) \tag{5.20}$$

where ω counts through all the valid index choice sets. An index choice set is an ordered set of values of all the indices summed over in (5.18). In other words it represents the choice of one particular summand. The set of all such sets Ω takes the form

$$\{(\mu_{ab},k_{ab},j_b|a=1,\ldots,r;b=1,\ldots,2d)\ \mu_{ab}\leq m;k_{ab},j_b<n\ \forall a,b\}$$

Note that S and T take the values 1 and 0 only.

In (5.20), G_ω is given by the product of all the elements below

$$\begin{array}{ccccccccc}
\xi_i^\nu & \xi_i^{\mu_{11}} & \xi_{k_{11}}^{\mu_{11}} & \xi_{k_{11}}^{\mu_{21}} & \xi_{k_{21}}^{\mu_{21}} & \cdots & \xi_{k_{(r-1)1}}^{\mu_{r1}} & \xi_{j_1}^{\mu_{r1}} & \xi_{j_1}^\nu \\
\xi_i^\nu & \xi_i^{\mu_{12}} & \xi_{k_{12}}^{\mu_{12}} & \xi_{k_{12}}^{\mu_{22}} & \xi_{k_{22}}^{\mu_{22}} & \cdots & \xi_{k_{(r-1)2}}^{\mu_{r2}} & \xi_{j_2}^{\mu_{r2}} & \xi_{j_2}^\nu \\
\vdots & & & & & & & & \\
\xi_i^\nu & \xi_i^{\mu_{1(2d)}} & \xi_{k_{1(2d)}}^{\mu_{1(2d)}} & \xi_{k_{1(2d)}}^{\mu_{2(2d)}} & \xi_{k_{2(2d)}}^{\mu_{2(2d)}} & \cdots & \xi_{k_{(r-1)(2d)}}^{\mu_{r(2d)}} & \xi_{j_{(2d)}}^{\mu_{r(2d)}} & \xi_{j_{(2d)}}^\nu
\end{array}$$

with all indices except ν and i given by $\omega \in \Omega$. The product has been written out in this grid form for convenience. The $2d$ rows above correspond to the replicas obtained when the $2d$ power of a sum (that is $E|N_{ir}^\nu|^{2d}$) is turned into a sum of products of elements.

Now this product contains many different ξ terms. If two of these terms have the same upper and the same lower indices, then the two terms are called a pair.

If all the terms in a product such as G_ω pair, then the product, and the corresponding index choice, are called pairings.

Now G_ω is a product of many different ± 1 random variables. Furthermore the set of all ξ_i^μ for different i and μ is a mutually independent set. Hence if any of those variables,

ξ_j^μ say, does not pair with any other variable, then it is independent of all the other terms. It can be taken out of the expectation in (5.20) and be averaged over separately ($E(AB) = E(A)E(B)$ if A and B are independent). But $\xi_j^\mu + \pm 1$ each with probability 1/2. Hence its expectation is zero. Therefore the expectation of the whole product is zero.

So $E(G_\omega)$ is only non-zero if it is a pairing. Furthermore if it is a pairing, its expectation is 1.

Because only pairings within G_ω contribute to the expectation, we want to count the total number of valid index choices which are pairings and which have $S = 1$ and $T = 1$. The problem has been transformed into a combinatoric one.

There are some index choices and corresponding products for which $S(\omega)$ and $T(\omega)$ are zero. Any index choice for which these are non-zero is called a constrained index choice (c-index choice) and the corresponding product a c-product. More generally the c prefix notation implies the constrained version of the relevant concept.

In order to reduce possible misunderstanding, the various concepts introduced so far are defined more rigorously.

Definition 7 (index choice) *An index choice is any member of the (ordered) index-choice set*

$$\{(\mu_{ab}, k_{ab}, j_b | a = 1, \ldots, r; b = 1, \ldots, 2d)\ \mu_{ab} \leq m; k_{ab}, j_b < n\ \forall a, b\}$$

An index choice corresponds to the choice of a single summand in (5.20).

Definition 8 (product) *For every index choice there is a corresponding product (which is written in a grid fashion)*

$$\begin{array}{ccccccccc}
\xi_i^\nu & \xi_i^{\mu_{11}} & \xi_{k_{11}}^{\mu_{11}} & \xi_{k_{11}}^{\mu_{21}} & \xi_{k_{21}}^{\mu_{21}} & \cdots & \xi_{k_{(r-1)1}}^{\mu_{r1}} & \xi_{j_1}^{\mu_{r1}} & \xi_{j_1}^\nu \\
\xi_i^\nu & \xi_i^{\mu_{12}} & \xi_{k_{12}}^{\mu_{12}} & \xi_{k_{12}}^{\mu_{22}} & \xi_{k_{22}}^{\mu_{22}} & \cdots & \xi_{k_{(r-1)2}}^{\mu_{r2}} & \xi_{j_2}^{\mu_{r2}} & \xi_{j_2}^\nu \\
\vdots & & & & & & & & \\
\xi_i^\nu & \xi_i^{\mu_{1(2d)}} & \xi_{k_{1(2d)}}^{\mu_{1(2d)}} & \xi_{k_{1(2d)}}^{\mu_{2(2d)}} & \xi_{k_{2(2d)}}^{\mu_{2(2d)}} & \cdots & \xi_{k_{(r-1)(2d)}}^{\mu_{r(2d)}} & \xi_{j_{(2d)}}^{\mu_{r(2d)}} & \xi_{j_{(2d)}}^\nu
\end{array}$$

where the indices of each element of the product is determined by the index choice. Products with the same elements, but in different positions are considered different. Hence counting the number of products is the same as counting the number of index choices.

The product is written in this grid fashion for convenience. This structure will play an important part in the later analysis. When we refer to the 'domino like' structure of the product, we are referring to the fact that one element shares an upper index with its neighbour, and that neighbour a lower index with its next neighbour and so on. Two elements in the same row and next to one another are said to be *neighbours*.

The interest in elements with the same upper and lower indices leads naturally to a graphical representation of the product.

Definition 9 (a graph of a product) *A graph of a product is a labelled graph formed* *from a given product by taking and labelling a node for every element of the product. Then a set of edges is chosen by following these rules.*

1. *The two nodes adjacent to any edge must correspond to elements in the product with the same upper indices, and the same lower indices.*
2. *Every node has at most degree 1.*
3. *No two nodes are adjacent (i.e. have a connecting edge), if their corresponding elements are neighbours.*
4. *The graph is maximal, in the sense that no other graph of the product can be formed with a larger number of edges.*

The nodes of the graph are positioned in the same row/column structure as the elements of a product. Again this is for convenience. In fact it gives a second way of referring to the labels of the graph through row-column reference.

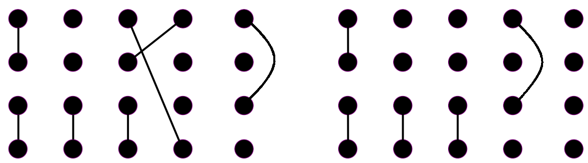
There can be many graphs for some products. A graph is said to be loosely a graph of a product if it is as above, but does not necessarily satisfy rules 3 or 4.

For convenience, we draw the graph to mirror the column/row structure we used to write out the original product.

For example, consider the product

$$\begin{array}{c} \xi_1^2\xi_3^2\xi_3^4\xi_4^4\xi_4^1 \\ \xi_1^2\xi_4^2\xi_4^4\xi_2^4\xi_2^1 \\ \xi_1^3\xi_2^3\xi_2^4\xi_4^4\xi_4^1 \\ \xi_1^3\xi_2^3\xi_2^4\xi_3^4\xi_3^1 \end{array}$$

Both the graphs



are loosely graphs of the above product. The left one is a graph of the above product, because it is maximal.

The conditions 3 and 4 in the above definition will be important in helping us to solve the problem. Basically if two adjacent elements are identical, $S = 0$ and $T = 0$. So we have in fact included one small element of the S, T constraints in choosing what graphs we allow.

Definition 10 (Set of product graphs) *The set of product graphs is the set of graphs which are of some product.*

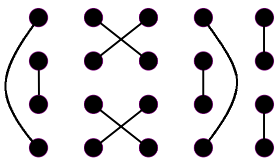
Definition 11 (a product from a graph) *A product is said to be generated from, or produced by, a given graph if the given graph is loosely a graph of the product. There are many products generated from a given graph (corresponding to choices for the indices of each element, and constrained by the structure of the graph).*

Equivalently, a product can be generated from a graph by creating an element for each node of the graph, and by then assigning values to each index. In doing this, the constraints provided by the 'domino' like structure of the product must be taken into account. In addition elements whose corresponding nodes are connected must have identical upper and lower indices.

It is worth noting that the definitions of 'graphs of products' and 'products from graphs' are not reciprocal. We tend to be sloppier in generating products from graphs. Hence if we start with a graph and generate some product from that graph, the original graph may not be a graph of the product. However, a graph of some product can always generate that original product.

Definition 12 (pairing) *A graph of a product is a pairing if and only if every node has degree 1. A product is a pairing if some graph of that product is a pairing. An index choice is a pairing if the corresponding product is a pairing. A product graph which is a pairing is called a paired graph. A product which is a pairing is called a paired product.*

For example the graph



is a pairing, and the product

$$\begin{array}{c}\xi_1^2\xi_3^2\xi_3^3\xi_5^3\xi_5^1\\ \xi_1^3\xi_3^3\xi_3^2\xi_5^2\xi_5^1\\ \xi_1^3\xi_4^3\xi_4^2\xi_5^2\xi_5^1\\ \xi_1^2\xi_4^2\xi_4^3\xi_5^3\xi_5^1\end{array}$$

is a pairing because the above graph is a graph of this product

Definition 13 (Constrained) *An index choice or corresponding product is called constrained, if and only if the index choice satisfies the constraints set by the requirement $S > 0$ and $T > 0$.*

So to calculate 5.20 we need to count the number of constrained index choices which are pairings.

Proposition 3 *The number of constrained index choices which are pairings is less than or equal to the sum of the number of constrained products generated from each possible paired graph.*

Proof There is a one to one relationship between index choices and products. Hence we can count products instead of index choices. By definition, any product which is a pairing is generatable from some paired graph. Hence by counting constrained products generated by each paired graph, every product which is a pairing must be counted once.

In fact more than one paired graph might produce a given paired product, and hence some products may be counted more than once. □

It turns out that this upper bound will be fine for our purposes. And so we bound 5.20 by

$$E|N_{ir}|^{2d} \leq \frac{1}{(n-1)^{2dr}} \sum_{\text{paired graphs G}} \text{c-products generated from G} \tag{5.21}$$

5.4.6.1 An isomorphic problem

For illustration purposes the basic problem can be recast in the following way. This clarifies where the term 'domino like' comes from.

Consider a playing board with $2d$ rows and $2r$ columns. A player is given a set of dominoes, which are unusual in that one half of each domino is red, and the other half blue. The red half has between 1 and n dots on; the blue half has between 1 and m. Each domino covers two squares of the board, and is in infinite supply.

The player plays dominoes by placing each domino horizontally on the board (covering two squares) matching red to red, and blue to blue, and also matching numbers. The first domino must have one spot in the red half, and this is placed in the top left, horizontally aligned and with red to the left.

Each piece is placed until the end of the row is reached. The last piece in the row must have one spot in the blue half. This process is repeated for each row. (In fact each piece placed must also be constrained in a way corresponding to S and T.)

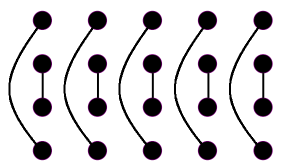
The question is how many possible ways are there of playing dominoes in this way, and getting an even number of each domino on the board.

5.4.6.2 Row pairing

To help to calculate the number of pairings, the concept of a row pairing is introduced.

Definition 14 (Graph: row pairing) *A graph of a product is called a row pairing if every node in one row is connected to the corresponding node in some second row. A graph which is a row pairing is called a row-paired graph*

Example:



Definition 15 (Product: row pairing) *A product is called a row pairing if some graph of the product is a row pairing. Such a product is called a row-paired product.*

Row pairings will help because we will see that they dominate the combinatorial calculation (5.21).

An upper bound to the number of products which are row pairings can be obtained by counting the number of products generated from graphs which are row pairings. In other words we count the number of graphs which are row pairings, and then multiply this by n for each free upper index, and m for each free lower index which the graph allows. In the case of row pairings, each index in the first row is free, but this determines the index in the second row. The following procedure allows us to build a row pairing:

- The elements of the first column pair automatically.
- Pair the rows of the graph up; hence each row has a 'row twin'.
- For each column connect the node in one row with the node in its row twin.
- A product is built from this graph by choosing the index values for one row of a pair. The indices in the second row of the pair are fully constrained by the choice in the first.

From this procedure, an upper bound on the total number of row pairings can be calculated.

Proposition 4 *The total number of row pairings of a product is less than*

$$\frac{(2d)!}{d!2^d} m^{dr} (n-1)^{dr}$$

Proof First of all the number of ways of pairing the rows is

$$\frac{(2d)!}{d!2^d} \tag{5.22}$$

Consider: the $2d$ elements are reordered ($(2d)!$ counts), and then element $2x-1$ is paired with element $2x$. However each pairing is then counted $d!$ times for each of the ways of permuting the element pairs, and 2^d times for each way of permuting within an element pair. Hence the total number of ways of pairing $2d$ elements is given by (5.22). There is only one possible graph for each row pairing.

For each of d row pairs, we can count m possible values of each upper index, and $n-1$ values of each lower index (lower index $k_{jr} \neq k_{(j-1)r}$ and $k_{1r} \neq i$), giving a total of $m^d(n-1)^d$ possibilities for the column. □

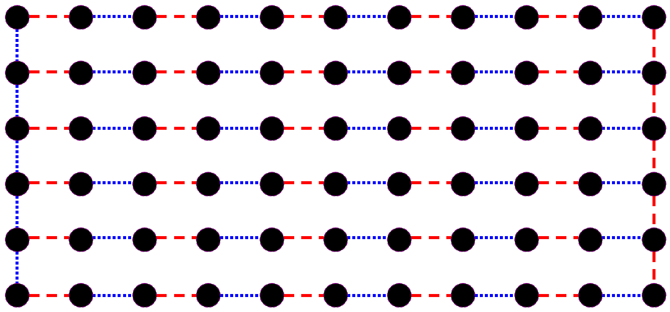
The reason for the domination of row pairings over other pairings is due to the fact when one element A is paired with another B in the same column, the two elements to the right automatically have identical upper (or possibly lower) indices. Hence pairing these two elements only constrains the remaining index, and is therefore less 'costly' than making a general pairing.

5.4.6.3 Non-row pairings

To show this more clearly, we need to develop a procedure for examining products which are pairings, but are not row pairings. It will be good to find a method of generating all the different possible product graphs which are pairings, while at the same time keeping count of the number of products which can be generated from each graph.

It will be helpful to introduce a coloured graphical representation similar to the simple graphs used above. This representation is used in the process of building graphs.

A graph building procedure is used. Start with a graph with the required $2d * 2r = 4dr$ nodes, but no edges. Blue and red edges are then used to represent the different possible constraints on the corresponding indices. A red edge is used to connect two nodes whose upper indices are constrained to be identical, a blue edge for those whose lower indices are constrained to be identical. Because of the 'domino like' effect of the product indices, the initial graph has alternate red and blue edges connecting the nodes in each row, with blue edges connecting all the first column, and red edges connecting the last column:



In this and all the following graphs, a dashed line is used to represent red edges, and a dotted line to represent blue edges.

The original black edges are still used to represent pairs. In other words they represent elements whose top and bottom indices are constrained to be identical. These black edges can be placed on a graph, connecting any two non-adjacent nodes not already connected by a black edge.

A pairing can be built by continuing the placement of black edges, until all nodes are connected to another node.

5.4.6.4 Product counting from graph counting

How many products can be generated by a given graph? Denote this value by P_G. It is straightforward to calculate P_G by considering the different constraints.

Proposition 5 *The graph with no edges has $P_G = (mn)^{2dr}$.*

Proof There are no constraints imposed by the graph on the index choices set. Each upper index takes at most m different values, and each lower index at most $n-1$ different values. There are $2dr$ upper/lower indices, giving the required result. □

Each edge can be considered to be imposing a constraint on the possible values these indices might take. The constraints can be seen in the following way. The elements of a product represented by any two nodes connected by a path containing edges, each of which is either blue or black (we call these elements blue-path connected) are lower index constrained. The elements represented by two nodes connected by a path of red or black edges (red-path connected) are upper index constrained.

