Classification and Regression using Mixtures of Experts

Steven Richard Waterhouse

Jesus College, Cambridge,
and
Department of Engineering, University of Cambridge

Dissertation submitted to the University of Cambridge
for the degree of Doctor of Philosophy

To Fran, Rick and Nathan Waterhouse.

Preface

This thesis is not substantially the same as any other that I have submitted for a degree or diploma or other qualification at any other university. This thesis is entirely the result of my own work over the last four years at Cambridge University Engineering Department. The length of this thesis is approximately 62,000 words, including appendices and bibliography. Some of the work in this thesis has previously been published in: Waterhouse and Cook [233], Waterhouse, Kershaw and Robinson [234], Waterhouse, MacKay and Robinson [235], Waterhouse and Robinson [236], Waterhouse and Robinson [237] and Waterhouse and Robinson [238]. Also, some of the chapters, were inspired by joint work with David MacKay, Gary Cook and Dan Kershaw. The level of collaboration was as follows.

Chapter 7: Bayesian Methods for Mixtures-of-Experts.

I first thought of using Bayesian methods for mixtures-of-experts after becoming frustrated with the singularity problems of the maximum likelihood algorithm [112]. I developed an approach based on MacKay's evidence framework [133] for multi-layer perceptrons which I subsequently discussed with David MacKay. The result of these discussions and my initial experiments was a joint paper [235]. During our discussions, however, David suggested the use of Neal and Hinton's [154] free energy framework for the mixtures-of-experts. This led to the ensemble learning ideas presented in this chapter, and also to a number of papers on ensemble learning by David [136, 134, 137]. Finally, all the experiments in the joint paper and in Chapter 7 were based on my sole implementation and performed by me.

Chapter 10: Speech Recognition using Mixtures of Multi-layer Perceptrons.

The idea for using mixtures-of-experts models for automatic speech recognition (ASR) originated in my MPhil thesis [232]. During a visit to the International Computer Science Institute (ICSI) at Berkeley, discussions with Nelson Morgan and Hervé Bourlard convinced me that mixtures of multi-layer perceptrons could be more effective than the single multi-layer perceptrons developed at ICSI [16] for speech recognition. David Johnson of ICSI kindly donated code for the MLPs to CUED as part of the SPRACH joint research exercise. After adapting this code to work within the CUED speech recognition system, ABBOT, I then extended the training algorithms and code to allow the use of mixtures of multi-layer perceptrons. At the same time, Gary Cook was independently working on David Johnson's code to adapt it to perform boosting. He has since published the results of his experiments in his PhD thesis [38]. I suggested the idea of using boosting to

initialise mixtures-of-experts to Gary, and together we performed work which led to a joint publication [233]. The boosting results presented in Chapter 10 are from experiments performed by Gary, whilst the mixtures experiments, including the ones where boosted networks were used as initial experts, were performed by me.

Chapter 11: Speaker Adaptation using Mixtures of Recurrent Neural Networks.

I first suggested the idea of using mixtures of recurrent neural networks (RNN) for speaker adaptation during discussions with Dan Kershaw on how to improve the existing adaptation method of the ABBOT system. These discussions led to the ideas presented in Chapter 11. The RNN code and linear input network (LIN) code were written by Dan Kershaw, and based on code developed by Mike Hochberg and Tony Robinson. I subsequently improved the performance and accuracy of the LIN and extended the code to support mixtures of RNN and LIN experts. The experiments on unsupervised adaptation, performed by Dan and myself led to a joint paper [234]. Further experiments on supervised adaptation were then performed by me and these are also presented in Chapter 11.

Summary

This thesis investigates a recent tool in statistical analysis: the *mixtures-of-experts* model for classification and regression. The aim of the thesis is to place mixtures-of-experts models in context with other statistical models. The hope of doing this is that we may better understand their advantages and disadvantages over other models. The thesis first considers mixtures-of-experts models from a theoretical perspective and compares them with other models such as trees, switching regression models and modular networks. Two extensions of the mixtures-of-experts model are then proposed. The first extension is a constructive algorithm for learning model architecture and parameters which is inspired by recursive partitioning. The second extension uses Bayesian methods for learning the parameters of the model. These extensions are compared empirically with the standard mixtures-of-experts model and with other statistical models on small to medium sized data sets. In the second part of the thesis the mixtures-of-experts framework is applied to acoustic modelling within a large vocabulary speech recognition system. The mixtures-of-experts is shown to give an advantage over standard single neural network approaches on this task. The results of both of these sets of comparisons indicate that mixtures-of-experts models are competitive with other state-of-the-art statistical models.

Original Contribution

The original contribution of this thesis may be summarised as follows:

- I have presented a detailed review of mixtures-of-experts in context with other statistical techniques;
- I have developed two new techniques for mixtures-of-experts: a growing algorithm for hierarchical mixtures-of-experts and a learning algorithm based on Bayesian methods;
- I have compared standard mixtures-of-experts models with my extensions and with state of the art statistical methods;
- I have developed new techniques based on the mixtures-of-experts framework for acoustic modelling in a large vocabulary speech recognition system and compared them empirically with existing acoustic models.

Acknowledgements

Many thanks to all the people who have made the last four years so interesting and enjoyable. To my supervisor, Tony Robinson, whose guidance and advice have been invaluable. To all the people in the Fallside lab who have advised me through the years, in particular Horacio Ayestaran, Simon Blackburn, Dan Kershaw, Jason Humphries, Gary Cook, Mahesan Niranjan, Carl Seymour, Marty Scott, and especially David Lovell.

Thanks also to Leo Breiman, Mike Hochberg, Mike Jordan, Ronjon Nag, Radford Neal, Steve Nowlan and Andreas Weigend for many helpful comments on my research over the last three years. I would like to thank David MacKay in particular for his advice and help. I would also like to acknowledge the financial support I received from Waterhouse Associates, Hewlett Packard, EPSRC, Jesus College, Cambridge University Engineering Department, the NIPS foundation and the Sprach ESPRIT project. Thanks also to my friends at Cambridge, especially Mark Davies, Susie Henstridge, Simon Newton, Tim Ritchey and Fiona Shields.

A special thanks to Phoebe for all her support and friendship. Finally, thanks to my parents and my brother, to whom this thesis is dedicated.

Contents

List of Figures

List of Tables

Notation

I have attempted to make the notation consistent within this thesis and the most commonly used symbols and abbreviations are listed below. To ensure simplicity of presentation som abuses of notation are made, although the intention should be clear from their context. The notation for vectors is as follows. A vector with d elements, $(x_1, x_2, \ldots, x_d)$, is represented by the bold face $\boldsymbol{x}$ and the transpose of $\boldsymbol{x}$ is represented by $\boldsymbol{x}^{(n)^T}$.

General

$\boldsymbol{x}$	—	an input vector;
$\boldsymbol{x}^T$	—	transpose of $\boldsymbol{x}$;
x_i	—	input variable i;
$\boldsymbol{y}$	—	a target vector;
y	—	a scalar target;
d	—	number of input variables (dimension of $\boldsymbol{x}$);
$\boldsymbol{x}^{(n)}$	—	an input vector at time n;
$\boldsymbol{y}^{(n)}$	—	a target vector at time n;
N	—	number of examples n in a training set;
$\{\boldsymbol{x}^{(n)}\}_1^N$	—	the set of vectors $\boldsymbol{x}^{(n)}$ for $n = 1$ to $n = N$;
X	—	the input data matrix, with N rows;
		each row consists of the transpose of an input vector $\boldsymbol{x}^{(n)^T}$;
Y	—	the target data matrix, with N rows;
		each row consists of the transpose of a target vector $\boldsymbol{y}^{(n)^T}$;
θ	—	a general set of parameters (vector or matrix);
$\boldsymbol{\theta}$	—	a parameter vector;
$\hat{\boldsymbol{y}}^{(n)}$	—	the prediction of $\boldsymbol{y}^{(n)}$;
c_k	—	class k;
$P(a)$	—	the probability of a, in the case of discrete a,
		or probability density of a in the case of continuous a;
$E[a]$	—	the expectation of a, defined as $\int aP(a)da$;

Mixtures-of-experts

$\mathbf{w}_i$	—	the parameters of expert i in a mixture of experts;
σ_i	—	the noise level variance parameter of expert i in a mixture of experts (for regression);
$\mathbf{v}$	—	the parameters of the gate in a mixture of experts;
$g_i^{(n)}$	—	output i of the gate at time n in a mixture of experts;
$\mathcal{E}_i$	—	expert i in a mixture of experts or node i in a hierarchical mixture of experts;
$Pa(\mathcal{E}_i)$	—	the parent of node $\mathcal{E}_i$
$An(\mathcal{E}_i)$	—	the ancestors of node $\mathcal{E}_i$
$Ch(\mathcal{E}_i)$	—	the children of node $\mathcal{E}_i$
$\mathcal{G}_j$	—	a gate rooted at node j in a HME
Z	—	a set of missing data in the EM algorithm
$z_i^{(n)}$	—	a missing data point n for a mixture component $\mathcal{E}_i$;
$\zeta_i^{(n)}$	—	the joint posterior probability of a node;
$\varphi_i^{(n)}$	—	the conditional posterior probability of a node;
α_i	—	the hyperparameter of the prior distribution on the parameter values of an expert;
μ_i	—	the hyperparameter of the prior distribution on the parameter values of a gate;
β_i	—	the hyperparameter of the prior distribution on the noise level of an expert.

Abbreviations

ME	—	mixtures-of-experts;
HME	—	hierarchical-mixtures-of-experts;
HMM	—	hidden Markov model;
RNN	—	recurrent neural network;
MLP	—	multi-layer perceptron;
GP	—	Gaussian process;
GLIM	—	generalised linear model;
CART	—	classification and regression tree;
MARS	—	multivariate adaptive regression splines;
LIN	—	linear input network;
PLP	—	perceptual linear prediction;
ML	—	maximum likelihood;
MAP	—	maximum a-posteriori;
EM	—	expectation maximisation;
KL	—	Kullback-Leibler;
MCMC	—	Markov chain Monte Carlo;
IRLS	—	iteratively re-weighted least squares;
DELVE	—	data for evaluating learning in valid experiments;
ICSI	—	International Computer Science Institute;
CUED	—	Cambridge Univeristy Engineering Department;
WSJ	—	Wall Street Journal;
DARPA	—	Defence Advanced Research Projects Agency;
NIST	—	National Institute for Standards and Technology;
LIMSI	—	Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur.

Chapter 1

Introduction

This thesis investigates a recent tool in statistical analysis: the *mixtures-of-experts* (ME) model of Jacobs, Jordan, Nowlan and Hinton [104]. The aim of the thesis is to place ME models in context with other statistical models. The hope of doing this is that we may better understand their advantages and disadvantages over other models. The thesis first considers ME models from a theoretical perspective and compares them with other models such as trees, switching regression models and modular networks. Two extensions of the ME model are then proposed and compared empirically with the standard ME model and with other statistical models on small to medium sized data sets. Finally the ME framework is applied to acoustic modelling within a large vocabulary speech recognition system. The results of both of these comparisons indicate that ME models are competitive with other state-of-the-art statistical models. Before the ME model is described further some important aspects of statistical modelling are reviewed.

The particular type of statistical models studied in this thesis are ones which predict outcomes from observed data. For example, in financial markets statistical models are used to predict the price of stocks based on their past performance. Statistical models are also used in optical character recognisers (OCR) in which they predict particular alphabetic, numeric or symbolic characters based on the data obtained by scanning a piece of paper. These two examples involve prediction of two different types of outcome: *continuous* variables in the stock prediction example, and *categorical* variables in the OCR example. Prediction of continuous variables is known as *regression* while prediction of categorical variables is known as *classification*. This thesis will be concerned solely with regression and classification models.

Statistical models for the prediction of outcomes can be described mathematically as follows. The aim of a model is to learn to predict the values of a vector of k *target* or *dependent* variables $\boldsymbol{y}$ from a vector of d *input* or *independent* variables $\boldsymbol{x}$. The prediction of $\boldsymbol{y}$ is denoted $\hat{\boldsymbol{y}}$. The prediction is made using a set of adjustable *parameters* θ. We may represent the prediction of the model by a function $f(\cdot)$ of the inputs and parameters:

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}^{(n)}, \theta). \tag{1.1}$$

The actual values of the parameters θ will determine how close the predictions are to the target variables. The parameter values are *learnt* or *inferred* from a set of N observations $\{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_1^N$, known as a

training set. Although sufficiently flexible models may often be able to learn to predict the set of targets in the training set very closely, such models will then typically have poor predictive performance on *unseen* or *test* data. This behaviour is known as *over-fitting*. Models that perform well on unseen data are said to have good *generalisation* power. Achieving good generalisation is a function of both the representative powers of a model, *i.e.*, the functions that the model can represent, and the *search* method used to find the parameters. Since the possible space of parameters is typically large, search methods must be also be efficient in time. In this thesis several representations and search methods are investigated for ME models with the aim of efficiently achieving good generalisation.

Having introduced the mathematical formalism of statistical prediction and some notation, let us now describe some of the methods that have been proposed for this task. The aim of this description is to provide a background against which the ME models may be compared rather than to provide a comprehensive or historical account of the field. The first models considered are conventional statistical models. These are followed by a description of models motivated by modern statistical methods, machine learning and neural networks. This is followed by a discussion of techniques for avoiding over-fitting.

Linear models

The earliest models studied in statistics for predicting outcomes were linear ones. Linear models make predictions using a *linear* combination of the input vector with a parameter vector $\boldsymbol{\theta}$. In the case of linear regression with a single target variable y, the prediction is given by:

$$\hat{y} = \sum_{l=1}^{d} \theta_l x_l + \theta_0, \tag{1.2}$$

which is frequently written as $\hat{y} = \boldsymbol{\theta}^T \boldsymbol{x}$ by augmenting the input vector with a 1, so that $\boldsymbol{x} = [1 x_1 \dots x_d]$. In classification, the analogy of linear regression is *logistic* regression, in which the model is given by:

$$\hat{y} = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \boldsymbol{x})}. \tag{1.3}$$

$\hat{y}$ is the prediction of a single category y, and may be interpreted as the probability of the category given the model: $\hat{y} = P(y|x, \boldsymbol{\theta})$. This is known as *binary* or dichotomous classification. The extension to more than one class, known as *multi-way* or polytomous classification, is straightforward. The parameters of the model are given by a matrix of parameters $\{\boldsymbol{\theta}_c\}_1^k$. For category c the probability $P(y_c|\boldsymbol{x}, \boldsymbol{\theta}_c)$ is given by:

$$\hat{y}_c = f(x) = \frac{\exp(\boldsymbol{\theta}_c^T \boldsymbol{x})}{\sum_{a=1}^{k} \exp(\boldsymbol{\theta}_a^T \boldsymbol{x})}, \tag{1.4}$$

where $f(\cdot)$ is known as the *multinomial logit* or *softmax* function. Both the linear regression and logistic regression models may be viewed in a unified framework known as *generalised linear models* (GLIM) [140]. In the GLIM framework a function of the prediction, $h(\hat{y})$, is modelled by the linear combination of input variables $\boldsymbol{\theta}^T \boldsymbol{x}$. Appropriate choices of this *link* function allow different models to be realised.

Linear models have been popular in statistics due both to their simplicity and their relative ease of interpretation. For example, the relative magnitude of each coefficient θ_i in a linear regression gives an indication of the relative importance of the variable x_i. In many problems, however, a linear fit to data is an inappropriate model which does not capture the relationships present in the data. In order to obtain a non-linear fit, one solution is to introduce extra terms into the input vector and model these using a linear model. For example, we might believe that $\log(x_j)$ was an important term in our model and so we would introduce this term into the input vector $\boldsymbol{x}$ and perform a regression on the augmented vector. Alternatively interaction terms such as $x_i x_j$ might be used. One problem with this approach is that it may be difficult to know which terms to introduce, especially when the number of input variables d is large. In modern statistical methods more explicit modelling of non-linear functions is employed. A number of these models will now be described. All of these models, linear or non-linear, can be considered as particular *learning methods* which learn to predict outcomes from data. A useful decomposition of learning methods is that of Fayyad, Piatetsky-Shapiro and Smyth [51][1] into three components: *model representation*, *estimation criterion* and *search method*. The first component, model representation, concerns the representational power of the model, *e.g.*, linear regression models make a prediction using a linear combination of the input variables. The second component, the estimation criterion, is the criterion used to determine how well a set of parameters fit the data, *e.g.*, linear regression models are fit by minimising the sum of squared errors on the training data. Finally the search method is the algorithm used to specify how the parameters are found that satisfy the model representation, estimation criterion and the training data. The next section discusses the representation, estimation and search aspects of a number of non-linear models.

Non-linear models

A natural extension of linear models is to use d non-linear functions of individual inputs x_i and combine them linearly:

$$\hat{y} = \theta_0 + \sum_{i=1}^{d} \phi_i(x_i) \tag{1.5}$$

where $\phi_i(\cdot)$ is a non-linear function of the input x_i and θ_0 a constant offset. This is known as an *additive* regression model and may be generalised in a similar fashion to generalised linear models[2]: via a *link* function $h(\hat{y})$, to give a *generalised additive model* (GAM)[80]. The functions $\phi_i(\cdot)$ are fitted using non-parametric methods such as smoothing splines. Although GAMs are interpretable they do not explicitly allow for interactions between the input variables.

One method for introducing interactions is to use linear combinations of the variables and combine M non-linear functions of these combinations:

$$\hat{y} = \theta_0 + \sum_{m=1}^{M} \theta_m \phi_m(\boldsymbol{w}^T \boldsymbol{x}). \tag{1.6}$$

[1] See also [25].

[2] This generalisation is also possible for all the models considered in the section.

This model is known in statistics as a *projection pursuit* regression (PPR) model [59]. In the neural networks community this model is known as a *multi-layer perceptron* (MLP) with a single hidden layer [201]. Although the two models have the same representative power, the search methods used for each model are different. In the PPR model each function $\phi_m(\cdot)$, is fitted separately using a smoothing function such as a spline. In an MLP these functions are usually chosen to be sigmoid functions and represent the outputs of the *hidden* units of the MLP. The parameters of an MLP are jointly estimated using the error back-propagation algorithm [244, 201].

A related approach is the radial basis function network in which the non-linear functions ϕ_m are represented by radial basis functions such as Gaussians:

$$\hat{y} = \theta_0 + \sum_{m=1}^{M} \theta_m \exp(-\tfrac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_m)^T \Sigma_m^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_m)) \tag{1.7}$$

where the mean vector $\boldsymbol{\mu}_m$ and the covariance matrix Σ_m are adaptive parameters. Finally, an alternative method for introducing interactions is via products of power splines as is done in the *multivariate adaptive regression splines* (MARS) model [58].

We now turn to two alternative approaches to regression and classification which are popular in the machine learning community: instance based classifiers and tree based methods. An example of an instance based classifier is the k nearest neighbour method [49]. For each point $\boldsymbol{x}^{(n)}$ in the test set, the nearest k neighbours in the training set are chosen and a vote taken over their categories. The result of the vote is used as the prediction of the category of $\boldsymbol{x}^{(n)}$. Nearest neighbour methods are also used in non-parametric regression and are related to the smoothing functions used in generalised additive models. Nearest neighbour methods are a simple but surprisingly successful method for regression and classification for some problems but have the disadvantage of being computationally expensive in terms of memory and time.

An alternative approach to predicting outcomes is that of tree based methods which use a recursive partitioning scheme to divide the space of inputs and fit local surfaces to the targets within the partitions. Two of the most well known examples are classification and regression trees (CART) [21] and C4.5 [186]. The functions used to partition the data are generally simple threshold decisions on one input variable at a time. The local predictions are also generally simple, consisting of piecewise constant functions. Classification and regression trees can also be converted to sets of rules which can be useful for interpretability of the model [186]. Tree based methods are closely related to the mixtures-of-experts models considered in this thesis. This connection is considered in further detail in Chapter 2. We now introduce the general framework of mixtures-of-experts models.

Mixtures-of-experts models are motivated by the concept that if a problem may be broken into smaller sub-problems, these sub-problems might be easier to solve than the overall problem. The general mixtures-of-experts framework specifies that a prediction is made up of a series of predictions from separate *experts*, each of which is a statistical model with parameters $\mathbf{w}_i$:

$$\hat{y} = \sum_{i=1}^{I} g_i \hat{y}_i \tag{1.8}$$

where I is the number of experts in the model, $\hat{y}_i$ the prediction of expert i and g_i the weighting on expert i. This weighting is itself the prediction of another model, known as the gate. The gate estimates the probability of each expert given the current input $\boldsymbol{x}$ and a set of parameters $\mathbf{v}$:

$$g_i = P(\mathcal{E}_i|\boldsymbol{x}, \mathbf{v}), \tag{1.9}$$

where $\mathcal{E}_i$ represents expert i. This is a general framework in which many different choices of statistical models for the experts and gates may be made. For example, in *hierarchical* mixtures-of-experts (HME) the experts are themselves chosen to be made up of mixtures-of-experts in a tree structure. HME models are closely related to classification and regression trees. This relationship is explored further in Chapter 2 which also gives a detailed review of other applications and related models to mixtures-of-experts. This thesis explores a number of different types of ME and HME models. In Part 1 of the thesis ME and HME models are studied and compared empirically with other statistical models. These ME and HME models use generalised linear models as experts and logistic regression models as gates. Part 2 investigates ME models for speech recognition. Two models are used: mixtures of MLP experts gated by an MLP and mixtures of recurrent networks gated by a logistic regression model.

This summarises some of the main tools of statistical prediction of outcomes. The next section considers the problem of searching for parameter values in statistical models. More detailed reviews of these models can be found in the following references. A classic introductory reference on pattern recognition is Duda and Hart [49]. A more recent reference which covers statistical, tree based, unsupervised methods and neural networks is Ripley [195]. Bishop [14] presents a comprehensive introduction to pattern recognition using neural networks. Russell and Norvig [202] consider machine learning within the overall context of artificial intelligence. Ripley [194] presents a comparison of neural networks and statistical methods. Cheng and Titterington [36] present a comprehensive review of neural networks from a statistical perspective. Finally the review by Jordan and Bishop [109] describes neural networks from a unified graphical model perspective.

Estimation and Search

For a linear regression model the parameters are fitted by minimising the sum of squared errors E between the predictions of the model and the targets:

$$E = \sum_n (\hat{y}^{(n)} - y^{(n)})^2. \tag{1.10}$$

This may be viewed in a probabilistic framework as follows. We assume that the probability $P(y|x, \theta)$ of generating y is modelled as a Gaussian:

$$P(y|x, \theta) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left(-\frac{(\hat{y} - y)^2}{2\sigma^2}\right) \tag{1.11}$$

If the data is independent and identically distributed (iid) then minimising E is equivalent to maximising the likelihood[3] L of the model given the data:

$$L = \prod_{n=1}^{N} P(y^{(n)}|\boldsymbol{x}^{(n)}, \theta). \tag{1.12}$$

Since $\log L$ in the case of a Gaussian is proportional to the sum squared error E plus a constant, the result follows. The principle of maximum likelihood (ML) may also be used to fit parameters of other models such as generalised linear models and neural networks. In mixtures-of-experts models maximum likelihood may be used as an estimation criterion. The likelihood for a mixture of I experts is given by:

$$L = \prod_{n=1}^{N} \sum_{i=1}^{I} P(\mathcal{E}_i|\boldsymbol{x}^{(n)}, \mathbf{v}) P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \mathbf{w}_i, \mathcal{E}_i) \tag{1.13}$$

One problem with ML as an estimation criterion is that it aims to minimise the difference between the targets and the predictions on the *training* data rather than on *unseen* or test data. If a sufficiently flexible model is trained via ML it may overfit the training data and have poor generalisation power on the unseen data. One solution is to use a more restrictive model, *e.g.,* by reducing the number of hidden units in an MLP, although this risks using a model that is too restrictive. Alternatively the training may be stopped *early* - before the parameters have converged. This latter approach is known as *early stopping* and is often used in multi-layer perceptrons. One method for choosing when to stop is *cross-validation* [216]. In cross-validation the training set is divided into a series of *estimation* and *validation* sets. A series of models are now trained on each division of the training set. For each model the performance as it is trained on the estimation set is measured on the validation set and training is stopped when the minimum on the validation set occurs. After all the models have been trained either the best model is chosen or the outputs of all the models combined. This method of complexity control via early stopping is used for mixtures-of-experts models in Chapter 5.

Cross validation may be used to control many other aspects of statistical model complexity, for example in tree based models the depth of the tree is controlled by recursively extending (*growing*) the tree on a estimation set and then reducing its size (*pruning*) based on its performance on a validation set. This tree growing via cross validation concept is extended to hierarchical mixtures-of-experts models in Chapter 6.

An alternative approach to complexity control is via Bayesian methods. Bayesian methods have been used in statistics for many years [*e.g.,* 18] and represent an alternative to the standard frequentist statistical paradigm[4]. Bernardo and Smith [13] and Berger [12] are classic references on Bayesian methods, whilst practical approaches are described in Ó Ruanaidh and Fitzgerald [162] and Gelman et al. [67]. Complexity control of neural networks by Bayesian methods is done by placing a prior on the parameters $P(\theta)$. Predictions are made by integrating over the posterior distribution $P(\theta|X, Y)$ of parameters. Two approaches have been proposed for training neural networks by Bayesian methods [29, 133, 153].

[3] Since the log function is monotonic, maximising the likelihood is also equivalent to maximising the log-likelihood.

[4] The minimum description length approach (MDL) Rissanen [196] is also often used in a similar way to Bayesian inference.

In the first a Gaussian approximation is made to the posterior distribution, the parameters fixed to their most probable values [29, 133] and the integral performed analytically. In the second, the integration over parameters is performed numerically using Monte Carlo techniques[153]. One advantage of the Bayesian approach is that there is no need to hold data aside as is done in cross validation. In small data sets this can be an important advantage. Bayesian methods have been used in mixtures-of-experts by Peng et al. [172] who used a Gibbs sampling [68] method. Chapter 7 describes an alternative approach to Bayesian inference in mixtures-of-experts which uses an approximation to the posterior distribution and minimises the distance between the approximating and true posterior distributions. We now turn our attention to the final aspect in our specification of models: search for parameters.

Search over parameters in non-linear models has two components. Firstly given a particular model form and estimation criterion, one direct approach to search is to take derivatives of the appropriate cost function, *e.g.*, the likelihood in ML, and use a suitable optimisation algorithm [*e.g.,* 53]. However, this will often not be an efficient approach and thus one aspect of search is in factorising the training algorithm to be more efficient, *e.g.,* the error back-propagation algorithm efficiently factorises the training of multi-layer perceptrons. After factorisation, the overall optimisation is typically decoupled into a set of simpler optimisations. In mixtures-of-experts models the search process is factorised by way of the expectation-maximisation (EM) algorithm into a series of separate optimisation problems; one for each expert and gate in the model. These optimisation problems may be solved by standard optimisation algorithms. Chapter 2 reviews the EM algorithm for ME and HME models whilst Chapter 4 describes the use of a conjugate gradient algorithm for optimisation of expert and gate parameters.

The second component of search is the method used to schedule or adjust the search so as to give better generalisation of the models. For example, early stopping via cross validation is one method of adjusting the ML training algorithm for neural networks. Many search adjustments are heuristics which are motivated by empirical investigations, but nevertheless are often essential for practical implementations of learning methods. This thesis describes a number of techniques for adjusting search, based mostly on early stopping, for mixtures-of-experts models.

Outline of the thesis

Part I of the thesis focuses on the application of mixtures-of-experts to small and medium sized problems in regression and classification. The emphasis is on comparison with other machine learning algorithms. This comparison is performed within the DELVE framework [190] which allows consistent comparisons with other researchers' results. Chapter 4 then describes the framework used for regression and classification mixtures-of-experts models and its implementation. Chapters 5, 6 and 7 describe the theory and implementation of five learning methods based on the framework introduced in Chapter 4. Following this, results on various DELVE data sets are presented in Chapter 8 and compared with the results of other learning methods including MLPs, MARS, CART and Gaussian processes. This chapter also summarises this part of the thesis and contains suggestions for future work.

Part II of the thesis focuses on large scale applications of mixtures-of-experts and describes their integration into a large vocabulary speech recognition system, known as the ABBOT system [199]. AB-

BOT is a hybrid connectionist-HMM speech recogniser which generally uses either recurrent neural networks or MLPs as its acoustic models. This part of the thesis introduces two new acoustic models based on mixtures of RNNs or MLPs and is organised as follows. Chapter 9 introduces the concepts underlying modern speech recognition systems, with particular reference to the ABBOT system. Following this, two applications of the mixtures-of-experts framework within ABBOT are described. These two applications are independent contributions and for this reason they are presented in two separate chapters which are fully self contained. Chapter 10 presents a new speech recognition system for ABBOT which uses mixtures of MLPs as the acoustic modelling component. Results are presented which show that the method yields significantly better performance than a single MLP with comparable numbers of parameters. Chapter 11 focuses on the use of the mixtures-of-experts framework within ABBOT for speaker adaptation. For this application mixtures of recurrent networks are used as the acoustic modelling component. Significantly improved adaptation performance for this system is also demonstrated in comparison to the standard method for speaker adaptation within ABBOT .

Part III of the thesis contains the final conclusions, appendices and the list of bibliographic references for all chapters.

Chapter 2

Mixtures of Experts

This chapter introduces a class of probabilistic models known as *mixtures-of-experts* (ME) [104]. Mixtures-of-experts models consist of a set of *experts*, which model conditional probabilistic processes, and a *gate* which combines the probabilities of the experts. The probabilistic basis for the mixture of experts is that of a mixture model [224] in which the experts form the input conditional mixture components while the gate outputs form the input conditional mixture weights. A straightforward generalisation of ME models is the *hierarchical mixtures-of-experts* (HME) class of models, in which each expert is made up of a mixture of experts in a recursive fashion.

One motivation for mixtures-of-experts is the *divide and conquer* principle common in computer science. This principle states that complex problems can be better solved by decomposing them into smaller tasks. In mixtures-of-experts the assumption is that there are separate processes in the underlying process of generating the data. Modelling of these processes is performed by the experts while the decision of which process to use is modelled by the gate.

This thesis also aims to place mixtures-of-experts in context with other supervised learning algorithms. Mixtures-of-experts have many connections with other algorithms such as tree-based methods, mixture models and switching regression. In this chapter, I consider how closely the mixtures-of-experts model resembles these other algorithms, and what is novel about it. In addition I review a number of the prominent applications of the mixtures-of-experts. Before this review I describe the main components of the general mixtures-of-experts framework.

2.1 Specification of the Mixtures-of-Experts Model

The ME class of models is based on the following decomposition of target data Y and input data X:

$$P(Y|X) = \sum_{Z} P(Z|X)P(Y|X,Z) \tag{2.1}$$

where Z is a set of *hidden* or *missing* data, and the sum is over all configurations of Z. The hidden data Z indicates which expert was responsible for generating each data point. Let us use the symbol $\mathcal{E}_i$ to

denote expert i. In a training set of N points, Z is made up of N vectors $\boldsymbol{z}^{(n)}$ in which:

$$z_i^{(n)} = \begin{cases} 1 & \text{if expert } \mathcal{E}_i \text{ generated } \boldsymbol{y}^{(n)} \text{ from } \boldsymbol{x}^{(n)}; \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

This decomposition implies the following assumption governing the generation of observed data: for each pattern $(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})$ there is a specific process, indicated by the state of the vector $\boldsymbol{z}^{(n)}$, generating the output $\boldsymbol{y}^{(n)}$ given the input $\boldsymbol{x}^{(n)}$. There are thus two tasks in modelling this decomposition: how to model the assignment of processes, Z, given inputs X, and how to model the generation of outputs, Y, given inputs X and specific processes Z. In a ME, these tasks are performed by the gate (or *gating network*) and experts (or *expert networks*) respectively. The gate models the probabilistic assignment of inputs to processes, $P(\boldsymbol{z}^{(n)}|\boldsymbol{x}^{(n)}, \mathbf{v})$ given parameters $\mathbf{v}$. Each expert models the probabilistic generation, for each process i, of outputs given inputs, $P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, z_i^{(n)} = 1, \mathbf{w}_i)$.