Proposition 6 *Suppose that the graph G^+ is created from graph G by placing a black edge between any two nodes, denoted for now by A and B. Then P_G^+ can be calculated from P_G by multiplying it by the following factors.*

- *If A and B are neither blue- nor red- path connected: a factor of $1/(mn)$.*
- *If A and B are only blue-path connected: a factor of $1/(n-1)$.*
- *If A and B are only red-path connected: a factor of $1/m$.*
- *If A and B are both red-path and blue-path connected: no factor.*

Proof If a black edge is placed between two nodes, then some upper and some lower index is fully constrained: only 1 of the mn original choices of that index are now valid.

If however either or both of those indices were already constrained, then the number of original possibilities is reduced accordingly, giving the result above. □

Row pairings again We return to the issue of row pairings again, and use this as an example of how to use the new coloured graph to calculate the relevant P_G^{UB} for a row paired graph.

The initial graph is set up: From this P_G^{UB} is set to $m^{2dr}(n-1)^{2dr}$.

All the first column are connected with red edges. Hence when each of the d edges in the first column is placed, P_G^{UB} is reduced by a factor of $1/m$ for each edge: $1/m^d$ in total. Hence $P_G^{UB} = m^{2dr-d}(n-1)^{2dr}$.

Now the nodes in second column are blue edge connected for each row pair. To see this, start at one node. Follow the blue edge to the previous column, then the black edge to the other row, and then the blue edge back to the original column. Hence when each of the row pairs for this column are connected with a black edge, P_G^{UB} is reduced by a factor of $1/(n-1)^d$.

The nodes in the third column are now red-edge connected for each row pair. The same procedure can then be followed for this row, reducing P_G^{UB} by a factor of $1/m^d$, and then each other row, until finally all nodes are connected. After all this P_G^{UB} will be reduced by a factor of $1/(m(n-1))^{dr}$, giving a final $P_G^{UB} = (m(n-1))^{dr}$, which is identical to the previous result.

Keeping pairings simple When non-row pairings are considered, however, things are not quite so simple. In fact it can soon become hard to keep track of red/blue connectivity for a general situation. A method is needed to simplify things and keep the connectivity structure in a simple form.

Now the nodes of a product graph are labelled. The row-column structure is used to simplify the position of the blue/red edges, and to easily represent the constraint that adjacent nodes in the same row cannot be paired.

The exact row-column position of nodes can be changed without detriment so long as they keep their original labels, and so long as the new row-column structure still keeps the blue/red edge structure, and the adjacent node constraint.

Note that the blue/red edges can also be moved so long as the resulting graph has the same blue/red edge path- connectivity: Only the path connectivity affects the factors in (6).

This facility is used to allow a non-row pairing to be transformed into a row-pairing for some other column-row structure. In this situation additional constraints will occur, which will be represented by additional red/blue connections.

Proceeding from a row pair We require a way of generating all the graphs which could be graphs of a product. Then for each of these graphs we need to know the value of P_G for that graph. An upper bound to the number of products which are pairings is given by

$$\sum_G P_G^{\text{UPPER BOUND}}$$

where G counts through all the possible graphs.

The following technique allows all the product graphs to be generated, while at the same time keeping track of all the red/blue edges, and hence the value of P_G for each graph G.

First we need a definition:

Definition 16 (twin) *Two rows are called twins if their first column elements pair. Two elements are called twins if they are in the same column, and are in twin rows. A graph/product is called a* twinned *graph/product if any paired element is paired with its twin, and any paired element is such that all nodes to its left are also paired. Note that not all elements need be paired.*

Now the following procedure can be used to pair the nodes of the graph.

1. The first column must pair within itself, due to the fact that the first column of a product is the only column whose elements have the lower index i. Pair this column.

2. For each row twin go along the columns placing a black edge between the two nodes in that column until some column is reached whose nodes will not be paired in the final graph.

3. Now we are in the situation where the first unpaired nodes in each twin are not to be paired. Take the first twin rows, and the first unpaired node (A) in the first of these rows. Let B be the second unpaired element. Choose a second node (C) to which that first node should be connected ($< 4dr$ possibilities), and place a black edge between them

4. Rearrange the nodes in the way elaborated below. Then the graph will have a twinned structure again.

5. Repeat from item 2 until all nodes are paired.

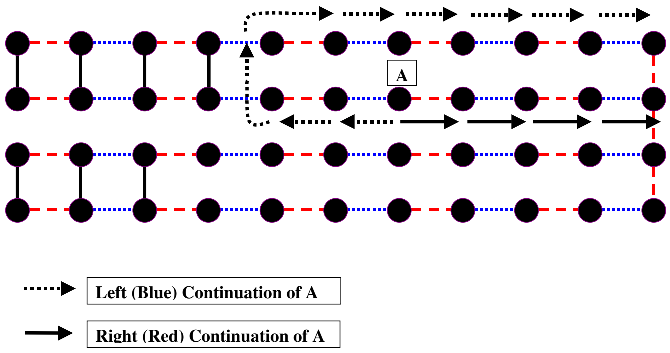
Rearrangement methods The rearrangement method is a little complicated, so it will be illustrated using a concrete example. First one further concept is needed.

Definition 17 (Left/right/blue/red row continuation of a node) *Suppose we have some node C of a twinned graph G. Suppose that C is connected by a red edge to its right-neighbour node. Then that neighbour node and all those to its right (maintaining that order) form the right row continuation of the node C. This is also the red row continuation of C. If instead C was connected by a blue edge to its right-neighbour node, this right row continuation of C would be the blue row continuation of C.*

Otherwise, suppose we have some node C of G, where C is connected by a red edge to its left-neighbour node. Then the left continuation of C can be built by adding one node at a time, starting from the left-neighbour, and continuing along the row until the last unpaired node is reached. The twin to this last unpaired node is then added, followed by the right row continuation of this twin.

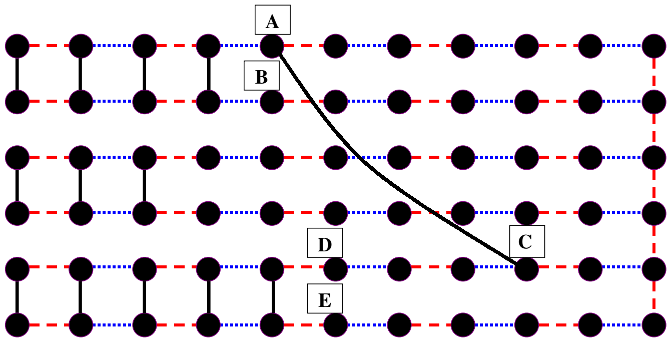
This left continuation of C is also called a red-continuation of C. If instead C were connected by a blue edge to its left-neighbour node, this left row continuation of C would be the blue row continuation of C.

More simply, the continuation of a node C is obtained by moving from that node in the direction of the edge of the relevant colour. If that direction happens to be left, then the movement wraps around onto the twin column. The movement stops when the end of a row has been reached, and all the nodes which are passed form the continuation of the node (in the order of the motion).

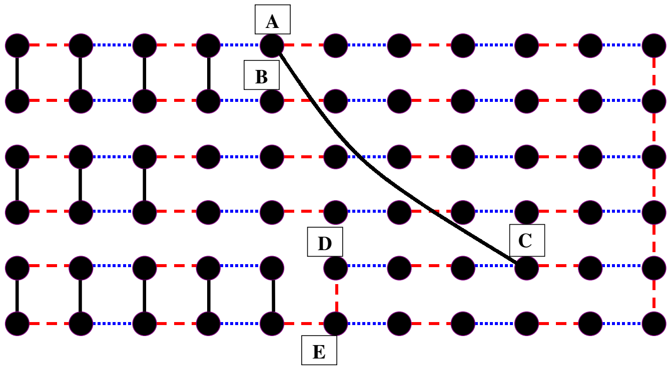


Now the rearrangement method can be given. The general method will be given, but will be graphically illustrated on a specific example.

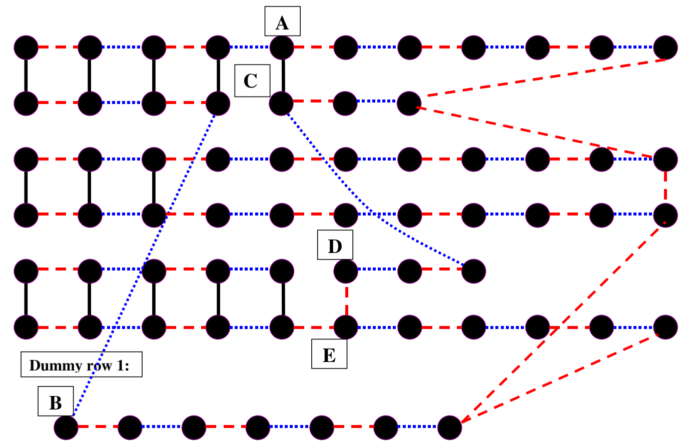
Consider some twinned graph G, where the first unpaired node in each row is not to be paired with its twin. Suppose the first unpaired node A is to be paired with some node C. Label the twin of A with B. Denote by D the leftmost unpaired node in the row that C is in. Let E be the twin of D. Let 'rcol' denote the colour of the edge from A to its right, and 'lcol' be the colour of the edge from A to its left. In the example below 'rcol'='red', So rcol-continuation would mean red-continuation. Likewise lcol-continuation would mean blue-continuation.



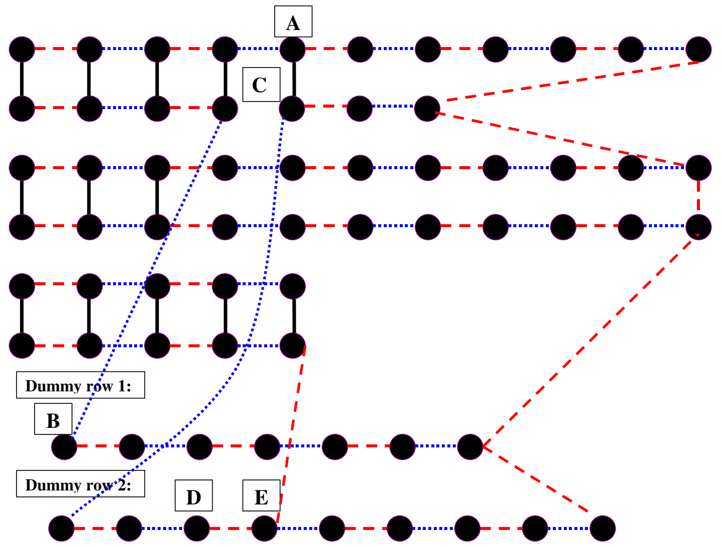
Step 1: The coloured edges can be manipulated so long as the red/blue path connectivity remains the same. So the edge from the D to the left of D can be replaced by one from D to E.



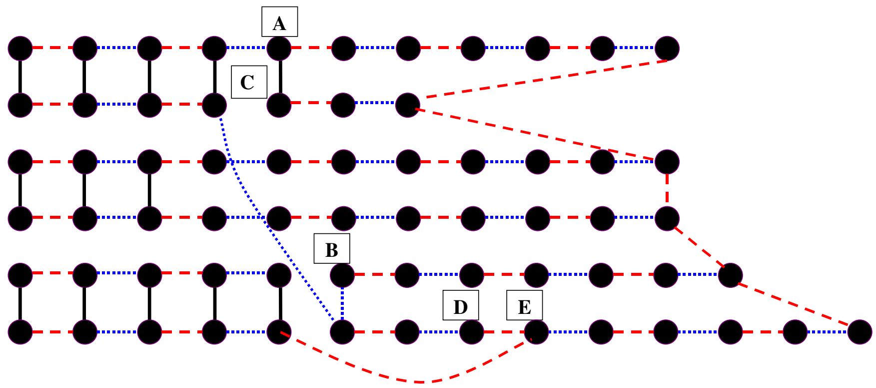
Step 2: B and its rcol-continuation are put in dummy row 1 and replaced by C and all the nodes to its right. The red edges in the last column can be rearranged (maintaining path-connectivity) to simplify the graph.



Step 3: We form a dummy row 2 with the lcol-continuation of C. Now E is colour-edge (in this case red-edge) connected to D, but will never be black edge connected because it does not pair with its twin. Hence these two nodes can now act as neighbour nodes: they have the relevant 'domino-like' constraints. So the nodes in dummy row two can reasonably form part of a row.



Step 4: Dummy row 1 is put where E originally was, and dummy row 2 is put where D originally was. The colour edge from C to B is replaced by an edge from B to its twin.



The end result is all nodes are paired within their (new) twin rows, and all first nodes in each twin are colour-path connected.

There are a number of differences between this new twinned graph and the standard twinned graph. Firstly the rows are no longer the same lengths. This might give further restrictions on what can later be connected to what, but need not be considered here because we only need an upper bound.

The second difference is that when the dummy rows are substituted, they might have different coloured edges in each column from the other rows. This does not matter because the two substituted rows are twins, and the only requirement is that they have the same column colouring as each other (In fact the rows can just be shifted up one column).

The third difference is that there are a few additional colour connections. However these colour connections path-connect single free nodes only with paired nodes, hence no pairing can be made between two nodes path-connected through these colour connections, and so they will not affect any future counting factors (see proposition 6).

Besides these differences, all the usual blue/red connections exist and the constraints about adjacent nodes not pairing still holds. Therefore this new twinned graph can form a base for making further twinned connections.

Note that pairing two nodes which are twins of one another puts a pairing between two nodes which are red-path or blue-path connected, and hence reduces P_G by a factor of $1/m$ or $1/(n-1)$. Pairing nodes which are not twins is pairing two nodes which are not path connected, and hence reduces P_G by a factor of $1/m(n-1)$.

Special case The above methods of rearrangement are not appropriate if the node A is to be connected to a node within its own row or the row of its twin C. In this case, suppose A is to be connected to some node D.

Situation 1:If D is in A's row, and its rcol-continuation is to the right then swap D and its rcol-continuation with C and its rcol-continuation.

Situation 2:If D is A's twin row, and its rcol-continuation is to the left, then flip (reverse the order) of the nodes from C to D inclusive.

If neither of these situations is the case, then consider node C, and its pair E. If C does not pair with a node in its row or its twin row, then rearrange using C as the base node instead. Continue as before. Otherwise exchange the rcol-continuation of E and the rcol-continuation of D (if D and E are in different rows) or flip nodes D to E inclusive (same row). Then either situation 1 or situation 2 must be true (rcol-continuation has changed direction). Return back to the beginning and continue as before.

Now all possible cases are covered: The rearrangement method is complete.

Some counting This algorithm will help us to do some counting.

We start with a blank graph, and with $P_G = (m(n-1))^{2dr}$. Let $P_{G_{row}}$ be the value of P_G if G is a row pairing. This equals $(m(n-1))^{dr}$

Step 1 involves pairing the elements of the first column. This can be done in

$$\frac{(2d)!}{d!2^d} \tag{5.23}$$

ways. This gives us the twin rows.

From now on each element is chosen in turn. It is either paired with its twin, and P_G is multiplied by a factor $1/m$ or $1/(n-1)$ (depending on its position), or its is paired with some other element ($< 4dr$ choices), and P_G is multiplied by a factor of $1/(m(n-1))$.

So an element can be paired with its twin, giving only one possible choice for the position of the edge, but only reducing the number of products which can be made from the graph by $1/m$ or $1/(n-1)$.

On the other hand an element can be paired with any other element, giving many possible choices for the position of the edge, and as a result many different possible final graphs. But the cost in this increase in the number of graphs made is that fewer products can be generated by each graph: P_G is reduced by a factor of $1/m(n-1)$.

Because $m < n-1$, the element and its pair contribute at most a factor of

$$\left(1 + \frac{4dr}{m}\right) F_{twin} \tag{5.24}$$

to the sum 5.20, where F_{twin} is the factor (either $1/m$ or $1/(n-1)$) contributed when an element is paired with its twin.

Now the rearrangement procedure is used to get the graph back into the required format. Then the next element can be chosen, and so on until all elements are paired. There are $2dr$ pairs in total.

As a result the sum 5.20 is given by (5.23) times the $2dr$ power of (5.24):

$$\frac{(2d)!}{d!2^d}\left(1 + \frac{4dr}{m}\right)^{2dr} m^{dr} n^{dr} \tag{5.25}$$

where $m^{dr} n^{dr}$ is the value of $P_{G_{row}}$ obtained when all elements are paired with their twins.

5.4.6.5 The upper index constraints

So far only very loose constraints on the index values have been considered. In fact the constraints on the upper indices given by $S > 0$ are much tighter than simply not allowing neighbour pairs.

Consider one row of the index choice set. Suppose that all the upper indices in that row are distinct from one another, but are not in any particular order. The constraint $S > 0$ says that they must be able to be ordered by taking the lowest, and then putting the next lowest either to the right or left of the lowest, and the same with the next, and so on. Furthermore the final ordering must not have the lowest on the right.

As a result only $2^{r-1} - 1$ of the original $r!$ possible index orders are valid.

Now for a row pairing, the validity of one row automatically implies the validity of a second. Hence in this situation the validity of the indices of one element implies the validity of its pair.

Hence we get one constraint for each of the d row pairs, so only $(2^{r-1} - 1)^d$ of the $r!^d$ unconstrained upper index orders satisfy the constraint $S > 0$.

If the pairing is not a row pairing this does not automatically hold. A smaller proportion of the pairings might be valid, because the validity of the indices for one element, might make its pair invalid. However we can use the proportion of valid index choices for a row pairing as an upper bound for the proportion for non row pairings.

Hence the total number of constrained index choices which are pairings is bounded above by

$$\frac{(2^{r-1}-1)^d}{(r!)^d} \sum_G P_G$$

Substituting (5.25) for $\sum_G P_G$ bounds the total number of constrained index choices by

$$\frac{(2d)!}{d!2^d} \frac{(2^{r-1}-1)^d}{(r!)^d} \left(1 + \frac{4dr}{m}\right)^{2dr} m^{dr} (n-1)^{dr} \tag{5.26}$$

Using (5.21) gives

$$E|N_{ir}|^{2d} \leq \frac{(2d)!}{d!2^d} \frac{(2^{r-1}-1)^d}{(r!)^d} \left(1 + \frac{4dr}{m}\right)^{2dr} \frac{m^{dr}}{(n-1)^{dr}} \tag{5.27}$$

5.4.6.6 Putting it all together

We can now put everything together. Equation (5.17) says

$$E^{\frac{1}{2d}} |N_i^\nu(n)|^{2d} \leq E^{\frac{1}{2d}} |w_{ii}|^{2d} + \sum_{r=1}^{m(n)} E^{\frac{1}{2d}} |N_{ir}^\nu(n)|^{2d}$$

We now know from (5.19) that

$$E|N_{i1}^\nu|^{2d} = 0$$

and from (5.27) that for $r > 1$

$$E|N_{ir}|^{2d} \leq \frac{(2d)!}{d!2^d} \frac{(2^{r-1}-1)^d}{(r!)^d} \left(1 + \frac{4dr}{m}\right)^{2dr} \frac{m^{dr}}{(n-1)^{dr}}$$

Substituting (5.19) and (5.27) into (5.17) gives us

$$E^{\frac{1}{2d}}|N_i^\nu(n)|^{2d} \leq E^{\frac{1}{2d}}|w_{ii}|^{2d} + \left(\frac{(2d)!}{d!2^d}\right)^{\frac{1}{2d}} \sum_{r=2}^{m(n)} \frac{m^{r/2}}{(n-1)^{r/2}} \left(\frac{(2^{r-1}-1)^{1/2}}{(r!)^{1/2}}\left(1+\frac{4dr}{m}\right)^r\right) \tag{5.28}$$

To perform the sum in (5.28), we split the summands into two different classes. First we choose some $a >> 4$. The exact value of a does not matter in the limits we are interested in; it only matters that it is constant as $n \to \infty$. The first class contains summands where $2 \leq r \leq a^2d^2$. The second consists of cases where $a^2d^2 < r \leq m$.

For summands in the first class, we use the fact that

$$(1+x)^k < \sum_{t=0}^{\infty}(kx)^t = \frac{1}{1-\min(1,kx)}$$

for $x > 0, k > 1$. This comes from bounding the binomial expansion with a geometric progression ($r^t > {}^rC_t$), and then summing the geometric progression to infinity.

This gives us

$$\left(1+\frac{4dr}{m}\right)^r < \frac{1}{1-min(1,4a^4d^5/m)}$$

For now let us denote $min(1,4a^4d^5/m)$ by $\epsilon(d,m)$. Later we will show that ϵ is in fact small in the situations which are important to us.

In addition to this we note that $(2^{r-1}-1)/r! < 1/2$ for $r \geq 2$. This gives an expression for the summand

$$\frac{(2^{r-1}-1)^{1/2}}{(r!)^{1/2}}\left(1+\frac{4dr}{m}\right)^r < \frac{1}{\sqrt{2}(1-\epsilon(d,m))} \tag{5.29}$$

for $2 \leq r < a^2d^2$.

For summands in the second class we have

$$\frac{(2^{r-1}-1)^{1/2}}{(r!)^{1/2}}\left(1+\frac{4dr}{m}\right)^r \leq \frac{1}{\sqrt{2}}\frac{(\sqrt{2})^r(1+4d)^r}{(r!)^{1/2}}$$

using $r \leq m$. For large enough, but constant, a ($a >> 4\sqrt{2}$), $[\sqrt{2}(1+4d)]^r = (2(1+4d)^2)^{r/2} < (r!)^{1/2}$, because $r > a^2d^2$.

Hence

$$\frac{(2^{r-1}-1)^{1/2}}{(r!)^{1/2}}\left(1+\frac{4dr}{m}\right)^r < 1/\sqrt{2} \tag{5.30}$$

for $a^2d^2 < r \leq m$.

Substituting (5.29) and (5.30) into (5.28) we get

$$\begin{aligned} E^{\frac{1}{2d}}|N_i^\nu(n)|^{2d} &< E^{\frac{1}{2d}}|w_{ii}|^{2d} + \left(\frac{(2d)!}{d!2^d}\right)^{\frac{1}{2d}} \sum_{r=2}^{a^2d^2}\left(\frac{m}{n-1}\right)^{\frac{r}{2}}\frac{1}{\sqrt{2}(1-\epsilon)} + \sum_{r=a^2d^2+1}^{m}\frac{m^{\frac{r}{2}}}{\sqrt{2}(n-1)^{\frac{r}{2}}} \\ &< \left(\frac{(2d)!}{d!2^{2d}}\right)^{\frac{1}{2d}} \sum_{r=2}^{\infty}\left(\frac{m}{n-1}\right)^{\frac{r}{2}}\frac{1}{(1-\epsilon)} \end{aligned}$$

The last sum above is now a geometric progression, and can be evaluated to give

$$E^{\frac{1}{2d}}|N_i^\nu(n)|^{2d} < E^{\frac{1}{2d}}|w_{ii}|^{2d} + \left(\frac{(2d)!}{d!2^{2d}}\right)^{\frac{1}{2d}} \frac{m/(n-1)}{1-\sqrt{m/(n-1)}} \left(\frac{1}{(1-\epsilon)}\right) \tag{5.31}$$

So far the diagonal elements have not been discussed. They are unproblematic, as they are an order of magnitude smaller than the noise term. In fact all the methods used so far on the noise term can equally well be applied to the expansion of the diagonal given by (5.12). The only difference is that there is one index fewer to be summed over, and the constraint is given by R and T rather than S and T. The result is that

$$E^{\frac{1}{2d}}|w_{ii}|^{2d} < \frac{2}{\sqrt{n-1}}B$$

where B is the upper bound we have already obtained for $E^{1/2d}|N_i^\nu - w_{ii}|^{2d}$.

We therefore have

$$E^{\frac{1}{2d}}|N_i^\nu(n)|^{2d} < \left(\frac{(2d)!}{d!2^{2d}}\right)^{1/2d} \frac{m/(n-1)}{1-\sqrt{m/(n-1)}} \left(\frac{1}{(1-\epsilon)}\right) \left(1+\frac{2}{\sqrt{n-1}}\right) \tag{5.32}$$

The required result follows shortly. Suppose that $m(n)$ is such that

$$\frac{m/(n-1)}{1-\sqrt{m/(n-1)}} \left(\frac{1}{(1-\epsilon(m,d))}\right) \left(1+\frac{2}{\sqrt{n-1}}\right) \leq \frac{1}{\sqrt{(2+\delta)\ln n}} \tag{5.33}$$

for some $\delta > 0$

then using (5.16) and the fact that $2^{2d} > (2d)!/(d!d!)$, we have

$$E[\mathrm{e}_n\{u_n(N_i^\mu(n))^2\}] < \sum_{d=1}^{D(n)} \frac{(2\ln n)^d (2d)!}{d!d!2^{2d}[(2+\delta)\ln n]^d} < 4\sum_{d=1}^{\infty} \frac{2^d}{(2+\delta)^d}$$

which is bounded and independent of n. Therefore using lemma 2 in the way outlined at the beginning of section 5.4, it is seen that this $m(n)$ is within capacity. Calculating $m(n)$ from (5.33) to highest order in n, we find that $m(n)$ is within capacity if

$$m(n) \leq \frac{(n-1)(1-\epsilon*)}{\sqrt{(2+\delta)\ln n}} \tag{5.34}$$

for any $\epsilon* > 0$. This follows because

$$\epsilon(m,d) = \min(1, 4a^4d^5/m)$$

where a is constant and $d < D(n) = [4\ln n] < 5\ln n$. With m given by (5.34), $4a^4d^5/m \to 0$. Hence for large enough n, $(1-\epsilon*)^{-1} > (1-\epsilon)^{-1}$

The network is within capacity with $m(n)$ given by (5.34) for all $\delta > 0$ and $\epsilon* > 0$ Hence the capacity of the network is at least $m(n) = (n-1)/\sqrt{2\ln n}$ which asymptotically equals $n/\sqrt{2\ln n}$.

We have shown that the capacity of the new learning rule is given by

$$m(n) = n/\sqrt{2\ln n}$$

5.5 Simulations

Simulations of the networks were performed, and the capacity results for the new learning rule were compared with those of the Hebb rule. Results were as the theory above predicted, and can be found in figure 5.3. A random set of patterns of length 400 was chosen, with each bit independent and of value ± 1 with probability $1/2$ each. For each network size n, these patterns (truncated to length n) were stored in the network one by one. For each storage level m, each of the stored patterns was tested to see if it was a fixed point. The lowest m for which some pattern failed to be a fixed point ($m^*(n)$ say) was taken to be the capacity, and the point $(n, m^*(n))$ was plotted on the graph. This was done identically for the new learning rule and for the Hebb rule. The results given are for one run for each value of n.

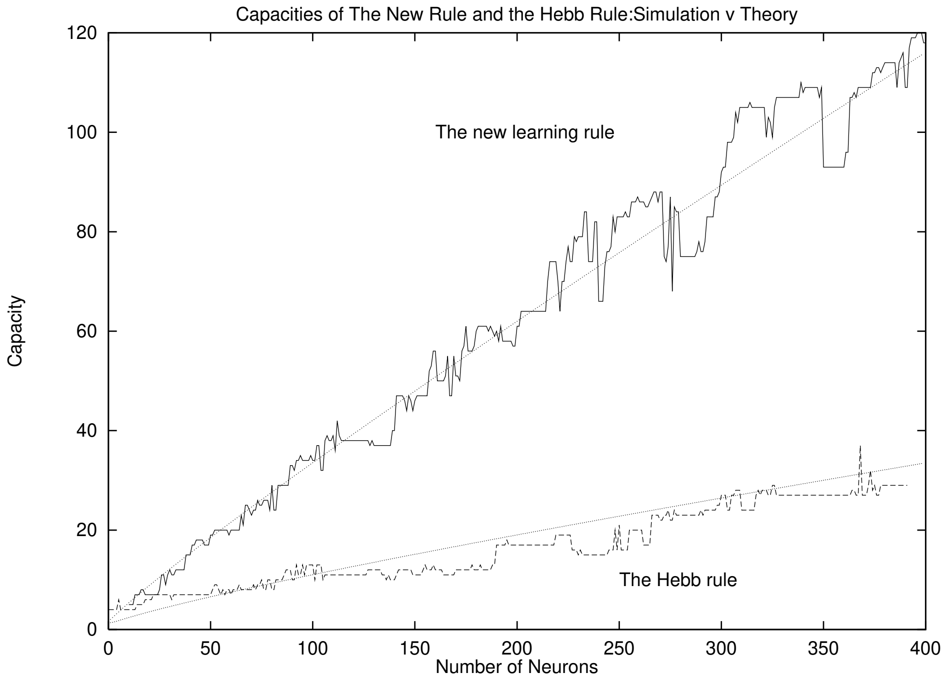


Figure 5.3: Capacity of new learning rule v Hebb

5.6 Further extensions of this learning rule

It is possible to extend this learning rule further, and continue to increase capacity by introducing a further term. Let the weight matrix be defined recursively by the rule

$$w_{ij}^0 = 0 \ \forall i, j \in \{1, 2, ..., n\}$$

$$\begin{aligned} w_{ij}^\mu &= w_{ij}^{\mu-1} + \frac{1}{n}\xi_i^\mu \xi_j^\mu - \frac{1}{n}\xi_i^\mu h_{ji}^\mu - \frac{1}{n} h_{ij}^\mu \xi_j^\mu + \frac{1}{n}\sum_k h_i^\mu h_j^\mu \text{ for } i \neq j \\ w_{ii}^\mu &= 0 \end{aligned} \tag{5.35}$$

where

$$h_i^\mu = \sum_{k=1}^{n} w_{ik}^{\mu-1} \xi_k^\mu$$

This rule is also local and incremental. It is more complicated than the earlier rule due to the introduction of the $h_i h_j$ term. However the price of this is small; it adds a further $o(n^2)$ calculation to the previous method. In return the capacity is slightly increased. The graph of the maximum storage is plotted for this rule in a similar way the same way in figure 5.6 (except now new patterns were chosen at for each n). When this is compared with figure 5.3 it can be seen that the capacity appears to have increased slightly. In fact the capacity of this rule seems to scale as

$$\frac{n}{\sqrt[3]{4 \ln n}}$$

There are other benefits of this extended rule. One such benefit is that it allows a clearer and more direct comparison with the pseudo inverse rule.

5.6.1 Incremental pseudo-inverse and the extended rule

It is possible to use the partitioned inverse equations to generate an incremental form of the pseudo-inverse learning rule. This is given by

$$w_{ij}^0 = 0 \ \forall i, j \in \{1, 2, ..., n\}$$

$$w_{ij}^\mu = w_{ij}^{\mu-1} + \frac{c}{n}\xi_i^\mu \xi_j^\mu - \frac{c}{n}\xi_i^\mu h_{ji}^\mu - \frac{c}{n} h_{ij}^\mu \xi_j^\mu + \frac{c}{n}\sum_k h_i^\mu h_j^\mu \tag{5.36}$$

where

$$h_i^\mu = \sum_{k=1}^{n} w_{ik}^{\mu-1} \xi_k^\mu$$

and

$$c = (1 - \frac{1}{n}\sum_{ij} \xi_i^\mu W_{ij}^{\mu-1} \xi_j^\mu)^{-1}$$

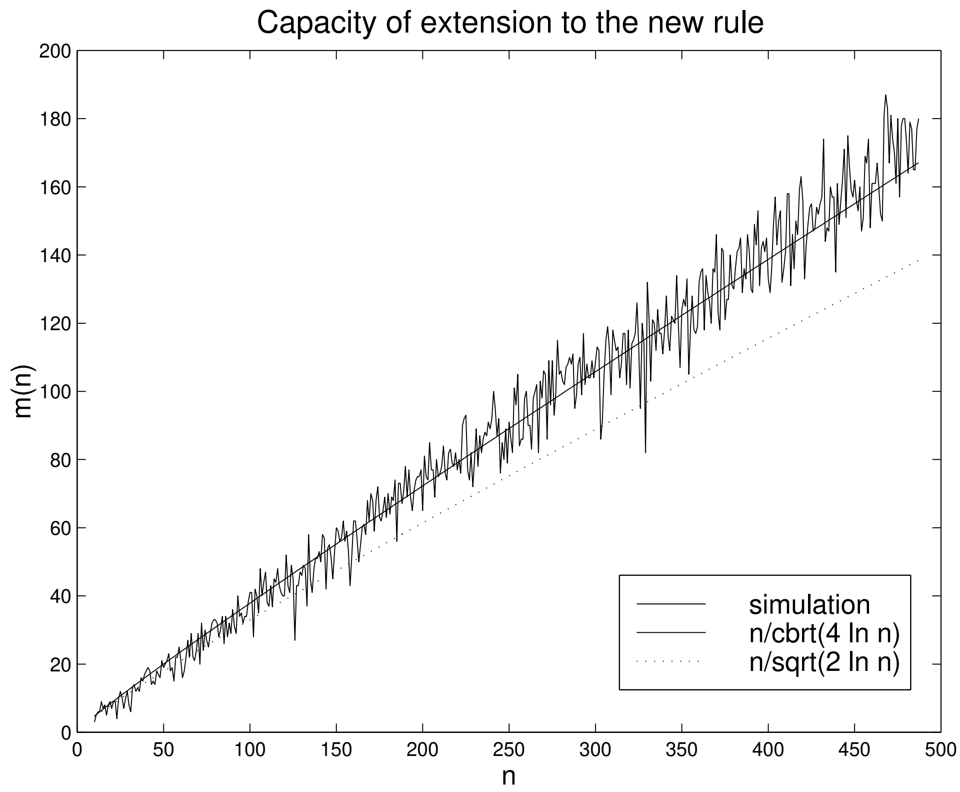


Figure 5.4: Capacity of the extended learning rule

This is not a local rule, because there is no way that $\boldsymbol{\xi}^T W \boldsymbol{\xi}$ can be calculated in a local manner. If instead of calculating c, it is approximated by a constant 1, then this becomes a form of the extended learning rule given above (5.35). This approximation is good for low memory loadings. So it can be seen that this learning rule is an approximation to the pseudo-inverse rule.