In a general mixture of experts architecture there are I experts $\{\mathcal{E}_i\}_1^I$ and a gate $\mathcal{G}$, which model the conditional density of targets $Y = \{\boldsymbol{y}^{(n)}\}_1^N$ given inputs $X = \{\boldsymbol{x}^{(n)}\}_1^N$ as:

$$P(Y|X, \boldsymbol{\theta}) = \prod_{n=1}^{N} \sum_{i=1}^{I} P(z_i^{(n)} = 1|\boldsymbol{x}^{(n)}, \mathbf{v}) P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, z_i^{(n)} = 1, \mathbf{w}_i) \tag{2.3}$$

where $\boldsymbol{\theta}$ is the overall set of parameters $\boldsymbol{\theta} = \{\mathbf{v}, \{\mathbf{w}_i\}_1^I\}$. I use the following shorthand notation for these quantities:

$$P(\mathcal{E}_i|\boldsymbol{x}^{(n)}, \mathbf{v}) \equiv P(z_i^{(n)} = 1|\boldsymbol{x}^{(n)}, \mathbf{v}) \tag{2.4}$$

$$P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \mathcal{E}_i, \mathbf{w}_i) \equiv P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, z_i^{(n)} = 1, \mathbf{w}_i) \tag{2.5}$$

For point predictions about targets $\boldsymbol{y}^{(n)}$, the expected value of the probability density function (2.3) is used. For pattern $(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})$, the expected value of $\boldsymbol{y}^{(n)}$ given $\boldsymbol{x}^{(n)}$ and the model is given by:

$$\begin{aligned} \hat{\boldsymbol{y}}^{(n)} = E(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \boldsymbol{\theta}) &= \sum_{i=1}^{I} P(\mathcal{E}_i|\boldsymbol{x}^{(n)}, \mathbf{v}) E(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \mathcal{E}_i, \mathbf{w}_i), \\ &= \sum_{i=1}^{I} g_i^{(n)} \hat{\boldsymbol{y}}_i^{(n)}, \end{aligned} \tag{2.6}$$

where the short-hand notation $g_i^{(n)} = P(\mathcal{E}_i|\boldsymbol{x}^{(n)}, \mathbf{v})$ and $\hat{\boldsymbol{y}}_i^{(n)} = E(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \mathcal{E}_i, \mathbf{w}_i)$ has been introduced.

This leads naturally to the following terminology. Given an input $\boldsymbol{x}^{(n)}$, each expert $\mathcal{E}_i$ makes a prediction $\hat{\boldsymbol{y}}_i^{(n)}$ of the target $\boldsymbol{y}^{(n)}$. The gate combines these predictions with its outputs $\{g_i^{(n)}\}_1^I$ to give the overall prediction of the model (2.6). This is shown schematically for two experts in Figure 2.1. The gate's outputs $\{g_i^{(n)}\}_1^I$ can be interpreted as estimates of the probabilities of selecting each of the I experts given the input $\boldsymbol{x}^{(n)}$. An appropriate parameterisation of the gate is therefore a probabilistic classifier, *e.g.,* a logistic regression model in the case of 2 experts:

$$P(\mathcal{E}_i|\boldsymbol{x}^{(n)}, \mathbf{v}) = \frac{1}{1 + \exp(-(\mathbf{v}^T \boldsymbol{x}^{(n)}))} \tag{2.7}$$

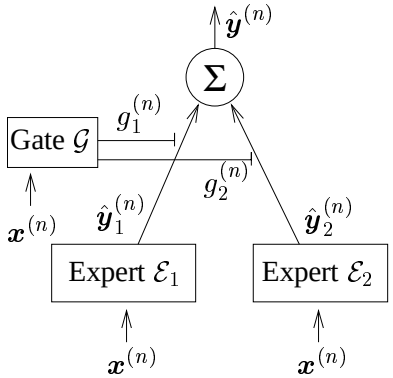

Figure 2.1: **A mixtures-of-experts model.** The model consists of 2 experts, $\mathcal{E}_1$ and $\mathcal{E}_2$ and a gate $\mathcal{G}$. Each expert predicts the value of the target $\boldsymbol{y}^{(n)}$, conditioned on the input $\boldsymbol{x}^{(n)}$. The prediction of expert $\mathcal{E}_1$ is given by $\hat{\boldsymbol{y}}_1^{(n)}$. The outputs of the experts are *gated* by the outputs of the gate $g_1^{(n)}$ and $g_2^{(n)}$. The overall output of the model is the convex sum of the gate and expert outputs: $\hat{\boldsymbol{y}}^{(n)} = g_1^{(n)}\hat{\boldsymbol{y}}_1^{(n)} + g_2^{(n)}\hat{\boldsymbol{y}}_2^{(n)}$, where $g_1^{(n)} + g_2^{(n)} = 1$.

in which the gate consists of a $d+1$ dimensional parameter vector[1] $\mathbf{v}$.

The form of the experts is chosen so as to fit the particular problem at hand. For regression problems, for example, one choice of expert is a linear regression model:

$$\hat{y}_i^{(n)} = \mathbf{w}_i^T \boldsymbol{x}^{(n)} \tag{2.8}$$

although in principle more complex experts can be used, *e.g.*, multi-layer perceptrons or radial basis functions. The experts can take any form such that the expected value of their probability density is consistent with the form of the problem, although it is often desirable to use experts which have a simple form since their optimisation is easier and they may be easily interpreted.

In summary, we can consider a mixtures-of-experts model as an input conditional mixture model [108].[2] In this model, the data are assumed to be generated from a series of processes. Each example $(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})$ is assumed to be generated from one process i only. Process i is assumed to be selected by a multinomial probability distribution $P(Z)$, such that $z_i^{(n)}$ represents the decision to use process i for data point n. The experts model the distinct processes in the data, and the gate models the decision to use one of these distinct processes. This allows us to write the augmented likelihood $P(Y, Z|X)$ as[112]:

$$P(Y, Z|X, \theta) = \prod_{n=1}^{N}\prod_{i=1}^{I}\left\{P(\mathcal{E}_i|\boldsymbol{x}^{(n)}, \mathbf{v})P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \mathcal{E}_i, \mathbf{w}_i)\right\}^{z_i^{(n)}}. \tag{2.9}$$

This probabilistic model is illustrated in Figure 2.2 which shows a *belief network* for the mixture of experts. A belief network (also known as a *Bayesian* network) is a graphical method for representing joint

[1] To simplify the notation I have assumed that the vector $\boldsymbol{x}^{(n)}$ is augmented by a constant 1 in each case: $\boldsymbol{x}^{(n)} = [1\ x_1^{(n)} x_2^{(n)} \ldots x_d^{(n)}]$.

[2] Although the mixture of experts may be considered as a mixture model, there are some subtle differences between it and standard mixture models. In the standard mixture model framework [224] the mixture weights are not conditional on the input $\boldsymbol{x}^{(n)}$ as in the mixture of experts. Also, the mixture components estimate input conditional densities *e.g.*, $P(\boldsymbol{x}^{(n)}|\mathcal{E}_i)$ rather than posterior probability densities of input to output mappings as in the mixture of experts *e.g.*, $P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \mathcal{E}_i)$.

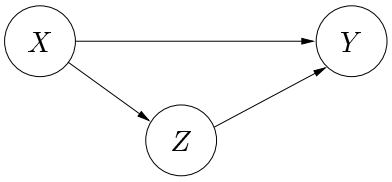

Figure 2.2: **Belief network for the mixtures-of-experts model.** Target data Y is dependent on the input data X and the choice of expert Z. The choice of expert Z is also dependent on the input X. In the mixtures-of-experts, $P(Z|X)$ is modelled by the gate while $P(Y|Z,X)$ is modelled by the experts.

probability statements. Recently, graphical models [126] and specifically Bayesian networks [106] have become a popular method in some applications of machine learning such as medical diagnosis [215]. One appealing property of graphical models is the ability to represent different models within a unified scheme. Buntine [27] summarises a number of such representations and describes general methods for learning with graphical models. Many graphical models, including mixture models [224] and hidden Markov models [130], require inference over *hidden* variables. This is typically done via the expectation algorithm (EM) [8, 45, 142]. In the mixtures-of-experts the hidden variable Z specifies the identity of the expert which is assumed to have generated the data. Section 2.3 describes an application of the EM algorithm to mixtures-of-experts models.

The belief network of Figure 2.2 expresses the assumption that the target Y is dependent on the input X and the multinomial random variable Z. One expression which will become useful in the rest of the discussion is the *posterior* probability of an expert, $P(\mathcal{E}_i|\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}, \mathbf{w}_i)$. For the set of indicator variables Z the posterior probability is given by:

$$P(Z|X,Y,\boldsymbol{\theta}) = \frac{P(Z|X,\boldsymbol{\theta})P(Y|X,Z,\boldsymbol{\theta})}{P(Y|X,\boldsymbol{\theta})}. \tag{2.10}$$

For a single pattern $(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})$, the posterior probability of expert $\mathcal{E}_i$ is denoted $\zeta_i^{(n)}$ and is given by:

$$\zeta_i^{(n)} = P(\mathcal{E}_i|\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}, \boldsymbol{\theta}) = \frac{P(\mathcal{E}_i|\boldsymbol{x}^{(n)}, \mathbf{v})P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \mathbf{w}_i, \mathcal{E}_i)}{\sum_j P(\mathcal{E}_j|\boldsymbol{x}^{(n)}, \mathbf{v})P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \mathbf{w}_j, \mathcal{E}_j)}. \tag{2.11}$$

This concludes our description of the representation of the mixtures-of-experts model. We will now describe the representation of the *hierarchical* mixtures-of-experts model. This is a mixtures-of-experts model in which each expert is *itself* a mixtures-of-experts model. Following this in Section 2.3 we describe how to train mixtures and hierarchical mixtures-of-experts.

2.2 Hierarchical Mixtures of Experts

In the mixtures-of-experts model proposed by Jacobs and Jordan [101] the experts were linear regressors and the gate was a logistic regressor. This model contains one component - the gate - which adds non-linearity to the possible predicted functions. As described in Chapter 1, the ability to model non-linear

functions is a desirable one in statistical models. However the non-linear functions that a ME model can represent are somewhat restricted since the gate can only form linear boundaries between adjacent expert regions in the input space. More non-linear functions can be modelled in two ways: either by making the experts more non-linear or by making the gates more non-linear. One approach to adding non-linearity has been to use multi-layer perceptrons as experts and gates. Examples of this kind of model will be presented in the review (Section 2.4). A complementary approach proposed by Jordan and Jacobs [112] is to use experts which are *themselves* mixtures-of-experts models. This approach yields a *hierarchical* mixtures-of-experts model (HME), which may be visualised as a tree structure. The terminal nodes, or *leaves*, of the tree contain the expert networks whilst the non-terminal nodes contain gating networks. This tree structure is shown schematically for a model with two levels of binary branching nodes in Figure 2.4. The interpretation of the HME is that each process that generated the data is itself composed of a decomposition into processes that are selected stochastically. The experts model the lowest level of process, whilst the gates model the successive selection of decomposable processes terminating with the experts.

In a similar manner to the mixtures-of-experts model the HME can be given a probabilistic interpretation. Consider generalising the belief network[3] of Figure 2.2 such that an extra pair of hidden states, W^2 and W^3 are added, and rewrite the original hidden state Z as W^1. These new states have the following conditional dependencies:

- state W^1 depends only on the input X;
- state W^2 depends on the input and state W^1;
- state W^3 depends on states W^2, W^1 and the input;
- the target Y depends on all states and the input.

The belief network which expresses these dependencies is shown in Figure 2.3. The meaning of these states can be understood from the tree diagram of Figure 2.4. Each state represents the set of conditional decisions about which branch to take at a particular non-terminal node in a tree, *e.g.*, if a two level binary tree is used, W^1 and W^2 would take values of either 0 or 1 denoting either a left or right branch taken at the top or 2nd levels of the tree respectively, thus node $\mathcal{E}_3$ in Figure 2.4 is specified by the choice $W^1 = 0, W^2 = 0$. Although this formalism is succinct, I prefer the use of the following alternative formalism, which will be used in the rest of the thesis. Each node i has an indicator variable $z_i^{(n)}$ which is 1 if node i if the node was in the path from the root node of the tree to the terminal node that was considered to have generated the data at time n, and is 0 otherwise. Given independent identically distributed data, we may write the conditional probability of Y given X for a HME model with 3 levels

[3]This belief network formalism is based on that of Jordan et al. [110]. Other Bayesian or belief network interpretations of mixtures-of-experts are given by Buntine [28] and Ripley [195].

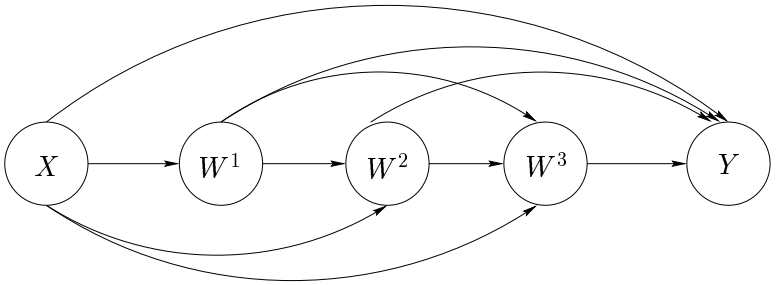

Figure 2.3: **Belief network for the hierarchical mixture of experts.** The target variable Y is dependent on the input X and each of the hidden states W^1, W^2 and W^3. These states replace the single hidden state Z of the mixtures-of-experts. Each state W^i is dependent on its previous states $\{W^j\}_1^{i-1}$ and the input X.

as $\prod_n P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)})$ where,

$$P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}) = \sum_i P(z_i^{(n)}=1|\boldsymbol{x}^{(n)}) \sum_j P(z_j^{(n)}=1|z_i^{(n)}=1,\boldsymbol{x}^{(n)}) \sum_k P(z_k^{(n)}=1|z_j^{(n)}=1,z_i^{(n)}=1,\boldsymbol{x}^{(n)}) P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)},z_k^{(n)}=1,z_j^{(n)}=1,z_i^{(n)}=1), \tag{2.12}$$

where the labels i, j and k index over the nodes in the first, second and third levels of the tree respectively. This conditional probability is a hierarchical mixture model in which each of the probability densities is represented by a statistical model. For example in Figure 2.4, gate $\mathcal{G}_1$ models $P(z_j^{(n)}=1|z_i^{(n)}=1,\boldsymbol{x}^{(n)})$ in which $i=0$ and $j=0$ for node 3 and $j=1$ for node 4.

Although conceptually a single gate could be used to model each joint probability $P(z_i^{(n)}=1,z_j^{(n)}=1|\boldsymbol{x}^{(n)})$, it is more natural to use individual gates to model each conditional probability. As before, a set of experts is used to model the conditional probabilities $P(\boldsymbol{y}^{(n)}|z_i^{(n)}=1,z_j^{(n)}=1,z_k^{(n)}=1,\boldsymbol{x}^{(n)})$. The parameters of expert $\mathcal{E}_i$ are represented by $\mathbf{w}_i$ and for gate $\mathcal{G}_j$ by $\mathbf{v}_j$. The conditional probability density of the model is given by:

$$P(Y|X,\boldsymbol{\theta}) = \prod_{n=1}^{N} \sum_i P(z_i^{(n)}=1|\boldsymbol{x}^{(n)},\mathbf{v}_0) \sum_j P(z_j^{(n)}=1|z_i^{(n)}=1,\boldsymbol{x}^{(n)},\mathbf{v}_i) \sum_k P(z_k^{(n)}=1|z_j^{(n)}=1,z_i^{(n)}=1,\boldsymbol{x}^{(n)},\mathbf{v}_j) P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)},z_k^{(n)}=1,z_j^{(n)}=1,z_i^{(n)}=1,\mathbf{w}_k). \tag{2.13}$$

If we write the prediction of expert $\mathcal{E}_k$ as $\hat{\boldsymbol{y}}_k^{(n)}$ and the k^{th} prediction of a gate as $g_k^{(n)}$, the expected value

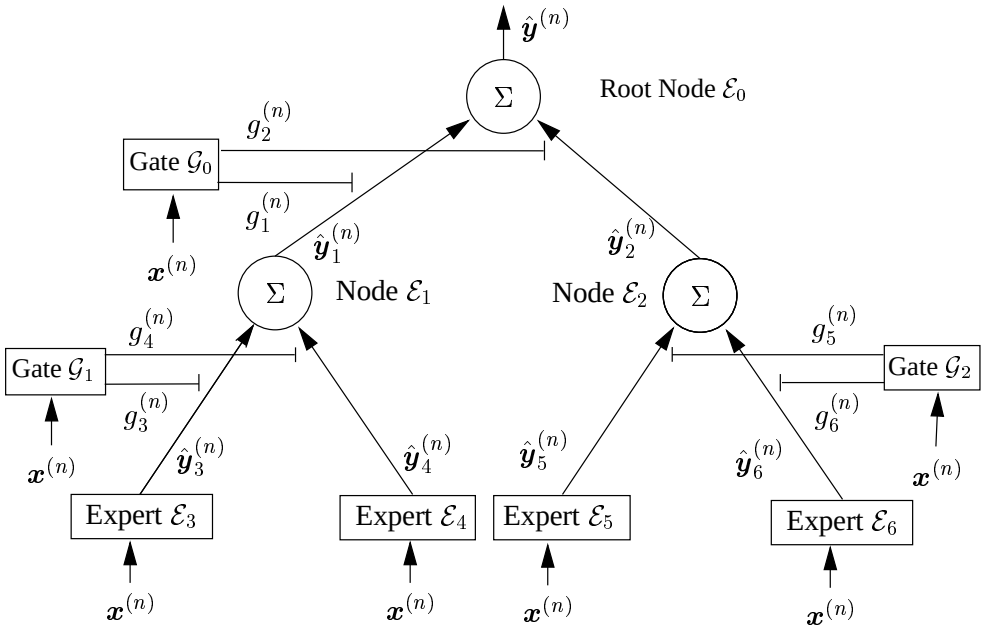

Figure 2.4: **Tree structure of a hierarchical mixtures-of-experts model.** The architecture of this model has binary branches at each non-terminal node, and a depth of 2. The output of the terminal experts $\mathcal{E}_3$ and $\mathcal{E}_4$ are given by $\hat{\boldsymbol{y}}_3^{(n)}$ and $\hat{\boldsymbol{y}}_4^{(n)}$. The outputs of the gate rooted at non-terminal node $\mathcal{E}_1$ are given by $g_3^{(n)}$ and $g_4^{(n)}$. The outputs of non-terminal node are then given by the convex combination of the gate and expert outputs: $\hat{\boldsymbol{y}}_1^{(n)} = g_3^{(n)}\hat{\boldsymbol{y}}_3^{(n)} + g_4^{(n)}\hat{\boldsymbol{y}}_4^{(n)}$. In a similar fashion the overall output of the model is given by $\hat{\boldsymbol{y}}^{(n)} = g_1^{(n)}\hat{\boldsymbol{y}}_1^{(n)} + g_2^{(n)}\hat{\boldsymbol{y}}_2^{(n)}$. Note that this HME model is represented in the belief network of Figure 2.3 if only two hidden states W^1 and W^2 are present, representing the decisions made at the two levels of the tree.

of $\boldsymbol{y}^{(n)}$ is given by[4] :

$$\begin{aligned}\hat{\boldsymbol{y}}^{(n)} &= E(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)},\boldsymbol{\theta}) \\ &= \sum_i P(z_i^{(n)}=1|\boldsymbol{x}^{(n)},\mathbf{v}_0)\sum_j P(z_j^{(n)}=1|z_i^{(n)}=1,\boldsymbol{x}^{(n)},\mathbf{v}_i) \\ &\sum_k P(z_k^{(n)}=1|z_j^{(n)}=1,z_i^{(n)}=1,\boldsymbol{x}^{(n)},\mathbf{v}_j)E(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)},z_k^{(n)}=1,z_j^{(n)}=1,z_i^{(n)}=1,\mathbf{w}_k) \\ &= \sum_i g_i^{(n)}\sum_j g_j^{(n)}\sum_k g_k^{(n)}\hat{\boldsymbol{y}}_k^{(n)}\end{aligned} \tag{2.14}$$

As shown in Figure 2.4, the output of the HME model is given by the combination of predictions of nodes with their parent gating network outputs.

Let us now introduce some terminology for the HME, which is motivated by the tree terminology common in machine learning [*e.g.,* 21]. For a general node $\mathcal{E}_p$ the following terminology will be used (see Figure 2.5):

- the immediate *parent* node: denoted by $Pa(\mathcal{E}_p)$, *e.g.,* the parent of node $\mathcal{E}_3$ is node $\mathcal{E}_1$;
- the *child* nodes, if the node is non-terminal, which are denoted by $Ch(\mathcal{E}_p)$ which have the property of all having parent node $\mathcal{E}_p$, *e.g.,* nodes $\mathcal{E}_3$ and $\mathcal{E}_4$ are the children of $\mathcal{E}_1$;
- the shortest path from $\mathcal{E}_p$ to the root node is the set of *ancestors* of $\mathcal{E}_p$, denoted by $An(\mathcal{E}_p)$;

If we also use a similar shorthand notation to that introduced for mixtures-of-experts:

$$P(z_k^{(n)}=1|z_j^{(n)}=1|z_i^{(n)}=1,\boldsymbol{x}^{(n)},\mathbf{v}) \equiv P(\mathcal{E}_k|An(\mathcal{E}_k),\boldsymbol{x}^{(n)},\mathbf{v}), \tag{2.15}$$

we may rewrite expression (2.14) as:

$$\hat{\boldsymbol{y}} = \sum_i P(\mathcal{E}_i|\boldsymbol{x},\mathbf{v}_0)\sum_j P(\mathcal{E}_j|An(\mathcal{E}_j),\boldsymbol{x},\mathbf{v}_i)\sum_k P(\mathcal{E}_k|An(\mathcal{E}_k),\boldsymbol{x},\mathbf{v}_j)E(\boldsymbol{y}|\boldsymbol{x},\mathcal{E}_k,An(\mathcal{E}_k),\mathbf{w}_k). \tag{2.16}$$

where I have dropped the superscript (n) from $\boldsymbol{y}^{(n)}$ and $\boldsymbol{x}^{(n)}$ for brevity. As in the mixtures-of-experts model, we will find the posterior probability of a node to be a useful quantity. For a given node $\mathcal{E}_i$, there are two posterior probabilities, the *joint* posterior probability of node $\mathcal{E}_i$ *and* its ancestors and the *conditional* posterior probability of node $\mathcal{E}_i$ *given* its ancestors. The joint posterior probability for node $\mathcal{E}_i$ is given by:

$$\zeta_i^{(n)} = P(\mathcal{E}_i, An(\mathcal{E}_i)|\boldsymbol{x}^{(n)},\boldsymbol{y}^{(n)},\boldsymbol{\theta}). \tag{2.17}$$

[4]In these equation I have avoided the use of multiple conditional subscripts on variables as was done by Jordan and Jacobs [112] in favour of a set description representation. This allows a more succinct description and generalises to arbitrary tree depths without confusion. For example, in the scheme of Jordan and Jacobs [112], node $\mathcal{E}_2$ might be represented by $\mathcal{E}_{1|1}$ and $\mathcal{E}_3$ and $\mathcal{E}_4$ by $\mathcal{E}_{1|1|1}$ and $\mathcal{E}_{2|1|1}$ respectively.

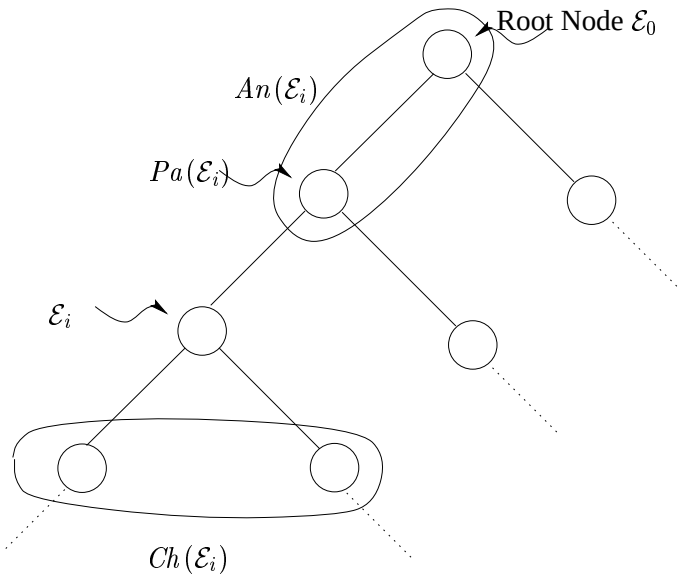

Figure 2.5: **Relationships between nodes in a hierarchical mixtures-of-experts model**

The joint posterior probabilities have the property that their sum over all nodes at the *same level* of the tree is equal to 1. In a similar fashion, the conditional posterior probability for node $\mathcal{E}_i$ is given by:

$$\varphi_i^{(n)} = P(\mathcal{E}_i | An(\mathcal{E}_i), \boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}, \boldsymbol{\theta}). \tag{2.18}$$

The conditional probabilities have the property that the sum of $\varphi_i^{(n)}$ over all *children* of $Pa(\mathcal{E}_i)$ is equal to 1:

$$\sum_{i \in Ch(Pa(\mathcal{E}_i))} \varphi_i^{(n)} \quad = \quad 1. \tag{2.19}$$

This concludes the discussion of the representation of the HME model. A number of concepts have been reviewed, including the structural and probabilistic specifications of the model either in a belief network or tree diagram. In the next section a method for training ME and HME models will be described. This uses the concept of hidden data in the models to efficiently factorise the training via the use of the expectation maximisation algorithm.

2.3 Training Mixtures and Hierarchical Mixtures-of-Experts

Unsupervised mixture models may be trained using a number of different techniques [224]. For example, the likelihood may be maximised directly using a gradient ascent method. Alternatively if a Bayesian method is used, Gibbs sampling may be employed to sample from the posterior distribution. In both these cases a useful factorisation of the likelihood can be made by the expectation maximisation (EM) algorithm of Dempster et al. [45]. In this section I review the application of the EM algorithm to ME and

HME models. This was first proposed by Jordan and Jacobs [112]. The EM algorithm is the method used in the remainder of the thesis to train ME and HME models. The EM algorithm is an attractive method since it enables the optimisation of a ME or HME model to be broken up into a set of optimisations, one for each expert and gate. In the remainder of this section I give a presentation of the EM algorithm as applied to mixtures-of-experts. This presentation is based on that of Jordan and Jacobs [112] but is extended to the case of a general hierarchy in this section. This extension is essential if the differences between the use of conditional and joint posterior probabilities in the algorithm are to be seen.

The EM algorithm and mixtures-of-experts

The EM algorithm is commonly used to train Gaussian mixtures and other mixture models. It is also prevalent in approaches to solving problems with missing data [45, 71]. The EM algorithm lends itself to problems in which data is missing or hidden. This missing data, denoted Z, may take any form but generally represents some knowledge which would make the problem easier to solve. In the case of mixtures-of-experts the missing data is chosen to be the identity of the process which generated the data *i.e.,* the set of indicator variables Z. The *complete* data likelihood is defined as the probability of the observed data $D = (X, Y)$ augmented by the missing data Z. Using the fact that Z is a multinomial random variable, we may write the complete data likelihood for a mixtures-of-experts model as:

$$L_c = P(Y, Z|X, \boldsymbol{\theta}) = \sum_n \sum_i \{P(\boldsymbol{y}^{(n)}, z_i^{(n)} = 1|\boldsymbol{x}^{(n)}, \boldsymbol{\theta})\}^{z_i^{(n)}} \tag{2.20}$$

which is distinct from the *incomplete* data log likelihood $\sum_n \log P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \boldsymbol{\theta})$ since the observed data is augmented with missing data. The concept behind the EM algorithm is that the values of the missing data Z are replaced by their estimated values given the current values of the parameters of the model. These estimated values are then treated as being provisionally correct and used to augment the observed data (X, Y). The resulting complete data log-likelihood is then maximised to give new parameter values. These new parameters are used to estimate new values for Z and the process iterated.

The EM algorithm increases the value of the incomplete data likelihood by iteratively maximising the expected value of the complete data log likelihood over successive distributions of the missing data $P(Z|X, Y, \boldsymbol{\theta}^{(t)})$. It does this by iteratively repeating two steps until convergence of the likelihood.

1. The algorithm starts with some value of the parameters $\boldsymbol{\theta}^{(t)}$. The *E step* postulates a distribution $P(Z|X, Y, \boldsymbol{\theta}^{(t)})$ over the missing data Z given this value of the parameters and the observed data $\{X, Y\}$.

2. The *M step* then finds the optimal parameters of the model that maximise the expected value of the complete data log likelihood, also known as the *auxiliary function* R:

$$R = \int P(Z|X, Y, \boldsymbol{\theta}^{(t)}) \log P(Y, Z|X, \boldsymbol{\theta}) dZ \tag{2.21}$$

   The parameters $\boldsymbol{\theta}^{(t+1)}$ that maximise this function are then used in the model and the E and M steps repeated.

In the case of an HME model with I terminal nodes, the complete data log likelihood is given by:

$$\log P(Y, Z, X|\boldsymbol{\theta}) = \sum_{n=1}^{N}\sum_{j=1}^{I} z_j^{(n)} \left\{ \log P(\mathcal{E}_j, An(\mathcal{E}_j)|\boldsymbol{x}^{(n)}, \boldsymbol{\theta}) + \log P(\boldsymbol{y}^{(n)}|\mathcal{E}_j, An(\mathcal{E}_j), \boldsymbol{x}^{(n)}, \boldsymbol{\theta}) \right\} \tag{2.22}$$

We use the fact that the expected value of $z_j^{(n)}$ over $P(Z|X, Y, \boldsymbol{\theta}^{(t)})$ is given by $P(\mathcal{E}_j, An(\mathcal{E}_j)|\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}, \boldsymbol{\theta}^{(t)})$. Let us now introduce the following shorthand notation for the joint and conditional posterior probabilities of node $\mathcal{E}_j$ given the set of parameters $\boldsymbol{\theta}^{(t)}$:

$$\zeta_j^{(n)}(\boldsymbol{\theta}^{(t)}) = P(\mathcal{E}_j, An(\mathcal{E}_j)|\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}, \boldsymbol{\theta}^{(t)}); \tag{2.23}$$

$$\varphi_j^{(n)}(\boldsymbol{\theta}^{(t)}) = P(\mathcal{E}_j|An(\mathcal{E}_j), \boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}, \boldsymbol{\theta}^{(t)}). \tag{2.24}$$

This gives the following auxiliary function for the HME:

$$R = \sum_{n=1}^{N}\sum_{j=1}^{I} \zeta_j^{(n)}(\boldsymbol{\theta}^{(t)}) \left\{ \log P(\mathcal{E}_j, An(\mathcal{E}_j)|\boldsymbol{x}^{(n)}, \boldsymbol{\theta}) + \log P(\boldsymbol{y}^{(n)}|\mathcal{E}_j, An(\mathcal{E}_j), \boldsymbol{x}^{(n)}, \boldsymbol{\theta}) \right\}. \tag{2.25}$$

We now separate the terms depending on the gate parameters from the terms depending on the expert parameters, since $P(\mathcal{E}_j, An(\mathcal{E}_j)|\boldsymbol{x}^{(n)}, \boldsymbol{\theta})$ can be separated into separate terms for each gate of:

$$P(\mathcal{E}_j, An(\mathcal{E}_j)|\boldsymbol{x}^{(n)}, \boldsymbol{\theta}) = \prod_{s \in \{\mathcal{E}_j, Pa(\mathcal{E}_j)\}} P(\mathcal{E}_s|An(\mathcal{E}_s), \boldsymbol{x}^{(n)}, \mathbf{v}_s), \tag{2.26}$$

where the product is over all gates. This gives the auxiliary function in the following form:

$$R = \sum_s R(g)_s + \sum_{j=1}^{I} R(e)_j \tag{2.27}$$

where $R(g)_s$ and $R(e)_j$ are the contributions to the auxiliary function from gate $\mathcal{G}_s$ and expert $\mathcal{E}_j$ respectively. These are given by:

$$\begin{aligned} R(g)_s &= \sum_{n=1}^{N} \zeta_s^{(n)}(\boldsymbol{\theta}^{(t)}) \sum_{j \in Ch(\mathcal{E}_s)} \varphi_j^{(n)}(\boldsymbol{\theta}^{(t)}) \log P(\mathcal{E}_j|An(\mathcal{E}_j), \boldsymbol{x}^{(n)}, \mathbf{v}_j), \\ R(e)_j &= \sum_{n=1}^{N} \zeta_j^{(n)}(\boldsymbol{\theta}^{(t)}) \log P(\boldsymbol{y}^{(n)}|\mathcal{E}_j, An(\mathcal{E}_j), \boldsymbol{x}^{(n)}, \mathbf{w}_j). \end{aligned} \tag{2.28}$$

The M step thus decomposes into the following separate maximisations:

- for each gate $\mathcal{G}_s$ maximise $R(g)_s$;
- for each expert $\mathcal{E}_i$, maximise $R(e)_i$.

The actual form that these maximisations take will depend on the choice of model for each expert and gate.

The EM training algorithm for HME models can be given an intuitive interpretation. The joint and conditional posterior probabilities of each node can be viewed as assigning *posterior credit* to each node. If a particular expert has a low probability of generating the targets Y given the inputs X, it will have low posterior credit, and hence will have only a small contribution to the auxiliary function. More importantly the expert will also be weighted less during the optimisation of the M step than other, better performing experts. The gate output for this expert will also be affected by the low posterior credit and will be adjusted to favour better performing experts. This process leads to a competitive process of expert selection during training which is embodied in the EM algorithm.

2.4 Review of Related Literature

We have now introduced the general mixtures-of-experts framework and the notation that will be used in this thesis. The specific framework for regression and classification used in this thesis is one in which each expert and gate is a GLIM. The theory behind this framework and its implementation is covered in Chapter 4. In this chapter I review the major related literature of mixtures-of-experts. This is divided into three sections: related work and inspirations, extensions of the ME model and formalism and finally applications of mixtures-of-experts. The aim of this review is first to place the ME in context with other statistical models, and second to present a background against which the original contribution of this thesis may be judged.

2.4.1 Related work and inspirations of mixtures-of-experts

The mixtures-of-experts model was first presented by Jacobs, Jordan, Nowlan and Hinton [104]. In that paper each expert was a single layer network trained using a squared error criterion to perform classification. Experiments on a small vowel classification problem (the Peterson and Barney vowel data) [175] indicated comparable performance with multi-layer perceptrons but with significantly faster training. More extensive studies of this model were performed by Nowlan and Hinton [160] [see also 159]. Nowlan and Hinton used multi-layer perceptron experts and gates and demonstrated significantly better generalisation than multi-layer perceptrons alone on the Peterson and Barney vowel data. Jacobs and Jordan [101] presented a summary of two applications of the ME: combined classification and spatial localisation of objects [see also 103], and control of a multi-payload robot [see also 102]). In all these applications the ME models were trained by direct gradient ascent of the log-likelihood.

The first description of a *hierarchical* mixtures-of-experts model was given by Jordan and Jacobs [111]. The HME was applied to a binary image classification problem and a system identification problem which involved learning the simulated dynamics of a four-joint, three dimensional robot arm (prediction of joint accelerations from joint positions, velocities and torques). Both the experts and gates used were single layer networks. On the image classification task the HME outperformed a CART model but was outperformed by an MLP. On the system identification task the HME outperformed both methods.

The learning algorithm was again based on gradient ascent of the log-likelihood.

The seminal paper on HME models was that of Jordan and Jacobs [112]. In this paper they presented the HME as a mixture model, and used this interpretation to motivate a learning algorithm based on the expectation-maximisation (EM) algorithm of Dempster et al. [45]. In addition, they strengthened the probabilistic framework of the experts and gates by considering them as generalised linear models (GLIMs) [140]. In a comparative study on system identification of a robot arm, the HME was outperformed by an MLP, but outperformed CART, MARS and linear regression. Using a Newton algorithm (iteratively re-weighted least squares (IRLS) [140]) for optimising the gates and experts, Jordan and Jacobs reported that the HME was "more than two orders of magnitude faster than back-propagation". However, it should be noted that IRLS is a second order optimisation technique and would therefore be expected to be faster than standard gradient descent in terms of epochs (*i.e.,* passes through the training data). Jordan and Jacobs do not give any details of sophisticated gradient descent techniques. If we assume that straightforward back-propagation via gradient descent was used their claim is somewhat misleading. A fairer comparison would be to use the same optimisers for back-propagation as for the HME, *e.g.,* a conjugate gradient algorithm.

The convergence properties of the HME were investigated by Jordan and Xu [113]. They showed that the convergence rate of the EM algorithm is linear with rate given by the condition number of the matrix $(P^{1/2})^T H (P^{1/2})$ where H is the Hessian of the log likelihood of the model and P is a diagonal matrix consisting of the diagonal elements of the covariance matrices of each of the parameters in the model, *i.e.,* $P = \text{diag}[\text{Cov}(\mathbf{v}), \{\text{Cov}(\mathbf{w}_i)\}]$.

The original mixtures-of-experts framework initially had some motivation in biological plausibility. Jacobs, Jordan and Barto [103] proposed a mixtures-of-experts model as a model of high-level vision. Recently Ghahramani and Wolpert [73] have proposed that human motor system uses a modular decomposition strategy to learn the visuomotor map and have presented empirical results on test subjects that suggest that the the human cognitive process may employ a modular decomposition strategy during learning.

The mixtures-of-experts framework has connections with many other statistical techniques including switching regressions, recursive partitioning, classification and regression trees and mixture models. The next section gives a brief description of some of these techniques and their connection with mixtures-of-experts.

Switching Regression

A switching regression model is one in which the observed y is assumed to have been generated by a number of distinct processes [224, pp.12]. One of the earliest references to switching regression models is that of Page [163] who investigates "problems in which a change in parameter occurs at an unknown point" [see also 164]. Page develops a test of the hypothesis that a set of data $\{x^{(n)}\}$ were generated by a single distribution $f(x^{(n)}|\boldsymbol{\theta})$ against the hypothesis that the data were generated by two distributions $f(x^{(n)}|\boldsymbol{\theta}_1)$ for $n < n^*$, and $f(x^{(n)}|\boldsymbol{\theta}_2)$ for $n \geq n^*$.

Quandt [182] considered the estimation of parameters of "a linear regression system obeying two

separate regimes", *e.g.*,

$$\hat{y}^{(n)} = \begin{cases} \mathbf{w}_1^T \boldsymbol{x}^{(n)} & \text{if } n < n^*; \quad \text{(regime 1)} \\ \mathbf{w}_2^T \boldsymbol{x}^{(n)} & \text{if } n \geq n^*, \quad \text{(regime 2)} \end{cases} \tag{2.29}$$

where n^* is the critical value at which the system changes from regime 1 to regime 2. Quandt developed a method for estimating the switching point of such systems by searching through all possible switch points and finding the maximum of an appropriate likelihood function. In later work Quandt [183] framed the problem in the following way. The density of target $\boldsymbol{y}^{(n)}$ given input $\boldsymbol{x}^{(n)}$ is considered as the sum of two terms:

$$P(y^{(n)}|\boldsymbol{x}^{(n)}) = \lambda P(y^{(n)}|\boldsymbol{x}^{(n)}, \mathbf{w}_1) + (1-\lambda) P(y^{(n)}|\boldsymbol{x}^{(n)}, \mathbf{w}_2) \tag{2.30}$$

where λ is the probability of regime 1 being chosen. In Quandt [183] the parameters of a 2 regime linear regression model were estimated using maximum likelihood. Quandt and Ramsey [184] summarised further results on this model and introduced the concept of *mixtures of regressions* to describe the switching regression model in terms of a finite mixture model. They used the connection between mixtures of Gaussians and mixtures of regressions to introduce the use of a moment generating function for estimation of the latter's parameters. In the comment to Quandt and Ramsey's paper, Hartley [79] described an EM algorithm for mixtures of regressions. It is interesting to note that this algorithm is similar to the EM algorithm for mixtures-of-experts of Jordan and Jacobs [112], apart from the specification of the mixing coefficient λ, which is dependent on the input $\boldsymbol{x}^{(n)}$ in the mixtures-of-experts, but is constant with respect to the input in the mixtures of regressions. This relationship was also noted by Jordan and Xu [113].

In more recent work on switching regression, Lau et al. [125] have developed a logistic regression model in which the function of the inputs obeys a switching regression:

$$P(y|\boldsymbol{x}, \boldsymbol{\theta}) = \begin{cases} 1/(1+\exp(\mathbf{w}_1^T \boldsymbol{x})) & \text{if } x_i < t_i; \\ 1/(1+\exp(\mathbf{w}_2^T \boldsymbol{x})) & \text{if } x_i \geq t_i, \end{cases} \tag{2.31}$$

where y is the target category, t_i is the unknown switch point for variable x_i and $\mathbf{w}_1$ and $\mathbf{w}_2$ represent the different parameter vectors associated with the 2 regimes. This may be considered as a mixtures-of-experts model in which the gate is a threshold function on one variable x_i.

The switching regression model of Quandt [182] was originally applied to econometric data. This model was extended by Hamilton [76] who considers the analysis of econometric time series "subject to changes in regime". Hamilton modelled time series over a set of m observations using a model with a joint probability:

$$P(\boldsymbol{y}^{(n)}|s^{(n)}, s^{(n-1)}, \ldots, s^{(n-m)}, \boldsymbol{y}^{(n-1)}, \boldsymbol{y}^{(n-2)}, \ldots, \boldsymbol{y}^{(n-m)}, \boldsymbol{\theta}), \tag{2.32}$$

in which the *state* variable $s^{(n)}$ obeys a first order Markov condition. Hamilton considered estimation of models with or without autoregressive components via the EM algorithm. The case of the Hamilton

model with autoregressive components is a mixtures-of-experts model in which the gate is a first order hidden Markov model (*c.f.*, [141, 110]).

Classification and Regression Trees

The HME has strong connections with classification and regression trees and recursive partitioning. These connections will be investigated in this section. A recursive partitioning procedure is defined as follows[58]:

$$\text{if } \boldsymbol{x} \in R_m, \text{ then } \hat{y} = f_m(\boldsymbol{x}) \tag{2.33}$$

where the M subregions $\{R_m\}_1^M$ partition the data D, and $\{f_m\}_1^M$ are a set of functions local to these subregions. The subregions are made up of nested subregions which are created by recursively splitting previous subregions. Starting with the overall data set D a split is made into a pair of subregions R_l and R_r according to the rule:

$$\begin{aligned} &\text{if } g(\boldsymbol{x}) < 0, \text{ then } \boldsymbol{x} \in R_l; \\ &\text{else } \boldsymbol{x} \in R_r, \end{aligned} \tag{2.34}$$

where $g(\boldsymbol{x})$ is a *splitting* function which decides whether to send the data into R_l or R_r. The functions f_m are generally chosen to be simple, and the most common is a constant function: $f_m(\boldsymbol{x}) = a_m$. Recursive partitioning can be considered as a regression *tree* in which subregions are represented by lower branches in the tree.

One of the first descriptions of a recursive partitioning procedure for regression was given by Breiman and Meisel [22]. They partitioned data into 2 distinct regions using a function $g(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x}^{(n)}$, where $\boldsymbol{\theta}$ is a *split* vector. They chose a split vector at random and adjusted its parameters such that it divided the data into half. In each of the resulting regions a linear regression was fit. If the squared error on the data was reduced significantly [5] by the split, the split was retained, otherwise another random split vector was chosen (a process which was continued for a fixed number of times, or until a good split was found). If the split was retained, the splitting process was continued until each region contained $2(d+1)$ or fewer points (where d is the dimension of the input space).

Friedman [57] also described a recursive partitioning procedure which used linear regression at the terminals. The split method was different to that of Breiman and Meisel, however, with the subregions defined by a threshold condition on one of the input attributes:

$$\begin{aligned} &\text{if } x_m < t_m \text{ then predict } \hat{y}_l; \\ &\text{else predict } \hat{y}_r, \end{aligned} \tag{2.35}$$

where $\hat{y}_l$ and $\hat{y}_r$ are the predictions of the left and right subregions respectively. This is equivalent to the choice of the split function $g(\boldsymbol{x}) = x_m - t_m$. Friedman also introduced the concept of using a backwards selection method for avoiding over-fitting of the regression tree. This concept was extended

[5] The significance of the reduction in squared error was measured by an F-test.

by Breiman et al. [21] who provided a definitive summary of the theory of Classification and Regression Trees (CART). CART models use piecewise constant functions at terminal nodes, and have predominately univariate splits. The search method used in CART is to *grow* (*i.e.*, recursively apply the splitting procedure) until the tree either reaches a pre-specified maximum size or no further improvement on the data can be achieved. This yields a tree which typically would overfit the data. To simplify the tree, and hence improve generalisation, a set of validation data is used to guide a *pruning* procedure which replaces non-terminal nodes by terminal nodes. Typically a cost-complexity term is also used in the pruning procedure to encourage small trees.

Quinlan [186] also described a tree based procedure, C4.5, which is used mainly for classification. Both C4.5 and CART have found wide usage in the machine learning and statistics communities. Tree models are popular both as a result of their good predictive accuracy and their degree of interpretability. There are two main extensions of tree based models which have relationships with hierarchical mixtures-of-experts: multivariate decision trees and soft splits. These will now be considered.

Multivariate decision trees

Standard trees such as CART are restrictive in that they only generally use one variable to make their splits. Multivariate decision trees generalise this to allow rules at non-terminal nodes to take the form:

$$\begin{aligned} &\text{if} \quad \sum_l a_l x_l + a_0 < t, \text{then predict } \hat{y}_l \\ &\text{else predict } \hat{y}_r \end{aligned} \tag{2.36}$$

where t is a threshold. These have been considered by Breiman et al. [21] and Brodley and Utgoff [26]. The gating network formalism of the mixture of experts is a generalisation of these concepts.

Breiman et al. [21, pp. 132-134] used a backward selection search method to delete attributes from the summation $\sum_l a_l x_l + a_0$, starting from the complete set of attributes $\{x_l\}_1^d$. Quinlan [186, pp. 96-98] also discussed this issue, concluding that although it is desirable to have linear combinations of attributes, there are two drawbacks. First, the search for linear combinations is typically slow in comparison with searches for univariate splits and second, the combination of attributes is less interpretable than univariate splits, especially if the overall space of attributes is large. Priors on small numbers of combinations are therefore desirable. Brodley and Utgoff [26] used a hill-climbing search procedure to find the parameters of the splitting rule, and reported that the resulting trees are more accurate than univariate trees.

Soft splits

Soft splits are those in which the split function is smoothed to allow for a continuous transition across a threshold t_j. These have been considered by a number of authors. Carter and Catlett [31] defined upper and lower cut points t_j^+ and t_j^- adjacent to the threshold t_j and used linear interpolation to smooth

between the following points:

$$\begin{aligned} &\text{if } x_j < t_j^-, \text{ choose } R_l \text{ with probability } 1; \\ &\text{if } x_j = t_j, \text{ choose } R_l \text{ or } R_r \text{ with probability } 0.5; \\ &\text{if } x_j > t_j^+, \text{ choose } R_r \text{ with probability } 1.0; \end{aligned} \tag{2.37}$$

The choice of t_j^+ and t_j^- was made using methods derived via empirical investigation. Quinlan [186, pp. 75-58] also discussed this method and suggested a method for choosing the cut points based on the distribution of misclassified training cases given the threshold and cut points. Furthermore, Ripley [195, pp. 239-240] has suggested use of a logistic function as the split function.

Multivariate adaptive regression splines

CART was also one of the motivations for Friedman's *multivariate adaptive regression splines* (MARS) [58]. Friedman considers CART as representing the predicted function as:

$$\hat{f}(\boldsymbol{x}) = \sum_{m=1}^{M} a_m B_m(\boldsymbol{x}), \tag{2.38}$$

where $\{B_m\}_1^M$ are a set of *basis functions*, and $\{a_m\}$ the set of piecewise constant functions of the terminal nodes. In CART the basis functions are products of Heaviside functions. A Heaviside function $H(a)$ takes the value 1 if $a \geq 0$, and 0 otherwise. These basis functions have the form $\prod_i H(s(x_i - t_i))$ where the product is over all levels of the tree and s is $+1$ or -1 depending on whether the split direction is left or right. MARS generalises regression trees by using truncated power spline functions in place of the Heaviside functions that partition the subregions of the tree. MARS also differs from trees in that it retains the parent basis function after a split has been made, allowing additive functions to be made. In the rejoinder to Friedman's MARS paper, O'Sullivan describes a smoothed version of CART that has connections with the mixtures-of-experts. *Smoothed* CART (SCART) has a similar form to CART but used basis functions of the form:

$$B_m = \frac{I_m}{\sum_m I_m} \tag{2.39}$$

so that $\sum_{m=1}^{M} B_m = 1$, and $I_m > 0$. The functions I_m are taken to be tensor products of B-splines. SCART models have a clear relationship with HME models, given that the basis function B_m has the form of a probability over lower splits in the tree, although in HME models B_m takes the form of a multinomial logit model.

The HME as a decision tree

The HME has many similarities to tree based models. Indeed, Jordan [108] describes the HME as a *probabilistic decision tree*. This section considers the connections and similarities between these models. In tree based terminology a regression HME may be considered as a multivariate regression tree with oblique soft splits in which the terminal regions estimate linear regressions rather than constant functions. As far as I am aware, there has been no other classification or regression tree system which used this exact form, although the individual components, *e.g.*, soft splits, oblique splits and multivariate decision trees, have been discussed in the literature, as described in the previous section. One reason for this could be the trend for *interpretability* of trees. In most classification and regression trees, the splits are made on only one input attribute (*e.g.*, is $x_j < t_j$? where t_j is a threshold for input attribute j) and the functions at the terminals are constants (*e.g.*, estimate y as $\hat{y}$ over the range of the terminal). These two conditions are potentially more interpretable than a multivariate split and a linear regression at the terminals. However, there is often a trade-off between interpretability and predictive accuracy. This point is discussed further in Chapter 8.

An additional difference between standard tree based models and HME models is the training methods. In HME models the architecture of the model, *i.e.*, the number of terminal and non-terminal nodes and their organisation, is fixed before training of its parameters. In standard tree based models such as CART, the structure of the model is inferred from the data by the process of recursive partitioning until convergence of the error on a training set. The tree is then pruned back by removal of terminal nodes till the minimum error is found on a separate validation set. During the growing process the parameters of the split functions and terminal nodes are inferred. This inference is straightforward if univariate splits and constant functions at the terminal nodes are used. In contrast the inference of parameters for HME models is more complex and is done after the structure has been fixed in advance.

Modular Neural Networks

The mixtures-of-experts framework is also related to the class of models known as *modular neural networks*. Loosely speaking a modular neural network contains a series of modules (analogous to experts) which are combined using some function (typically additive) and which are trained to focus on different parts of a problem. For example, Anand et al. [4] considered a modular neural network for polytomous classification in which a k-class problem was broken into k 2-class problems.

Bishop and Nabney [15] describes a class of models known as *mixture density networks* (MDN). In MDNs a mixture model is used to approximate the probability density of targets given inputs:

$$P(\boldsymbol{y}|\boldsymbol{x}) = \sum_{m=1}^{M} \alpha_m(\boldsymbol{x}) P(\boldsymbol{y}|\boldsymbol{x}, z_m) \tag{2.40}$$

where $\{z_m\}_1^M$ is a set of mixture components with mixing weights $\alpha_m(\boldsymbol{x})$. In the case of regression the mixture components are chosen to be Gaussian. The means and variances of the mixture are modelled by the outputs of a multi-layer perceptron. This is closely related to the "adaptive mixture of local

experts" model of Jacobs, Jordan, Nowlan and Hinton [104], with the difference being that Jacobs, Jordan, Nowlan and Hinton [104] model the mixture means using different experts, whilst Bishop and Nabney use a single network to model the set of mixture means and variances.

One restriction of mixtures-of-experts is that only one *cause* is assumed to have generated each target pattern in the data. In some problems it may be more desirable to have more than one cause or expert accounting for a single example. Models of this form are sometimes referred to as *factorial*, and have recently been considered in the literature. Dayan and Zemel [44] consider *multiple-cause* models that learn cooperation between inputs to generate targets. Cooperation was also considered by Nowlan [159] and Jacobs [98] in the original form of the mixtures-of-experts, but not in the context of multiple causes. Experimental analyses by both Nowlan and Jacobs showed that competitive training was more effective than cooperative training for mixtures-of-experts. Factorial learning has also been considered in the context of vector quantisers and hidden Markov models [72].

2.4.2 Extensions to the mixtures-of-experts model

Now that we have seen some of the work that is related to and has influenced the mixtures-of-experts formalism, let us consider some of the extensions to this formalism. I have divided these into *theoretical extensions* which have suggested alternative theoretical viewpoints of the framework, *structural extensions* which have proposed new models based on the mixtures-of-experts and finally extensions which have focussed on using the mixtures-of-experts framework for *classification* rather than regression.

Theoretical Extensions

Amari [3] presents a discussion in which he shows that the conditional distributions $P(Y, Z|X)$ of mixtures-of-experts belong to the exponential family. Amari notes that the *em* algorithm [see also 2] can thus be applied to the mixtures-of-experts. The *em* algorithm is motivated from an information geometric perspective. Amari [3] presented relationships between the E and M steps of the EM algorithm and the *e* and *m* projections of the *em* algorithm.

A generalisation of the EM algorithm from statistical physics is also considered by Yuille and Kosowsky [251]. This viewpoint is closely connected with that of Hathaway [82], Neal and Hinton [154] and Csiszár and Tusnády [41]. Yuille et al. [252] used this generalisation to describe a more general statistical physics based learning algorithm for mixtures-of-experts. They showed that deterministic annealing may be used to minimise the energy function:

$$E = -\frac{1}{\beta}\left(\sum_i Z_i\{\log P(Y|X, Z_i, \mathbf{w}_i) + \log P(Z_i|X, \mathbf{v})\} + \log P(\{\mathbf{w}_i\}_i^I, \mathbf{v})\right) \tag{2.41}$$

in which β is an extra free parameter in the minimisation which controls the "coarseness" of the solution found (*i.e.*, how many distinct regions are assigned to each expert). Typically β is started small (close to 0) and increased towards 1 according to an annealing schedule. The motivation for this scheduling of β is to avoid local minima in the search through parameter space. Deterministic annealing of a hierarchical *unsupervised* clustering model has also recently been considered by Miller and Rose [145].

Jacobs [99] considers alternative methods for combining probability assessments of experts using a *decision maker*. He separates these methods into two classes of aggregation methods: supra-Bayesian procedures and linear opinion pools. In supra-Bayesian procedures the expert opinions are treated as separate data sources with prior distributions, and are combined by a decision maker using Bayes' rule. In a linear opinion pools method the decision maker forms a linear combination of the expert opinions. Jacobs shows that linear opinion pools work best when the experts have relatively *independent* opinions, such as the experts in a mixtures-of-experts model. In a bias-variance analysis of ME models, however, Jacobs [100] demonstrates empirically that the mixtures-of-experts training procedure generates *biased* experts whose estimates are *negatively correlated*. The overall bias of the ME models was, however, found to be close to zero. Jacobs also validates the claims of Jordan and Jacobs [112] that the variance of a ME model with soft splits is lower than one with hard splits (and thus more like a CART model).

The statistical mechanics of the mixtures-of-experts has been considered by Kang and Oh [114] who show that there exists a phase transition related to the symmetry of the model. Below the phase transition there are too few examples to allow the gate to break symmetry and the expert parameters are similar. Above the phase transition the gate learns the partition of the data and the experts specialise. Kang et al. [115] also generalised this work to a committee of generalised linear models.

Bayesian inference

The standard EM algorithm for mixtures-of-experts is a maximum likelihood algorithm. Often, in order to control complexity, it is desirable to use Bayesian inference instead of maximum likelihood inference. In general mixture modelling, Hathaway [81] considered constrained maximum likelihood for normal mixtures. Mixtures from the exponential family such as normal mixtures are straightforward to apply priors to [*e.g.,* 27], since they have sufficient statistics and simple conjugate priors [13]. Most formulations of mixtures-of-experts, however, are less straightforward, since the probability densities for the gates and experts do not have conjugate priors.

Peng et al. [172] presented a Bayesian inference framework for mixtures-of-experts which used Gibbs sampling [68]. Applications of this framework to classification of a small vowel data set yielded impressive results, although the time to train these models may be prohibitive for large data sets. Jacobs et al. [105] extended this framework to perform limited model selection based on evaluating the averaged posterior contribution of each expert using the Gibbs sampler.

An empirical Bayesian framework for inference in mixtures-of-experts was presented by Waterhouse, MacKay and Robinson [235]. This work is presented and extended further in Chapter 7. This method generalises the EM algorithm via the formulation of Neal and Hinton [154] to produce a free energy quantity which is alternately minimised with respect to parameters, hyper-parameters and node assignments. Related work on Bayesian inference of self organising maps by Utsugi [228, 229] also considered the EM algorithm from this viewpoint.

Structural Extensions

Now that we have considered some of the theoretical extensions to the ME framework let us now

turn to some of the structural extensions that have been motivated by this framework. First I consider some miscellaneous alternative models which have been inspired by the mixtures-of-experts or arrived at independently. Second I describe the extension of the ME framework to model sequential processes through the addition of *recurrency*. Finally I review the *localised* mixtures-of-experts model which is a popular alternative to the standard model for regression.

Alternative models

A similar approach to the ME formalism is that of Leite and Hancock [129] who use local splines to perform contour organisation on images. Their model used a Gaussian mixture defined over the set of spline coverings. The splines are limited in extent by a Gaussian windowing function in a similar fashion to the gating component of a mixtures-of-experts model.

An incremental learning algorithm known as *receptive field locally weighted regression* was presented by Schaal and Atkeson [210]. This algorithm consists of linear regression experts and a set of Gaussian receptive field units which gate the expert outputs. Each of the experts are trained independently in this algorithm.

Pawelzik et al. [171] consider the problem of segmentation of a time series by an ensemble of radial basis functions, which they describe as a mixture of experts *without* a gating network. They derive a weighting factor for each expert in the ensemble based on its likelihood of explaining the current sample in the time series, and show good segmentation of switching time series. Although the focus is on segmentation, they also show better prediction accuracy on a real time series.

An interesting alternative derivation of a mixtures-of-experts model for classification is the *stochastic neural nets* of Nadas [151]. Nadas derives an EM algorithm for this class of models which used sigmoid functions for experts and a set of Gaussian densities for the gate. The derivation is made without reference to the mixtures-of-experts and seems to have been done independently from this work.

Recurrency

The standard mixtures-of-experts model is designed for modelling of static data. In the case of time series data it seems likely that the mixtures-of-experts might perform well given that time series often move from one regime to another — regimes that could conceivably be modelled by different experts. In order to generalise the framework to model time series, a number of approaches have been proposed. Most of these focus on reformulating the gating network so that it has a dependency on previous observations.

The mixtures of controllers model of Cacciatore and Nowlan [30] used a recurrent network to gate a set of experts which are linear or non-linear controllers. The gate is a recurrent network in which the input is made up of $\boldsymbol{x}$ and its output vector $\boldsymbol{g}$. In this model the indicator variable $z_i^{(n)}$ for each expert obeys a first order Markov assumption: $P(z_i^{(n)} = 1 | \boldsymbol{z}^{(n-1)}, \boldsymbol{x}^{(n)})$. An on-line gradient ascent procedure is used to maximise the likelihood of this model. Cacciatore and Nowlan show that this model is capable of controlling plants which contain switching or jump effects more effectively than a standard mixtures-of-experts model.

The input-output HMM (IOHMM) of Bengio and Frasconi [10] is a further generalisation of the mixtures-of-controllers model. The IOHMM consists of I experts which predict the target $\boldsymbol{y}^{(n)}$ given the current input $\boldsymbol{x}^{(n)}$, and whose outputs are gated by the outputs of a gating network. The difference with the standard mixtures-of-experts model lies in the gating network. The gate in the IOHMM consists a recurrent mixture of experts. The aim of this architecture is to factor learning into a *temporal* component, which is learnt by the gate and a *static* component, which is learnt by the experts. Modularity is encouraged in both components by the use of the mixtures-of-experts framework. Bengio and Frasconi [10] described an application of the EM algorithm to learn parameters of IOHMM models. This has intuitive connections with both the EM algorithm for mixtures-of-experts models and the Baum Welch algorithm [8] of hidden Markov models. Other applications of recurrent mixtures-of-experts models are in motion estimation and scene segmentation for computer vision [243], and in control Meilă and Jordan [141].

Potentially the most sophisticated extension of the HME to include recurrency is that of the hidden Markov decision tree (HMDT) of Jordan et al. [110]. The HMDT marries a hidden Markov model with an HME and the result can be viewed as a sequence of HME models in which the decision about which path to take in the tree is conditioned on the decisions in the HME at the previous time. Alternatively the HMDT can be viewed as a HMM with factorised state variables which are coupled in a decision tree structure. This complicated structure is optimised using the mean field framework of Saul et al. [207].

Localised mixtures-of-experts

An interesting extension of the mixtures-of-experts framework is the *localised* mixtures-of-experts model of Xu et al. [248]. In this model, the multinomial logit model used in the gate of standard ME models is replaced by a set of Gaussian basis functions, each localised to a specific expert. The prediction of the overall model is therefore:

$$E(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)},\boldsymbol{\theta}) = \sum_{i=1}^{I} \frac{\alpha_i \phi_i(\boldsymbol{x}^{(n)})}{\sum_{j=1}^{I} \alpha_j \phi_j(x)} E(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)},\mathbf{w}_i,\mathcal{E}_i) \tag{2.42}$$

where ϕ_i is a multivariate Gaussian with mean $\boldsymbol{\mu}_i$ and covariance matrix Σ_i and α_i is a prior probability on this expert. Xu et al. [248] note that direct maximum likelihood estimation of this model is not possible in a closed form, and so maximise the joint probability $P(Y,X)$ via the EM algorithm. This joint probability is given by:

$$P(Y,X) = \prod_n \sum_i P(\boldsymbol{x}^{(n)}|\mathbf{v}_i,\mathcal{E}_i) P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)},\mathbf{w}_i,\mathcal{E}_i) \tag{2.43}$$

where $P(\boldsymbol{x}^{(n)}|\mathbf{v}_i,\mathcal{E}_i) = \phi_i(\boldsymbol{x}^{(n)})$. The use of Gaussian basis functions as experts for classification has also been considered by Fritsch [60]. Other researchers have found the localised ME model to be successful, including Ramamurti and Ghosh [187].

Classification using mixtures-of-experts

Classification was considered in the original forms of the mixtures-of-experts model by Nowlan [159] and Jacobs, Jordan, Nowlan and Hinton [104] who applied the model to the Peterson-Barney vowel data.

On this data they found that the model gave good predictive performance, outperforming an MLP in empirical comparisons [160], but also gave intuitive segmentation into regions with similar types of vowels.

As described in Chapter 1, within the generalised linear model framework [140], the multinomial logit function (or *softmax* function [23]) which is encapsulated in a multinomial logit model, is an appropriate one for dichotomous or polytomous classification. The use of multinomial logit models as experts in an HME model was first suggested by Jordan and Jacobs [112]. Waterhouse and Robinson [236] subsequently applied this model to simulated classification tasks. Peng et al. [172] also used this form of model on experiments with the Peterson-Barney data but employed a Bayesian inference method via Gibbs sampling.

Alpaydin and Jordan [1] used the multinomial logit for the experts in their mixtures-of-experts model but employed the localised gating network form of Xu et al. [248]. In an empirical comparison, they demonstrated performance on par with radial basis functions and MLPs on a handwritten digit problem. Other alternative structures for classification via HME models have been proposed by Fritsch [60] who used normalised Gaussian radial basis functions as gates (as in the localised ME) *and* experts. The MultiClass model [200] is another variant on this idea, but motivated from a Bayesian perspective as a supervised version of the AutoClass model [33].