5.7 Conclusions

This improvement of capacity for the Hopfield network is very significant. At network sizes of a million neurons, the absolute capacity of the new learning rule is five times greater than that of the Hebb rule, increasing the storage for a given network size.

In this thesis we have focused only on absolute capacity. The Hebb rule has a relative capacity (where imperfect recall is allowed) of $0.14n$ [47], which scales better than this absolute capacity for the new learning rule. Although it may be possible to calculate a relative capacity for the new learning rule, it should be noted that even at sizes of a million neurons, the absolute capacity of the new learning rule is greater than the relative capacity of the Hebb.

The analysis given in this thesis provides a new way of tackling the issue of Hopfield network capacity. The analysis makes significant use of the binary structure of the pattern space, and the independence of different pattern elements. This enables learning rules made from products of many different elements to be analysed. Learning rules defined in an incremental fashion will often be of this form. The analysis makes use of some old but powerful tools of probability, and some simple graphical approaches.

Results on the eigenvalues of sample covariance matrices are used to provide a comparison of the pseudo-inverse, the new learning rule and the Hebb rule. It can be seen clearly that the new rule is a better approximation to the pseudo-inverse than the Hebb rule is. An extension of the new learning rule is given which is better still.

This chapter provides a firm grounding for further analysis of the new learning rule which has been introduced.

Chapter 6

The Basins of Attraction

6.1 Introduction

[1] In the last chapter, the capacity of the new learning rule was looked at. This was done by looking at the fixed points of the Hopfield dynamics. However this is not enough for the Hopfield network to be useful. These fixed point must also have significant attraction basins for the network to be of any use. Hence the nature of the basins of attraction of a Hopfield network is as important as the capacity. In the previous chapter, the attraction basins were not considered. Here the attraction basins of the new rule are studied and are compared with those of other learning rules.

Three important characteristics of basins of attraction are considered: indirect and direct basins of attraction, distribution of sizes of basins of attraction and the shape of the basins of attraction.

The results for the new learning rule are compared with those of the Hebb rule. It will be shown that the size of direct and indirect basins of attractions are generally larger for the new rule than for the Hebb rule, the distribution of sizes is more even, and the shape of the basins more round.

In order to use networks such as the Hopfield [3] networks as autoassociative content addressable memories it is not enough to ensure that the required patterns are fixed points of the network dynamics. The required patterns must be attractors, and as such must have substantial attraction basins in order to perform in the desired fashion.

6.2 Learning rule characteristics

In order to perform in the way described, the network must have a learning algorithm which sets the connection weights between all pairs of neurons so that it can perform this task.

As mentioned in the previous chapter, these learning rules can have a number of characteristics. Firstly a rule can be local. If the update of a particular connection depends only on information available to the neurons on either side of the connection,

[1]Much of the content of this chapter has been published in [62]

then the rule is said to be local. Locality is important, because it provides a natural parallelism to the learning rule, which, when combined with the local update dynamics, make a Hopfield network a truly parallel machine.

Secondly a rule can be incremental. If the learning process can modify an old network configuration to memorise a new pattern, without needing to refer to any of the previously learnt patterns, then an algorithm is called incremental. Clearly incrementality makes the Hopfield network adaptive, and therefore more suitable for changing environments or real time situations.

Thirdly a rule can either perform an immediate update of the network configuration, or can be a limit process. The former makes for faster learning.

Fourthly a learning algorithm has a capacity. This is some measure of how many patterns can be stored in a network of a given size. The absolute capacity [44] of a network is given by the number of patterns that can be stored with correct recall as an asymptotic function of the number of neurons.

Finally there is the concern of this chapter: the sizes and shapes of the basins of attraction of the fixed points of the network. Patterns must not only be stored, but be able to be recalled. The recall process involves moving from the network start state into some fixed state. The basin of attraction of a fixed point is defined to be the set of states that are attracted to that fixed point. Different learning rules produce different basins of attraction. Clearly capacity and attraction basins are related. Here we consider what extra information about the performance of the new learning rule can be gained from the study of attraction basins.

The following are desirable features of the basins of attraction:

1. They should be large.
2. They should be rounded: states with similar Hamming distances from the fixed point should behave in the same fashion.
3. They should be evenly distributed.

Large attraction basins are desirable, because then more states are attracted to stored patterns rather than spurious fixed points. Round attraction basins make the network more predictable: whether or not a state is attracted to a stored pattern is reliably dependent on its Hamming distance away from that pattern. An even distribution of basin sizes also increases predictability and ensure there is no large bias against any particular pattern.

Attraction basins and capacity are not independent, and there have been methods to try to use the attraction basin size in a capacity-like definition. One such method can be found in [63]. There the concept of a completion capacity is introduced. This is an information theoretic concept, and it is used to calculate what is effectively the maximal information gain through the action of an associative memory for a given network size. Such a definition can be useful to represent the information processing power of a particular network. The analysis in [63] assumes both the independence of the components of the original patterns and the independence of the components after they have passed through the Hopfield update rule. This latter case is problematic, as the learning rule mixes the original independent components using the weight matrix. Hence the final components can be significantly correlated. A result of this problem is that the above approach, though generally useful, cannot be followed for the analysis undertaken here, such as that involved with looking at the shape and skew of attraction basins.

6.3 The Hopfield network

Once again we note that the Hopfield network is an attractor neural network governed by the difference equation

$$x_i(t+1) = \mathrm{sgn}(\sum_{j \neq i} w_{ij} x_j(t))$$

where $x_i(t)$ is the ± 1 state of neuron i, w_{ij} is the symmetric weight matrix and $x_i(n) = \pm 1$ is the nth update of the ith neuron where updates are performed asynchronously. We will be comparing the new learning rule with the Hebb rule and pseudo inverse rule as defined in the previous chapter.

6.3.0.1 The new learning rule

In order to overcome the problems of both of these learning methods a form of the new learning rule introduced in the last chapter will be used. In this chapter we will use the following definition.

Definition 18 (The new learning rule) *The weight matrix w_{ij} of an attractor neural network is said to follow the new learning rule if it obeys*

$$w_{ij}^0 = 0 \ \forall i,j \ \textit{and} \ w_{ij}^{\nu} = w_{ij}^{\nu-1} + \frac{1}{n}\xi_i^{\nu}\xi_j^{\nu} - \frac{1}{n}\xi_i^{\nu}h_{ji}^{\nu} - \frac{1}{n}h_{ij}^{\nu}\xi_j^{\nu} \tag{6.1}$$

where $h_{ij}^{\mu} = \sum_{k=1,k\neq i,j}^{n} w_{ik}^{\mu-1}\xi_k^{\mu}$ is a form of local field at neuron i (the input to the neuron i) , and ξ^{μ} is the new pattern to be learnt ($\xi_i^{\mu} = \pm 1$)

This rule is local: w_{ij} depends only on information available at the two adjacent neurons (the values of ξ_i, ξ_j, h_i, h_j). It is clear from the recursive nature of (18) that it is also incremental. It also has an absolute capacity of $m_c(n) = n/\sqrt{2\ln n}$ and performs better than the Hebb rule with correlated patterns (see the next chapter, and [64]).

It should be noted that the weight matrix obtained is not independent of the presentation order. This important matter is discussed in a later chapter, and in the paper [65]. There it is shown that this rule (or rather a minor variant of it) acts as a palimpsest or forgetful learning rule: it does not suffer from catastrophic forgetting when capacity is reached, but instead retains recent memories at the expense of old ones.

The dependence of the weight matrix on the presentation order will not affect the following analysis of macroscopic properties such as the size of attraction basins. This is because the patterns we will use will be statistically independent and identically distributed. Hence there is no structural order to the patterns, and they can perfectly well be interchanged without changing the relevant probability distributions.

6.3.1 The patterns

Here we consider training the network with the usual m patterns $\xi_i^\mu = \pm 1$ of length n $(i = 1, \ldots, n,\ \mu = 1, \ldots, m)$ with each bit chosen independently to be 1 with probability 1/2.

6.4 Direct and indirect basins of attraction

There are two fundamentally different types of basins of attraction. These are commonly called direct and indirect attraction basins. A state is within the direct basin of attraction of a fixed point if the dynamics of the network will always take the system state to the fixed point, and the network state will never move away from the fixed point throughout this process.

On the other hand a state not in the direct attraction basin, but which is eventually attracted to the fixed point (with a high probability) is said to be within the indirect attraction basin. The reason why the term 'with a high probability' is used is that whether or not attraction occurs may well depend on the update order. We use the term complete attraction basin (or sometimes just attraction basin) to denote the union of the direct and indirect attraction basins.

It is one thing to know whether a state is within the attraction basin of a fixed point, and another thing to measure the size of the attraction basin. Here we represent attraction size by the hamming radius of the basin of attraction as is common in the literature [66, 67, 44, 68]. (Note that the hamming radius is equivalent to the overlap radius up to a linear transformation.) This in itself is not well defined, because attraction basins tend not to be Hamming hyper-spherical. One common measure is the minimum radius [66, 44]:

$$r(p) = \inf\{d = < p, q >; q \in Basin(p)\} \tag{6.2}$$

where $< p, q >$ is the Hamming distance between states p and q in Q, and Q is the set of all stored patterns. Then to compare two learning rules, we can compare the average size of attraction basin over all the stored patterns

$$r = \frac{1}{|Q|} \sum_{p \in Q} r(p) \tag{6.3}$$

6.5 Direct basins of attraction

Direct attraction basins are important because the attraction rate is faster within the direct attraction basin than outside it. For synchronous dynamics, attraction occurs in one step. See section 6.8.3 for the reasons for this.

The definitions (6.2) and (6.3) were used to compare the size of direct attraction basins for different learning rules at different network loadings.

The direct attraction basins were calculated as follows:

- For each pattern, ξ^μ, find the state x the shortest distance d from ξ^μ for which

$$\xi_i^\mu \sum_j w_{ij} x_j$$

  is less than zero for some i. This implies that when the Hopfield update rule is applied to the neuron i of the network it returns $-\xi_i^\mu$, which is an incorrect update.

- The radius of the direct basin of attraction is therefore $d - 1$.

The graph 6.1 shows the average direct basin sizes for three learning rules. Here a network size of 300 neurons was used. The graph for each learning rule terminates when absolute capacity is reached. The pseudo-inverse is included for comparison, even though it is neither a local nor incremental rule.

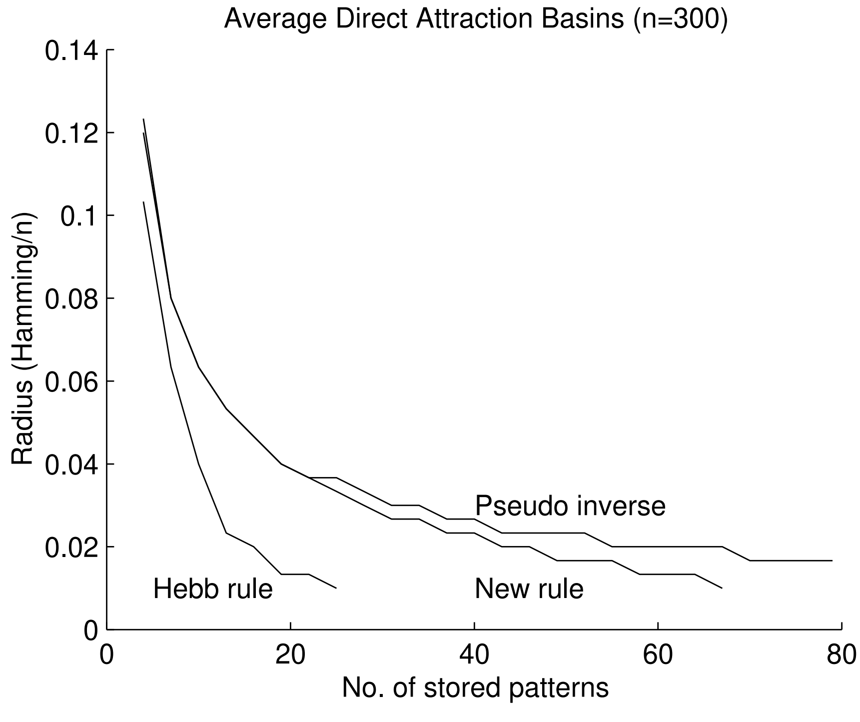


Figure 6.1: Comparing the sizes of direct attraction basins

6.6 Indirect basins of attraction

Of course the direct basin of attraction is not of the greatest importance. The size of the complete attraction basin is that which most affects the ability of the network to perform as an associative memory. In this section we look at the following characteristics of complete attraction basins:

1. The distribution of sizes of attraction basins across different stored patterns.
2. The shapes of the attraction basins.
3. The effect that increasing the number of stored patterns has on the above factors.

Once again we need some way of measuring the sizes of the attraction basins. This is harder than it was with direct attraction basins. We chose a process similar to that used in [66]. Our method is outlined below.

1. Choose some fixed point μ corresponding to a stored pattern.
2. Set the initial normalised Hamming radius $r = r_0 < 1$. We set $r_0 = 0$
3. Let the set A be all states of Hamming radius nr from the fixed point.
4. Sample 100 states from A.
5. Calculate how many of these states are attracted to the fixed point. Denote this number $t_\mu(r)$.
6. Increment the normalised Hamming radius r by a suitable amount and repeat from (3) until $r = 1$ or few states are attracted.
7. Repeat for each stored pattern, μ.

The benefit of this method is that $t_\mu(r)$ gives an estimate for the percentage of states distance nr from the fixed point μ which are attracted to the fixed point. We arbitrarily define the radius R of attraction to be the largest value of r such that $t_\mu(s) \geq 90$ for all $s \leq r$. In other words most states within a Hamming sphere radius R are attracted to the fixed point. Note that $t_\mu(r)$ can also give a measure of the skew of the basin of attraction. The skew of an attraction basin is some measure of how much it differs in shape from a hypersphere. The more elongated or contorted, the greater the skew.

If $t_\mu(r)$ decreases rapidly as r increases from R then the basin of attraction is approximately hyperspherical: at one radius almost all states are attracted, and at a slightly higher radius very few are attracted. If, however, the gradient of $t_\mu(r)$ is shallow, then the basin of attraction is skew: Some states are attracted from further away than others.

6.7 Results

The graphs of figure 6.2 give the values of $t_\mu(r)$ for each pattern μ, for a network size of 150 neurons, and a number of different network loadings.

From such results we can compare the size of the complete attraction basins for the Hebb rule and the new learning rule. Figure 6.4 gives the distributions of attraction basin sizes for each of the rules for different loadings.