A framework for handling missing data in classification was also developed by Miller and Uyar [146]. This model takes the form of a mixture model in which the mixture components are counts and the mixture weights given by a normalised Gaussian basis function. This model for the gate is similar to the localised mixtures-of-experts model [248], and the overall framework is related to, but less general than that of Tresp et al. [227]. Finally, the mixtures-of-experts framework has also been applied to the problem of combining multiple, pre-trained, classifiers [249].

2.4.3 Applications of mixtures-of-experts

Applications of the mixtures-of-experts model have been widespread in the last few years, although it is fair to say that there have been few applications in large scale industrial problems. In addition there has been little investigation into how mixtures-of-experts models perform in comparison to other statistical models on a *range* of tasks. This leaves a significant hole in the understanding of the applicability of the mixtures-of-experts framework that will be addressed in the first part of the thesis.

Two successful areas of application of mixtures-of-experts models are in speech recognition and time series prediction. Both these areas involve modelling sequential data in which there is a potential for multiple processes to be present. For example in prediction of financial stocks there is evidence of switching behaviour between different regimes [76]. This section considers general applications of mixtures-of-experts models and applications in time series prediction. Speech recognition will be considered further in Chapter 9.

General applications

One of the first applications of the mixtures-of-experts was in control of dynamical systems [102].

Mixtures-of-experts are likely to be successful in control problems where a change in plant dynamics occurs. Murray-Smith and Johansen [150] contains a good review of multiple model approaches to control which contain many models similar in form to mixtures-of-experts. Kecman [116] also considered the ME for system identification of a car engine throttle valve position.

Control of dynamic systems via reinforcement learning (see Sutton [217] for a good review) has been a focus of connectionist research in the last decade. Applications of mixtures-of-experts to reinforcement learning have attempted to encourage decomposition of complex tasks into simpler ones, thus hopefully learning faster and more accurately. Tham [220] used an HME in which cerebellar model articulation controller (CMAC) networks are used as gates and experts. Compositional Q-learning [214] was used to successfully decompose composite tasks into their elemental tasks within this model. Sabes and Jordan [203] also used a mixtures-of-experts model for reinforcement learning, with a training algorithm based on matching the model's probability distribution with a probability distribution based on the reward signal from the environment.

Miscellaneous applications of HME models include the work of Barlow [7] who used an HME to predict protein structure, Bregler and Malik [19] who used hierarchical mixtures of *generalised second moments* to classify images of cars based on their local texture features, and Hu et al. [93] who used a ME model to classify ECG beats. Mixtures of linear models have been used for handwritten digit recognition by Hinton et al. [88]. Finally, Hering et al. [83] used an HME framework for VLSI design.

An approach to Kalman filtering based on mixtures-of-experts is presented by Chaer et al. [32], in which each expert is a Kalman filter. The Kalman filtering framework is closely related to the "on-line" algorithms for recursive identification of HME models [112]. Related on-line algorithms have also been considered by Tham [219].

Applications to time series prediction

As described previously, one of the original motivations for switching regression models was to identify switching patterns in econometric time series. It is unsurprising therefore that mixtures-of-experts have found many applications in time series prediction, both as a generalisation of autoregressive models and as an alternative to traditional connectionist models such as MLPs.

As far as I am aware the first application of mixtures-of-experts to time series prediction was by Waterhouse and Robinson [237] who used linear regression experts gated by a multinomial logit model to predict future values of a sunspots time series, with results comparable to two other techniques: MLP models [241] and the threshold autoregressive (TAR) model of Tong and Lim [225]. The TAR is itself closely related to switching regression models and mixtures-of-experts. More extensive work on time series prediction using ME models has been carried out by Zeevi, Meir and Adler [256] (see also [254] for a fuller description), including applications to sunspots and other time series. They also developed a universal approximation proof for mixtures-of-experts [255] which suggests that ME models are capable of approximating the same functions as MLP models. Comparisons of radial basis functions with HME models on a variety of time series including stock prices have also been done by Nikovski [156, 157].

An interesting application of the mixtures-of-experts framework to time series prediction was de-

scribed by Bengio et al. [9]. In this work each expert consisted of an MLP with the same number of hidden units but different lag values (the number of previous time series values used to predict the next value). They showed good performance of the model in terms of predictive accuracy and also showed how the different experts partitioned the series in an intuitive fashion. Their use of different models for each expert is an interesting one which could be useful in situations where the most appropriate model for a problem is not known.

A related class of models to mixtures-of-experts known as predictive modular neural networks (PREMONNs) have been studied by Kehagias and Petridis [117] for time series classification, *i.e.*, the segmentation of time series into distinct regions. The model is similar to the mixtures-of-experts but employs a recursive method, the *partition* algorithm [176], for estimation of the mixing factors of the experts' predictions.

Finally Weigend et al. [242] considered the use of MLPs as gates and experts in an architecture they called *gated experts* [see also 240]. Weigend has applied gated experts to various problems, including prediction of a laser time series, to good effect.

2.5 Conclusions

In the previous sections I have introduced the mixtures-of-experts framework. I have described related techniques such as switching regression models and decision trees. I have also given an overview of selected applications and theoretical extensions of mixtures-of-experts. In the remainder of the thesis I will build on this introduction. The aim of this work is to place the mixtures-of-experts model in context with other pattern recognition algorithms and to investigate their practical usage.

Part I

Extensions of Mixtures-of-Experts and their Evaluation using DELVE

Chapter 3

Introduction to Part I

The aim of Part I of this thesis is, first, to describe three learning methods based on mixtures-of-experts, and second, to compare these methods with other established learning methods. The point of the comparisons is to place mixtures-of-experts in their correct context with respect to other machine learning techniques. The comparisons are performed within a common framework known as DELVE, which is described further in this chapter.

Comparison of learning methods requires careful planning and consideration. This chapter describes DELVE, which is the framework which I chose to evaluate my learning methods. DELVE is an attractive suite to use since it implements a consistent framework for comparisons. This allows fair comparisons to be made with other algorithms and ensures a degree of consistency in testing and evaluation. Because of the framework of DELVE and the sheer number of comparisons done in this part of the thesis, I found it necessary to create completely automatic learning algorithms - *i.e.*, ones which required no human intervention. There are a number of issues to consider when designing these automatic learning algorithms which are considered in this chapter.

A common criticism of machine learning research, especially that done in the neural networks community, is that accounts of algorithms and techniques are not descriptive enough [178]. Notable exceptions include the excellent description of C4.5 [186]. My aim in this thesis has been to describe my algorithms carefully, including descriptions of implementation wherever necessary. I stop short of including source code for the learning methods, although I have made it publically available (see Appendix A), but do include pseudo-code where appropriate. This should make it possible for future researchers to re-implement my methods if necessary.

3.1 Issues in Comparing Learning Methods

Statistical comparison of learning algorithms has become the focus of a number of articles in the last few years. Cohen [37] described empirical methods for evaluating general AI programs and gave a good overview of the field. Prechelt [179] discussed the problem of statistical comparison in the context of neural network research [see also 178]. The “Prechelt test” is defined as the test of whether a research article contains more than one real world example:

> *An algorithm evaluation is called acceptable if it uses a minimum of two real or realistic problems and compares the results to those of at least one alternative algorithm.* Prechelt [179]

In Prechelt's research he found that many of the current research articles in neural network journals failed this test. Similar results for computer science in general were obtained by Tichy et al. [223].

A number of comparison studies have been performed for different learning algorithms including [47, 91, 177, 193, 213]. A comprehensive review of many comparison studies is given in Michie et al. [144]. StatLog [144] is perhaps the most well known. In the StatLog project, comparisons were made between a number of learning algorithms on both classification and regression tasks. King et al. [119] also summarised the classification portion of StatLog. The majority of the data sets used in StatLog were natural. The comparison procedure used in StatLog was either to train on a fixed portion of the data and test on an independent set in the case of large data sets, or to use 10-fold cross validation in the case of smaller data sets.

Considerable care must be taken when comparing algorithms. Simply stating that a particular algorithm gives an improvement is not sufficient. The improvement must be shown to be *significant*, according to an appropriate test methodology. Flexer [54] notes that very few neural network research publications describe significance tests between their proposed algorithms and existing ones. Experimental design is therefore of paramount importance when comparing learning methods. Salzberg [205] recommends an approach for comparing classifiers based on cross validation and the use of a binomial test to assess significance. Rasmussen [189], however, stresses that cross validation produces dependent training sets which make statistical comparisons difficult. Rasmussen therefore proposes the use of disjoint training sets and either a common, or a set of disjoint, test sets. The results of these experiments are analysed using an Analysis of Variance (ANOVA) method[1]. This scheme is implemented in the DELVE framework which is the framework that I use in this part of the thesis. DELVE will be described further in Section 3.2.

One of the problems with a number of the data sets used in comparisons is that they are too simple. The UC Irvine (UCI) collection of data sets [143] is a common source of data for comparisons. Holte [92] reports results on some of the most popular of the UCI data sets which show that the average performance of an extremely naive classifier is comparable with the performance of much more sophisticated classifiers such as CART. Cohen [37, pp. 80] denotes this the "ceiling effect" and explains it by the fact that the test data is too easy to solve, therefore making comparisons between algorithms difficult to perform. A conjugate effect occurs when the data sets are too difficult, denoted by Cohen as the *floor* effect. An example of this effect can be seen in the DARPA Switchboard speech recognition evaluations [75]. In this task, word error rates of over 40% are the norm. In such a situation it is difficult to assess whether the alteration of a component of the model results in improved performance since the baseline error rate is so high.

[1] See Cohen [37, pp.192-194] for an accessible introduction to ANOVA.

3.2 The DELVE Framework

> *The aim of DELVE is to help researchers and potential users to assess learning methods in a way that is relevant to real-world problems and allows for statistically-valid comparisons of different methods. Improved assessments will make it easier to determine which methods work best for various applications, and will promote the development of better learning methods by allowing researchers to easily determine how the performance of a new method compares to that of existing methods.* Rasmussen et al. [190]

DELVE is an attempt by researchers at Toronto University, including Rasmussen et al. [190], to overcome some of the previous problems with comparison of learning methods. DELVE provides a framework in which statistical comparison between different methods can be evaluated in a consistent manner. It consists of a set of tools which both designs an experiment for the user and assess the results of the method after it has been run through the experiment.

There is various terminology associated with DELVE which will be used in this part of the thesis. Firstly, each data set may contain many *prototasks*. A prototask is a supervised learning problem consisting of certain target attributes to be predicted from the values of input attributes. A single prototask may have many *tasks* which generally differ in the amount of training data (and potentially in the amount of prior information specified, although this aspect is not explored in this thesis). Each task then contains a number of *task instances* each of which has a *training set* and a *test set*. The task instances all contain disjoint training sets and either disjoint test sets in the case of large data sets (a *hierarchical* testing scheme) or overlapping test sets in the case of smaller data sets (a *common* testing scheme). This framework is illustrated in Figure 3.1. As an example, consider the Boston housing data set described in Section 3.3. The data set is given the name Boston and the prototask studied in this thesis is the prediction of house prices, Boston/price. This prototask has three tasks, Boston/price/std.32, Boston/price/std.64 and Boston/price/std.128 which have 32, 64 and 128 data points in their training sets respectively. Each of these tasks contains 8, 4 and 2 task instances respectively.

The hierarchical testing scheme uses a hierarchical Analysis of Variance (ANOVA) design and tests significance via a t-test. The common testing scheme uses a 2-way ANOVA design and a F-test of significance. Since the hierarchical testing scheme is the simpler of the two, I will restrict my discussion to this. Further details of both schemes and other aspects of DELVE can be found in either Rasmussen [189] or the DELVE manual [190].

To assess a learning method via DELVE, the first step is to generate the tasks and train the method on each of the training sets in each task instance and generate predictions for each of the test sets. The tools of DELVE are then used to compute the expected loss and standard error of this loss for each of the task instances in a task. For a set of J cases in each of I task instances, the average loss for task instance i and the overall average loss is given by [189]:

$$\bar{y}_i = \frac{1}{J}\sum_j y_{ij}, \quad \text{and} \quad \bar{y} = \frac{1}{IJ}\sum_i \sum_j y_{ij} \tag{3.1}$$

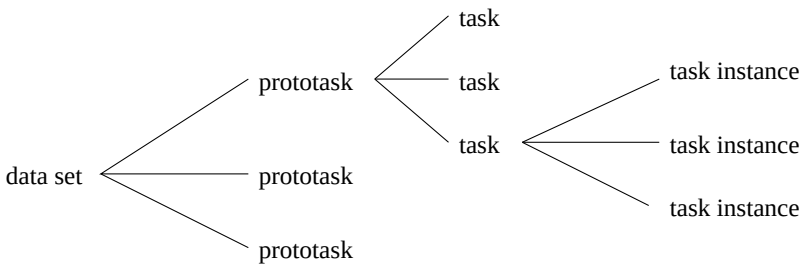

Figure 3.1: **A schematic diagram illustrating the structure of the DELVE framework.**

where y_{ij} is the loss on case j of task instance i. For hierarchical testing schemes, the standard error is given by:

$$SE = \left(\frac{1}{I(I-1)} \sum_i (\bar{y}_i - \bar{y})^2 \right)^{\frac{1}{2}}. \tag{3.2}$$

When comparing two learning methods the null hypothesis we wish to test is that the average losses of the two methods are equal. In hierarchical testing schemes a t-test on this hypothesis is used. If the *difference* between the losses on case j of task instance i is given by z_{ij}, the t-statistic may be written as:

$$t = \bar{z} \left(\frac{1}{I(I-1)} \sum_i (\bar{z}_i - \bar{z})^2 \right)^{-\frac{1}{2}} \tag{3.3}$$

which has $I - 1$ degrees of freedom. If the p-values resulting from this t-statistic are low, they indicate that the observed difference between the learning methods is unlikely to be due to chance. If $p < 0.05$ we say that the difference between the methods is *significant*.

3.3 The Data Sets used in Part I

In Chapters 5 to 7 five new learning methods based on the mixtures-of-experts framework are introduced. Comparison of these methods with each other or with existing methods in DELVE is deferred until Chapter 8. However, some of the data sets in DELVE are used to illustrate the behaviour of the methods during Chapters 5 to 7. For this reason the DELVE data sets are introduced in this chapter. Before this, however, let us introduce one of the artificial data sets that was used for design and validation of the methods.

An artificial data set

Many of the design issues described in Chapters 5, to 7 require empirical tests. During the design of each of the five methods I used a variety of simple data sets to validate their performance. I avoided the use of the DELVE data sets, using instead artificial data sets, such as the one shown in Figure 3.2 which is used in the rest of this part as an example. Whilst the use of artificial data sets is no substitute for real or realistic data, I expect that the methods resulting from their use does not bias them overly to such data. In any learning method design process, however, some feedback from empirical trials is essential and typically affects the design choices made. It is hoped that the insights that I provide in this part of the thesis and in the results of Chapter 8 will benefit future work into mixtures-of-experts learning methods.

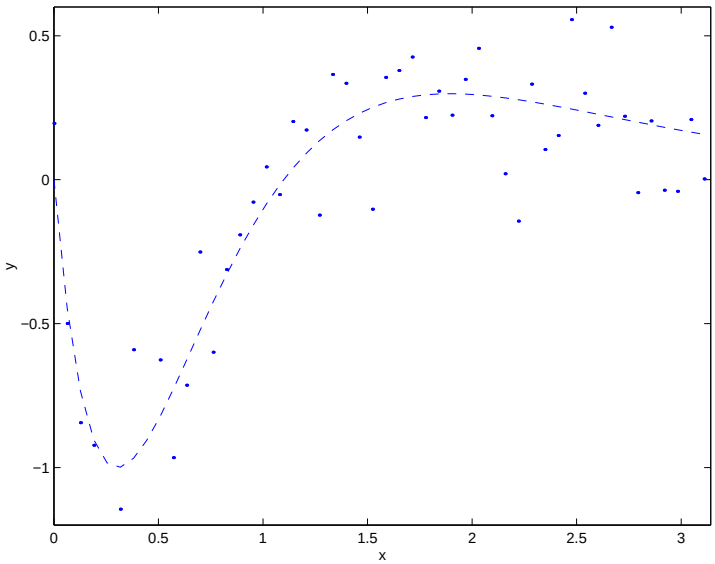

Figure 3.2: **An artificial data set.** This data set is based on an example by Wahba & Wold (1975). It consists of 50 points generated by adding zero-mean Gaussian noise with standard deviation 0.2 to the function $y = 4.26(\exp(-x) - 4\exp(-2x) + 3\exp(-3x))$.

The DELVE data sets

The data sets used can be divided into regression and classification tasks. These are shown in summary form in Tables 3.1 and 3.2. Further details are given in the remainder of this section. All the data sets were taken from the DELVE archive. The data sets were chosen as being the ones on which the largest number of comparisons had already been performed. On the regression tasks seven other methods have previously been investigated by Rasmussen [189]. On the classification tasks three other methods have been investigated by Mike Revow[2]. These methods are described in Chapter 8. Although other studies such as [144] have used more data sets and compared more methods, I chose the DELVE framework and the data sets shown since I believe DELVE provides a consistent manner for future researchers to compare my results with theirs. Hopefully as more data sets and methods are added to DELVE the methods and results described in this thesis will be able to be judged in a wider context. At present, however, it is possible to still judge the results in comparison with the existing results in DELVE on the data sets described here.

For the regression tasks the only natural data set used is the Boston housing data [78]. The other data sets represent particular versions of two families of data sets: the Kin family and Pumadyn family. These are realistic simulations of the kinematics and dynamics of robot arms. The Kin and Pumadyn data sets span a range of different types of problems: from medium to high noise and fairly linear to non-linear interactions. These allow some conclusions to be drawn about the specific behaviour of different methods on these different tasks. The classification tasks consist of three data sets, all of which are real data and have been used in previous studies. Results on both the regression and classification tasks are given in Chapter 8, although some of the data sets are used as examples in Chapters 5, 6 and 7.

[2]Details available at http://www.cs.utoronto.ca/∼delve/.

Name	Total Cases	Training Set Sizes	Test Set Size	Inputs	Selection Method (H/C)
Boston	506	32, 64, 128	250	13	C
Kin-32fh	8192	64, 128, 256, 1024	4096	32	H
Kin-32fm	8192	64, 128, 256, 1024	4096	32	H
Kin-32nh	8192	64, 128, 256, 1024	4096	32	H
Kin-32nm	8192	64, 128, 256, 1024	4096	32	H
Kin-8fh	8192	64, 128, 256, 1024	4096	8	H
Kin-8fm	8192	64, 128, 256, 1024	4096	8	H
Kin-8nh	8192	64, 128, 256, 1024	4096	8	H
Kin-8nm	8192	64, 128, 256, 1024	4096	8	H
Pumadyn-32fh	8192	64, 128, 256, 1024	4096	32	H
Pumadyn-32fm	8192	64, 128, 256, 1024	4096	32	H
Pumadyn-32nh	8192	64, 128, 256, 1024	4096	32	H
Pumadyn-32nm	8192	64, 128, 256, 1024	4096	32	H
Pumadyn-8fh	8192	64, 128, 256, 1024	4096	8	H
Pumadyn-8fm	8192	64, 128, 256, 1024	4096	8	H
Pumadyn-8nh	8192	64, 128, 256, 1024	4096	8	H
Pumadyn-8nm	8192	64, 128, 256, 1024	4096	8	H

Table 3.1: **The regression tasks considered in this study.** The table shows the total number of cases available in each data set (Total Cases) and how these are divided into training sets (Training Set Sizes) and test sets (Test Set Size). Also shown are the number of inputs in the task; all regression tasks have univariate targets. Finally the selection method is marked as C for common and H for hierarchical. This denotes the method used to select training and test sets. See Section 3.2 for further details on selection methods.

Name	Total Cases	Training Set Sizes	Test Set Size	Inputs	Classes	Selection Method (H/C)
Splice	3175	100, 200, 400	1575	61	3	C
Image	2310	70, 140, 280	1190	19	7	C
Letter	20000	390, 780, 1560	10640	16	26	H

Table 3.2: **The classification tasks considered in this study.** The table shows the total number of cases available in each data set (Total Cases) and how these are divided into training sets (Training Set Sizes) and test sets (Test Set Size). Also shown are the number of inputs and the number of classes in each task. Finally the selection method is marked as C for common and H for hierarchical. This denotes the method used to select training and test sets. See Section 3.2 for further details on selection methods.

Boston

Inputs			
Name	c/u	Range	Description
CRIM	u	$\{0:\infty\}$	per capita crime rate by town
ZN	u	$\{0:100\}$	proportion of residential land zoned for lots over $25,000$ sq.ft.
NDUS	u	$\{0:100\}$	proportion of non-retail business acres per town
CHAS	u	$\{0/1\}$	Charles River dummy variable (1 if tract bounds river; 0 otherwise)
NOX	u	$\{0:\infty\}$	nitric oxides concentration (parts per 10 million)
RM	u	$\{0:\infty\}$	average number of rooms per dwelling
AGE	u	$\{0:100\}$	proportion of owner-occupied units built prior to 1940
DIS	u	$\{0:\infty\}$	weighted distances to five Boston employment centres
RAD	u	$\{0:\infty\}$	index of accessibility to radial highways
TAX	u	$\{0:\infty\}$	full-value property-tax rate per $\$10,000$
PTRATIO	u	$\{0:\infty\}$	pupil-teacher ratio by town
B	u	$\{0:396.9\}$	$1000(Bk-0.63)^2$ where Bk is the proportion of blacks by town
LSTAT	u	$\{0:100\}$	Percentage lower status of the population
Target			
Name	c/u	Range	Description
MEDV	u	$\{0:\infty\}$	Median value of owner-occupied homes in $\$1000$s

Table 3.3: **The Boston Housing data set** Each of the attributes of the Boston data set are shown with its name, whether it is categorical or uniform (c/u), its range and its description. the attributes are separated into 13 inputs and a single target.

The Boston housing data set was originally published in [78]. The data contains information about housing in the Boston Massachusetts area collected by the United States Census Service. The data is also available in the StatLib archive[3].

The data set has been widely used throughout statistical literature as a benchmark. The prototask considered in this thesis is Boston/price which involves the prediction of the median value of a home given its other 11 attributes.

There are 3 subtasks with 32, 64 and 128 examples. There are 8 training sets of 32 examples, 4 of 64 examples and 2 of 128 examples. There are 506 examples in the total data set. The training and test sets are drawn from the total data set so as to ensure 250 points in each of the test sets. The test set selection method is *common*.

[3] http://lib.stat.cmu.edu/datasets/boston

The Kin Family

The Kin family of data sets consists of 8 different data sets which share the same model but which have different numbers of inputs (8 or 32) and differing levels of noise and degrees of linearity. The model for the Kin family is a simulation of the forward dynamics of an 8 link all-revolute robot arm. The task in all data sets is to predict the proximity of the end-effector of the arm from a target. The inputs are shown in Table 3.4.

All the data sets in the Kin family have the common base name Kin which is appended by a dash and followed by:

1. the number of input attributes, either 32 or 8,
2. the degree of linearity: either f for "fairly linear" or n for "non-linear",
3. the level of noise, either m for medium noise or h for high noise.

In total there are 8192 cases in each of the data sets. The training set sizes in the 4 subtasks are 64, 128, 256, 512 and 1024. The 1024 example subtask has 4 training sets, whilst all others have 8 training sets. The test set selection method is *hierarchical*. Further details of the design of the Kin data sets can be found in Ghahramani [69].

The Pumadyn Family

The Pumadyn family is similar in form to the Kin family with the distinction that the model used is a simulation of the dynamics of a Puma 560 robot arm. The task is to predict the angular acceleration of one of the links of the robot. Once again there are 8192 total examples in each data set in the Pumadyn family with training sets of size 64, 128, 256, 512 and 1024 examples. The data set naming is as for the Kin family, with the prefix Pumadyn rather than Kin. Further details of the Pumadyn data sets can be found in Ghahramani [70]

Inputs			
Name	c/u	Range	Description
theta1	u	$\{-3.1416 : 3.1416\}$	angular position of joint 1 in radians
theta2	u	$\{-3.1416 : 3.1416\}$	angular position of joint 2 in radians
theta3	u	$\{-3.1416 : 3.1416\}$	angular position of joint 3 in radians
theta4	u	$\{-3.1416 : 3.1416\}$	angular position of joint 4 in radians
theta5	u	$\{-3.1416 : 3.1416\}$	angular position of joint 5 in radians
theta6	u	$\{-3.1416 : 3.1416\}$	angular position of joint 6 in radians
theta7	u	$\{-3.1416 : 3.1416\}$	angular position of joint 7 in radians
theta8	u	$\{-3.1416 : 3.1416\}$	angular position of joint 8 in radians
alpha1	u	$\{-3.1416 : 3.1416\}$	link 1 twist angle
alpha2	u	$\{-3.1416 : 3.1416\}$	link 2 twist angle
alpha3	u	$\{-3.1416 : 3.1416\}$	link 3 twist angle
alpha4	u	$\{-3.1416 : 3.1416\}$	link 4 twist angle
alpha5	u	$\{-3.1416 : 3.1416\}$	link 5 twist angle
alpha6	u	$\{-3.1416 : 3.1416\}$	link 6 twist angle
alpha7	u	$\{-3.1416 : 3.1416\}$	link 7 twist angle
alpha8	u	$\{-3.1416 : 3.1416\}$	link 8 twist angle
a1	u	$\{0 : \infty\}$	link 1 length
a2	u	$\{0 : \infty\}$	link 2 length
a3	u	$\{0 : \infty\}$	link 3 length
a4	u	$\{0 : \infty\}$	link 4 length
a5	u	$\{0 : \infty\}$	link 5 length
a6	u	$\{0 : \infty\}$	link 6 length
a7	u	$\{0 : \infty\}$	link 7 length
a8	u	$\{0 : \infty\}$	link 8 length
d1	u	$\{0 : \infty\}$	link 1 offset distance
d2	u	$\{0 : \infty\}$	link 2 offset distance
d3	u	$\{0 : \infty\}$	link 3 offset distance
d4	u	$\{0 : \infty\}$	link 4 offset distance
d5	u	$\{0 : \infty\}$	link 5 offset distance
d6	u	$\{0 : \infty\}$	link 6 offset distance
d7	u	$\{0 : \infty\}$	link 7 offset distance
d8	u	$\{0 : \infty\}$	link 8 offset distance
Target			
Name	c/u	Range	Description
y	u	$\{0 : \infty\}$	Cartesian distance of end point from $(0.1, 0.1, 0.1)$

Table 3.4: **The Kin family of data sets**. Each of the attributes of the Kin family of data sets are shown with its name, whether it is categorical or uniform (c/u), its range and its description. the attributes are separated into 32 inputs and a single target. Two subsets of tasks are created from this data set: tasks with 32 input variables and tasks with 8. For the 8 input variable tasks, only the first 8 attributes are used - theta1 to theta8.

Inputs			
Name	c/u	Range	Description
theta1	u	$\{-3.1416 : 3.1416\}$	angular position of joint 1 in radians
theta2	u	$\{-3.1416 : 3.1416\}$	angular position of joint 2 in radians
theta3	u	$\{-3.1416 : 3.1416\}$	angular position of joint 3 in radians
theta4	u	$\{-3.1416 : 3.1416\}$	angular position of joint 4 in radians
theta5	u	$\{-3.1416 : 3.1416\}$	angular position of joint 5 in radians
theta6	u	$\{-3.1416 : 3.1416\}$	angular position of joint 6 in radians
thetad1	u	$\{-\infty : \infty\}$	angular velocity of joint 1 in rad/sec
thetad2	u	$\{-\infty : \infty\}$	angular velocity of joint 2 in rad/sec
thetad3	u	$\{-\infty : \infty\}$	angular velocity of joint 3 in rad/sec
thetad4	u	$\{-\infty : \infty\}$	angular velocity of joint 4 in rad/sec
thetad5	u	$\{-\infty : \infty\}$	angular velocity of joint 5 in rad/sec
thetad6	u	$\{-\infty : \infty\}$	angular velocity of joint 6 in rad/sec
tau1	u	$\{-\infty : \infty\}$	torque on joint 1
tau2	u	$\{-\infty : \infty\}$	torque on joint 2
tau3	u	$\{-\infty : \infty\}$	torque on joint 3
tau4	u	$\{-\infty : \infty\}$	torque on joint 4
tau5	u	$\{-\infty : \infty\}$	torque on joint 5
dm1	u	$\{0 : \infty\}$	proportion change in mass of link 1
dm2	u	$\{0 : \infty\}$	proportion change in mass of link 2
dm3	u	$\{0 : \infty\}$	proportion change in mass of link 3
dm4	u	$\{0 : \infty\}$	proportion change in mass of link 4
dm5	u	$\{0 : \infty\}$	proportion change in mass of link 5
da1	u	$\{0 : \infty\}$	proportion change in length of link 1
da2	u	$\{0 : \infty\}$	proportion change in length of link 2
da3	u	$\{0 : \infty\}$	proportion change in length of link 3
da4	u	$\{0 : \infty\}$	proportion change in length of link 4
da5	u	$\{0 : \infty\}$	proportion change in length of link 5
db1	u	$\{0 : \infty\}$	proportion change in viscosity of friction of link 1
db2	u	$\{0 : \infty\}$	proportion change in viscosity of friction of link 2
db3	u	$\{0 : \infty\}$	proportion change in viscosity of friction of link 3
db4	u	$\{0 : \infty\}$	proportion change in viscosity of friction of link 4
db5	u	$\{0 : \infty\}$	proportion change in viscosity of friction of link 5
Target			
Name	c/u	Range	Description
thetadd6	u	$\{-\infty : \infty\}$	angular acceleration of joint 6

Table 3.5: **The Pumadyn family of data sets**. Each of the attributes of the Pumadyn family of data sets are shown with its name, whether it is categorical or uniform (c/u), its range and its description. the attributes are separated into 32 inputs and a single target. Two subsets of tasks are created from this data set: tasks with 32 input variables and tasks with 8. For the 8 input variable tasks, the following attributes are used as inputs: theta1 to theta3, thetad1 to thetad3, tau1 and tau2.

Image

Inputs			
Name	c/u	Range	Description
region-centroid-col	u	$\{0:255\}$	The column of the centre pixel of the region.
region-centroid-row	u	$\{0:255\}$	The row of the centre pixel of the region.
short-line-density-5	u	$\{0:1\}$	Low contrast line count. This is the result of a line extraction algorithm that counts how many lines of length 5 (any orientation) with low contrast (less than or equal to 5) go through the region.
short-line-density-2	u	$\{0:1\}$	High contrast line count. This is the same as short-line-density-5 but counts lines of high contrast (greater than 5).
vedge-mean	u	$\{0:\infty\}$	Mean vertical contrast. This measures the contrast of horizontally adjacent pixels in the region. This attribute is used as a vertical edge detector.
vedge-sd	u	$\{0:\infty\}$	Standard deviation of horizontal contrast
hedge-mean	u	$\{0:\infty\}$	Mean vertical contrast. This measures the contrast of vertically adjacent pixels. Used for horizontal line detection.
hedge-sd	u	$\{0:\infty\}$	Standard deviation of vertical contrast
intensity-mean	u	$\{0:\infty\}$	Average intensity of red (R), green (G) and blue (B) (R+G+B)/3
rawred-mean	u	$\{0:\infty\}$	Average red over areas
rawblue-mean	u	$\{0:\infty\}$	Average blue over areas
rawgreen-mean	u	$\{0:\infty\}$	Average green over areas
exred-mean	u	$\{-\infty:\infty\}$	Excess red (2R - (G + B))
exblue-mean	u	$\{-\infty:\infty\}$	Excess blue (2B - (G + R))
exgreen-mean	u	$\{-\infty:\infty\}$	Excess green (2G - (R + B))
value-mean	u	$\{-\infty:\infty\}$	3-d non linear transformation of RGB
saturation-mean	u	$\{-\infty:\infty\}$	same as for value-mean
hue-mean	u	$\{-\infty:\infty\}$	same as for value-mean
Target			
Name	c/u	Range	Description
pixel-class	u	—	the class of the current pixel

Table 3.6: **The Image data set** Each of the attributes of the Image data set are shown with its name, whether it is categorical or uniform (c/u), its range and its description. the attributes are separated into 18 inputs and a single target of one out of 7 classes.

The task of the Image data set is to classify the centre pixel of a 3x3 patch of an image into one of 7 categories. There are 18 inputs which are the image processing features shown in table Table 3.6. The classes are *brickface*, *sky*, *foliage*, *cement*, *window*, *path*, and *grass*. The Image data set was created by Carla Brodley of the Vision Group, University of Massachusetts. Each of the instances was drawn randomly from a database of 7 outdoor images which were hand segmented to classify each pixel into

one of the 7 classes. There are 30 instances per class for the training data and 300 instances per class for the test data. The data set was modified by Mike Revow for DELVE. The modifications from the original data set were as follows. The data and test files were combined and stratified so as to give an even distribution of the 7 classes in each of the task instances. Also, one attribute (the region-pixel-count) was deleted since it is constant (value 9) for the data set. This data set originally came from the UCI collection [143]. The algorithm for the 3-d non-linear transformations of value-mean, saturation-mean and hue-mean can be found in Foley and van Dam A. [55].

There are 3 subtasks with 8 training sets of 70 and 140 examples and 4 training sets of 280 examples. There is a common test set of 1190 examples in each case. The test set selection method is *common*.

Letter

Inputs			
Name	c/u	Range	Description
x-box	u	$\{0:\infty\}$	horizontal position of box
y-box	u	$\{0:\infty\}$	vertical position of box
width	u	$\{0:\infty\}$	width of box
high	u	$\{0:\infty\}$	height of box
onpix	u	$\{0:\infty\}$	total on pixels
x-bar	u	$\{0:\infty\}$	mean x of on pixels in box
y-bar	u	$\{0:\infty\}$	mean y of on pixels in box
x2bar	u	$\{0:\infty\}$	mean x variance
y2bar	u	$\{0:\infty\}$	mean y variance
xybar	u	$\{0:\infty\}$	mean xy correlation
x2ybr	u	$\{0:\infty\}$	mean of $x \times x \times y$
xy2br	u	$\{0:\infty\}$	mean of $x \times y \times y$
x-ege	u	$\{0:\infty\}$	mean edge count left to rig
xegvy	u	$\{0:\infty\}$	correlation of x-ege with y
y-ege	u	$\{0:\infty\}$	mean edge count bottom to t
yegvx	u	$\{0:\infty\}$	correlation of y-ege with x
Target			
Name	c/u	Range	Description
class	u	A:Z	letter class

Table 3.7: **The Letter data set** Each of the attributes of the Letter data set are shown with its name, whether it is categorical or uniform (c/u), its range and its description. the attributes are separated into 16 inputs and a single target of one out of 26 classes.

This data set consists of sets of 16 attributes derived from raster scan images of 26 capital letters. The 16 attributes are shown in Table 3.7. The data set was created and originally used by Frey and Slate [56]. The character images were based on 20 different fonts. Each letter within these 20 fonts was randomly distorted to create $20,000$ unique patterns.

There are 3 subtasks, with 6 training sets per subtask of size 390, 780 and 1560 examples. The test set selection method is *hierarchical*.

Splice

During protein creation, splice junctions are sites in a DNA sequence at which superfluous DNA is removed. The sequence spliced out is the *intron* portion, whilst the *exon* is the portion of sequence retained. The origin of this data set is Genbank 64.1 [4]. The donors [158], took the intron:exon and exon:intron examples from primates in the Genbank 64.1 database and the non-splice (neither class) from sequences known not to include a splicing site.

The task in this data set is to predict whether a particular position in a DNA sequence is a donor (intron:exon boundary), or an acceptor (exon:intron boundary), or neither of these types. The data set consists of a window of 60 base pairs (or *nucleotides*) around the position at which the boundary is to be predicted. The base pairs are one of A, G, T or C (which represent the different possible base pairs), which are encoded using a 1 of 4 encoding scheme (so that A is 1000, G is 0100, T is 0010 and C is 0001), giving a total number of input attributes of 240. There are 3175 total cases in the data set, divided into 2000 for training and 1575 for testing. The data set originally came from the UCI collection [143]. There are 8 training sets of size 100 and 200 points and 4 of size 400 points. The test set selection method is *common*.

3.4 Issues in Designing Learning Algorithms

Modern machine learning techniques, such as the ones considered in this thesis, typically have many free parameters which affect their performance, *e.g.*, the number of hidden units in an MLP or the choice of splitting and pruning criteria for decision trees. Comparison of different methods is therefore only fair if these free parameters are set in a consistent fashion. There are two approaches for doing this. The first is to use a human expert in the method to set the free parameters, *e.g.*, according to the characteristics of the problem, the performance of the method on validation data or via certain heuristics. The alternative, and the approach used in this thesis, is to design *automatic* learning methods which have few or, ideally, no free parameters.

The use of experts to set free parameters was considered in the StatLog project. King et al. [119, pp.298] describe the evaluation methodology used in StatLog. A two phase procedure was used in which experienced experts in each method performed the initial trials and filed a report on the techniques used to set each free parameter. Using this report a novice user then also performed secondary trials and compared the results with those of the expert. The two trials were then evaluated by a third party and if the results were close[5] they were accepted, if not a further evaluation was used.

The major problem with the use of experts is firstly the degree of reproducibility of the results, and

[4] ftp://genbank.bio.net

[5] The measure of closeness is not defined - ideally a significance test should have been used, but this would have been difficult given the cross validation framework used for testing.

secondly the variability of the experts which is difficult to account for in a comparison, *i.e.,* if different experts were used a different conclusion might be drawn. The approach of StatLog goes some way to addressing the problems of using experts but still ignores the choice of heuristics or techniques for setting free parameters. Ideally such heuristics should be encoded in an automatic learning algorithm so that a novice user could always get the same performance as an expert without having to consult a report. This issue is particularly important when a large amount of data sets are used in a comparison, as is done in this study, or when the final choice of algorithm is also dependent on its ease of use.

Automatic learning methods are thus a fairer way of comparing techniques and also force researchers to make explicit the choices made in the design of a learning method. By using automatic methods we are also comparing particular *instances* of general learning methods. For example it is inappropriate to say that C4.5 [186] is better or worse than multi-layer perceptrons on a particular data set since C4.5 is an instance of a general framework of decision tree algorithms and uses particular choices of splitting rules and pruning criteria whereas "MLPs" encompass a wide range of possible instances of learning methods, some of which may well be better or worse than C4.5. If we compare C4.5 with a particular instance of an MLP, *e.g.,* `mlp-ese-1` [189], we are now making explicit the methods used in the comparison. Given these instances we can then compare different design choices in their implementation. For example ID3 [185] and C4.5 are both instances of decision trees which use different design choices and implementations — ID3 was an early version of C4.5. Shavlik et al. [213] compared ID3 and found it to be worse than an MLP at some classification tasks. Subsequent studies on the same data have found that C4.5 performs as well or better than MLPs.

Descriptions of automatic learning methods are fairly common in statistics, *e.g.,* MARS [58], and symbolic approaches, *e.g.,* CART [21] and C4.5 [186], but less so in neural networks. A notable exception is the work of Rasmussen [189] which has inspired many of the concepts in this discussion.

The decomposition of learning methods introduced in Chapter 1 can be further generalised to the concepts of model, estimation and search *biases* [26]. A particular learning method will exhibit different biases in model, estimation and search which will bias it towards one data set and away from another. This leads to, and is exhibited in, the issue of *selective superiority* [26] of learning methods in which one method does not perform universally well on all tasks.

One danger of comparing automatic learning methods is that it ignores prior information. It is possible that some methods could benefit greatly from prior knowledge in a comparison. Neglecting such information could therefore lead one to conclude that a particular method was better than another when in fact additional information could reverse the results of the comparison. In such cases, however, tests should be done in which the prior knowledge is presented to all learning methods in a consistent manner so that those methods which can perform best with this extra information can be judged. Obviously this raises the question of how to encode prior knowledge which is a subject of current research in fields such as belief networks [215].

One final problem with any comparison of learning methods is that they are always in some sense artificial comparisons. This is because in many applications the researcher will have some idea as to which method will work well for their problem which they have worked out either through trial and error or through a consideration of the features of the data. For example in speech recognition hidden

Markov models (HMMs) have emerged as the dominant technology for learning. HMMs are well suited to speech due to their sequential modelling and issues such as representation, estimation and search methods as well as design of model architectures have been evaluated experimentally over many years. For such a domain it is unlikely that the lessons learnt in a comparison will be of benefit, unless a new model class is proposed for the domain. Comparisons thus represent only one part of the *process* of designing learning methods which may include feedback from results of applying the method to the domain [25]. Other issues include ease of use of the method, time taken during training and testing and interpretability of the parameters of the models. Of these issues, only the final predictive performance is considered in this study, although a discussion of the other points is given in Chapter 8.

3.5 Outline of Part I

Part I of this thesis is organised as follows. Chapter 4 describes a framework for regression and classification using mixtures-of-experts. This framework addresses the choice of models for the experts and gates, methods of posterior propagation and methods for optimisation of parameters. This framework is then used in Chapters 5, 6 and 7 which describe three different approaches to *complexity control* in mixtures-of-experts models.

The first approach to complexity control, described in Chapter 5, is based on early stopping by cross validation. The architecture of the models is determined in advance of training and is either a fixed width mixture or a fixed depth hierarchy. The chapter describes two learning methods: `me-ese-1` and `hme-ese-1`. In Chapter 6 early stopping is also used but the architecture of hierarchical mixtures-of-experts models is determined during training by recursive splitting of terminal nodes. This method is called `hme-grow-1`. Finally, in Chapter 7 Bayesian inference methods are used to control complexity within a fixed architecture. This approach is used to develop two methods: `me-el-1` and `hme-el-1`.

Chapter 8 presents the results obtained from these five methods on the DELVE data sets introduced in this chapter. These are compared with the existing learning methods in DELVE. Finally, this part of the thesis is concluded in Chapter 8 with a discussion of the issue of *selective superiority* for mixtures-of-experts and other learning methods and gives suggestions for future work in the area of mixtures-of-experts.

Chapter 4

Mixtures of Experts for Classification and Regression

4.1 Introduction

The aim of this chapter is to describe a framework for regression and classification using mixtures-of-experts models. The first issued addressed is how to propagate posterior credit through hierarchical mixtures-of-experts models. This is followed by a description of an algorithm that implements the E step of the EM algorithm presented in Chapter 2. The choice of conditional densities and the functional forms of the experts and gates is then described. Generalised linear models [140] are used for both experts and gates in this part of the thesis. In this chapter the choice of optimisation algorithm for experts and gates is also discussed. The original paper on the use of GLIMs in hierarchical mixtures-of-experts by Jordan and Jacobs [112] advocated the use of a Newton based optimisation algorithm. This chapter describes an alternative conjugate gradient based optimisation algorithm which is believe to be preferable in most circumstances. Some issues in efficient implementation and initialisation are also discussed. Finally, a summary is given of the maximum likelihood training algorithm which will be used in Chapters 5 and 6 and which forms the basis of the Bayesian training algorithm described in Chapter 7.

4.2 The E step - Posterior Propagation and Segmentation

In Section 2.3 the EM algorithm for mixtures and hierarchical mixtures-of-experts developed by Jordan and Jacobs [112] was described. This algorithm consisted of two parts: a propagation of posterior credit to each node in the model (the E step) and a maximisation of a likelihood function weighted by the posterior credit of each node (the M step). This chapter discusses how to implement both of these steps. Section 4.3 describes the functional form of the experts and gates used in this part of the thesis and the implementation of the M step for these models. This section describes how to implement the E step of the EM algorithm for general choices of experts and gates.

The form of the computations in the E step varies according to the type of experts and gates used. The E step involves computation of two probabilities. First, for each expert $\mathcal{E}_i$, compute the probability

of the target $\boldsymbol{y}^{(n)}$ given the input $\boldsymbol{x}^{(n)}$ and current parameters $\mathbf{w}_i$: $P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \mathbf{w}_i)$. Secondly, for each child node of each non-terminal node $\mathcal{E}_j$ compute the set of probabilities $P(\,Ch(\mathcal{E}_j)|\boldsymbol{x}^{(n)}, \mathbf{v}_j)$ given by the outputs of gate $\mathcal{G}_j$. Computation of these two probabilities is usually straightforward, but its form depends on the type of experts and gates used. The next phase of the E step involves combining and propagating these probabilities so as to compute the posterior probabilities of each node in the model. Before we describe how to perform this propagation process, let us introduce a concept which is considered in the E step which we call the *segmentation*. The segmentation of an HME model is defined as the set of joint posterior probabilities $\{\zeta_i^{(n)} = P(\mathcal{E}_i, An(\mathcal{E}_i)|\boldsymbol{y}^{(n)}, \boldsymbol{x}^{(n)})\}$ for all nodes and all examples $(\boldsymbol{y}^{(n)}, \boldsymbol{x}^{(n)})$.

As described in Section 2.3, there are two types of posterior probability of nodes in a mixture of experts model: *conditional* and *joint* posterior probabilities. The conditional probability of node $\mathcal{E}_i$ is given by:

$$\varphi_i^{(n)} = P(\mathcal{E}_i|An(\mathcal{E}_i), \boldsymbol{y}^{(n)}, \boldsymbol{x}^{(n)}, \boldsymbol{\theta}), \tag{4.1}$$

where $An(\mathcal{E}_i)$ is the set of ancestors nodes from $\mathcal{E}_i$ to the root node and $\boldsymbol{\theta}$ is the set of parameters of $\mathcal{E}_i$ and its ancestors. The joint probability is given by

$$\zeta_i^{(n)} = P(\mathcal{E}_i, An(\mathcal{E}_i)|\boldsymbol{y}^{(n)}, \boldsymbol{x}^{(n)}, \boldsymbol{\theta}). \tag{4.2}$$

Using these forms, in conjunction with the belief network decomposition of Figure 2.2 and Bayes' rule, we may write down the following expressions:

$$\varphi_i^{(n)} = \frac{P(\boldsymbol{y}^{(n)}|\mathcal{E}_i, An(\mathcal{E}_i), \boldsymbol{x}^{(n)})P(\mathcal{E}_i|An(\mathcal{E}_i), \boldsymbol{x}^{(n)})}{P(\boldsymbol{y}^{(n)}|An(\mathcal{E}_i), \boldsymbol{x}^{(n)})} \tag{4.3}$$

for the conditional probability, and

$$\zeta_i^{(n)} = \frac{P(\boldsymbol{y}^{(n)}|\mathcal{E}_i, An(\mathcal{E}_i), \boldsymbol{x}^{(n)})P(\mathcal{E}_i, An(\mathcal{E}_i)|\boldsymbol{x}^{(n)})}{\sum_j P(\boldsymbol{y}^{(n)}|\mathcal{E}_j, An(\mathcal{E}_j), \boldsymbol{x}^{(n)})P(\mathcal{E}_j, An(\mathcal{E}_j)|\boldsymbol{x}^{(n)}))} \tag{4.4}$$

for the joint probability, where j sums over all nodes on the same level in the tree as $\mathcal{E}_i$. The individual terms of these equations are given by:

- $P(\boldsymbol{y}^{(n)}|\mathcal{E}_i, An(\mathcal{E}_i), \boldsymbol{x}^{(n)})$: the probability of the target at time n given node i and its ancestors. For a terminal node this is a function of the form of the expert at the terminal node (and will be described in the next section). For a non-terminal node this may be computed via an upwards propagation of the appropriate terms from the siblings of the node. For example, the denominator of Equation 4.3 $P(\boldsymbol{y}^{(n)}|An(\mathcal{E}_i), \boldsymbol{x}^{(n)})$ is given by:

$$P(\boldsymbol{y}^{(n)}|An(\mathcal{E}_i), \boldsymbol{x}^{(n)}) = \sum_{j \in Ch(\mathcal{E}_k)} P(\boldsymbol{y}^{(n)}|\mathcal{E}_j, An(\mathcal{E}_j), \boldsymbol{x}^{(n)})P(\mathcal{E}_j|An(\mathcal{E}_j), \boldsymbol{x}^{(n)}), \tag{4.5}$$

  where $\mathcal{E}_k = Pa(\mathcal{E}_j)$, the immediate parent of node $\mathcal{E}_i$.

- For all examples $\{\boldsymbol{y}^{(n)}, \boldsymbol{x}^{(n)}\}$:
  - For each terminal node, compute $P(\boldsymbol{y}^{(n)}|\mathcal{E}_i, An(\mathcal{E}_i), \boldsymbol{x}^{(n)})$.
  - For each terminal node, compute via a downwards recursion from the root node, $P(\mathcal{E}_i, An(\mathcal{E}_i)|\boldsymbol{x}^{(n)})$ as the product of terms $P(\mathcal{E}_j|An(\mathcal{E}_j), \boldsymbol{x}^{(n)})$ for each node $\mathcal{E}_j$ in $An(\mathcal{E}_i)$.
  - For each terminal node $\mathcal{E}_i$ form the product $\phi_i = P(\boldsymbol{y}^{(n)}|\mathcal{E}_i, An(\mathcal{E}_i), \boldsymbol{x}^{(n)})P(\mathcal{E}_i, An(\mathcal{E}_i)|\boldsymbol{x}^{(n)})$.
  - For each node $\mathcal{E}_i$ divide each product term ϕ_i by the summation $\sum_j \phi_j$ over all nodes in the same level as $\mathcal{E}_i$ which gives the joint posterior probability $\zeta_i^{(n)}$ for $\mathcal{E}_i$.

Figure 4.1: **Algorithm for posterior propagation in hierarchical mixtures-of-experts models**

- $P(\mathcal{E}_i|An(\mathcal{E}_i), \boldsymbol{x}^{(n)})$: the probability of node $\mathcal{E}_i$ given its ancestors and the input at time n. This is given by the i^{th} output of the gate rooted at $Pa(\mathcal{E}_i)$.
- $P(\mathcal{E}_i, An(\mathcal{E}_i)|\boldsymbol{x}^{(n)})$: the joint probability of node $\mathcal{E}_i$ and its ancestors given the input at time n. This is given by the product of terms $P(\mathcal{E}_i|An(\mathcal{E}_i), \boldsymbol{x}^{(n)})$ over all ancestors of $\mathcal{E}_i$ along the path to the root node from $\mathcal{E}_i$, *i.e.,* the product of all the gate outputs for the path from $\mathcal{E}_i$ to the root node.

The algorithm shown in Figure 4.1 is used to compute the joint posterior probabilities of the terminal nodes. Once the joint posterior probabilities of the terminal nodes have been computed, the joint posterior probabilities of all other nodes can be computed by an upwards propagation to the root node, using the fact that the joint posterior probability of a non-terminal node is the sum of the joint posterior probabilities of its children:

$$\zeta_i^{(n)} = \sum_{j \in Ch(\mathcal{E}_i)} \zeta_j^{(n)} \tag{4.6}$$

Finally, the conditional probabilities may be computed by the following expression:

$$P(\mathcal{E}_i|An(\mathcal{E}_i), \boldsymbol{y}^{(n)}, \boldsymbol{x}^{(n)}) = \frac{P(\mathcal{E}_i, An(\mathcal{E}_i)|\boldsymbol{y}^{(n)}, \boldsymbol{x}^{(n)})}{\sum_{Ch(Pa(\mathcal{E}_i))} P(\mathcal{E}_i, An(\mathcal{E}_i)|\boldsymbol{y}^{(n)}, \boldsymbol{x}^{(n)})} = \frac{\zeta_i^{(n)}}{\zeta_k^{(n)}} \tag{4.7}$$

where the sum in the denominator is over all the children of $\mathcal{E}_k = Pa(\mathcal{E}_i)$.

This is not the only possible method of posterior propagation but is the one I prefer since it allows easy extension to other ideas, such as initialisation of the expert posterior probabilities. The form of the propagation algorithm is an example of more general methods for calculating posterior probabilities on directed graphs. Other possible methods are discussed by Jordan [108]. Note also that in the case of more complicated variants and extensions of the HME, such as hidden Markov decision trees [110], the E step often becomes intractable and approximate methods must be used.

1. Compare a to b. If $a < b$, reverse a and b.
2. Compute the difference $d = b - a$.
3. If $d > -10$,
   - set $c = a + \log(1 + \exp(d))$,
4. else
   - if $a < -0.5e10$, set $c = -1.0e10$,
   - else set $c = a$.
5. Return c.

Figure 4.2: **The `log-add` algorithm.** Given two arguments, a and b, `log-add` computes the result $c = \log(\exp(a) + \exp(b))$.

Use of logs to avoid underflow

Since some of the product terms in these algorithms can run over many nodes, the resulting probabilities can become small. For this reason logarithms are used to avoid numerical underflow. The products then become summations. In order to compute the summations of products of probabilities, *e.g.*, $\sum_i \phi_i$, I use an algorithm called `log-add` which is shown in Figure 4.2. This algorithm is based on the expansion,

$$\begin{aligned} \log(\exp(a) + \exp(b)) &= \log(\exp(a)(1 + \exp(b - a))) \\ &= a + \log(1 + \exp(b - a)). \end{aligned} \tag{4.8}$$

Expanding the log in this form saves one $\exp()$ computation. However, further savings can be made examining the magnitude of the difference $b - a$ as shown in Figure 4.2. This algorithm was motivated by an implementation used in Young et al. [250]. It is also used for the softmax function for multinomial logit experts and gates.

4.3 Choice of Expert and Gate Forms and Implementation of the M step

As described in Chapter 1 there have been a number of suggestions in the literature for choices of expert and gating networks. In this part of the thesis generalised linear models are used for experts and gates. In a generalised linear model the conditional density $P(y|\boldsymbol{x}, \boldsymbol{\theta})$ is restricted to the exponential family. A general form for the expected value of y is given by:

$$h(\hat{y}) = E(y|\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{x} \tag{4.9}$$

where $h(\cdot)$ is called the *link* function. In the case of linear regression, for example, the link function is the identity function. I consider two types of generalised linear models: linear regression models with

constant variance for experts in the case of regression, and multinomial logit models for gates and experts in the case of classification.

One of the advantages of choosing generalised linear models is that their optimisation is straightforward, in contrast, for example, to MLPs. In addition, generalised linear models are generally accepted as being more "interpretable" than more complex models such as MLPs. It is questionable, however, whether a deep hierarchy with many experts and gates is still interpretable. This question of interpretability will be considered in more depth in Chapter 8. Let us now consider the models used in this part of the thesis: linear regression experts, multinomial logit gates and multinomial logit experts.

As described in Section 2.3 the auxiliary function for a hierarchical mixtures-of-experts model is given by the sum of two terms, one for the gates and one for the experts in the model:

$$R = \sum_s R(g)_s + \sum_i R(e)_i, \tag{4.10}$$

where the sum s is over all gates, and the sum i over all experts. This auxiliary function is maximised in the M step of the EM algorithm. In the following sections we describe how to implement this maximisation.

Linear Regression experts

For regression problems I use linear regression experts, which have the following form. In the case of 1-dimensional outputs, the output of expert i is given by

$$\hat{y}_i^{(n)} = \sum_{l=1}^{d} w_{il} x_l^{(n)} + w_0 \tag{4.11}$$

where $\mathbf{w}_i = \{w_{il}\}_0^d$ is the parameter vector for expert $\mathcal{E}_i$. The intercept term w_0 is sometimes referred to as the *bias* term in neural network research. I lump it together with the parameter vector and augment the input vector $\boldsymbol{x}^{(n)}$ with an extra constant element of 1, so that the output can be written as $\hat{y}_i^{(n)} = \mathbf{w}_i^T \boldsymbol{x}^{(n)}$.

As described in the previous section, the E step of the EM algorithm utilises the conditional density of the target given the input and the model in its calculation. In the case of regression over 1-dimensional targets, this distribution takes the form of a Gaussian:

$$P(y^{(n)}|\boldsymbol{x}^{(n)}, \mathbf{w}_i, \sigma_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{1}{2\sigma_i^2}(y^{(n)} - \hat{y}_i^{(n)})^2\right), \tag{4.12}$$

where σ_i^2 is the variance parameter for expert i. In the case of multivariate regression, the variance parameter becomes a covariance matrix Σ_i. I also use the notation β_i to refer to the reciprocal of the variance $\beta_i = 1/\sigma_i^2$.

Training linear regression experts

For linear regression experts the contribution of node i to the auxiliary function is given by:

$$R(e)_i \quad = \quad -\tfrac{1}{2}\log(2\pi\sigma_i^2)\textstyle\sum_n \zeta_i^{(n)} - \frac{1}{2\sigma_i^2}\sum_n \zeta_i^{(n)}(y^{(n)} - \mathbf{w}_i^T \boldsymbol{x}^{(n)})^2 \tag{4.13}$$

where $\zeta_i^{(n)}$ is the joint posterior probability of expert $\mathcal{E}_i$. The second term in this expression is the sum of weighted squared errors between targets and predictions of expert i. The weights are given by the joint posterior probability of expert i divided by the variance of expert i. Maximisation of this function corresponds to a weighted least squares regression, with weights given by $\zeta_i^{(n)}/\sigma_i^2$. The first and second derivatives of this function are given by:

$$\frac{\partial R(e)_i}{\partial \mathbf{w}_i} = \sum_n \frac{\zeta_i^{(n)}}{\sigma_i^2}(y^{(n)} - \hat{y}_i^{(n)})\boldsymbol{x}^{(n)}, \tag{4.14}$$

$$\frac{\partial^2 R(e)_i}{\partial \mathbf{w}_i \partial \mathbf{w}_i^T} = -\sum_n \frac{\zeta_i^{(n)}}{\sigma_i^2}(\boldsymbol{x}^{(n)})(\boldsymbol{x}^{(n)})^T. \tag{4.15}$$

The process of credit assignment can be seen clearly in these derivatives. For an expert at node $\mathcal{E}_i$ the standard gradient contributions of a linear regression model $(y^{(n)} - \hat{y}_i^{(n)})\boldsymbol{x}^{(n)}/\sigma_i^2$ are weighted by the joint posterior probability of the node $\zeta_i^{(n)}$. The second derivative terms are also weighted in a similar way. This is effectively assigning credit to parameters of experts which have a high probability of being selected according to the probabilistic model.

To maximise $R(e)_i$, these derivatives may be used in a standard optimisation algorithm such as conjugate gradients, Newton methods or quasi-Newton methods. Alternatively, since $R(e)_i$ is quadratic an exact solution can be written down:

$$\mathbf{w}_i = (X^T W_i X)^{-1} X^T W_i Y \tag{4.16}$$

where X is the input data matrix, Y is the target data vector and W_i is an $N \times N$ diagonal matrix whose n^{th} element is given by $\zeta_i^{(n)}/\sigma_i^2$. This expression may be solved using matrix methods such as LU decomposition or singular value decomposition (SVD). In Section 4.4 a conjugate gradient algorithm is described which is used in this work. The variance may be computed exactly by setting $\partial R(e)_i/\partial\sigma^2$ to zero:

$$\sigma_i^2 = \frac{\sum_{n=1}^{N} \zeta_i^{(n)}(y^{(n)} - \hat{y}_i^{(n)})^2}{\sum_{n=1}^{N} \zeta_i^{(n)}} \tag{4.17}$$

This takes the form of the weighted sum of squared errors divided by the sum of weights $\zeta_i^{(n)}$.

Practical issues

There are a number of practical issues which arise in the implementation of linear regression experts. Foremost is the behaviour of the algorithm in regions of low noise in the data set. Consider a region in which an expert has low contribution to the total prediction (*i.e.,* $\sum_n \zeta_i^{(n)}$ is small) and the error of this expert is also small (*i.e.,* $\sum \zeta_i^{(n)}(y^{(n)} - \hat{y}_i^{(n)})^2$ is small). This implies the variance estimate approaches a singularity giving rise to an unbounded likelihood. This problem is also found in other

mixture models, such as mixtures of Gaussians, as noted by many authors including Gelman et al. [67, pp.103] and Titterington et al. [224, pp.83]. In general, a simple approach to solving this problem is to place a minimum bound on the variance, based on some prior knowledge. This approach was used by Weigend et al. [242] for the prediction of laser activity by a mixture of experts. The threshold they used was based on the accuracy of the equipment used to record the training data.

In this work I place a prior on the variance parameter, and maximise the *a-posteriori* distribution rather than the likelihood. This is done purely for computational purposes so as to avoid the singularity in the variance solution arising from the maximum likelihood solution. Potentially, many other methods could be used, such as not updating the variance when the number of observations for an expert falls below a threshold or flooring the variance to a specific level. I would expect these methods to turn out to be identical to the approach outlined here. Personally I prefer the use of a prior since it embodies our assumptions mathematically.

Other authors, *e.g.*, MacKay [136] have proposed Gamma priors [13, pp. 268] on the reciprocal of the variance $\beta_i = 1/\sigma_i^2$ for linear or non-linear regression models. I do the same here for the variance of a single expert in the mixtures-of-experts model. This prior has two parameters, ρ and υ, and has the following form:

$$P(\log \beta_i|\rho,\upsilon) = \frac{1}{\Gamma(\rho)}\left(\frac{\beta_i}{\upsilon}\right)^{\rho}\exp(-\beta_i/\upsilon), \tag{4.18}$$

The resulting estimate of the variance for expert $\mathcal{E}_i$ is then given by:

$$\sigma_i^2 = \frac{\sum_n \zeta_i^{(n)}(y^{(n)} - \hat{y}_i^{(n)})^2 + 2/\upsilon}{\sum_n \zeta_i^{(n)} + 2\rho} \tag{4.19}$$

The interpretation of these hyper-parameters is that $2/\upsilon$ defines the prior sum squared error that we might expect a single expert to have. 2ρ defines the prior number of observations that we might expect an expert to see. I re-parametrise these hyper-parameters so that

$$1/\upsilon = \frac{\rho}{2}\sigma_{prior}^2 \tag{4.20}$$

where σ_{prior}^2 is the prior noise level which I set to 0.01 by default. I then set ρ to be a function of the number of terminal nodes I:

$$2\rho = 0.01\left(\frac{N}{I}\right) \tag{4.21}$$

where N is the number of examples in the data set. Note that the choice of σ_{prior}^2 will depend on the range and scale of the data. In all the examples studied here the data is standardised so as to have zero mean and unit standard deviation.

One problem with using priors such as this one, is the effect that the choice of hyper-parameters may have on the final model after training[1]. For example, if the prior noise level is set too high, the model

[1] As noted by David Spiegelhalter [personal communication]

may degenerate into modelling the data using only one expert. Ideally, for each new data set we should perform a number of runs using different values for the hyper-parameters and examining the parameters of the model. In practice this may be difficult when a large number of parameters are present. In this work I found the choices of hyper-parameters outlined above to be adequate to both bound the likelihood safely so as to avoid singularities, and to not lead to degenerate solutions. The prior became most useful in situations where a large number of experts were present, *e.g.*, deep trees, in which the contribution to each expert $\sum_n \zeta_i^{(n)}$ becomes small.

Multiple targets

In the case of multiple regression, where the target variable $\boldsymbol{y}^{(n)}$ is a k-dimensional vector, the appropriate Gaussian distribution for $P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \mathcal{E}_i)$ is given by:

$$P(\boldsymbol{y}^{(n)}|\mathcal{E}_i, \boldsymbol{x}^{(n)}) = \frac{1}{(2\pi)^{\frac{k}{2}}|\Sigma_i|^{\frac{1}{2}}} \exp\left(-\tfrac{1}{2}(\boldsymbol{y}^{(n)} - \hat{\boldsymbol{y}}_i^{(n)})^T \Sigma_i^{-1} (\boldsymbol{y}^{(n)} - \hat{\boldsymbol{y}}_i^{(n)})\right) \tag{4.22}$$

where Σ_i is the variance-covariance matrix of the targets $\boldsymbol{y}^{(n)}$ and predictions $\hat{\boldsymbol{y}}_i^{(n)}$ of expert $\mathcal{E}_i$. The choice of form for Σ_i reflects our belief in the form of the prediction task. If Σ_i is chosen to be diagonal, we are saying that the noise in each target variable y_i is independent but has different variance. Alternatively we might be even more restrictive and say that the covariance matrix be modelled by $\Sigma_i = \sigma_i^2 I$, *i.e.*, with common variance for each target variable. At the other end of the spectrum, a full covariance matrix Σ_i models all dependencies between the noise in the target variables.