Lastly we look at how the skew of each attraction basin varies with the size of the attraction basin. Here, skew is defined as $t_\mu^{-1}(90) - t_\mu^{-1}(40)$. This slightly arbitrary, but simple definition suffices to approximate the gradient of $t(r)$. The larger the skew, the

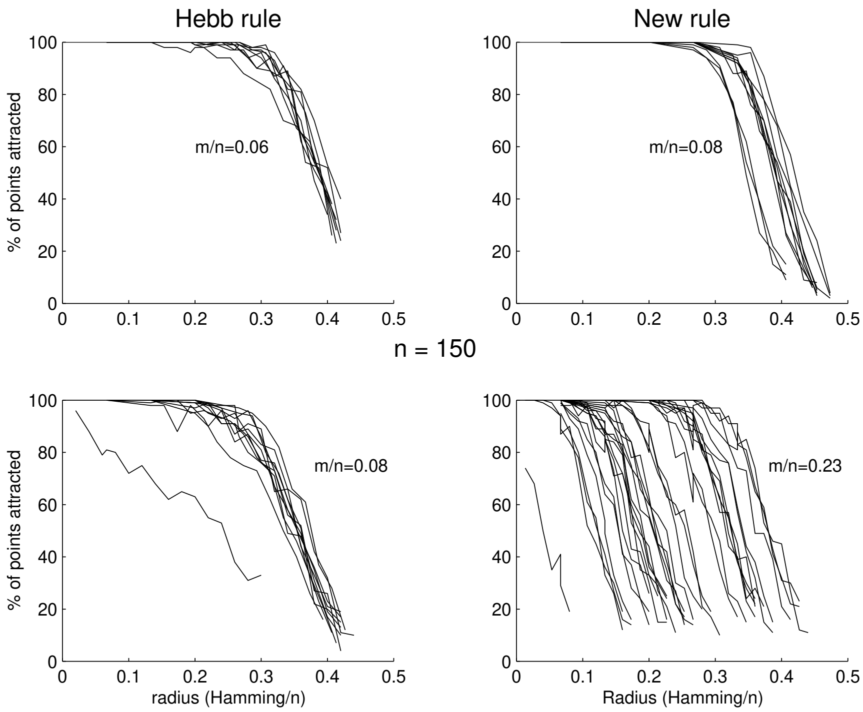


Figure 6.2: $t(r)$ for m stored patterns

greater the variety of distances from which states are attracted to the stored pattern. This implies that the attraction basin is less rounded. Figure 6.3 gives a comparison of the skew for the Hebb rule and the new rule. Note that different loadings (m/n) need to be used for the different rules. This is because the network needs to be near capacity to get any variation in basin size, and the capacity of the new rule is greater than that of the Hebb.

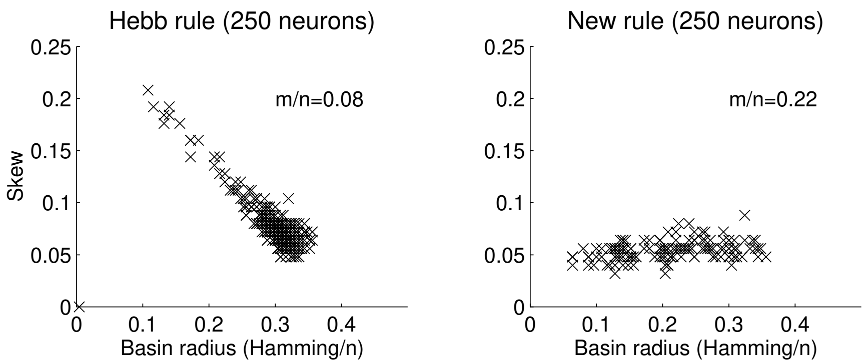


Figure 6.3: Comparing the skew of attraction basins

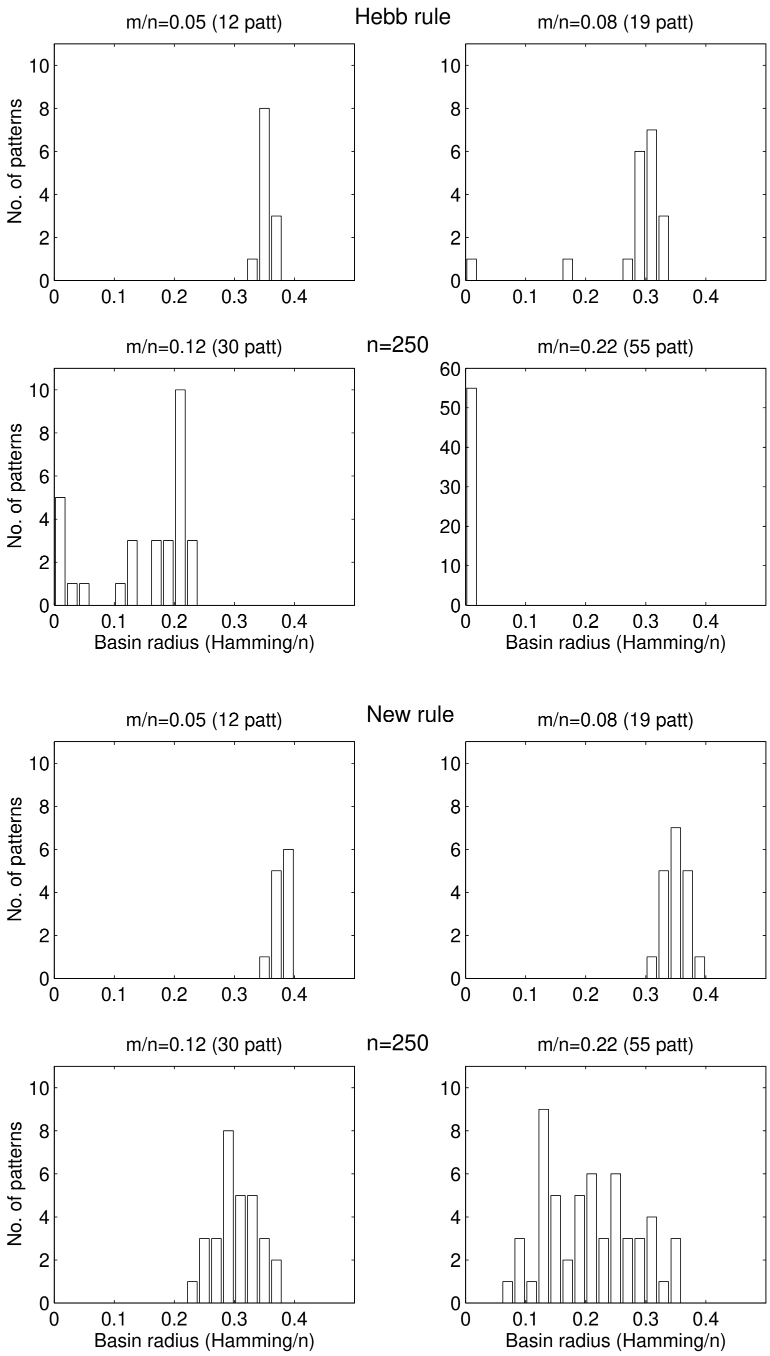


Figure 6.4: Graph shows the distribution of attraction basin sizes across different stored patterns

6.8 Analysis

6.8.1 Basin Sizes

It is clear from figures 6.1 and 6.4 that the attraction basins are generally larger for the new rule than for the Hebb rule. The difference is there, but not highly significant at very low loadings. As the number of stored patterns increases this difference becomes more noticeable. Near the capacity of the new rule, the Hebb rule basin size for most stored patterns has collapsed. On the other hand the new rule still has large basin sizes for the majority of patterns.

6.8.2 Basin distribution and network predictability

Not only do the average radii of the attraction basins differ, but the spread of the distribution of radii is different. For the Hebb rule the distribution of attraction basin radii is peaked at low network loadings. (figure 6.4) As the Hebb rule capacity is approached, one of the attraction basins suddenly begins to loose size. The rest of the attraction basins remain in a peaked distribution. One by one, as the network loading is increased, other attraction basins 'leave the pack' and quickly loose size. At capacity the first attraction basin disappears. Long before the Hebb rule network reaches the capacity loading for the new rule, the attraction basins of the stored patterns have all disappeared.

The behaviour of the new rule is different. It also has a tight distribution of basin sizes for low network loadings. However, as the loading increases, attraction basins do not loose size quickly. Instead the distribution of basin sizes slowly spreads out. Capacity is reached when the lower tail of the distribution of attraction basin sizes touches zero, and one of the stored patterns is no longer an attractor. At capacity, most stored patterns still have substantial attraction basin sizes.

From this we ascertain that the behaviour of the new rule is more predictable than the Hebb rule: the distribution of basin sizes does not change dramatically for small changes in network loadings. Furthermore, the distribution of attraction basin sizes is more even: The distribution is a bell-shaped distribution with a standard deviation that increases slowly as the network loading increases, not a bimodal distribution like that of the Hebb rule near capacity.

6.8.3 Direct attraction basins

Not only are the complete attraction basins larger for the new rule than the Hebb rule, but the direct attraction basins are larger, and decay slower as m increases (see figure 6.1) . Network performance is better within the direct attraction basins, because attraction rates are faster. Network states are attracted to the memory in one synchronous time step. In asynchronous terms this means that the attraction rate is exponential because at each step the probability of moving 1 hamming unit towards the fixed point is proportional to the distance from the current state to the fixed point, and the probability of moving away is zero. Outside the direct attraction basin the attraction rate is generally sub exponential, because the probability of moving away from the fixed point is non-zero.

We note (also from figure 6.1) that the direct attraction basins are not greatly different in size from those of the pseudo-inverse.

6.8.4 Shape of attraction basins

Lastly we consider the shape of the attraction basins. Attraction basins for the Hebb rule are not round but have a convoluted shape [69]. This can create problems with misclassification because initial states which are close to a particular stored pattern are actually attracted to another pattern which is further away. More likely, states are attracted to spurious memories, and convoluted basins mean that some states close to the stored pattern are attracted to spurious states, whereas other states initially further away are attracted to the pattern. The results obtained are therefore highly unpredictable.

Note that in general, for a content addressable memory (CAM), hyper-spherical attraction basins would not be expected. For example if the three patterns A,B and C are such that A and B are close, but A and B are far from C, then the basin of attraction about A would be smaller in the direction of B than in the direction of C.

However, the attraction basins of the Hebb rule attractors are not skew in the above way. This can be shown as follows:

Consider m random patterns of length n, $\xi_i^\mu = \pm 1$ defined as before. Let the distance between any two patterns μ and ν, $d(\mu, \nu)$ be the hamming distance.

The distribution of the hamming distance between any two patterns has mean $n/2$ and variance $n/4$. We want to find the probability that any two patterns are less than $n/2 - \alpha n$ apart or greater than $n/2 + \alpha n$ apart. We need both of these inequalities to account for the fact that the Hopfield model effectively stores both a pattern and its negative.

Proposition 7 *The probability that there is some pair of patterns (length n), say ξ^a and ξ^b, in a set of patterns of size m that can be found with $|d(\xi^a, \xi^b) - n/2| > \alpha n$ is less than*

$$\frac{(1/2 - \alpha)^{(\alpha - 1/2)n} m^2}{2^{n-1}(1/2 + \alpha)^{(\alpha + 1/2)n}}$$

Proof Let A_μ^ν be the event that $|d(\mu, \nu) - n/2| > \alpha n$. Then the event that this is true for any pattern pair

$$A_\mu = \bigcup_{\nu \neq \mu} A_\mu^\nu$$

has probability

$$P(A_\mu) < \sum_{\nu \neq \mu} P(A_\mu^\nu) < m P(A_1^2)$$

using the fact that the distance between any two patterns is identically distributed. Also the event $A = \bigcup_\mu A_\mu$ has probability $P(A) < \sum_\mu P(A_\mu)$ and so $P(A) < m^2 P$ where $P = P(A_1^2)$

By symmetry $P(A_1^{2+}) \stackrel{\text{def}}{=} P[d(1,2) > n/2 + \alpha n] = P[d(1,2) < n/2 - \alpha n]$, and these two events are mutually exclusive. Hence $P = P(A_1^2) = 2P(A_1^{2+})$.

Now

$$\begin{aligned} P = 2P[d(1,2) > n/2 + \alpha n] &< \frac{2\mathbf{E}|\exp[td(1,2)]|}{\exp(\alpha nt + nt/2)} \text{ for all } t \\ &= \frac{2(1+e^t)^n}{2^n e^{\alpha nt} e^{nt/2}} \end{aligned}$$

by a Chebyshev inequality [70, p 186]. We have used the fact that the moment generating function of a binomial$(n, 1/2)$ distribution is $(1+e^t)^n/2^n$. By equating the derivative of P with respect to t with zero, we find that the P is minimised by choosing $t = \ln(1/2 + \alpha) - \ln(1/2 - \alpha)$. Substituting this in and using $P(A) < m^2 P$ we get the required result. □

In our case, for the Hebb rule, we have $n = 250$, $m = 19$. If we take $\alpha = 0.2$, then $P(A) < 9 \times 10^{-7}$. In addition, this probability tends to zero as $n \to \infty$ with $m = o(n)$.

The above shows the probability that the distance between any two patterns is significantly different (more than $0.2n$) from $n/2$ is negligible. This means that the radius of attraction should not decrease below $0.3n$ in any direction, if the variation in the radius is due to the above effect. Furthermore the skew of patterns should not be more than 0.1: With the above effect, the midpoint between two patterns defines the boundary of the basins of attraction to each point (assuming no closer pattern). The distance from a pattern to the midpoint is half the distance between the two points.

The noticed shape of the attraction basins for the Hebb rule cannot be attributed to the above effect. The skew is often larger than 0.1 and the radii of attraction basins is often well below 0.3 in *all* directions (see figures 6.3 and 6.4). We find that the attraction basins for the Hebb rule become more skew as they become smaller. For the new rule, the attraction basins tend to remain quite rounded, whatever their size.

6.8.5 Other details

The use of a fixed sample size of 100 might seem small for this application. However because independent sampling is made every step in radius (of 2 hamming units), the real sample size is actually much larger: results for close radius sizes are expected to be similar. The difference between 5 independent runs of this technique was found to be insignificant.

Experiments were done with different network sizes. The results were similar, although the differences between the Hebb and new learning rules became more marked as the network sizes increased.

6.9 Conclusions

It is already known that the new learning rule has a higher capacity than the Hebb rule, and does not suffer significant capacity loss when patterns with medium correlation are stored.

In this chapter we have established some initial results about the size and shapes of the attraction basins. Analytical methods (similar to those in [66]) have been used to gain knowledge of the sizes and shapes of the basins for independent unbiased storage patterns. The work of this chapter has shown that the new rule has larger, more evenly distributed and more rounded attraction basins than those of the Hebb rule.

This new rule provides a different approach to the problem of increasing capacity and the sizes of attraction basins. Other approaches include the introduction of dynamic or noisy external fields [71, 72, 73], self feedback [74] or the minimum overlap method of [75, 76]. Here, however, we do not only focus on the sizes of the attraction basins, but the shape and distribution as well. The warped attraction basins noticed in [69] are eradicated by this method.

The algorithm we propose achieves its results while maintaining important functionality: unlike the pseudo-inverse rule or minimum overlap rule, it is incremental and local, and it is relatively fast because it does not involve the calculation of an inverse of an m by m matrix.

It should be noted that we have not mentioned the external field issue in this chapter, but just examined the learning rule as it stands. There is no reason that the use of external fields cannot also be applied to this rule to increase attraction basin sizes.

Chapter 7

High capacity storage of time-correlated patterns

7.1 Introduction

[1]In this chapter, the new local and incremental learning rule is examined for its ability to store patterns from a time series in an attractor neural network. It is already known that this learning rule has a higher capacity than the Hebb rule. Here it is shown that it suffers significantly less capacity loss as the correlation between patterns increases.

There are two rules regularly used for training Hopfield networks. The most common of these, the Hebb rule, suffers severe degradation when training patterns are correlated. As a result, practitioners have generally reverted to using the pseudo-inverse rule in such circumstances. However the pseudo-inverse method suffers from significant problems. Firstly, it is not incremental: if a new pattern is to be trained, all the old patterns have to be retrained. Secondly it is not local: the network cannot naturally be trained in a parallel manner, and so is not easily amenable to high speed parallel techniques. Lastly the training method is slow because it involves inverting an $m \times m$ matrix, where m is the number of patterns to be stored.

These problems are most significant when the patterns to be trained are a time series and real time training is needed. The slow speed of the pseudo-inverse, and more importantly its inability to add patterns incrementally, make it of little use for this task. Furthermore time series usually include significant correlations between readings from adjacent time steps. This makes the Hebb rule unsuitable.

Here, we examine the new learning rule introduced in [43] for its effectiveness with time correlated patterns. This learning rule does not suffer from the disadvantages of the pseudo-inverse: it is both local and incremental. It also has a higher capacity than the Hebb rule. We demonstrate that this new learning rule maintains its high capacity with significant correlations in patterns. This contrasts with the Hebb rule, where the capacity reduces much faster as the correlation increases. We show how this can be accounted for

[1]Much of the content of this chapter has been published in [64]

by the relationship between the new rule and the pseudo-inverse.

In this chapter we look at a model of pattern correlations, which resemble the correlations in patterns which form a time series. Many time series are digitisations of continuous events, digitised both in the time and space domains. For example video footage consists of frames of digitised video information. Any two frames of such time series are inevitably correlated, the level of correlation depending on the speed of movement, the sampling rate, and how close in time the two frames are to one another.

7.2 The Hopfield network

Once again we will be looking at the Hopfield network as an attractor neural network which acts as a content addressable memory. The job of the learning rule of a Hopfield network is to find some weight matrix w_{ij} which stores the required patterns as fixed points of the network.