Jordan and Xu [113] present a framework for the mixture of experts using a full covariance matrix. As far as I am aware, no other authors have used the mixture of experts for prediction of multiple targets. In this work I choose a diagonal covariance matrix, so that the log of the probability distribution for each expert $\mathcal{E}_i$ can be written as:

$$\log P(\boldsymbol{y}^{(n)}|\mathcal{E}_i, \boldsymbol{x}^{(n)}) = -\frac{k}{2}\log 2\pi - \tfrac{1}{2}\sum_{l=1}^{k} \log \sigma_{il}^2 - \sum_{i=1}^{k} \tfrac{1}{2\sigma_{il}^2}(y_i^{(n)} - \hat{y}_{il}^{(n)})^2 \tag{4.23}$$

where $\hat{y}_{il}^{(n)}$ and σ_{il}^2 are the predictions and the variance of expert $\mathcal{E}_i$ for target variable $y_l^{(n)}$.

Multinomial Logit Experts

In the case of classification, the experts can take any form such that their output is a prediction of the class probability distribution $P(Y|X)$ where Y is the target class distribution, *i.e.*, $y_r^{(n)} = 1$ if class r is the correct class at time n otherwise $y_r^{(n)} = 0$. In this work I use multinomial logit models in which the estimate of the probability of class r takes the form:

$$\hat{y}_{i,r}^{(n)} = P(y_r^{(n)} = 1|\boldsymbol{x}^{(n)}, \mathbf{w}_i) = \frac{\exp(\mathbf{w}_{ir}^T \boldsymbol{x}^{(n)})}{\sum_p \exp(\mathbf{w}_{ip}^T \boldsymbol{x}^{(n)})} \tag{4.24}$$

where $\mathbf{w}_{ip}$ is the parameter vector for expert $\mathcal{E}_i$, parameterising the estimate for class p. This function allows linear decision boundaries to be constructed between adjacent classes [14, pp.79]. For expert $\mathcal{E}_i$

the distribution of targets given inputs and parameters takes the form of the exponential of the negative of the cross entropy between the targets $\{\boldsymbol{y}^{(n)}\}$ and the outputs of the expert $\{\hat{\boldsymbol{y}}_i^{(n)}\}$:

$$P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \mathbf{w}_i) = \exp\left(\sum_{r=1}^{k} y_r^{(n)} \log \hat{y}_{i,r}^{(n)}\right), \tag{4.25}$$

where $\hat{y}_{ir}^{(n)}$ is the r^{th} output of expert $\mathcal{E}_i$.

Training multinomial logit experts

In the case of multinomial logit experts the contribution of expert i to the auxiliary function is given by:

$$R(e)_i = \sum_{n=1}^{N} \zeta_i^{(n)} \sum_{r=1}^{k} y_r^{(n)} \log \hat{y}_{i,r}^{(n)}. \tag{4.26}$$

Maximisation of this function corresponds to maximising the negative of the cross entropy between the targets $\boldsymbol{y}^{(n)}$ and the predictions $\hat{\boldsymbol{y}}^{(n)}$, weighted by the joint posterior probability of the expert. Once again we wish to find the derivatives of this function. Bishop [14, pp. 237-40] shows that for a cross entropy function E given by:

$$E = -\sum_{n}\sum_{c=1}^{k} y_c^{(n)} \log \hat{y}_c^{(n)} \tag{4.27}$$

the first derivative is given by:

$$\frac{\partial E}{\partial a_c} = \sum_{n} (\hat{y}_c^{(n)} - y_c^{(n)}), \tag{4.28}$$

where a_c is some function of the inputs $f(\boldsymbol{x}^{(n)})$. In an MLP f is given by the outputs of the hidden units. In a generalised linear model, f is simply the linear combination of the parameters $\mathbf{w}_c$ with the inputs $\boldsymbol{x}^{(n)}$, *i.e.*, $f(\boldsymbol{x}^{(n)}) = \mathbf{w}_c^T \boldsymbol{x}^{(n)}$. This allows the derivative of $R(e)_i$ with respect to parameter vector $\mathbf{w}_{ic}$ to be written as:

$$\frac{\partial R(e)_i}{\partial \mathbf{w}_{ic}} = \sum_{n} \zeta_i^{(n)} (y_k^{(n)} - \hat{y}_{i,c}^{(n)}) \boldsymbol{x}^{(n)}. \tag{4.29}$$

The second derivative is given by:

$$\frac{\partial^2 R(e)_i}{\partial \mathbf{w}_{ic} \partial \mathbf{w}_{ic}^T} = -\sum_{n} \zeta_i^{(n)} \hat{y}_{i,c}^{(n)} (1 - \hat{y}_{i,c}^{(n)}) (\boldsymbol{x}^{(n)}) (\boldsymbol{x}^{(n)})^T. \tag{4.30}$$

Multinomial Logit Gates

The gate can take any functional form that is consistent with the estimation of probabilities. In this chapter the gates are multinomial logit models which can form only linear boundaries between expert regions. The outputs in this case are given by:

$$g_i^{(n)} = \frac{\exp(\mathbf{v}_i^T \boldsymbol{x}^{(n)})}{\sum_j \exp(\mathbf{v}_j^T \boldsymbol{x}^{(n)})}, \tag{4.31}$$

where $\{\mathbf{v}_i\}_{i=1}^I$ is the set of parameters for the gate. The vector $\mathbf{v}_i$ parameterises the i^{th} output of the gate.

Training multinomial logit gates

The contribution of gate $\mathcal{G}_i$ rooted at node $\mathcal{E}_i$ is given by:

$$R(g)_i = \sum_{n=1}^{N} \zeta_i^{(n)} \sum_{j \in Ch(\mathcal{E}_i)} \varphi_j^{(n)} \log g_j^{(n)}. \tag{4.32}$$

where the sum is over the child nodes of node $\mathcal{E}_i$. Maximising this function maximises the negative of the cross entropy between the target distribution for the missing data $\varphi_j^{(n)}$ and the output distribution of the gates $g_j^{(n)}$, weighted by the joint posterior probability of the node $\zeta_i^{(n)}$. In a similar manner to the multinomial logit experts the derivatives of $R(g)_i$ with respect to parameter vector $\mathbf{v}_j$ are given by:

$$\frac{\partial R(g)_i}{\partial \mathbf{v}_j} = \sum_{n=1}^{N} \zeta_i^{(n)} (\varphi_j^{(n)} - g_j^{(n)}) \boldsymbol{x}^{(n)}, \tag{4.33}$$

$$\frac{\partial^2 R(g)_i}{\partial \mathbf{v}_j \partial \mathbf{v}_j^T} = -\sum_{n=1}^{N} \zeta_i^{(n)} g_j^{(n)} (1 - g_j^{(n)}) (\boldsymbol{x}^{(n)})(\boldsymbol{x}^{(n)})^T. \tag{4.34}$$

4.4 Optimisation of Experts and Gates

Given the specification of posterior probabilities which are computed by the posterior propagation algorithm, it remains only to specify how the experts and gates are optimised. Given that each gate and expert is a generalised linear model[2], the optimisation process reduces to solving the set of equations [140, pp. 41]. Specifically, if Equations 4.14, 4.29 and 4.33 are set to zero, they may be written in the generic form:

$$\sum_{n=1}^{N} \zeta_i^{(n)} (t_l^{(n)} - \hat{t}_l^{(n)}) \boldsymbol{x}^{(n)} = 0 \tag{4.35}$$

[2] As noted by Jordan and Jacobs [112, pp. 211], the multinomial logit function is a special case of the generalized linear model family in which the probabilistic component is the multinomial density or Poisson density.

in which $t_l^{(n)}$ represents target l at time n, which is either the actual target $y_l^{(n)}$ in the case of linear regression or multinomial logit experts, or the conditional posterior probability $\varphi_l^{(n)}$ in the case of multinomial logit gates. The term $\hat{t}_l{}^{(n)}$ represents the prediction of $t_l^{(n)}$ by the model, which is either the actual prediction $\hat{\boldsymbol{y}}_l^{(n)}$ of an expert, or the output $g_l^{(n)}$ of a gate. An algorithm for solving these equations based on a Fisher scoring method, known as iteratively re-weighted least squares (IRLS) is described by McCullagh and Nelder [140]. IRLS was proposed by Jordan and Jacobs [112] for solving these equations in mixtures-of-experts models.

In the special case of linear regression experts, IRLS reduces to weighted least squares, with the weights given by posterior probabilities as shown in Equation 4.16. These equations may be efficiently solved using a matrix decomposition method such as Cholesky decomposition, LU decomposition or Singular Value Decomposition (SVD). In practice, the use of SVD method for least squares may be preferred since this allows solutions when the Hessian matrix is nearly singular. Near-singularity of the Hessian matrix is likely when the sum of posterior probabilities $\sum_n \zeta_i^{(n)}$ is close to the dimension k of the Hessian matrix.

Rather than using IRLS, in this thesis I use a conjugate gradient algorithm for both linear regression and multinomial logit models. This is because I find empirically the IRLS algorithm to be too unstable in the presence of the extra weighting induced by the posterior probabilities $\{\zeta_i^{(n)}\}$, in particular when the number of observations for a node is very small. In addition, the IRLS algorithm requires the computation of the Hessian matrix (requiring $\mathcal{O}(Nd^2)$ calculations) which may be a costly operation in the case of large input dimensionality d. An alternative to conjugate gradients is the use of variable metric methods (see Press et al. [180] or Fletcher [53] for more details).

I now describe the conjugate gradient algorithm which I use for optimising experts and gates. This is called `macopt` and was originally written by David MacKay. [3] Briefly, the major difference between `macopt` and other conjugate gradient algorithms is its use of gradient information. This difference comes in the way that `macopt` performs line searches. Standard conjugate gradient algorithms do a line search for the minimum along the conjugate direction by evaluating the value of the function it is minimising. `macopt`, however, uses the gradient of the function to *bracket* the minimum of the function. This bracketing is based on the principle that on either side of a minimum, the inner product between the gradient at and the line search direction will have opposite signs. `macopt` also only evaluates the minimum of the line search approximately. On average the algorithm takes approximately 2 gradient evaluations per line minimisation. The following parameters control the behaviour of `macopt`:

tol: the value of magnitude of gradient below which the algorithm terminates. I set this to 0.001.

itmax: the maximum number of iterations of the algorithm performed before it terminates (if tol has not been already reached). I set this to 10.

These values for the parameters were found to be adequate to ensure good training of linear models in empirical trials.

[3] See Appendix A for details of the availability of `macopt`

4.5 Initialisation

Examination of the likelihood function for the HME indicates that choice of initial parameter values for experts and gates is important. For example, for a mixture of two experts, if the values of the expert parameters are the same, and the gate parameters are set to zero, training will not change these values. In order to "break symmetry" either the expert parameters must be different or the gate parameters must be sufficiently different from zero so as to give an advantage to one expert or the other. If the predictions of each expert are too close to one another (perhaps by poor choice of starting parameters), the training algorithm falls into a *degenerate* solution in which each expert has near-identical parameters and the gate has near-zero parameter values.

I experimented with a number of different initialisation strategies for expert and gate parameters. One method is to randomise the expert and gate parameters using a uniform or Gaussian distribution in a pre-specified range. Choice of this range, however, is somewhat ad hoc. Another method I tried was to train a single expert on the data and use the magnitude of the parameters of this expert as the range for randomisation of the experts. This approach was somewhat successful, but still led to degenerate solutions for small tasks.

A more successful method of initialisation, which forms the basis for all the mixture of experts models in Chapters 5 to 7, is based on the concept of *segmentation*, described in Section 4.2. I randomise the segmentation values $\zeta_i^{(n)}$ for each example n and each expert i given the constraint that the sum $\sum_i \zeta_i^{(n)} = 1$ for nodes $\{\mathcal{E}_i\}$ in the same level of the tree. Each terminal node segmentation value is drawn from a uniform random distribution and normalised by the sum of segmentation values of all nodes at this level of the tree. I then propagate these probabilities up the tree using the propagation algorithm of Section 4.2, and run the training algorithms for all experts and gates to give initial values. This method is more stable than the randomisation of parameters method (*i.e.*, leads to less degenerate solutions), and has the practical advantage of not requiring choice of initial values for the randomisation, since the randomisation is over probabilities rather than parameter values.

4.6 Summary

In this chapter I have described a framework for regression and classification using mixtures-of-experts. A number of issues have been addressed including the choice of models for experts and gates and optimisation and initialisation of parameters. These are summarised in the algorithm `ml-train` shown in Figure 4.3. `ml-train` forms the basis of Chapters 5, 6 and 7. These chapters address the issue of complexity control and architecture selection in mixtures-of-experts based learning methods.

Initialisation

1. Initialise the segmentation of the terminal nodes to random values between 0 and 1, normalise to 1 and propagate the posteriors through the tree.
2. Compute the gradients of each expert and gate and use `macopt` to update their parameters.

Training

3. Compute the likelihood of the model L^t.
4. Compute the posterior probabilities $\{\varphi_i^{(n)}\}_1^N$ and $\{\zeta_i^{(n)}\}_1^N$ for each node in the tree using the posterior propagation algorithm of Section 4.2.
5. Compute the gradients of each expert and gate and use `macopt` to update their parameters.
6. Compute the new likelihood L^{t+1}.
7. If $L^{t+1} - L^t > 10^{-4}$ goto (3) else terminate.

Figure 4.3: **The `ml-train` algorithm**

Chapter 5

Training Mixtures-of-Experts using Early Stopping

5.1 Introduction

This chapter describes the first two of my proposed methods for regression and classification based on the mixtures-of-experts. The background theory on which these methods are based was described in Chapters 2 and 4. Sections 5.2 and 5.3 describe how these methods are implemented. Included in these section are answers to such issues as how to initialise the models, how many parameters to use and how to reduce over-fitting of the data. The two methods are given the names `me-ese-1` which is a mixtures-of-experts model and `hme-ese-1` which is a hierarchical mixtures-of-experts model. Both methods use early stopping via cross-validation to control the complexity of their models.

The two methods described in this chapter were motivated by Carl Rasmussen's [189] method for multi-layer perceptrons with early stopping, `mlp-ese-1`. Rasmussen found this method to be competitive with other learning methods such as MLPs trained by Markov Chain Monte Carlo (MCMC) techniques (results summarised in Chapter 8, Appendix B and Appendix C). The technique used by Rasmussen was followed closely so as to see how well a ME or HME model would do in comparison to a MLP given similar constraints on computing resources and time.

5.2 Avoiding Over-fitting

The mixture of experts consists of a series of experts, each modelling different processes assumed to be underlying causes of the data. Since each expert may focus on a different subset of the data which may be arbitrarily small, the possibility of over-fitting is increased when using a large number of experts. There is an added potential for the whole architecture to over-fit the data through the gates. This effect occurs when the distribution of missing data $P(Z|X,Y)$ becomes sharp, focusing on small groups of data points at a time. The net effect is to cause the gates to select a single expert for each small group of points. In the limit, the effect is to assign one expert for each pair of data points. This leads to disjointed functions being predicted by the model - an effect akin to "joining the dots".

A side effect of over-trained gates is the potential for *pinch off* to occur. Pinch off is caused by the magnitude of the parameters in the gate becoming large, so that the outputs of the gate tend to one and zero. This effect is observed in mixtures when the number of experts is greater than the number of actual processes in the data. In a hierarchy, however, the effect can lead to many parameters being unused when a high level branch is pinched off.

Chapter 4 described how to train the mixtures-of-experts model using a combination of the EM algorithm and a conjugate gradient algorithm. This section describes how to reduce over-fitting of the training data using the method of *early stopping*. An alternative method will be described in Chapter 7 based on Bayesian methods.

The technique of early stopping states simply that a model should be trained on the training data until its error on a validation set reaches a minimum. Figure 5.1 shows the behaviour of the training and validation errors for the `hme-ese-1` method.

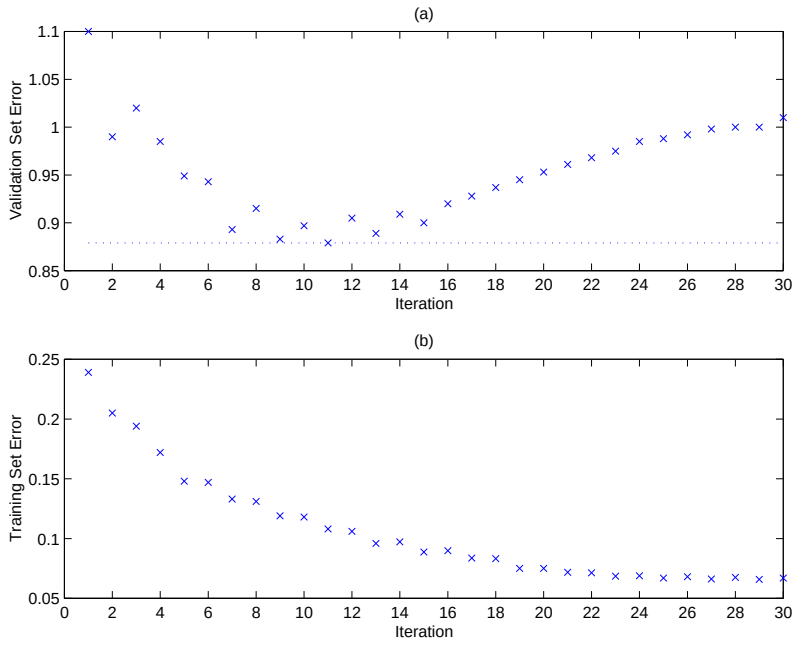

Figure 5.1: **Early stopping of an HME model on the Boston housing data.** This figure shows the behaviour of early stopping for an HME on one of the Boston data training sets (128 cases) using the `hme-ese-1` method. The upper figure (a) shows the evolution of the error on the validation set which is a randomly chosen third of the total data (43 patterns) against iterations of the algorithm. The lower figure (b) shows the evolution of the error on the training data which is the remainder of the total data (85 patterns). The minimum validation error is shown in sub-figure (a) via a dashed horizontal line at the value 0.879. This minimum is achieved after 11 iterations of the algorithm.

The first time I ran the `me-ese-1` training algorithm on the Boston housing data I checked when the minimum on the validation set was achieved. I noticed that the minimum was achieved after only the 2nd iteration of the EM cycle. This indicated to me that the conjugate gradient algorithm was actually working *too* well, and causing the minimum of the validation error set to be missed. To remedy this I

reduced the number of line searches (the variable *itmax* in `macopt`)[1] from 10 to 3, a value which was computed empirically by looking at the minimum found over a series of runs of the training algorithm. Subsequent comparison of the errors on unseen data justified this move, as errors for $itmax = 3$, were significantly lower than for $itmax = 10$.

In order not to stop training too early, the algorithm searches for additional minima after finding the first minimum so that the final iteration is at least 1.5 times the iteration at which the last minimum occurred. In addition, at least 30 iterations of the algorithm are performed, with a maximum of 200 total iterations allowed.

In the algorithms `me-ese-1` and `hme-ese-1` the total data available for training were divided into a training set consisting of two thirds of the total data and a validation set consisting of one third of the total. These sets were chosen at random every time a new pair of {train:validate} sets was required. After training a specified number of models on the different sets created by this method, I stored these models and used the average of their predictions, in a *committee* [14, pp. 364], [174] to form the overall prediction on unseen data. If we represent the prediction of the m^{th} committee member by $\hat{y}(m)^{(n)}$, the prediction of the committee is given by:

$$\hat{y}^{(n)} = \frac{1}{M} \sum_{m=1}^{M} \hat{y}(m)^{(n)}, \tag{5.1}$$

where M is the total number of members in the committee.

In order to decide how many committee members to use, the method of Rasmussen [189] was followed. In this method, a minimum of 3 committee members and a maximum of 50 are created. After each committee member has been trained, the overall time elapsed (in seconds) since starting to train the members is examined. If this time is greater than a critical threshold T_{crit}, no more members are created. T_{crit} is chosen to be $1.8765 \times N$ seconds where N is the number of points in the training set. The value of T_{crit} was chosen so that a training set with 1024 points would take 32 minutes to train on [189].

Ideally, it would be nice to use the time constraint to decide on the number of points to leave out in the cross validation process. However, since in practice the length of time taken by each run of the training algorithm may vary, and is not known in advance, the ideal method is not realisable. An approximation to the ideal method could be implemented by training on a small subset of the training data and using information about the scaling properties of the algorithm to make a decision about the dimension of the CV process. In practice I believe the method used here provides a reasonable solution to the choice of free parameters in the early stopping process.

Table 5.1 shows the behaviour of algorithm `hme-ese-1` as the number of committee members are reduced for the task Pumadyn-32nh/accel/std.64 (see Chapter 3). It seems that for this task increasing the number of committee members over 5 gives no significant improvement in loss. Up to 5 members, however, each new member gives a significant decrease in loss.

[1] The `macopt` algorithm was introduced in Section 4.4.

Members	Loss	Standard Error	p-value
1	0.00210	1.62	1.24e-08
2	0.00177	1.37	1.47e-05
3	0.00166	1.28	0.003
4	0.00163	1.26	0.036
5	0.00160	1.24	0.517
6	0.00158	1.22	0.498
10	0.00158	1.23	0.645
20	0.00159	1.23	0.547
30	0.00159	1.23	0.738
40	0.00159	1.23	N/A

Table 5.1: **Effect of committee size on the performance of the `hme-ese-1`method.** The table shows the effect of the numbers of committee members on the performance of algorithm `hme-ese-1` on the task Pumadyn-32nh/accel/std.64. The p-values are the result of pairwise t-tests relative to the default committee with 40 members.

5.3 Choosing the Number of Experts and Depth of Hierarchy

One point I have missed out so far in the description of the `me-ese-1` and `hme-ese-1` methods is the specification of architecture - both the shape and depth of the mixtures and hierarchies. Both methods used fixed architectures, *i.e.*, the number of experts and gates is fixed in advance and remains the same throughout training and for each member of the committee.

The number of experts and gates in each model is decided by balancing the number of parameters in the architecture and the number of points in the training set. This is a simple method for architecture selection used by Rasmussen [189] in his `mlp-ese-1` method for specifying the number of hidden nodes in a multi-layer perceptron.

In the `me-ese-1` method only a mixture of experts is used, so only one gate is included. The number of experts I is given by:

$$I = \frac{N}{(d+1)(k+1)} \tag{5.2}$$

where N is the number of training examples, d is the dimension of $\boldsymbol{x}$ and k is the dimension of $\boldsymbol{y}$. In the `hme-ese-1` method a hierarchical mixture of experts is used. The number of branches at each node in the hierarchy is fixed at 2, *i.e.*, the architecture takes the form of a binary tree. In this case the free parameter is the depth of the tree, M, which is given by:

$$M = \frac{1}{\log 2}\left(\log(N + 2(d+1)) - \log(d+1) - \log(k+2)\right) \tag{5.3}$$

It is interesting to note that the `me-ese-1` method will allocate proportionally more resources to expert networks than the `hme-ese-1` method which will allocate more gating networks.

1. Randomly divide the training data into an estimation set and validation set in the ratio $2:1$.
2. Initialise a new committee member $\mathcal{H}$ using either a fixed width mixture with number of experts set by Equation 5.2 (for `me-ese-1`) or a fixed depth binary hierarchy with depth set by Equation 5.3 (for `hme-ese-1`).
3. Set $iter = 0$.
4. Perform an EM step of `ml-train` for $\mathcal{H}$ on the estimation set.
5. If the error on the validation set $VE^{(iter)}(\mathcal{H})$ is less than the best error $VE^{*}(\mathcal{H})$, set $iter^{*} = iter$.
6. Set $iter = iter + 1$.
7. If [$iter < 30$ or $((iter < 1.5 \times iter^{*})$ and $(iter < 200))$] go to 2.
8. If (time elapsed $> T_{crit}$ and number of committee members > 3) terminate, else go to 1.

Figure 5.2: **The `me-ese-1` and `hme-ese-1` methods**

5.4 Summary

This chapter has described two learning methods based on the theory of mixtures-of-experts outlined in Chapter 2 and Chapter 4 which are summarised in Figure 5.2. These two learning methods are based on early stopping using a fixed architecture. In Chapter 8 results on the DELVE data sets will be given for these methods. In Chapters 6 and 7 three more learning methods are described which address complexity control in different ways to the methods presented in this chapter.

Chapter 6

Growing Mixtures of Experts

6.1 Introduction

In Chapter 5 two learning methods for classification and regression tasks using mixtures-of-experts were described. These methods used a prior choice of architecture based on balancing the number of parameters in the model to the number of data points in the training set. This choice of architecture is crude, however, since it fails to take account of the complexity of the tasks. In some cases the number of parameters in the models may be too large, resulting in potentially overly complex functions, whilst in other cases, there may be too few parameters, resulting in *under-fitting* of the data. Indeed, given an arbitrary complex function, there is no reason to expect that increasing the number of samples from the function should require more parameters to model it, and such a rule is also therefore likely to produce models that will tend to overfit the data. In the previous chapter these problems were addressed by the use of committees of models trained by early stopping on validation sets. This method was mostly successful, as will be shown on regression and classification tasks in Chapter 8, but suggests further refinements which are investigated in this chapter.

This chapter approaches the problem of architecture selection via a recursive procedure which recursively adds parameters to a model in the fashion of a tree growing algorithm [21]. At each step of the growing algorithm a terminal node is split into a a pair of terminal nodes and a gate. Nodes are split in a best first manner, with the node that increases the likelihood of the model by the greatest amount retained. After each addition to the tree, the whole tree is retrained using the EM described in Chapter 4.

Complexity control of the growing algorithm is achieved as in Chapter 5 by the use of a committee whose members are trained by early stopping on validation sets. In the growing algorithm, however, the search for the minimum of the validation set takes on an extra dimension - a search over architecture sizes. In standard tree growing algorithms, this method has not been found to result in good performance. Both CART and C4.5 grow trees to convergence on the training data and subsequently prune back nodes to select the optimal rooted subtree on a validation set. In this chapter I describe why this pruning algorithm is not appropriate for growing an HME, and give empirical justification for the early stopping method.

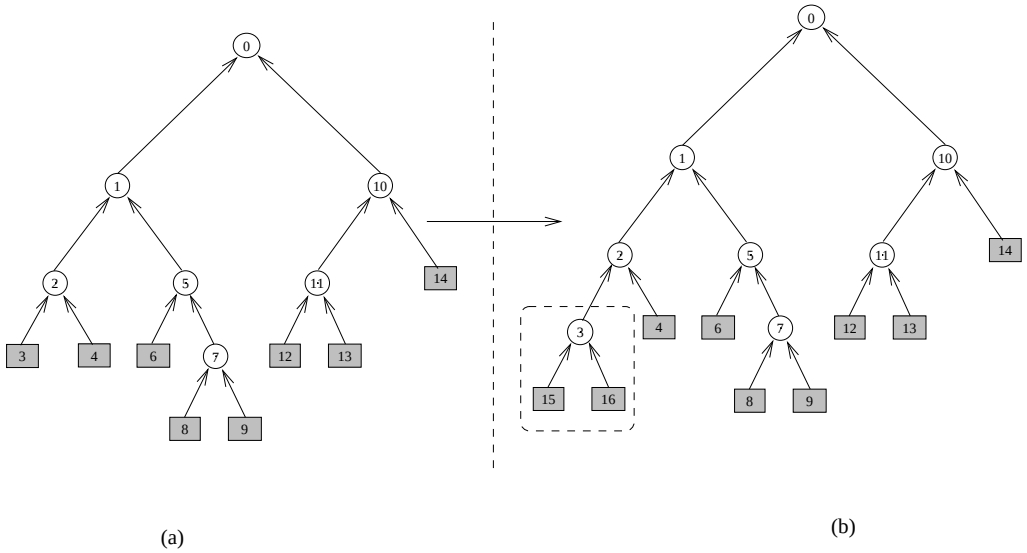

Figure 6.1: **Tree growing for HME models**. Two schematic diagrams of hierarchical mixtures-of-experts models are shown. Each terminal node of the trees contains an expert network, and each non-terminal node contains a gate. The original tree (a) has been extended to give the new tree (b) by splitting terminal node 3 into a pair of terminal nodes 15 and 16.

6.2 Design of the Growing Algorithm

In this section I describe the development of the `hme-grow-1` algorithm. In summary, this algorithm constructs a hierarchical mixture of experts from data, learning both the structure (or *architecture*) of the hierarchy as well as the parameters of the gates and experts. This is in contrast to the `me-ese-1` and `hme-ese-1` methods which have predetermined architectures based purely on number of attributes and cases in a data set. `hme-grow-1` starts from a single expert (either linear regression or multinomial logit models) and makes a split of this expert into a mixture of 2 experts and a gate. How this split is made is described in the next section. This mixture is subsequently retrained to convergence and another split of each expert made. At this point, the split which results in the largest *local* improvement in the auxiliary function is kept and the overall architecture retrained. For node $\mathcal{E}_i$, the difference between the auxiliary function after a split, R_i^2, and the auxiliary function before the split was made, R_i^1, is given by:

$$\delta R_i = R_i^2 - R_i^1 \tag{6.1}$$

where

$$\begin{aligned} R_i^1 &= \sum_n \zeta_i^{(n)} \left(\log P(\boldsymbol{y}^{(n)} | \boldsymbol{x}^{(n)}, \mathcal{E}_i, An(\mathcal{E}_i), \mathbf{w}_i) + \log P(\mathcal{E}_i, An(\mathcal{E}_i) | \boldsymbol{x}^{(n)}, \theta) \right) \\ R_i^2 &= \sum_n \sum_{j \in Ch(\mathcal{E}_i)} \zeta_j^{(n)} \left(\log P(\boldsymbol{y}^{(n)} | \boldsymbol{x}^{(n)}, \mathcal{E}_j, An(\mathcal{E}_j), \mathbf{w}_j) + \log P(\mathcal{E}_j, An(\mathcal{E}_j) | \boldsymbol{x}^{(n)}, \theta) \right). \end{aligned} \tag{6.2}$$

The splitting process continues by alternately retraining the model and making splits of the terminal nodes. Since it is impractical to search over all possible tree shapes the search is confined to binary trees.

The overall growing algorithm for `hme-grow-1` can be summarised as:

1. Train whole tree to convergence using standard algorithm `ml-train`.
2. Choose a node to split.
3. Split the chosen node and train.
4. If the training error has converged then exit, else, go to (1).

This growing algorithm is used to generate a model which fits the training data well. In order to control the complexity of the model, early stopping using a validation set is used. I now describe the components of this general algorithm, starting with the method used to choose a node to split.

Choosing a node to split

In deciding which node to split, the growing algorithm must decide which node will give, in the short term at least, the best improvement in likelihood when it is split. There are two different methods that have been explored in the literature to decide how to choose the node to split: those based on statistics gathered before the split is made (prior-stats methods), and those based on iterative search through all possible splits (iterative-search). Most recursive partitioning methods [*e.g.,* 21, 58] utilise the iterative-search method. In other HME growing algorithms, Waterhouse and Robinson [238] used an iterative-search method, whilst Saito and Nakano [204] and Fritsch et al. [62] used prior-stats methods. Critics of the iterative-search method have pointed out that the computational cost of evaluating each split can be large if the parameters of each split have to be trained in order to be evaluated. However, the major problem with the prior-stats methods seems to be the lack of guarantee of choosing the best node, *i.e.,* the one which will give the best local improvement in the performance of the model. In this chapter I use the iterative-search method, which I have found empirically to be the most robust method for choosing which node to split. In summary, to choose a node to split, `hme-grow-1` splits *all* terminal nodes, evaluates the benefit of each split, and keeps the best. I now turn to how exactly a split is made in this algorithm.

How to make a split ?

In previous work on growing HME models, [*e.g.,* 238, 204, 62], there has been little discussion of how to actually make a split of a node. Most of the description has focussed on the choice of node, as discussed in the previous section. Previously, in order to split a node, the following method was used:

- Replace the original expert network with a pair of expert networks and a gate.
- Initialise the parameters of the expert networks with copies of the original expert network parameters, optionally corrupted with random noise.
- Initialise the parameters of the new gate with random values in a pre-specified range.

The use of randomly perturbed versions of the previous expert's parameters is to break symmetry. Clearly if the same values were used, the resulting mixture would not differ from the original expert, even if the gate was initialised with random values.

I investigated the performance of this algorithm for creating splits, but found a number of problems with it. The degree of perturbation of the expert's parameters has a large effect on the success of the split (where success is defined as a split which improves upon the original expert). If the random perturbation is too small, the new experts fail to break out of the symmetric solution, and end up with the same parameters as the original expert. If the random perturbation is too large, it dominates the expert parameters and the training suffers from the same problems of random initialisation as described in Section 4.5. Indeed, I found that random initialisation of the posterior probabilities of the new experts and appropriate scaling by the parent posterior probabilities gave more stable results than the use of random perturbation of expert and gate parameters. This is disappointing since it ignores all that has been learnt by the previous expert about the region.

A geometric argument

One alternative to the use of random perturbations of expert parameters is to appeal to a geometric argument. In this method, the two new expert parameters are initialised by a deterministic adjustment of the original expert parameters, according to the rule:

$$\mathbf{w}_{left} = \mathbf{w}_{orig} + \delta\mathbf{w}, \quad \mathbf{w}_{right} = \mathbf{w}_{orig} - \delta\mathbf{w}, \tag{6.3}$$

where $\delta\mathbf{w}$ is computed via a geometric argument. In this, the original parameter vector is rotated about its mid point relative to the data local to the expert. The angle of rotation is related to the noise level σ_i^2 for the expert, and is used to define $\delta\mathbf{w}$ using trigonometric rules.

Empirical results on simple 1-d functions (not shown) failed to show that the geometrical argument could generate good splits. Analysis of the training after initialisation suggested that the new parameter vectors were too close to the original parameter vector. Retraining of these vectors therefore caused them to revert to the original rather than diverge towards new configurations. This suggests that more radical adjustment of the parameter vectors is required. Ideally, a full search through all adjustment angles would be preferred. This is clearly impractical even in the case of 1-d functions. In the case of higher dimensional problems, the geometric argument also appears impractical even if a good adjustment could be found.

A threshold search method

Empirical investigation of both the random perturbation and geometric methods failed to show good performance for either algorithm. In this section a more successful technique for splitting nodes is described, which is based on techniques of recursive partitioning. A good introduction to recursive partitioning is given by Breiman et al. [21]. The algorithm I use, `best-split`, is described in Figure 6.2. For each terminal node in an existing tree, `best-split` investigates making a split into a pair of terminal nodes. First the data local to each node i is determined by sorting the data $D = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_1^N$ into a

local sub-region D_s if the joint posterior probability of the node $\zeta_i^{(n)}$ is greater than 0.01. This sub-region is then divided into two regions. The division is made by using a threshold on one of the attributes of $\boldsymbol{x}$, *e.g.*,

$$\begin{aligned} &\text{if } x_j < t_j \text{ then put } \boldsymbol{x} \text{ into left region;} \\ &\text{else put } \boldsymbol{x} \text{ into right region.} \end{aligned} \tag{6.4}$$

To find the best attribute of $\boldsymbol{x}$ on which to make the division, and the best threshold, the following procedure is used. For each element j of $\boldsymbol{x}$ the data is sorted[1]. Given this sorted data, each possible threshold may be tested. For the n^{th} data point in the sorted data the threshold $t_j^{(n)}$ tested is the mid point of the current and next data point:

$$t_j^{(n)} = \frac{x_j^{(n+1)} + x_j^{(n)}}{2}. \tag{6.5}$$

Each threshold defines a possible division into left and right regions of the data. Ideally, we would like to fit the two regions using linear regression experts. However, this would be computationally very expensive since for a sub-region with N_s points in it, $2N_s$ linear fits would need to be computed. Therefore, two approximations are made: either to fit each of the two regions using piecewise constant functions and compare the resulting error on the sub-region with a single constant fit, or to consider a smaller number of sub-regions than N_s and to fit linear regressions to the splits made in these sub-regions. In both approximations the best split is selected and a pair of experts and a gate trained on this split. The trade offs between these two methods are discussed in the next section. Given the new experts, the increase in local likelihood made by this split is computed. For each terminal node the attribute and threshold which give the largest increase in local likelihood are retained. Comparison of the increases for each terminal node then gives the best node to split.

Once the best node to split has been found, the `best-split` algorithm terminates, returning the best feature and associated threshold to use. These are then used to initialise the gate and pair of experts of the new split. This initialisation is performed by dividing the data into two regions and defining the posterior probabilities of the left and right experts according to the division. These posterior probabilities provide targets for the gate and weighting factors for the experts in the usual way. However, as will be shown in the following sections, division of the data via a hard threshold can lead to undesirable behaviour, and so a smoothing operation is performed on the posterior probabilities of the new experts.

Making a test split

In order to make the algorithm above practical, it is necessary to make each test split and its evaluation as efficient as possible, since there are $N_s \times d$ splits to be evaluated, where N_s is the number of training examples in the reduced data set D_s for each node and d the dimension of $\boldsymbol{x}$, for each node. I chose two methods to do this, one based on exact evaluation of a reduced set of possible splits, and the other based on approximate evaluation of the full set of splits.

[1]The sorting was performed using the QuickSort algorithm [180].

- For the node to be split, create a sub-region of data specific to this node, D_s, which represents the data most likely to be used by this split. This is done by filtering the data according to the joint probability of the node $\zeta_p^{(n)}$:
  - For all $\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}$:
  - If $\zeta_p^{(n)} > 0.01$, put $\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}$ into the sub-region D_s for this node,
  - else discard $\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}$.
- For each attribute j of $\boldsymbol{x}^{(n)}$, apply the following algorithm:
  - Sort D_s along the attribute j.
  - For all examples $x_j^{(n)}$ in D_s:
    - Set the threshold $t_j = (x_j^{(n)} + x_j^{(n+1)})/2$.
    - Generate a test split using the threshold t_j and record the score of this split.
- Record the best split, characterised by the best feature j^* and best threshold t_j^*.

Figure 6.2: **The `best-split` algorithm**

In the reduced set method, I divide the sorted data into 4 regions. In each region I make a split into two sub-regions, setting the threshold at the mid point of the region. I then use this threshold to initialise a complete pair of experts and a gate, using the algorithm of the next section. I choose the split out of the 4 possible splits that gives the best increase in likelihood.

In the full set method, I evaluate splits at all possible thresholds t_j, but use an approximate method to evaluate the improvement that each split makes. Each approximate split is a pair of piecewise constant functions, with values computed by the average of the targets to the left and right of the threshold, weighted by the posterior probabilities $\zeta_p^{(n)}$ of the original expert in each case, and normalised by the sum of posterior probabilities:

$$\hat{\boldsymbol{y}}_{left}^{(n)} = \sum_{n_{left}} \zeta_p^{(n)} \boldsymbol{y}^{(n)} / \sum_{n_{left}} \zeta_p^{(n)} \tag{6.6}$$

$$\hat{\boldsymbol{y}}_{right}^{(n)} = \sum_{n_{right}} \zeta_p^{(n)} \boldsymbol{y}^{(n)} / \sum_{n_{right}} \zeta_p^{(n)} \tag{6.7}$$

where $\{n_{left}\}$ and $\{n_{right}\}$ are the set of examples lying to the left and right of the threshold t_j. This gives a fast but approximate method of computing the increase that would be made by splitting the node into a pair of experts and a gate, but without actually doing so. This is the method I use in `hme-grow-1` in preference to the reduced set method since it is less computationally demanding and still leads to good splits in practice.

- For all data points, $\{\boldsymbol{x}^{(n)}\}$,

$$\text{if } x_j^{(n)} < t_j, \quad \text{set } \varphi_0^{(n)} = 1, \varphi_1^{(n)} = 0, \zeta_0^{(n)} = \zeta_p^{(n)}, \zeta_1^{(n)} = 0, \tag{6.10}$$

$$\text{else} \quad \text{set } \varphi_0^{(n)} = 0, \varphi_1^{(n)} = 1, \zeta_0^{(n)} = 0, \zeta_1^{(n)} = \zeta_p^{(n)}. \tag{6.11}$$

  where $\zeta_p^{(n)}$ is the joint posterior probability of the expert which was split to generate the mixture (and hence the joint posterior probability of the parent of experts $\mathcal{E}_0$ and $\mathcal{E}_1$). The subscripts 0 and 1 denote the left and right experts in the split $\mathcal{E}_0$ and $\mathcal{E}_1$.

- Use posteriors $\{\varphi_0^{(n)}, \varphi_1^{(n)}\}$ as targets for gate in the maximum likelihood equations, weighted by $\zeta^{(n)}$.

- Use posteriors $\{\zeta_0^{(n)}, \zeta_1^{(n)}\}$ as weights for the experts $\mathcal{E}_0$ and $\mathcal{E}_1$ in the maximum likelihood equations.

Figure 6.3: **The `split-posts` algorithm**

From rules to gates

The search algorithm, described in the previous section, over all attributes generates a threshold t_j for a particular attribute x_j. This implies a rule:

$$\text{if } x_j < t_j \qquad \text{use left expert} \tag{6.8}$$

$$\text{else} \qquad \text{use right expert} \tag{6.9}$$

which could in principle be used as the gate in a mixture of experts. In this case, the mixture would be more akin to a recursive partitioning model such as CART. In this work, however, rather than using the rule *as* the gate, I use it to *initialise* the gate and pair of experts of the new split.

A rule such as Equation 6.9 can be visualised as a Heaviside or *step* function. This defines a segmentation of the data around the threshold t_j. I use this segmentation to generate posterior probabilities $\zeta_0^{(n)}$ and $\zeta_1^{(n)}$ for the left and right experts of the new split as described in algorithm `split-posts` shown in Figure 6.3. The posterior probabilities resulting from the rule are used as targets in conjunction with the training data to initialise the gate and experts.

Smoothing the split

The algorithm described above to generate posterior probabilities of nodes from rules was successful at reducing the error rate on training data, but led to "spiky" functions in its predictions. This effect is shown in Figure 6.4. These spiky functions are piecewise linear in form, a mode which is at one end of the model space for the mixture of experts. Clearly the search bias imposed by the split initialisation algorithm has encouraged this extreme of the model space.

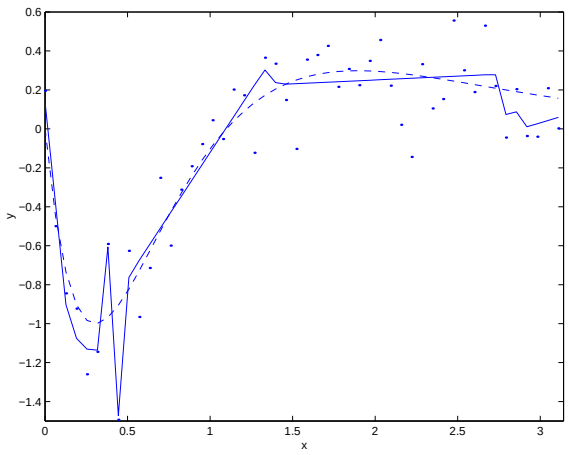

Figure 6.4: **Splitting using hard targets.** A "spiky" function resulting from the growing algorithm using hard targets in `split-posts`. The prediction (solid line) is a poor representation of the underlying function (dashed line) used to generate the training data (shown with dots) is which is the artificial data set introduced in Chapter 3.

I believe the spiky functions are less representative of true functions than a smooth function would be. This belief represents a prior on smooth functions which I would prefer to see as predictions of the model. Whether this prior is a good one will depend on the type of underlying functions in the training set. In this work I encourage smooth functions by altering the search method in the split initialisation of `split-posts`. In order to smooth the split implied by the rule $x_j^{(n)} < t_j$, I consider the probabilistic interpretation of the rule, *i.e.*,

$$\text{if} \quad x_j^{(n)} < t_j, \quad P(\mathcal{E}_0|\boldsymbol{x}^{(n)}) = 1, \quad P(\mathcal{E}_1|\boldsymbol{x}^{(n)}) = 0 \tag{6.12}$$

$$\text{else} \quad P(\mathcal{E}_0|\boldsymbol{x}^{(n)}) = 0, \quad P(\mathcal{E}_1|\boldsymbol{x}^{(n)}) = 1, \tag{6.13}$$

where $P(\mathcal{E}_0|\boldsymbol{x}^{(n)})$ is the probability that $\mathcal{E}_0$ generated the data $\boldsymbol{x}^{(n)}$. Since the outputs of the gate are given by $P(\mathcal{E}_0|\boldsymbol{x}^{(n)}, \mathbf{v})$, and $P(\mathcal{E}_1|\boldsymbol{x}^{(n)}, \mathbf{v})$, in order to *compile* the rule into a gate, a method for setting the parameter vector $\mathbf{v}$ is required. In the previous section I used the posterior probabilities implied by the rule as targets to train the gate parameter vector. The neater solution described in the sequel also yields the smoother functions desired by the prior described previously.

Direct conversion of the rule into the parameter vector is not possible, since in order to represent the step function exactly, the parameter vector must have terms approaching infinity. This can be seen by consideration of the sigmoid function represented by the gate. The gate output $P(\mathcal{E}_0|\boldsymbol{x}^{(n)}, \mathbf{v})$ has the following form:

$$P(\mathcal{E}_0|\boldsymbol{x}^{(n)}, \mathbf{v}) = \frac{1}{1 + \exp(-a)} \tag{6.14}$$

where $a = \sum_{l=1}^{d} v_l x_l^{(n)} + v_0$. For a particular attribute x_j only, a reduces to $a = v_j x_j^{(n)} + v_0$. However, rearrangement of Equation 6.14 gives another expression for a:

$$a = \log\left(\frac{p}{1-p}\right) \tag{6.15}$$

where $p = P(\mathcal{E}_0|\boldsymbol{x}^{(n)}, \mathbf{v})$. Values for p of 1.0 and 0.0 give values for a of ∞ and $-\infty$ respectively. At the boundary of the rule ($x_j^{(n)} = t_j, p = 0.5$), the value of a is 0.0.

In order to get a smooth function for p, I convert the rule into a sigmoid function in the following manner. The data in the region of the node is sorted into a data set D_s. The following conditions are then used for p, the probability of the left expert $\mathcal{E}_0$:

- $p = 0.95$ at x_{left}, the 1st interquartile position of D_s;
- $p = 0.5$ at x_{mid}, the median of D_s;
- $p = 0.05$ at x_{right}, the 3rd interquartile position of D_s.

Using the fact that $a = v_j x_j^{(n)} + v_0$ together with these conditions gives the following simultaneous equations for v_j and v_0:

$$v_j x_{left} + v_0 = \log\left(\frac{0.95}{1.0 - 0.05}\right); \tag{6.16}$$

$$v_j x_{mid} + v_0 = \log\left(\frac{0.5}{0.5}\right). \tag{6.17}$$

Note that the condition at x_{right} is not used since the smooth is symmetrical about x_{mid}. These values for the gate parameters are used to generate probabilities for the left and right experts and to set the values of the conditional posterior probabilities $\varphi_0^{(n)}$ and $\varphi_1^{(n)}$:

$$p = \frac{1}{1 + \exp[-(v_0 + v_j x_j^{(n)})]} \tag{6.18}$$

$$\varphi_0^{(n)} = p, \quad \varphi_1^{(n)} = (1-p) \tag{6.19}$$

$$\zeta_0^{(n)} = p \times \zeta_p^{(n)}, \quad \zeta_1^{(n)} = (1-p) \times \zeta_p^{(n)}. \tag{6.20}$$

These posterior probabilities are then used in conjunction with the data to train the experts and gate. The use of this initialisation for the gate within `split-posts` gives smoother functions, as shown in Figure 6.5.

Pruning or Early Stopping

In order to control the complexity of the models built by the tree growing procedure I investigated two methods: *pruning* and *early stopping*. The pruning method works as follows. The training data is divided into a training set and a validation set. The tree growing algorithm is iterated till convergence

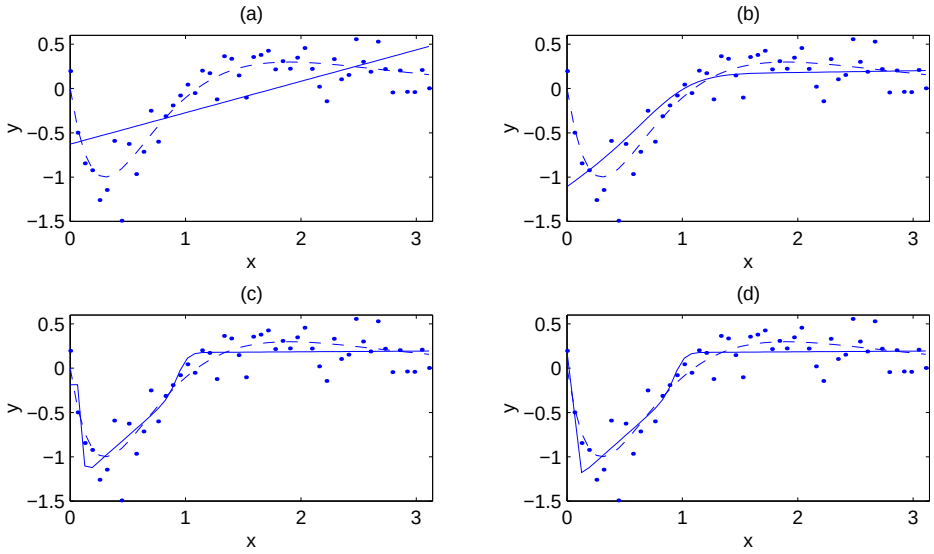

Figure 6.5: **Growing a smoother function.** Each sub-figure (a-d) shows the prediction of the model as a solid line, the generating function as a dashed line and the training data as dots. The predictions shown in each of the sub-figures are generated from models with successively more nodes. Sub-figure (a) shows the prediction of the initial model - a single linear regression expert. After one cycle of the growing algorithm, a mixture of 2 experts is obtained whose prediction is shown in sub-figure (b). Sub-figures (c) and (d) show the predictions after 3 and 4 terminal nodes have been grown onto the model. After 4 terminal nodes there is no significant decrease in the training error.

on the training set, resulting in a large tree that typically overfits the training set. This large tree is then *pruned* using the validation set. Pruning involves selecting the optimal rooted sub-tree that gives the minimum error on the validation set.

During pruning, each non-terminal node of the tree is iteratively replaced by a terminal node which is retrained using the posterior probabilities of the non-terminal node and training set. I also investigated retraining new terminals using the validation set and the relevant posteriors inferred from it. Since the search space of possible sub-trees is large, only the best non-terminal / terminal replacement is retained at each stage of the pruning algorithm. This procedure is iterated until no further improvement on the validation set is found by pruning.

Unfortunately, although the pruning algorithm was somewhat successful on small simulations, it was not successful on real world or realistic problems, as shown in Table 6.1. On many of the training sets of the Boston data, for example, the pruning algorithm failed to reduce the size of the grown tree at all, resulting in an over-trained model which generalised poorly. One possible reason for the failure of the pruning algorithm lies in the distinction between conventional decision and regression tree algorithms and the HME and its growing algorithm. In conventional tree growing, once a decision about a split has been made (and retained in the tree), this decision is kept fixed. Only lower levels of the tree are then updated according to the growing algorithm. During pruning, therefore, the merging of terminal nodes together yields models which could have also been grown, since the lower levels of the tree are

Data Set	Size	me-ese-1	hme-grow-0	p-value
Boston	32	0.366	0.402	0.274
Boston	64	0.234	0.525	0.033
Boston	128	0.160	0.216	0.086
Kin-8nm	64	0.597	0.659	0.006
Kin-8nm	128	0.489	0.525	0.054
Kin-8nm	256	0.387	0.440	1.1e-5
Kin-8nm	512	0.301	0.395	8.7e-13
Kin-8nm	1024	0.230	0.385	2.7e-5

Table 6.1: **Pruning HME models.** A comparison of squared error loss for methods me-ese-1 and hme-grow-0 on two tasks Boston and Kin-8nm using different numbers of training examples (Size). hme-grow-0 is the growing algorithm for HME models using pruning to control complexity. The p-values in the fifth column are the results of pairwise t or F tests between hme-grow-0 and me-ese-1. p-values less than 0.05 indicate that the hme-grow-0 method was significantly outperformed at the 95% level by the me-ese-1 method. Only the Boston data with 32 data points shows no significant difference between the methods.

independent of the higher levels during growing. In contrast, the HME growing algorithm (and general training algorithm), updates *all* parameters of the tree during training, and after each new node is added to the tree. To understand this, consider for example, the case of two experts in a mixture with the regions of influence of each expert overlapping significantly. Now let us split both of the experts in turn. If the left expert gives a much better improvement than the right expert, it will be retained. As a result, the gate will reallocate the regions so that the left node has more importance over the right expert and the degree of overlap will reduce. In this way, by splitting lower levels of the tree, the upper levels have been affected. Simply replacing a non-terminal node by a terminal node will therefore not be a reversal of the growing procedure, without retraining of the whole tree.

Because of the failure of the pruning algorithm I used an early stopping method for controlling the size of the trees. The early stopping algorithm for hme-grow-1 is similar to that of me-ese-1 and hme-ese-1. The division of the training data into training and validation sets is random, with $2/3$ for training and $1/3$ for validation. A series of committee members are generated by training and validating on different training and validation sets. The predictions of the committee members on out of sample data are then averaged.

There are two aspects to the search for the minimum on the validation set. After a split has been made to the tree and retained, the training algorithm is applied to the tree. The minimum on the validation set is then found and this model is saved. As was done for me-ese-1 and hme-ese-1, the speed of the conjugate gradient algorithm macopt was reduced by setting *itmax* to 3. Training proceeds until convergence on the training set is achieved, at which point a new split is made to the tree. In order to avoid stopping when too simple a model is found, the tree growing is allowed to proceed for at least 3 iterations. A new split is made if the best error on the validation set for the current tree shape is less than the best error for the previous tree shape. A maximum of 50 iterations are allowed. In order to place a limit on the number of committee members, once again a minimum number of 3 members are allowed.

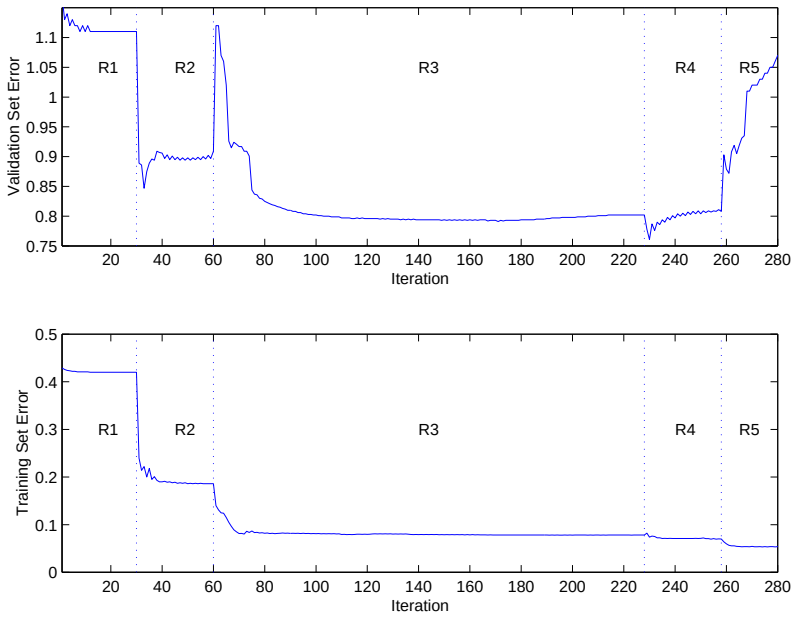

Figure 6.6: **Growing using early stopping.** The two figures show the evolution of the tree growing algorithm on the Boston data using a training set of 128 points. The top figure shows the evolution of the validation set squared error against the iterations of the training algorithm, while the lower figure shows the evolution of the training set squared error. The plots are divided into 5 regions, $R1$ to $R5$, which signify different sizes of trees grown. $R1$ is the initial tree shape with only one expert. $R2$ is the tree after one split has been made and so on. On the training set the error decreases monotonically each time a split is made and subsequently levels out to 0.05. A new split is made when the training error for a fixed tree size has converged. On the validation set the error also decreases from $R1$ to $R2$ and $R3$, but increases after an initial drop when $R4$ is entered, and steadily increases in $R5$. The minimum error is achieved in $R4$ at around iteration 230, and so this size tree is chosen as the final model. One interesting point is the jump in validation set error between $R2$ and $R3$ followed by the settling down of this error after 10 iterations to a lower value. This noisiness in the validation error demonstrates the need for appropriate search heuristics, such as used here, when using validation sets to choose model complexity rather than stopping when the validation set error increases.

If the overall time elapsed is less than $1.8765 \times N$ a new committee member is trained, up to a maximum of 50 committee members. An example of a run of the growing algorithm using early stopping is shown in Figure 6.6.

The growing algorithm behaves in an interesting manner on the Pumadyn data sets as shown in Figure 6.8. The histograms show the distribution of different sized trees in the committees formed for each of the Pumadyn tasks. The behaviour of the algorithm is markedly different for the 32 dimensional tasks than for the 8 dimensional tasks. For the 32 dimensional tasks smaller trees are built and the most frequent trees contain only 1 expert. For the 8 dimensional tasks the most frequent trees are still the ones with 1 expert but there are more trees with up to 10 experts in them and the distributions are less peaked towards most trees have one expert only. Another point of interest is that the number of larger trees increases as the non-linearity of the task increases and the level of the noise decreases. This implies

1. Start with a single expert in the model. Set *iter* to 1.
2. Train the model to convergence on the training set, storing the best performing model on the validation set $M^{(iter)}$. If the validation error of current model $VE(M^{(iter)})$ is less than the currently best overall model $VE(M^*)$, replace M^* by $M^{(iter)}$.
3. If $(iter > 3)$ and $VE(M^{(iter)}) > VE(M^{(iter-1)})$ (*i.e.,* no improvement has been made from the last model to the current), terminate.
4. For each terminal node in the model, find the best threshold and attribute using algorithm `best-split`. Generate posterior probabilities for the test split using algorithm `split-posts`. Train a new gate and pair of terminals using these posterior probabilities and record the increase in auxiliary function δR_i for the node that was split.
5. Keep the split which gives the biggest increase in auxiliary function and delete all other test splits.
6. Set $iter = iter + 1$.
7. Go to 2.

Figure 6.7: **The `hme-grow-1`method**

that the growing algorithm allocates more resources to modelling non-linear data but reverts to simpler trees for higher noise / more linear problems.

Comparison of the speed of `hme-grow-1` on 32 dimensional data sets with its speed on 8 dimensional data sets indicates that it takes 1.8 times longer on average on the 32 dimensional data than on the 8 dimensional data. This compares with a figure of 1.5 times longer for the methods `me-ese-1` and `hme-ese-1`. The growing algorithm is therefore slightly slower as the dimensionality of the data increases, as might be expected from the nature of the search algorithm employed.

6.3 Related Work

Model selection for mixture of experts has also been considered by a number of authors. Zeevi, Meir and Adler [254] used a recursive procedure based on incrementally adding more experts in a flat mixture to an original configuration of 2 experts. If, after retraining, the new expert parameters are sufficiently different, the new expert is retained in the model, otherwise the algorithm is terminated. This somewhat ad-hoc procedure (which has no description of how to decide when the experts are sufficiently different) did yield good performance in their studies of time series prediction on sunspots and Canadian Lynx series. A companion paper, [255] presented a theoretical analysis of error bounds of mixture of experts models, and suggested the use of an information criterion similar to the Network Information Criterion (NIC) [149].

Jacobs et al. [105] use a Bayesian approach to perform model selection in hierarchical mixtures of experts architectures. Their approach starts with initially complex HME architectures which are pruned

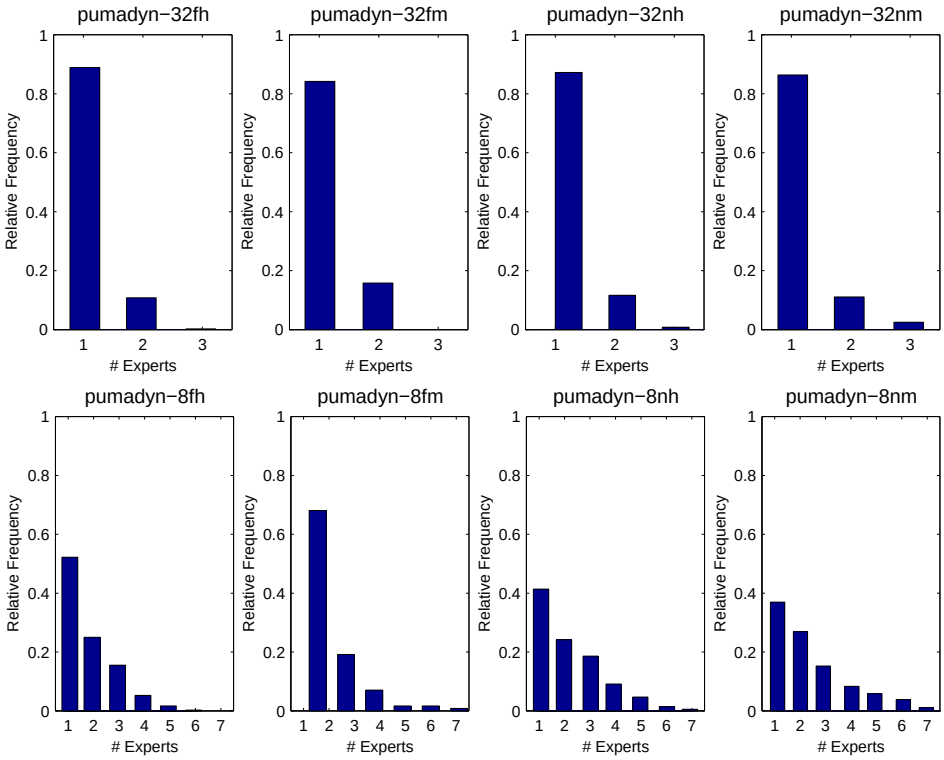

Figure 6.8: **Distribution of tree size for the growing algorithm.** The plots show the distribution of the number of terminal nodes (experts) in the committees of trees for each of the Pumadyn data sets. The vertical axis is the relative frequency which is the number of trees in each of the bins normalised by the total number of trees in all the bins.

using *worth indices* for experts computed using Gibbs sampling. The worth index for an expert is defined as:

$$\text{worth index for expert } \mathcal{E}_i = \frac{1}{N}\sum_{n=1}^{N} z_i^{(n)} \tag{6.21}$$

i.e., the normalised sum of random indicator variables $z_i^{(n)}$. The worth indices are realised using simulated values from a Gibbs sampler and give an indication of how useful an expert is in a model. Jacobs et al. [105] use them to either prune away experts from a model or to indicate that a more complex model should be trained. The approach was shown to be successful by Jacobs et al. [105] at selecting architectures for two classification tasks: breast cancer [246] and phoneme classification [175][2]. One disadvantage of this approach, however, is the high degree of computational cost apparent in the Gibbs sampling process. Results by Rasmussen [189], however suggest that even when computational resources are fixed, MLPs and Gaussian processes implemented using Markov chain Monte Carlo techniques outperformed those trained by other methods. Whether this result will extend to other models, such as mixtures of experts, and other data sets will surely be the subject of future research in this area.