Locality and incrementality have already been discussed in previous chapters. These are both important characteristics of a learning rule. In addition to these a learning rule has a capacity. This is some measure of how many patterns can be stored in a network of a given size. Once again, in this chapter, we consider the absolute capacity [44] of the network. This is given by the asymptotic number of patterns that can be stored without error as a function of the number of neurons, as the network size tends to infinity.

7.2.1 Different learning rules

Three learning rules are used in this chapter. The Hebbian learning rule is local and incremental, but has a low absolute capacity of $n/(2\ln n)$ where n is the number of neurons [44]. This capacity decreases significantly if patterns are correlated.

The pseudo inverse has a capacity of n [45], but does not have the functionality of the Hebb rule. It is not incremental or local, because it involves the calculation of an inverse.

7.2.1.1 The new learning rule

In order to overcome the problems of both of these learning methods we use the new learning rule. In this chapter we use the following simple form

Definition 19 (The new learning rule) *The weight matrix w_{ij} of an attractor neural network is said to follow the new learning rule if it obeys*

$$w_{ij}^0 = 0 \ \forall i,j \ \textit{and} \ w_{ij}^{\nu} = w_{ij}^{\nu-1} + \frac{1}{n}\xi_i^{\nu}\xi_j^{\nu} - \frac{1}{n}\xi_i^{\nu}h_{ji}^{\nu} - \frac{1}{n}h_{ij}^{\nu}\xi_j^{\nu} \tag{7.1}$$

where $h_{ij}^{\mu} = \sum_{k=1,k\neq i,j}^{n} w_{ik}^{\mu-1}\xi_k^{\mu}$ is a form of local field at neuron i (the input to the neuron i) , and ξ^{μ} is the new pattern to be learnt ($\xi_i^{\mu} = \pm 1$)

This rule is local: w_{ij} depends only on information available at the two adjacent neurons (the values of ξ_i, ξ_j, h_i, h_j). It is clear from the recursive nature of (19) that it is also incremental. It has an absolute capacity of $n/\sqrt{2\ln n}$ [43].

7.3 Capacity

The measures of absolute capacity given above, and analysed in earlier chapters, only hold for independent patterns. Here we consider what happens when patterns are correlated. We are specifically interested in the types of correlations which occur in time series. To model this we form the patterns using the following simple Markov chain.

$$P(\xi_i^\mu = \xi_i^{\mu-1}) = \rho;\ P(\xi_i^1 = 1) = \frac{1}{2};\quad \frac{1}{2} \leq \rho < 1 \tag{7.2}$$

where $\mu = 1, 2, \ldots, m$ labels the position of the pattern in the time series and $\xi_i^\mu = \pm 1$ $\forall i, \mu$.

The correlation between patterns ξ_i^μ and $\xi_i^{\mu+t}$ is seen to be

$$\frac{E(\xi_i^\mu \xi_i^{\mu+t})}{(E((\xi_i^\mu)^2)E((\xi_i^\mu)^2))^{1/2}} = [2\rho - 1]^{|t|}$$

by application of the Chapman-Kolmogorov equations. This shows that the correlation between patterns decays as their time separation increases. Henceforth when we use the term correlation, we refer to the value of the correlation between two consecutive patterns: $2\rho - 1$.

The analysis in earlier chapters only deals with the uncorrelated case. In real life situations it is vital that any storage and retrieval method should deal well with correlations between different patterns. Unfortunately the Hebb rule is not very good in such situations. Because the new learning rule is a better approximation to the pseudo inverse, it would be hoped that it would perform better with time-correlated patterns.

Here we compare the absolute capacity of the new learning rule with the Hebb rule as the correlation increases. In all cases 30 sets of patterns ξ^μ of length 500 were chosen according to (7.2). For each set, the patterns were presented to the network one by one until one of the patterns was no longer an attractor (i.e. absolute capacity had been reached). We plot the minimum mean and maximum capacity over the 30 sets.

The above simulations were performed, and the capacity results for the new learning rule were compared with those of the Hebb rule (Figure 7.1). It is clear from this graph that not only does the new rule have a significantly higher capacity for uncorrelated patterns, but it also suffers less from an increase in the correlation. The new rule seems to be unaffected by small correlations, whereas the Hebb rule capacity decays fast as the correlation increases from zero. In addition the new rule still has greater storage ability at correlations of about 0.5 than the Hebb rule does for uncorrelated patterns. The Hebb rule on the other hand is useless for correlations above that of 0.4. Results were very similar for other network sizes.

7.3.1 The relationship between the learning rules

We noted in the previous chapter that the new learning rule can be seen as an approximation to the pseudo-inverse. We can consider the new learning rule to be an intermediate

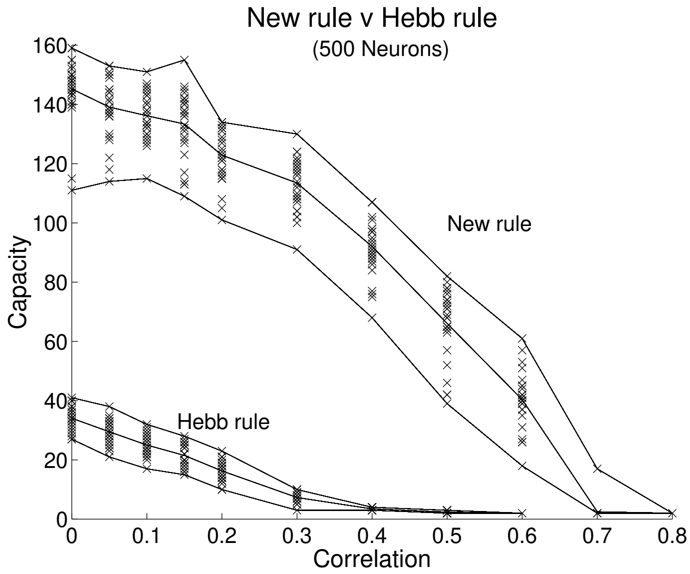


Figure 7.1: Capacity of new learning rule v Hebb (min-mean-max graph)

between the Hebb rule and the pseudo-inverse. It inherits some of the ability of the pseudo-inverse to deal with correlated patterns.

7.4 Conclusions

It is little real life benefit to be able to store independent patterns in an associative network. All meaningful patterns tend to have some form of correlations. Here we have looked at a simple case, and have seen how the Hebb rule deteriorates rapidly with these types of time correlations. The new learning rule still has all the important functionality of the Hebb rule, but has both higher capacity and greater resilience to correlations. It appears that the new learning rule is the only one of the three rules considered which is suitable for real time storage of patterns that form a time series.

Chapter 8

Palimpsest memories

8.1 Outline

[1]Palimpsest[2] or forgetful learning rules for attractor neural networks do not suffer from catastrophic forgetting. Instead they selectively forget older memories in order to store new patterns. Standard palimpsest learning algorithms have a capacity of up to $0.05n$, where n is the size of the network. Here a new learning rule is introduced. This rule is local and incremental. It is shown that it has palimpsest properties, and it has a palimpsest capacity of about $0.25n$, much higher than the capacity of standard palimpsest schemes. It is shown that the algorithm acts as an iterated function sequence on the space of matrices, and this is used to illustrate the performance of the learning rule.

8.2 Introduction

The Hopfield network must have a learning algorithm which sets the connection weights between all pairs of neurons so that it can perform autoassociation. One problem with many such learning rules is that they suffer from catastrophic forgetting. As patterns are presented to the network, it stores them for future retrieval. This continues until the network reaches capacity. Then if patterns continue to be presented the network is quickly unable to retrieve *any* of the stored patterns. In other words at storage levels above capacity the network forgets all it has learnt.

A number of learning rules were developed to get round this problem. They are called palimpsest or forgetful learning rules [77]. These work in a different way. Patterns are stored until capacity is reached. Then as patterns continue to be presented the network forgets the older patterns, preferring to remember the newer ones. The network will continue to remember the most recent p patterns where p is the palimpsest capacity of the network.

The problem with palimpsest memories is that they tend to have very low capacities:

[1]Much of this chapter can be found in [65]

[2]The term *palimpsest* was used by Amit for memories such as these. A palimpsest is a manuscript where earlier writing has been erased to make way for something new.

less than $0.05n$, compared with $0.14n$ for the Hebb rule [78, 79]. Here we introduce a new learning rule which has a palimpsest capacity of estimated to be roughly about $0.25n$. Furthermore this rule has all the important characteristics of a Hopfield learning scheme, including locality and incrementality.

8.3 Learning scheme characteristics

Hopfield learning rules can have a number of characteristics. Firstly a rule can be local. If the update of a particular connection depends only on information available to the neurons on either side of the connection (including weighted information these neurons receive from elsewhere), then the rule is said to be local. Locality is important, because it provides a natural parallelism to the learning rule, which, when combined with the local update dynamics, make a Hopfield network a truly parallel machine.

Secondly a rule can be incremental. If the learning process can modify an old network configuration to memorise a new pattern, without needing to refer to any of the previously learnt patterns, then an algorithm is called incremental. Clearly incrementality makes the Hopfield network adaptive, and therefore more suitable for changing environments or real time situations.

Thirdly a rule can either perform an immediate update of the network configuration, or can be a limit process. The former makes for faster learning.

Fourthly a learning algorithm has a capacity. This is some measure of how many patterns can be stored in a network of a given size. The palimpsest capacity is the number of recent patterns which are retrievable after many patterns are stored in the network.

8.4 Hopfield learning rules

As mentioned earlier, Hopfield networks are usually trained by the Hebb rule

$$w_{ij}^{0} = 0 \ \forall i, j$$

$$w_{ij}^{\mu} = w_{ij}^{\mu-1} + \frac{1}{n}\xi_i^{\mu}\xi_j^{\mu}$$

which has a (relative) capacity of about $0.14n$. It is both local and incremental. However it suffers from catastrophic forgetting. The pseudo-inverse rule is not incremental, and can never store more that n memories. It is of little use in situations where adaptive update occurs.

In chapter 5 a new learning rule was introduced:

$$w_{ij}^{0} = 0 \ \forall i, j \in \{1, 2, ..., n\}$$

$$w_{ij}^{m} = w_{ij}^{m-1} + \frac{1}{n}\xi_i^{m}\xi_j^{m} - \frac{1}{n}\xi_i^{m}h_{ji}^{m} - \frac{1}{n}h_{ij}^{m}\xi_j^{m} \tag{8.1}$$

where

$$h_{ij}^m = \sum_{k=1,k\neq i,j}^{n} w_{ik}^{m-1}\xi_k^m$$

It has a higher capacity than that of the Hebb rule, and is local and incremental. It deals well with correlated patterns [64], and has larger, more even basins of attraction than those obtained by the Hebbian scheme [62]. As it stands it also suffers from a catastrophic loss of memory after capacity is reached.

8.4.1 Palimpsest schemes

There are a number of different palimpsest learning rules, but they have all been based on the same principle: keeping the size of the weight matrix elements bounded. Hopfield suggested a forgetful scheme in his original paper [3]. Others have developed this framework [80, 77, 79, 81, 78, 82, 83]. Palimpsest storage prescriptions are given generally by the local rule

$$w_{ij}^m = \frac{1}{n}\phi(nw_{ij}^{m-1} + \epsilon\xi_i^m\xi_j^m)$$

where $\boldsymbol{\xi}^m$ is the pattern to be stored, ϕ is some function, and n is the size of the network. $w_{ij}^m = w_{ji}^m$ is the weight matrix after the mth pattern is stored.

Parisi formalised Hopfield's original proposal by choosing the function

$$\phi(x) = \mathrm{sgn}(x)\min(1, |x|)$$

Nadal, Toulouse, Changeux and Dehaene [77] examined a number of learning methods including what they called the marginalist scheme ($\phi(x) = \lambda_N x$) and the smooth scheme ($\phi(x) = \tanh(x)$). The largest palimpsest capacity obtained for such schemes is about $0.05n$.

Here we modify the learning rule of [43] very slightly to obtain a palimpsest rule with a capacity greater than $0.25n$, many times that of other palimpsest schemes, and larger than the (non-palimpsest) capacity of the Hebb rule:

$$w_{ij}^0 = 0 \ \forall i,j \in \{1,2,...,n\}$$

$$w_{ij}^m = \begin{cases} w_{ij}^{m-1} + \frac{1}{n}\xi_i^m\xi_j^m - \frac{1}{n}\xi_i^m h_j^m - \frac{1}{n}h_i^m\xi_j^m & \text{for } i \neq j \\ 0 \text{ for } i = j \end{cases} \tag{8.2}$$

where

$$h_i^m = \sum_{k=1}^{n} w_{ik}^{m-1}\xi_k^m$$

8.5 Results

Simulations of this network were performed. Random independent zero mean ± 1 patterns were presented to a network of 400 neurons. The palimpsest storage is plotted against number of patterns presented

Definition 20 (Relative/absolute palimpsest storage) *Store m patterns in the network. For each pattern, starting with the most recent and moving back through time, check whether it is recalled with an error within a given tolerance. Stop when one pattern fails this test. The total number of pattern recalled within the tolerance is called the palimpsest storage for m patterns. For absolute palimpsest storage, this tolerance level is zero. For relative palimpsest storage, the tolerance level is small, but non-zero.*

Here we make one assumption to speed up the computation of the relative capacity. We assume that if the stored pattern has only a small number of unstable bits, then it is within the direct attraction basin of some stable point. This assumption is often fair. It fails when flipping one bit of the pattern induces instability in a whole number of other bits which were previously stable. But the affect of a single bit change is usually relatively small [62].

The benefit of this assumption is that we can test the number of unstable neurons when the stored pattern is presented, rather than searching for network fixed points. We take a tolerance level of 5 percent, So 95 percent of the bits of the nearest fixed point must be correct.

Figure 8.1 gives the palimpsest storage at different memory loadings. The network recalls all patterns until capacity is reached. The storage level then decreases, tending to the palimpsest capacity of the network. These results were obtained by storing random patterns, one by one, in a network of size $n = 400$. After each new pattern is stored, the palimpsest capacity is given by the size of the largest contiguous block of patterns (including the pattern just presented) which are all fixed points of the network.

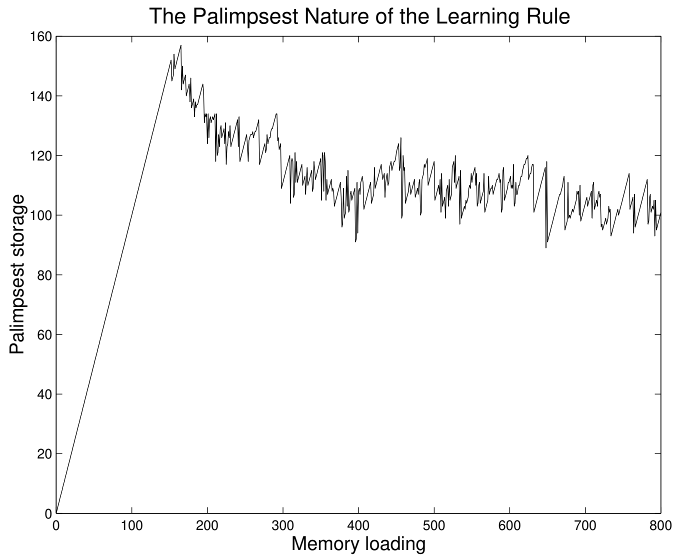


Figure 8.1: The palimpsest storage at different memory loadings for a network size $n = 400$

In order to find the palimpsest capacity, the palimpsest storage is averaged for a number of high memory loadings for different network sizes. Figure 8.2 plots the palimpsest capacity as the number of neurons in the network increases. The straight line on the graph

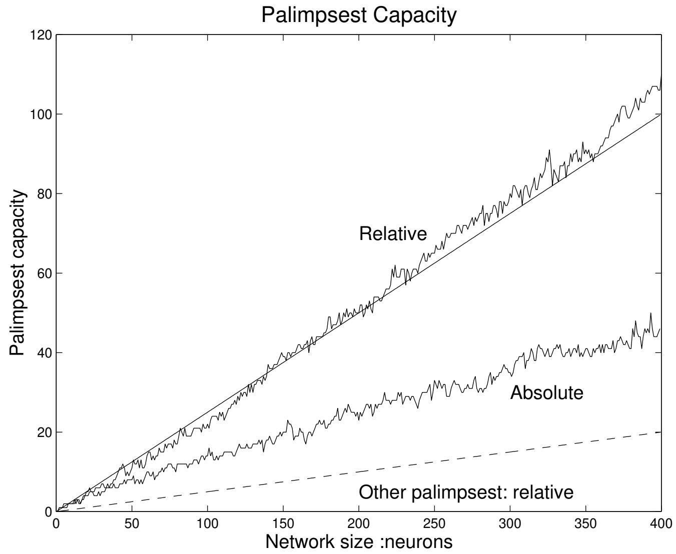


Figure 8.2: The palimpsest storage at different memory loadings

corresponds to $0.25n$, and gives a lower bound to the capacity scaling of the network. The dotted line gives the $0.05n$ capacity of the standard palimpsest rules.

Because of the assumptions made in calculating the relative capacities, a plot of the absolute palimpsest capacity is also given on the same graph. This gives a definite lower bound for the relative capacity, as well as being interesting in its own right.

8.6 How the learning rule works

The recursive nature of the learning rule makes it hard to analyse, but it is possible to illustrate why the learning rule acts as a palimpsest. We demonstrate that it acts as an iterated function sequence (IFS) with probabilities.

Definition 21 (IFS with probabilities) *An IFS with probabilities, denoted by*

$$IFS(X, d; f_1, f_2, \dots, f_r; p_1, p_2, \dots, p_r)$$

consists of a complete metric space (X,d) together with a finite set of contraction mappings $\{f_1, f_2, \dots, f_r\}$ *and a set of probabilities* $\{p_1, p_2, \dots, p_r\}$ *where* p_i *is the probability of choosing transformation* f_i.

IFSs are important because they have a unique invariant fractal measure. The importance of an invariant fractal distribution for palimpsest rules is elaborated in [78] and [83].

The learning scheme (8.2) is now reformulated in IFS terminology. Let X be the space of weight (i.e. zero diagonal symmetric) matrices, with the Frobenius metric

$$d(W^1, W^2) = \sum_{i,j\neq i}^{n} (w_{ij}^2 - w_{ij}^1)^2 \tag{8.3}$$

where for later convenience we make the fact that the weight matrix has zero diagonal explicit in the summation.

Define $f_{\boldsymbol{\xi},\boldsymbol{\xi}'}(W)$ by the action of the update rule (8.2) on W as two distinct patterns $\boldsymbol{\xi}, \boldsymbol{\xi}'$ are stored consecutively. Let $p(\boldsymbol{\xi}, \boldsymbol{\xi}') = 1/(2^n(2^n-1))$ for all possible distinct $\boldsymbol{\xi}$ and $\boldsymbol{\xi}'$.

Proposition 8 $(X, d, \{f(\boldsymbol{\xi}, \boldsymbol{\xi}')\}; \{p(\boldsymbol{\xi}, \boldsymbol{\xi}')\})$ *is an IFS with probabilities, and defines the learning rule (8.2) where identical pattern pairs are disallowed.*

Proof Consider two possible weight matrices W^1 and W^2. The Frobenius distance between these two matrices is given by (8.3).

Now consider what happens to this distance as the two matrices are updated on the arrival of pattern $\boldsymbol{\xi}$. Let W^{1new} and W^{2new} be these updated weight matrices. Then

$$\sum_{i,j\neq i}^{n} (w_{ij}^{2new} - w_{ij}^{1new})^2 = \sum_{i,j\neq i}^{n} (w_{ij}^2 - w_{ij}^1)^2 \tag{8.4}$$

$$+ \quad \frac{1}{n^2} \sum_{i,j\neq i}^{n} \left[\sum_{k\neq i}^{n} (w_{ik}^2 - w_{ik}^1)\xi_k\xi_j + \sum_{k\neq i}^{n} \xi_i\xi_k(w_{jk}^2 - w_{jk}^1) \right]^2 \tag{8.5}$$

$$- \quad \frac{2}{n} \sum_{i,j\neq i}^{n} (w_{ij}^2 - w_{ij}^1) \left[\sum_{k\neq i}^{n} (w_{ik}^2 - w_{ik}^1)\xi_k\xi_j + \sum_{k\neq j} \xi_i\xi_k(w_{kj}^2 - w_{kj}^1) \right] \tag{8.6}$$

The part of this equation labelled (8.5) simplifies to

$$\frac{2(n-1)}{n^2} \sum_{i}^{n} \left[\sum_{k\neq i}^{n} (w_{ik}^2 - w_{ik}^1)\xi_k \right]^2 + \frac{2}{n^2} \left[\sum_{i,k\neq i}^{n} \xi_i (w_{ik}^2 - w_{ik}^1)\xi_k \right]^2 \tag{8.7}$$

where we use $\sum_{j\neq i} \xi_j\xi_j = n-1$. Clearly the second part of the RHS of (8.7) is smaller than the first part by a form of triangle inequality (the minimum hypotenuse length for fixed total length of the other sides is for an isosceles triangle), and both parts are positive (they are sums of square terms). Hence (8.5) is less than

$$\frac{4(n-1)}{n^2} \sum_{i}^{n} \left[\sum_{k\neq i}^{n} (w_{ik}^2 - w_{ik}^1)\xi_k \right]^2$$

On the other hand the part-equation labelled (8.6) simplifies to

$$-\frac{4}{n} \sum_{i}^{n} \left[\sum_{k\neq i}^{n} (w_{ik}^2 - w_{ik}^1)\xi_k \right]^2$$

Putting all this together we get

$$\sum_{i,j\neq i}^{n}(w_{ij}^{2new}-w_{ij}^{1new})^2<\sum_{i,j\neq i}^{n}(w_{ij}^2-w_{ij}^1)^2-\frac{4}{n^2}\sum_{i}^{n}\left[\sum_{k\neq i}^{n}(w_{ik}^2-w_{ik}^1)\xi_k\right]^2$$

Now we see that on the whole the distance between two weight matrices is reduced by putting them through the update equation. The only exception to this is if each row of $w_{ij}^2-w_{ij}^1$ lies in a hyperplane perpendicular to the new pattern ξ. For now, let us constrain $w_{ij}^2-w_{ij}^1$ to have some row a for which

$$\sum_{j}(w_{aj}^2-w_{aj}^1)\xi_j\geq\alpha\sum_{i,j\neq i}^{n}(w_{ij}^2-w_{ij}^1)^2 \tag{8.8}$$

for α small and positive. This excludes matrices for which all row vectors are within a small angle of the aforementioned hyperplane. Then

$$\sum_{i,j\neq i}^{n}(w_{ij}^{2new}-w_{ij}^{1new})^2\leq(1-\alpha)\sum_{i,j\neq i}^{n}(w_{ij}^2-w_{ij}^1)^2 \tag{8.9}$$

If on the other hand the weight matrix difference does not satisfy the constraint (8.8) for this ξ, then it will undergo only a small perturbation when updated by (8.2), and will certainly satisfy the constraint for a new and different pattern $\boldsymbol{\xi}'$ which arrives next. Hence after the weight matrices have been updated by two different incoming patterns, equation 8.9 is satisfied for all w_{ij}^1 and all w_{ij}^2. Therefore we have a contraction mapping on the space of weight matrices. This enables us to define an IFS with probabilities.

It is easiest if we restrict the incoming patterns so that each pair of incoming patterns cannot be identical (or opposite). This constraint is negligible for large network sizes. Then for each pattern pair $s=(\boldsymbol{\xi},\boldsymbol{\xi}')$, we have a contraction mapping, denoted ϕ_s, and a probability $p_s=1/(2^n(2^n-1))$, and so we have an IFS with probabilities $(\phi_1,\phi_2,\ldots;p_1,p_2,\ldots)$ on the space of matrices with a Frobenius metric.

The contraction mapping is given by the application of the learning rule, and the probabilities are the probabilities of choosing a given pattern pair for zero mean ±1 patterns chosen without replacement. Hence this IFS is isomorphic to the learning rule where identical pattern pairs are disallowed. □ The constraint on pattern pairs is negligible for large networks, and in fact an identical second pattern will serve only to reinforce that same pattern. Hence we have shown that the learning rule acts as an IFS.

Consider two points W^1, W^2 in weight space chosen with respect to the invariant measure of the learning rule. This corresponds to choosing two points with different histories. The distance between these two weight matrices measures the difference in the recall performance of the Hopfield networks corresponding to each weight matrix.

Now the learning rule for any given pattern acts as a contraction mapping in the space of weight matrices. Hence as each weight matrix is updated with incoming patterns, W^1 and W^2 are mapped to points closer to one another. In other words the effect of the different histories is reduced as new training patterns arrive: the learning rule enables gradual forgetting of the past, favoring more recent patterns. This is an indication of the palimpsest effect of the learning rule. The Hebb rule and other non-palimpsest rules do not act as contraction mappings. As a result the effect of the old patterns does not decay. On the other hand this (and other palimpsest rules) do act as IFS's.

The above result also helps elucidate why the learning rule has such a high capacity. Furthermore the decay rate for each row of the weight matrix depends on its dot product with the training pattern. Hence each row vector is pushed towards the hyperplane perpendicular to the training vector before the Hebbian term is added. This reduces the interference of old patterns with the recall of the new pattern (recall depends on the same dot product). One consequence of this is that there is no reduction in the information stored in the components of the matrix rows which are perpendicular to the training pattern. This contrasts with standard palimpsest rules where exponential decay occurs to all elements of the weight matrix indiscriminately.

8.7 Conclusion

The new learning rule introduced in this chapter provides a new approach to producing palimpsest learning schemes. Instead of forcing the weight matrix to exhibit exponential decay, the patterns are stored in the weight matrix in the normal way. However at each stage the components of the weight matrix which interfere with the recall of that pattern are significantly reduced. Hence old memories are not forced to decay if it is not necessary.

This effect can be seen by recognising that the learning rule acts as an iterated function system (IFS) on the space of weight matrices. The IFS definition consists of 2^n contraction mappings: one for each configuration of an input pattern. Because the rule acts as an IFS, we know that the strength of the effect of past patterns is reduced to make way for new ones.

The result of this is a palimpsest memories with capacities larger than standard non-palimpsest learning schemes such as the Hebb rule. This improvement is at least a factor of five improvement over previous palimpsest rules.

Chapter 9

Conclusions

This thesis has outlined certain developments and improvements in Gaussian process modelling techniques, and in Hopfield model learning algorithms.

9.1 Hopfield networks

Hopfield networks have been interesting to researchers because of their interesting mathematical structure [47], their highly parallel nature, the possibility that their form might provide an analysable but idealised analogue to parts of the brain [2], and the possibility of at least some practical use [3]. Locality and incrementality are vital prerequisites for both parallelism and biological plausibility, and increase the areas where Hopfield networks might be of practical use.

9.1.1 The importance of locality and incrementality

Unfortunately one of the best algorithms for training Hopfield networks, the pseudo-inverse rule, is neither local nor incremental. It is possible to recast the pseudo-inverse in an incremental form (5.36), or a local form [46], but not both. The emphasis in this thesis is on finding and analysing alternative learning rules for Hopfield networks with local and incremental forms. In addition the algorithms should be reasonably simple, and allow fast training. Ideally it should also be possible to analyse some elements of the learning rule analytically.

9.1.2 The new learning rules

In this thesis, a new Hopfield learning rule, and some variants of it, are introduced. These learning rules are all local and incremental. They are related to the pseudo-inverse in that they can be seen to be approximations of it, and are significant improvements over the standard Hebb rule in many ways.

The improvements the new learning rules make are analysed one by one, and the structure and nature of the new learning rules are investigated. In some cases analytic proofs are given about the characteristics of the algorithms, in other cases simulations and sampling methods are used to assess the properties that these learning rules possess.

9.1.3 Capacity issues

The first and most significant characteristic of a Hopfield learning rule is its capacity. The capacity of the Hopfield network has been the subject of large amounts of study for a number of different algorithms. The techniques used in these studies varies from replica spin glass methods [47], simulations [66], heuristic methods, which usually make some distributional or independence assumptions [69], limit distributional analysis [44], use of the Markov inequality and independence structure [60, 61]. Some of these methods, notably replica approaches, have been used for a variety of learning rules. Others, notably Markov inequality approaches and limit distributional analysis, are only immediately applicable to the Hebb rule. These methods also involve slightly different definitions of capacity, depending on the definition amenable to that particular approach.

In this thesis an analytic proof of the capacity of the new learning rule is given, which shows that the absolute capacity of the Hopfield network increases significantly when moving from the use of the Hebb rule to the new rule. This proof does not make assumptions about the distribution of the 'noise' term in what has become known as the signal to noise method of analysing capacity ([44] notes this problem). Neither does it make assumptions about the independence of the noise terms for different neurons and different patterns. It is most similar in approach to the methods used in [60], but differs in the form of Markov inequality used, and in the fact that for the new learning rule, the system cannot be reduced to various independent elements. As a result the analysis is significantly harder.

The methods employed involve the reduction of the problem to a set of combinatorial pairing problems, each of which which is a form of domino grid pairing. Solution of this gives the value of the absolute capacity of the new learning rule to be $n/\sqrt{2\ln n}$ for a network size n, which for $n = 3000$ say, makes the absolute capacity of the new rule over four times that of the Hebb rule.

Simulations of the Hopfield network were performed, which verified that the theoretical capacity measures given were realistic even for small network sizes (the theoretical measure is asymptotically valid for network sizes tending to infinity).

A variant of the new learning rule was also introduced, which is also both local and incremental. It was shown that this learning rule is an approximation to the incremental form of pseudo-inverse, where the non-local terms are taken to be constant. This approximation is good for low memory loadings, but becomes worse as the memory loading increases. As a result, the capacity is not as high as that of the pseudo-inverse. Simulations show that this variant has a higher capacity than the new rule, at a marginal increase in computational cost.

9.1.4 Basins of attraction

The basins of attraction determine the number of noisy patterns which will be correctly classified. This issue is complicated by the fact that basins of attraction may differ from one pattern to another, and can take a variety of different shapes. In this thesis both the direct and indirect attraction basins are analysed for the different learning rules. The distribution of sizes, and the shapes of the attraction basins are examined.

Certain types of analysis were inappropriate for this task. The information theoretic approach in [63] made independence assumptions about the neural outputs which are in general untrue, and as a result did not leave room for analysis of the shapes of the attraction basins. The sampling approach taken in [66] was favoured, in that it could give, reasonably straightforwardly, a number of different characteristics of the attraction basins all at once. The sampling methods were extended to look at not only the size of the attraction basins, but the distribution and shape of them too.

The analysis found that the new learning rule has larger direct and indirect attraction basins than the Hebb rule. Furthermore the behaviour of the attraction basin size for the new rule is not as erratic as is the case for the Hebb rule. With the Hebb rule certain attraction basins collapse unpredictably as the network capacity is approached. With the new rule, the attraction basin sizes are unimodally distributed, and the variance of that distribution gradually increases.

The new learning rule differs from the Hebb rule in another area: that of attraction basin shape. In this thesis it has been shown that the attraction basins for the Hebb rule is Skew. This mirrors the findings about basin shapes in [69]. On the other hand, the new rule has more rounded attraction basins than those of the Hebb rule, especially when near capacity. This increases the predictability of the network.

9.1.5 Correlated patterns

The standard capacity proofs for Hopfield networks tend to assume that both the bits of each pattern, and the patterns themselves are independent. This is an idealisation which makes the networks more analysable, but is not often practically true. Here the new learning rule has been assessed on its ability to store certain types of correlated patterns. More specifically patterns were ordered, and nearby patterns were assumed more correlated than patterns which are far apart. This is a common type of correlation, and incidentally is the type also assumed in much of the Gaussian process modelling section.

It was found that the new learning rule coped well with correlations between patterns. It certainly did not suffer the significant loss of capacity that the Hebb rule did in the same circumstances.

9.1.6 Palimpsest rules

Lastly there is the issue of catastrophic forgetting. The Hebb rule suffers when attempts are made to storage a number of patterns which is larger than the capacity. In these situations none of the patterns will be recalled correctly. All will be forgotten [84]. The alternatives to this involve the selective forgetting of earlier patterns in order to keep later ones. Such algorithms are called ‘forgetful’ rules or ‘palimpsest’ rules. Most of these types of rules work by exponentially decaying the effect of the older patterns [81, 79] or by bounding the synaptic weights [85]. The spin-type capacities of such learning rules are small, generally about $0.02n$ [83, 77]

Here it has been shown that a simple modification to the new learning rule acts as a high capacity palimpsest rule. The reason for this high capacity is that no simple exponential decay rule is followed, and hence components of a given pattern are not automatically reduced, whether this is necessary or not.

The way the learning rule works has been illustrated using iterated function system (IFS) theory. The learning rule acts as an IFS in the space of weight matrices. This approach has been used in the past for the standard forgetful memories [78], but there he contraction mappings can be defined on the single neuron space, ie the real line. Because the new learning rule is recursive, the learning rule does not decay each neuron independently, and so no such simple contraction mappings can be formed. Instead, contraction mappings on the space of complete weight matrices are used.

The working of an IFS is characteristic of forgetful learning rules. For example because an IFS has a stationary distribution, it can be seen that the effective embedding of the older patterns in the weight matrix decays.

Simulations of the network trained with the learning rule show palimpsest activity, and give estimates of a palimpsest capacity of about $0.25n$, five times that of standard palimpsest rules.

9.2 Gaussian processes

9.2.1 Switching between regimes

Research into the use of Gaussian process priors has shown that Gaussian processes are a competitive tool for regression and classification [17]. However the Gaussianity requirements of the joint sample point distribution can be restrictive in a number of circumstances.

Gaussian process methods rely on Gaussian distributed noise to maintain the Gaussianity of the final prior and to provide an analytic solution. However, here was been noted that certain types of non-Gaussian noise can be represented as a mixture of Gaussians. This leads to a mixture of Gaussian processes model. These models can be extended beyond the t-distributed noise used in [30] to other distributions. The possibilities were extended further by incorporating a variable mean term into the noise formulation. The use of these techniques for a mixture of digital and Gaussian noise was demonstrated. When these methods are combined with other suitable apriori transformations, or transformation mixtures the possible functionality and scope of Gaussian process methods is significantly increased.

Many interesting practical situations concern signals which consist of a number of different regimes. Given the proven ability of Gaussian process techniques, it is useful to develop Gaussian process methods for dealing with these situations. Such a development has been made here. A switching process latent variable model was proposed where the latent variables represent which of a number of independent sources the current signal comes from. This is a similar formation to that in [33] among others, except that for Gaussian processes, a continuous system is needed, rather than one relating to a time series.

Developing extensions of Gaussian process techniques for switching models involves finding suitable underlying covariance matrices for the relevant process. The model which has been developed here involves the use of a block covariance structure, where the covariance between points in different regimes is zero. This corresponds to the different regimes being independent. Within a regime, the covariance can be of whatever form the modeller chooses.

Which point is in which regime is not known. Hence a hierarchical Bayesian model was developed with latent variables used to represent which regime a data point comes from. Because such variables are unknown the correct Bayesian approach involves marginalisation. This is not tractable, and so was approximated using Gibbs sampling techniques.

In this thesis methods are developed for efficiently sampling from the latent variable space. This involved the use of the partitioned inverse equations to update the inverse covariance matrix for well chosen 'walks' around the latent space hypercube.

The methods are tested on a toy problem, from which reasonable results were obtained.

9.2.2 Toeplitz methods

Gaussian process inference methods suffer from scaling problems. This is because their implementation involves the inverse of an $n \times n$ matrix where n is the size of the training set. In this thesis a method of improving this scaling is given for systems where the sample points can be controlled, or are in some regular form. This method utilises the Toeplitz structure of the covariance matrices for these systems.

Use of these methods for grid based systems relies on the use of truncated covariances. These are covariance functions which are zero outside a certain radius. This enables the space to be warped onto a cylinder, a space for which Toeplitz matrices can be defined. These methods were tested on a toy problem.

Truncated covariances are a useful tool in Gaussian process modelling, because they remove many of the near zero matrix terms which are seen using standard covariance functions. Because of this the resulting matrices are sparser, and hence are faster to invert.

9.3 Final remarks

This thesis has tackled a number of diverse problems, all of which relied on the characterisation of inverse covariance matrices. Techniques for localising the changes to such matrices, using the partitioned inverse equations, has enabled sampling methods for latent variable Gaussian process models to be speeded up. They have allowed the extended new Hopfield learning rule to be developed from an incremental form of the pseudo-inverse method. Toeplitz covariance matrix structures have been utilised to speed up covariance matrix inversion, and results on the eigenvalue spectrum of sample covariances have aided comparisons between the pseudo-inverse, the Hebb rule and the new learning rule.

Each of the models considered in this thesis is interesting in its own right. Each has been developed further. The new learning rule and its variants have a firm grounding in theory. Understanding of their behaviour has been developed. Gaussian process methods have been extended for use in new situations, with increased efficiency. They both benefit from covariance-based approaches.

Bibliography

[1] R. M. Neal. Priors for infinite networks. Technical report, Department of Computer Science, University of Toronto, 1994.

[2] D. J. Amit. *Modeling Brain Function.* Cambride University Press, 1989.

[3] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America: Biological Sciences*, 79(8):2554–2558, 1982.

[4] J. J. Hopfield and D. W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[5] C. Mead. *Analog VLSI and Neural Systems.* Addison Wesley, 1989.

[6] M. Yasunga, N. Msuda, M. Yagyu, M. Asai, M. Yamada, and A. Masaki. Design, fabrication and evaluation of a 5-inch wafer scale neural network lsi composed of 576 digital neurons. In *Proceedings of the International Joint Conference on Neural Networks*, pages 527–535, 1990.

[7] T. F. Pawlicki, D. S. Lee, J. J. Hull, and S. N. Sihari. Neural network models and their application to handwritten digit recognition. In *Proceedings of the IEEE International Conference on Neural Networks*, volume II, pages 63–70, 1988.

[8] K. Funahashi. On the approximate realisation of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[9] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netwokrs*, 2:359–366, 1989.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[11] C. Williams. Regression with Gaussian processes. In S. W. Ellacott, J. C. Mason, and I. J.Anderson, editors, *Mathematics of Neural Networks: Models, Algorithms and Applications.* Kluwer, 1995. Published 1997.

[12] A. O'Hagan. On curve fitting and optimal design for regression. *Journal of the Royal Statistical Society B*, 40:1–42, 1978.

[13] D. Barber and C. K. I. Williams. Gaussian processes for Bayesian classification via hybrid Monte Carlo. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9.* MIT Press, 1997.

[14] M. N. Gibbs and D. J. C. MacKay. Variational Gaussian process classifiers. Technical report, Department of Physics, University of Cambridge, May 1997.

[15] G. Wahba. Improper priors, spline smoothing and the problem of guarding against model errors in regression. *Journal of the Royal Statistical Society*, 6:1–18, 1978.

[16] G. Wahba. *Spline Models for Observational Data.* SIAM, 1990.

[17] C. E. Rasmussen. *Evaluation of Gaussian Processes and other Methods for Non-Linear Regression.* PhD thesis, University of Toronto., 1996.

[18] M. N. Gibbs and D. J. C. MacKay. Efficient implementation of Gaussian processes. Preprint, 1997.

[19] D. Barber and C. K. I. Williams. Regression with input-dependent noise: A gaussian process treatment. In M. I. Jordan, M. J. Kearns, and S. A. Solla., editors, *Advances in Neural Information Processing Systems 10.* MIT Press, 1998.

[20] P. M. Williams. Bayesian regularization and pruning using a laplace prior. *Neural Computation*, 7:117–143, 1995. Published 1997.

[21] D. J. C. MacKay. A practical bayesian framework for back-propagation networks. *Neural Computation*, 4:448–472, 1992.

[22] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison Wesley, 1991.

[23] D. J. C. Mackay. Hyperparameters: Optimise or integrate out? In G. Heidbreder, editor, *Maximum Entropy and Bayesian Methods, Santa Barbara 1993.* Kluwer, 1994.

[24] D. H. Wolpert. On the use of evidence in neural networks. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 539–546. Morgan Kaufmann, 1993.

[25] R. M. Neal. *Lecture Notes in Statistics 118: Bayesian Learning for Neural Networks.* Springer Verlag, 1996.

[26] C. M. Bishop. *Neural Networks fro Pattern Recognition.* Oxford University Press, 1995.

[27] S. Barnett. *Matrix Methods for Engineers and Scientists.* McGraw-Hill, 1979.

[28] A. J. Storkey. Gaussian processes for switching regimes. In L. Niklasson, M. Boden, and T. Ziemke, editors, *ICANN98*, pages 621–626, 1998.

[29] D. J. C. Mackay. Introduction to Monte Carlo methods. Review paper at Erice Summer School, 1999.

[30] R. M. Neal. Monte Carlo implementation of Gaussian process models for Bayesian regression and classification. Technical report, Department of Statistics, University of Toronto, Jan 1997.

[31] M. Loeve. *Probability Theory.* Van Nostrand, 1963.

[32] R. M. Neal and G. E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 353–358. Kluwer Academic Publishers, 1998.

[33] Z. Ghahramani and G. E. Hinton. Switching state space models. To be published in Neural Computation.

[34] R. H. Swendsen and J-S. Wang. Non-universal critical dynamics in monte-carlo simulations. *Physical Review Letters*, 58:86–88, 1987.

[35] C. K. I. Williams and D. Barber. Bayesian classification with gaussian processes. submitted to IEEE PAMI.

[36] M. Hodgson and P. J. Green. Investigating markov model discrimination for ion channels. In *Proceedings of the Royal Society of London Series A*, volume 455, pages 3425–3448, 1998.

[37] J. Skilling. Bayesian numerical analysis. In P. Milonni, editor, *Physics and Probability.* Cambridge University Press, 1993.

[38] W. F. Trench. An algorithm for the inversion of finite toeplitz matrices. *Journal of SIAM*, 12:515–522, 1964.

[39] G. H. Golub and C. F. Van Loan. *Matrix Computations.* John Hopkins, third edition, 1996.

[40] Chan R, J. Nagy, and R. Plemmons. Circulant preconditioned toeplitz least squares iterations. *SIAM Journal of Matrix Analysis and Applications*, 15:80–87, 1994.

[41] R. Chan and M. Ng. Conjugate gradient methods for solving toeplitz systems. *SIAM Review*, 38:427–482, 1996.

[42] D. J. C. Mackay. Gaussian processes - a replacement for supervised neural networks? In *NIPS97 Tutorial*, 1997.

[43] A. J. Storkey. Increasing the capacity of the Hopfield network without sacrificing functionality. In W. Gerstner, A. Germond, M. Hastler, and J Nicoud, editors, *ICANN97: Lecture Notes in Computer Science 1327*, pages 451–456. Springer-Verlag, 1997.

[44] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh. The capacity of the Hopfield associative memory. *IEEE Transactions on Information Theory*, 33(4):461–482, 1987.

[45] I. Kanter and H. Sompolinsky. Associative recall of memory without errors. *Physical Review A: General Physics*, 35(1):380–392, 1987.

[46] V. S. Dotsenko, N. D. Yarunin, and E. A. Dorotheyev. Statistical mechanics of Hopfield-like neural networks with modified interactions. *Journal of Physics A: Mathematical General*, 24(10):2419–2429, 1991.

[47] D. J. Amit, H. Gutfreund, and H. Sompolinsky. Statistical mechanics of neural networks near saturation. *Annals of Physics*, 173(1):30–67, 1987.

[48] W. S. McCulloch and W. A. Pitts. A logical calculus of the ideas immanent in neural networks. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[49] S. R. Kelso and T. H. Brown. Differential conditioning of associative synaptic enhancement in hippocampal brain slices. *Science*, 232:85–87, 1986.

[50] B. G. Cragg and H. N. V. Temperley. The organisation of neurons: A cooperative analogy. *EEG and Clinical Neurophysiology*, 6:85–92, 1954.

[51] W. A. Little. The existence of persistent states in the brain. *Mathematical Biosciences*, 19:101–120, 1974.

[52] L. Personnaz, I. Guyon, and G. Dreyfus. Information storage and retrieval in spin-glass-like neural networks. *Journal de Physique Lettres*, 46:359–365, 1985.

[53] L. Personnaz, I. Guyon, and G. Dreyfus. Collective computational properties of neural networks: New learning algorithms. *Physical Review A*, 34:4217–4228, 1986.

[54] J. F. Fontanari and W. K. Theumann. On the storage of correlated patterns in Hopfields model. *Journal de Physique*, 51:375–386, 1990.

[55] J. W. Silversein. Eigenvalues and eigenvectors of large dimensional sample covariance matrices. *Contemporary Mathematics*, 50:153–159, 1986.

[56] V. A. Marcenko and L. A. Pastur. Distribution of eigenvalues for some sets of random matrices. *Mathematics of the USSR. Sbornik*, 1:457–483, 1967.

[57] U. Grenander and J.W. Silverstein. Spectral analysis of networks with random topologies. *SIAM Journal of Applied Mathematics*, 32:499–519, 1977.

[58] K. W. Wachter. The storng limits of random matrix spectra for sample matrices of independent elements. *Annals of Probability*, 6:1–18, 1978.

[59] R. M. Everson and S. J. Roberts. Inferring the eigenvalue spectrum of covariance matrices from limited, noisy data. Submitted to IEEE Transactions in Signal Processing., June 1998.

[60] A. Bovier and V. Gayrard. Rigorous bounds on the storage capacity of the dilute hopfield model. *Journal of Statistical Physics*, 69:597–627, 1992.

[61] C. M. Newman. Memory capacity in neural network models:rigourous lower bounds. *Neural Networks*, 1:223–238, 1988.

[62] A. J. Storkey and R. Valabregue. The basins of attraction of a new Hopfield learning rule. *Neural Networks*, 12(6):869–876, 1999.

[63] F. Schwenker, F. T. Sommer, and G. Palm. Iterative retrieval of sparsely coded associative memory patterns. *Neural Networks*, 9:445–455, 1996.

[64] A. J. Storkey and R. Valabregue. A new Hopfield learning rule with high capacity storage of correlated patterns. *Electronics Letters*, 33(21):1803–1804, 1997.

[65] A. J. Storkey. Palimpsest memories: a new high-capacity forgetful learning rule for Hopfield networks. Submitted to Physical Review E, 1998.

[66] B. M. Forrest. Content-addressability and learning in neural networks. *Journal of Physics A: Mathematical and General*, 21:245–255, 1988.

[67] H. Horner, D. Bormann, M. Frick, H. Kinzelbach, and A. Schmidt. Transients and basins of attraction in neural network models. *Zeitschrift fur Physik B*, 76:381–398, 1989.

[68] L. Viana and A.C.C. Coolen. Attraction domains in neural networks. *Journal de Physique 1*, 3:777–786, 1993.

[69] S. I. Amari and K. Maginu. Statistical neurodynamics of associative memory. *Neural Networks*, 1:63–73, 1988.

[70] G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes*. Oxford University Press, 1982.

[71] T. Wang. Improving recall in associative memories by dynamic threshold. *neural networks*, 7(9):1379–1385, 1994.

[72] H. W. Yau and D. J. Wallace. Enlarging the attractor basins of neural networks with noisy external fields. *Journal of Physics A - Mathematicaland General*, 24(23):5639–5650, 1991.

[73] H. W. Yau and D. J. Wallace. Basins of attraction in a neural network model trained with external fields. *Physica A*, 185(1-4):471–480, 1992.

[74] A. M. C. L. Ho and P. DeWilde. General transient length upper bound for recurrent neural networks. *Lecture Notes in Computer Science*, 930:202–208, 1995.

[75] W. Krauth and M. Mezard. Learning algorithms with optimal stability in neural networks. *Journal of Physics A - Mathematical General*, 20(11):L745 –L752, 1987.

[76] J. Y. Chang and C. C. Wu. Desing of hopfield-type associative memory with maximal basin of attraction. *Electronics Letters*, 29(24):2128–2130, 1993.

[77] J. P. Nadal, G. Toulouse, J. P. Changeux, and S. Dehaene. Networks of formal neurons and memory palimpsests. *Europhysics Letters*, 1:535–542, 1986.

[78] U. Behn, J. L. van Hemmen, R. Kuhn, A. Lange, and V. A. Zagebnov. Multifractality in forgetful memories. *Physica D*, 68:401–415, 1993.

[79] M. Mezard, J. P. Nadal, and G. Toulouse. Solvable models of working memories. *Journal de Physique*, 47:1457–1462, 1986.

[80] G. Parisi. A memory which forgets. *Journal of Physics A: Mathematical and General*, 7:L617–L620, 1986.

[81] J. L. van Hemmen, G. Keller, and R. Kuhn. Forgetful memories. *Europhysics Letters*, 5(7):663–668, 1988.

[82] D. J. Amit and S. Fusi. Learning in neural networks with material synapses. *Neural Computation*, 6:957–982, 1994.

[83] P. J. Potts. The storage capacity of forgetful neural networks. Master's thesis, Department of Computing, Imperial College, 1995.

[84] J. L. van Hemmen, D. Grensing, A. Huber, and R. Kuhn. Nonlinear neural networks ii. information processing. *Journal of Statistical Physics*, 50:259–293, 1988.

[85] K. Y. M. Wong, P. E. Kahn, and D. Sherrington. A neural network model of working memory exhibiting primary and recency. *Journal of Physics A: Mathematical and General*, 24:1119–1135, 1991.