Other approaches to *tree growing* in hierarchical mixtures of experts have been proposed. In work which predates this chapter, I investigated the properties of tree growing on a small simulated classification task, the parity data set [238]. The method used was similar to the one described in this chapter, consisting of evaluation of the maximal increase in node likelihood after splitting of every terminal node. The splitting method was not robust, however, consisting of randomisation of the gate parameters, and random perturbation of the split expert parameters. In addition, no method for complexity control was discussed (the parity data is an exact classification problem).

Fritsch et al. [62] suggested an alternative approach to tree growing in HME models [see also 60]. Their method used *expert dependent scaled likelihoods* computed during training to select the terminal node to split. For expert $\mathcal{E}_i$, the scaled likelihood is given by:[3]

$$l_k = \sum_n P(\mathcal{E}_i, An(\mathcal{E}_i)|\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}, \boldsymbol{\theta}) \log[P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, \mathcal{E}_i)]. \tag{6.22}$$

The expert with the minimum l_k was selected from the tree, and split using random parameters for the gate, and random perturbation of the expert parameters to give a pair of new experts, in a similar manner to Waterhouse and Robinson [236]. The new tree is then trained till a minimum error is found on a validation set, at which point a new tree shape is hypothesised. The growing algorithm is evaluated on a phoneme classification task [175] and on a portion of Switchboard [75], a large vocabulary speech recognition task as part of the JANUS [230] speech recogniser. The growing method for HME models outperformed fixed architecture HME models on both tasks, although the results on Switchboard

[2]The breast cancer and phoneme data sets are available at the UCI Repository [143].

[3]This differs slightly from the formulation of Fritsch et al. [62] and Fritsch [60], who appear to use the probabilities $P(\mathcal{E}_i, An(\mathcal{E}_i)|\boldsymbol{x}^{(n)}, \boldsymbol{\theta})$ in place of $P(\mathcal{E}_i, An(\mathcal{E}_i)|\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}, \boldsymbol{\theta})$, but then describe the probability that they use as "the probability that expert k is responsible for generating the observed data" which is clearly the latter function.

were at least 20% worse than the state of the art systems, which brings into question the validity of the comparisons.

Saito and Nakano [204] also consider a constructive algorithm for HME architectures. In their method, the expert with the largest error on the training set is selected and split into 2 experts and a gate. The parameters of the new experts are initialised to the previous expert parameters, whilst the gate parameters are set to random values between -1 and $+1$. The authors evaluate this method on two simulated examples: a piecewise linear regression problem and classification of 8-bit parity strings.

Growing using the localised mixture of experts model [248] has also been investigated by Ramamurti and Ghosh [187] [see also 188]. In their approach the experts are ranked using the following weighted mean squared error score obtained on a *validation* data set:

$$E_j = \sum_n \zeta_j^{(n)} (y^{(n)} - \hat{y}_j^{(n)})^2 / \sum_n \zeta_j^{(n)}. \tag{6.23}$$

The experts are ranked in order of increasing error E_j and the expert with highest error selected for splitting. If this split does not decrease the error on the validation set it is rejected and the expert with next highest rank selected. The splitting algorithm works by copying the expert parameters into 2 new experts, whilst initialising the gate parameters using a weighted k-means algorithm in the locality of the original expert. The use of the localised mixture allows this neat initialisation method for the gate. Results of this method were given on simulated regression data sets and on two real world examples: classification of patient data into heart disease risk categories and prediction of energy demand for a building. Initial results for this method were impressive, and the authors also extended the method to an on-line algorithm which they use to model time varying regression functions.

6.4 Summary

In this chapter I have described a learning method based on a hierarchical mixtures-of-experts model which determines its structure in a recursive manner. This growing algorithm has been motivated from recursive partitioning ideas used frequently in decision and regression trees. Unlike most tree growing algorithms however, the method used here does not prune back branches after growing to exhaustion, but instead uses early stopping. I have demonstrated the behaviour of the growing algorithm on a simple task and on some of the DELVE data sets. Initial results suggest that the method makes sensible use of resources according to the complexity of the problem. Whether this behaviour translates into good performance will be seen in Chapter 8.

Chapter 7

Bayesian Methods for Mixtures of Experts

7.1 Introduction

Complexity control, in the form of preventing over-fitting, is a key issue in machine learning. In Chapters 5 and 6 I described methods for controlling the complexity of mixture of experts models based on early stopping via cross validation in either fixed (Chapter 5) or adaptive architecture models (Chapter 6). Early stopping can be an effective method for complexity control, as was shown empirically in Chapters 5 and 6, and in more detail in Chapter 8, via experiments on the DELVE data sets. However, early stopping has a number of critics in the literature. One of the cited disadvantages of early stopping is the fact that some training data must be left aside for validation. This is a somewhat obscure point, especially when cross validation and committees used, since all the training data is used in the committee. Also, although the validation data is not used to actually train the parameters of the models, it is used to control the complexity, and is thus effectively used for training. The second most commonly cited criticism of early stopping is that the resulting search path through complexity space is not a good one, in the sense that some parameters of the model are less well trained than others. Hopefully, the use of cross validation and committees averages over this latter effect, leading to well trained models. However, there remains some doubt as to whether the models resulting from early stopping, even when using committees and cross validation, are actually good models, and whether a better scheme could be used.

In this chapter I describe an alternative method for controlling complexity of mixtures-of-experts models. This approach aims to use all the available training data to infer the parameters of the models and also to control the model complexity. The approach used is Bayesian in flavour and owes much to the evidence framework of MacKay [133] and the variational free energy view of the EM algorithm of Neal and Hinton [154]. I use *ensemble learning*, a technique originally proposed by Hinton and van Camp [87] and subsequently extended by MacKay [136], to motivate an alternating minimisation procedure that combines the standard EM training algorithm with re-estimation of hyper-parameters of priors on gate and expert parameters.

The outline of this chapter is as follows. I separate the chapter into two overall parts, the *theory* of ensemble learning and Bayesian methods for mixtures-of-experts is described in Section 7.2 whereas the *implementation* of this theory in two learning methods, `me-el-1` and `hme-el-1`, is covered in

Section 7.3. Finally, related work in this area is discussed in Section 7.4.

7.2 Background Theory

This section describes the background theory of ensemble learning in mixtures-of-experts. It is organised in the following way. First, I discuss the possible priors which can be used in a mixtures-of-experts model and introduce the specific priors which I use in this chapter. I then introduce the concept of ensemble learning and show how it can be used for learning from data in mixtures-of-experts. Following this I describe how prediction can be performed in the presence of priors.

Priors for mixtures-of-experts models

The choice of priors for a model is an important one in Bayesian inference. Priors embody the assumptions about such aspects as the generative processes in the data, form of the model and shape or desired accuracy of the output function. The priors on a model are typically placed either on the structure, the output function or the parameters. In this chapter I consider priors on the parameters only.

The mixtures-of-experts model is implicitly a modular architecture in which the parameters of the experts and gate may be assumed to be independent. This assumption motivates the use of a separable prior distribution on the parameters of the experts and gate. In addition, I use separate priors on each parameter vector of the experts and gates which is similar to the framework described by MacKay [133] for multi-layer perceptrons. This yields an overall separable prior on the parameters of the model. In the simple case of a mixture of I experts, this prior is given by:

$$P(\mathbf{v}, \mathbf{w}|\mu, \alpha) = \prod_{i=1}^{I} P(\mathbf{v}_i|\mu_i) \prod_{l=1}^{k} P(\mathbf{w}_{il}|\alpha_{il}) \tag{7.1}$$

where i indexes the I experts in the model, and l indexes the k outputs of each expert. The prior for each expert parameter vector $\mathbf{w}_{il}$ is parameterised by a hyper-parameter α_{il}. Each gate parameter vector $\mathbf{v}_i$ has a hyper-parameter μ_i. Given this form for the overall priors on the model parameters, how do we choose priors for these parameters? In this chapter I use zero mean Gaussian priors on each distinct parameter vector $\mathbf{w}_{il}$ and $\mathbf{v}_i$. Gaussian priors correspond to the traditional method of ridge regression [90] in statistics or weight decay [86] in neural networks. For expert parameter vector $\mathbf{w}_{il}$ the prior is given by:

$$P(\mathbf{w}_{il}|\alpha_{il}) = \left(\frac{\alpha_{il}}{2\pi}\right)^{d/2} \exp\left(-\frac{\alpha_{il}}{2}\mathbf{w}_{il}^T\mathbf{w}_{il}\right), \tag{7.2}$$

and for gate parameter vector $\mathbf{v}_i$ the prior is given by:

$$P(\mathbf{v}_i|\mu_i) = \left(\frac{\mu_i}{2\pi}\right)^{d/2} \exp\left(-\frac{\mu_i}{2}\mathbf{v}_i^T\mathbf{v}_i\right). \tag{7.3}$$

The hyper-parameters μ_i and α_{il} represent the reciprocal of the variance of these Gaussian distributions. Small values for the hyper-parameters therefore imply wide priors for the parameters. Large values imply tight priors, constraining the parameters to be close to 0.

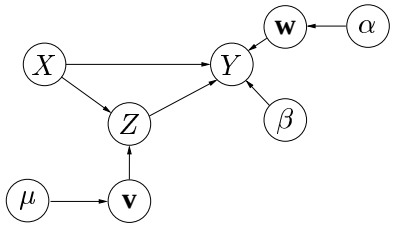

Figure 7.1: **Belief network for the mixtures-of-experts with priors in the case of regression**

In the case of linear regression I also place a prior on the variance for each expert. For the variance of an expert $\mathcal{E}_i$ I use the same Gamma hyper-prior on $\log \beta_i$ as in Section 4.3:

$$P(\log \beta_i|\rho, \upsilon) = \frac{1}{\Gamma(\rho)} \left(\frac{\beta_i}{\upsilon}\right)^{\rho} \exp(-\beta_i/\upsilon), \tag{7.4}$$

where υ, ρ are the *hyper*-hyper-parameters which specify the range in which the noise levels β_i are expected to lie as described in Section 4.3. Once again I set the hyper-parameters to the fixed values given in Section 4.3. Unlike the hyper-parameters of the gate and expert parameter vectors, I do not attempt to infer the variance hyper-parameters υ since these parameterise only a single variable β. Once again, as in Chapter 4 the Gamma prior is used mostly as a computational device, to avoid the possibility of singularities in the variance estimate when the number of observations at an expert is very small.

Potentially, Gamma priors could also be placed on the hyper-parameters μ_i and α_{il} since these are also scale parameters [13]. This was done by MacKay [136] in his presentation of ensemble learning for linear regression, and also suggested by Neal [153] for Markov chain Monte-Carlo methods for multi-layer perceptrons, although this is not considered here.

Combining the priors with the distributions for data and missing data gives the following posterior distribution for a mixtures-of-experts model for regression:

$$\begin{aligned} P(\mathbf{w}, \mathbf{v}, \alpha, \beta|X, Y) = & \prod_{i=1}^{I} P(\mathbf{v}_i|\mu_i) P(\mathbf{w}_i|\alpha_i) P(\beta_i|\rho, \upsilon) \\ & \prod_{n=1}^{N} P(z_i^{(n)} = 1|\boldsymbol{x}^{(n)}, \mathbf{v}_i) P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, z_i^{(n)} = 1, \mathbf{w}_i, \beta_i) \end{aligned} \tag{7.5}$$

This framework is represented in the belief network of Figure 7.1. Comparing this to Figure 2.2 we can see that the parameters **w**, and **v** have been introduced, as well as the hyper-parameters α, β and μ.

Ensemble learning for mixtures-of-experts

Given the specification of priors for the mixtures-of-experts model, many possible methods for inference of the parameters exist. If we simply fixed the hyper-parameters to particular values and inferred the parameters, we would get *maximum a-posteriori* (MAP) estimates of the parameters. Whilst this approach is possible in some circumstances and for certain types of models, it is generally difficult to choose the

values for the hyper-parameters in advance, especially when there are many such hyper-parameters as in this model. Alternatively we can adopt a *Bayesian* method for inferring the parameters *and* hyper-parameters of the model.

There are broadly two approaches to Bayesian inference in neural networks, that which uses approximations to integrals [29, 133] and another based on Markov chain Monte Carlo methods to find integrals numerically [153]. Bishop [14] gives a good review and tutorial on this area. In this chapter I use an approach to Bayesian inference which is motivated by the evidence framework of MacKay [133]. Other motivations include the ensemble learning method of Hinton and van Camp [87] and MacKay [136], the EM viewpoint of Neal and Hinton [154] and the mean field theory method of Saul and Jordan [209]. Whilst it is acknowledged that many Bayesian methods could be used for the mixtures-of-experts, I attempt in this chapter to generalise the maximum likelihood method described in Chapter 2 in a logical way. Hopefully this will allow the attractive properties of the EM based learning algorithm of Jordan and Jacobs [112] to be retained.

The starting point for the framework is the EM algorithm viewpoint of Neal and Hinton [154]. In this section I reproduce Neal and Hinton's argument in a slightly more general form by considering a prior distribution over parameters $P(\boldsymbol{\theta}|\boldsymbol{\phi})$, where $\boldsymbol{\phi}$ is a set of hyper-parameters for this prior. I assume the following decomposition of the joint distribution:

$$P(\boldsymbol{\theta}, \boldsymbol{\phi}, Z, D) = P(\boldsymbol{\theta}|\boldsymbol{\phi})P(\boldsymbol{\phi})P(Z, D|\boldsymbol{\theta}). \tag{7.6}$$

In the mixtures-of-experts model the parameters $\boldsymbol{\theta}$ encompass the expert parameters $\mathbf{w}$ and gate parameters $\mathbf{v}$. The hyper-parameters $\boldsymbol{\phi}$ encompass the expert and gate complexity hyper-parameters α and μ and the expert noise hyper-parameters β in the case of regression. The data D is the set of input and target data: $D = (X, Y)$, and the missing data Z is the set of node assignments $\{z_j^{(n)}\}_{n=1}^N$ for all nodes $\mathcal{E}_j$. The complete data $P(Z, D|\boldsymbol{\theta})$ in the mixtures-of-experts model is given by:

$$P(Z, D|\boldsymbol{\theta}) = \prod_n \prod_i [P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, z_i^{(n)} = 1, \mathbf{w}_i)P(z_i^{(n)} = 1|\boldsymbol{x}^{(n)}, \mathbf{v}_i)]^{z_i^{(n)}} \tag{7.7}$$

where the product is over all discrete choices of Z.

Following the framework of MacKay [136] I approximate the posterior distribution $P(\boldsymbol{\theta}, \boldsymbol{\phi}, Z|D)$ by a distribution $Q(\boldsymbol{\theta}, \boldsymbol{\phi}, Z)$. In order to measure the quality of the approximation I use the *variational free energy* F:

$$F(\boldsymbol{\theta}, \boldsymbol{\phi}, Z) \quad = \quad -\int Q(\boldsymbol{\theta}, \boldsymbol{\phi}, Z) \log \left[\frac{P(\boldsymbol{\theta}, \boldsymbol{\phi}, Z, D)}{Q(\boldsymbol{\theta}, \boldsymbol{\phi}, Z)} \right] d\boldsymbol{\theta}\, d\boldsymbol{\phi}\, dZ. \tag{7.8}$$

The free energy function F is a central concept in statistical physics. If we take the values of Z, $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ to represent physical states, the energy of each state is given by $P(\boldsymbol{\theta}, \boldsymbol{\phi}, Z|D)$. The free energy is also equivalent to the Kullback-Leibler distance [123], [14, pp.59] between $Q(\boldsymbol{\theta}, \boldsymbol{\phi}, Z)$ and $P(\boldsymbol{\theta}, \boldsymbol{\phi}, Z, D)$. Since the joint distribution may be written as $P(\boldsymbol{\theta}, \boldsymbol{\phi}, Z|D)P(D)$, the free energy may also be rewritten as:

$$F = KL\left[Q(\boldsymbol{\theta}, \boldsymbol{\phi}, Z) \,||\, P(\boldsymbol{\theta}, \boldsymbol{\phi}, Z|D)\right] - \log[P(D)] \tag{7.9}$$

i.e., the sum of the *Kullback-Leibler distance* or *asymmetric divergence* between Q and the posterior probability $P(\boldsymbol{\theta}, \boldsymbol{\phi}, Z|D)$ and the negative of the log of the evidence $P(D)$ of the model. This implies that the free energy F is lower bounded below by the negative log evidence:

$$F \geq -\log[P(D)] \tag{7.10}$$

so that minimising F is equivalent to maximising the evidence. We now make the following approximation of assuming that the distribution $Q(\boldsymbol{\theta}, \boldsymbol{\phi}, Z)$ is *separable*:

$$Q(\boldsymbol{\theta}, \boldsymbol{\phi}, Z) = Q(\boldsymbol{\theta})Q(\boldsymbol{\phi})Q(Z) \tag{7.11}$$

which is equivalent to the mean field approximation [168, pp.24], [209]. This is a fairly restrictive assumption that implies that each approximating distribution is independent of the other variables. Although this assumption allows the free energy to be evaluated in a tractable way it is unlikely that the true posterior distribution is separable in this way. The quality of the approximation is measured by the free energy, and the log evidence will be underestimated by the KL distance between Q and the true posterior $P(\boldsymbol{\theta}, \boldsymbol{\phi}, Z|D)$.

We now find the optimal $Q(\boldsymbol{\theta}, \boldsymbol{\phi}, Z)$ by alternately minimising F with respect to each of the variables $\boldsymbol{\theta}$, $\boldsymbol{\phi}$ and Z while the others are held fixed. This will lead to three *optimal* distributions $Q^*(Z)$, $Q^*(\boldsymbol{\theta})$ and $Q^*(\boldsymbol{\phi})$. This is possible since we have made the separability assumption about Q. Rewriting F in terms of the three variables and using Equation 7.6 gives:

$$\begin{aligned}
F_Z &= -\int Q(Z) \left\{ \int Q(\boldsymbol{\theta}) \log[P(Z, D|\boldsymbol{\theta})] \, d\boldsymbol{\theta} - \log Q(Z) \right\} dZ + C_1 \\
F_{\boldsymbol{\theta}} &= -\int Q(\boldsymbol{\theta}) \left\{ \int Q(Z) \log[P(Z, D|\boldsymbol{\theta})] \, dZ + \int Q(\boldsymbol{\phi}) \log[P(\boldsymbol{\theta}|\boldsymbol{\phi})] \, d\boldsymbol{\phi} - \log[Q(\boldsymbol{\theta})] \right\} d\boldsymbol{\theta} + C_2 \\
F_{\boldsymbol{\phi}} &= -\int Q(\boldsymbol{\phi}) \left\{ \int Q(\boldsymbol{\theta}) \log[P(\boldsymbol{\theta}|\boldsymbol{\phi})P(\boldsymbol{\phi})] \, d\boldsymbol{\theta} - \log[Q(\boldsymbol{\phi})] \right\} d\boldsymbol{\phi} + C_3
\end{aligned} \tag{7.12}$$

where C_1, C_2 and C_3 are constants. In order to compute *optimal* distributions the following principle [136], [14, pp.59] is used:

$$\begin{aligned}
&\textit{The divergence} -\int Q(\omega) \left\{ \log[P(\lambda)] - \log[Q(\omega)] \right\} d(\omega) \\
&\textit{is minimised by setting } Q(\omega) = P(\lambda).
\end{aligned} \tag{7.13}$$

Some simple algebra then gives optimal distributions for Z, $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ as:

$$\begin{aligned}
\log Q^*(Z) &= \int Q(\boldsymbol{\theta}) \log[P(Z, D|\boldsymbol{\theta})] \, d\boldsymbol{\theta} \; + C_1' \\
\log Q^*(\boldsymbol{\theta}) &= \int Q(Z) \log[P(Z, D|\boldsymbol{\theta})] \, dZ + \int Q(\boldsymbol{\phi}) \log[P(\boldsymbol{\theta}|\boldsymbol{\phi})] \, d\boldsymbol{\phi} + C_2' \\
\log Q^*(\boldsymbol{\phi}) &= \int Q(\boldsymbol{\theta}) \log[P(\boldsymbol{\theta}|\boldsymbol{\phi})P(\boldsymbol{\phi})] \, d\boldsymbol{\theta} + C_3'
\end{aligned} \tag{7.14}$$

where C_1', C_2' and C_3' are different constants. Two viewpoints are now possible - if Z is ignored, we get the ensemble learning framework of Hinton and van Camp [87] and MacKay [136] in which the parameters and hyper-parameters of a single model (as distinct from a *mixture* of models) are inferred in a scheme which converges to the maximum evidence framework [136]. Alternatively, if $\boldsymbol{\phi}$ is ignored (*i.e.,* a flat prior on $\boldsymbol{\theta}$ is used), and $Q(\boldsymbol{\theta})$ is given by a delta function about a fixed point $\bar{\boldsymbol{\theta}}$:

$$Q(\boldsymbol{\theta}) = \delta(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}}), \tag{7.15}$$

we get the free energy EM framework of Neal and Hinton [154] in which F is bounded by the negative log likelihood $-\log[P(D|\boldsymbol{\theta})]$. In this case, the optimal distributions are given by:

$$\log Q^*(Z) = \int Q(\boldsymbol{\theta}) \log P(Z, D|\boldsymbol{\theta})\, d\boldsymbol{\theta} + C_1'' \tag{7.16}$$

$$\log Q^*(\boldsymbol{\theta}) = \int Q(Z) \log P(Z, D|\boldsymbol{\theta})\, dZ + C_2'' \tag{7.17}$$

From this we have that $Q^*(Z)$ is proportional to $P(Z, D|\bar{\boldsymbol{\theta}})$. Since we also have the constraint that $\sum_Z Q^*(Z) = 1$ [154], the optimal distribution $Q^*(Z)$ is given by:

$$Q^*(Z) = P(Z|D, \bar{\boldsymbol{\theta}}) \tag{7.18}$$

i.e., the posterior distribution of missing data given data and parameters. Maximising $\log Q^*(\boldsymbol{\theta})$ over $Q^*(Z)$ is then clearly equivalent to the EM algorithm.

A number of other researchers have also considered the EM algorithm from this viewpoint. Csiszár and Tusnády [41] considered the EM algorithm from an information geometry viewpoint and also generalised the framework to a more general class of alternating minimisation procedures. Hathaway [82] described a derivation of the EM algorithm using the method of coordinate descent [253] which alternately minimises the free energy with respect to $P(Z)$ and $\boldsymbol{\theta}$. Coordinate ascent minimises a function by alternately finding the minimum in different coordinate directions. For example in three dimensions, the coordinate directions would be $[\pm 1\ 0\ 0]^T$, $[0 \pm 1\ 0]^T$ and $[0\ 0\ \pm 1]^T$.

The concept behind the variational free energy principle has also been applied to a number of systems by Saul and Jordan [209] who consider it in terms of mean field theory. In their framework a set of states $\{S\}$ is considered, where the states are for example hidden variables in the EM algorithm. The energy $E\{S\}$ and probability $P\{S\}$ of the states are related by the Boltzmann distribution:

$$P\{S\} = \frac{\exp(-\beta E\{S\})}{\mathcal{Z}} \tag{7.19}$$

where β is the inverse temperature and $\mathcal{Z}$ is the *partition function*. In order to perform inference about the states, averages must be made over $P\{S\}$. Given that this is generally intractable, a simpler distribution $Q\{S\}$ is postulated over the states which typically takes the form of the mean field approximation which is identical to the separable assumption for Q made above. The next stage is to minimise the Kullback-Leibler divergence between $Q\{S\}$ and $P\{S\}$. In the case where $\mathcal{Z}$ is a likelihood function, minimisation of the KL divergence is equivalent to maximising the likelihood since the negative log likelihood forms

a lower bound on the KL divergence. This is identical to the case of ensemble learning considered above where the log evidence formed the lower bound on the free energy F. The mean field approach has been widely applied to models in which the true likelihood is intractable, such as sigmoid belief networks [207], factorised hidden Markov models [72], and Boltzmann machines [208, 206]. Recently Jaakkola and Jordan [96] have also computed *upper* bounds for likelihoods in intractable networks in addition to lower bounds resulting from the KL divergence. The concept of using a bound on likelihoods and minimising the bound rather than the actual likelihood has also been used by Dayan et al. [43] in the Helmholtz machine.

Having given the general outline of the ensemble learning approach, I now turn my attention to specific instances of its use in Bayesian inference of mixtures-of-experts. In the following sections I consider the form of the optimal distributions for ME models with multinomial logit gates and linear regression experts or multinomial logit experts.

Linear regression experts

As described in Chapter 4, the complete data likelihood for a mixture of I experts is given by:

$$P(Y, X, Z|\theta) = \prod_{n=1}^{N} \prod_{i=1}^{I} P(z_i^{(n)} = 1|\boldsymbol{x}^{(n)}, \mathbf{v}_i) P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, z_i^{(n)} = 1, \mathbf{w}_i). \tag{7.20}$$

Combining this with the priors specified in Section 7.2, gives the joint distribution in the case of regression for a mixture of I experts as:

$$\begin{aligned} P(\mathbf{w}, \mathbf{v}, \alpha, \beta, Z, X, Y) &= P(\mathbf{v}|\mu) P(\mathbf{w}|\alpha) P(\beta|\rho, \upsilon) P(Y, X, Z|\theta) \\ &= \prod_{i=1}^{I} P(\mathbf{v}_i|\mu_i) P(\mathbf{w}_i|\alpha_i) P(\beta_i|\rho, \upsilon) \\ &\quad \prod_{n=1}^{N} \left\{ P(z_i^{(n)} = 1|\boldsymbol{x}^{(n)}, \mathbf{v}_i) P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, z_i^{(n)} = 1, \mathbf{w}_i, \beta_i) \right\}^{z_i^{(n)}}. \end{aligned} \tag{7.21}$$

We postulate an approximating ensemble distribution Q to the posterior distribution $P(\mathbf{w}, \mathbf{v}, \alpha, \mu, \beta, Z)$. Q is once again constrained to be separable:

$$Q(\mathbf{w}, \mathbf{v}, \alpha, \mu, \beta, Z) = Q(\mathbf{w}) Q(\mathbf{v}) Q(\alpha) Q(\mu) Q(\beta) Q(Z). \tag{7.22}$$

The optimal separable distribution Q is found by minimisation of the free energy F between the joint distribution of Equation 7.21 and the approximating distribution of Equation 7.22. This minimisation is performed over each separate ensemble component $Q(\cdot)$ with all other components fixed. For each variable the optimal distribution is found in the following manner, in which we take the example of the variable **w**:

1. Express F as a function of $\mathbf{w}$:

$$F = -\int Q(\mathbf{w}) \left[\int Q(\alpha) \log[P(\mathbf{w}|\alpha)] \, d\mu + \int Q(Z)Q(\beta) \log[P(Y, Z|X, \mathbf{w}, \beta)] \, d\beta \, dZ - \log Q(\mathbf{w}) \right] d\mathbf{w} + C_1. \tag{7.23}$$

2. Use the principle of Equation 7.13 to write the log of the optimal distribution for $\mathbf{w}$ as:

$$\log Q^*(\mathbf{w}) = \int Q(\alpha) \log[P(\mathbf{w}|\alpha)] \, d\mu + \int Q(Z)Q(\beta) \log[P(Y, Z|X, \mathbf{w}, \beta)] \, d\beta \, dZ + C_2 \tag{7.24}$$

This gives the optimal distribution for the expert parameters $\mathbf{w}$ given the current $Q(Z)$, $Q(\mathbf{v})$, $Q(\alpha)$ and $Q(\beta)$. Performing the same process on each of the variables in Equation 7.22 gives the optimal distributions for the gate parameters, $Q^*(\mathbf{v})$, for the complexity hyper-parameters, $Q^*(\alpha)$ and $Q^*(\mu)$, and for the missing data, $Q^*(Z)$. To minimise F, each of these optimal distributions is computed in turn, whilst holding the other variables fixed. The optimal distributions are given by the following equations. The dependencies between these expressions are summarised in Table 7.1.

Term	$Q^*(\mathbf{w})$	$Q^*(\mathbf{v})$	$Q^*(\alpha)$	$Q^*(Z)$	$Q^*(\beta)$	$Q^*(\mu)$
$\int Q(Z)\log[P(Y,X,Z\|\mathbf{w},\beta)]\,dZ$	$\diamond$				$\diamond$	
$\int Q(Z)\log[P(Z,X\|\mathbf{v})]\,dZ$		$\diamond$				
$\int Q(\mathbf{w})\log[P(\mathbf{w}\|\alpha)]\,d\mathbf{w}$			$\diamond$			
$\int Q(\mathbf{v})\log[P(Z\|X,\mathbf{v})]\,d\mathbf{v}$				$\diamond$		
$\int Q(\mathbf{w})\log[P(Y\|Z,X,\mathbf{w},\beta)]\,d\mathbf{w}$				$\diamond$	$\diamond$	
$\int Q(\beta)\log[P(Y,X,Z,\|\mathbf{w},\beta)]\,d\beta$	$\diamond$			$\diamond$		
$\int Q(\alpha)\log[P(\mathbf{w}\|\alpha)]\,d\alpha$	$\diamond$					
$\int Q(\mu)\log[P(\mathbf{v}\|\mu)]\,d\mu$		$\diamond$				
$\int Q(\mathbf{v})\log[P(\mathbf{v}\|\mu)]\,d\mathbf{v}$						$\diamond$

Table 7.1: **Dependency of terms in the optimal distributions.** Each optimal distribution is shown across the top of the table. A dependency of a distribution on one of the terms at the left of the table is shown by way of a diamond $\diamond$ in the table. See also Equations 7.25 to 7.30.

1. Expert and Gate parameters:

$$\log Q^*(\mathbf{w}) = \int Q(Z)Q(\beta)Q(\alpha)\log[P(Y,Z,X|\mathbf{w},\beta)P(\mathbf{w}|\alpha)]\,dZ\,d\beta\,d\alpha + C_1 \quad (7.25)$$

$$\log Q^*(\mathbf{v}) = \int Q(Z)Q(\mu)\log[P(Z,X|\mathbf{v})P(\mathbf{v}|\mu)]\,dZ\,d\mu + C_2 \quad (7.26)$$

2. Expert and gate complexity hyper-parameters:

$$\log Q^*(\alpha) = \int Q(\mathbf{w})\log[P(\mathbf{w}|\alpha)P(\alpha)]\,d\mathbf{w} + C_3 \quad (7.27)$$

$$\log Q^*(\mu) = \int Q(\mathbf{v})\log[P(\mathbf{v}|\mu)P(\mu)]\,d\mathbf{v} + C_4 \quad (7.28)$$

3. Expert noise hyper-parameter:

$$\log Q^*(\beta) = \int Q(\mathbf{w})Q(Z)Z\log[P(Y,X,Z|\mathbf{w},\beta)P(\beta|\rho,\upsilon)]\,dZ\,d\mathbf{w} + C_5 \quad (7.29)$$

4. Missing data Z:

$$\begin{aligned}\log Q^*(Z) = &\int Q(\mathbf{w})Q(\beta)\log[P(Y|X,Z,\mathbf{w},\beta)]\,d\mathbf{w}\,d\beta \\ &+ \int Q(\mathbf{v})\log[P(Z|X,\mathbf{v})]\,d\mathbf{v} + C_6\end{aligned} \quad (7.30)$$

Given the set of optimal distributions in Equations 7.25 to 7.30, we have seemingly solved the problem. Using these distributions for the variables we may minimise F and thus better approximate the posterior distribution $P(\mathbf{w},\mathbf{v},\alpha,\mu,\beta,Z|D)$. The actual form of each of the approximating distributions

depends on the choice of expert and gate. In the case of generalised linear models used here the integrations are straightforward. If more complex experts and gates were used, *e.g.*, multi-layer perceptrons, two courses of action could be taken — either to use numerical integration via MCMC techniques or to make approximations to the integrands and perform the integrations analytically. In this work most of the integrations can be performed analytically without approximations. In some cases a Gaussian approximation is made to the integrand and the resulting approximation integrated analytically.

In the case of the missing data the optimal distribution $Q^*(Z)$ gives a new set of node joint and conditional posterior probabilities: $\bar{\zeta}_j^{(n)}$ and $\bar{\varphi}_j^{(n)}$. In the case of the distributions of parameters and hyper-parameters the parameters of each of the optimal distributions are found. The means of each of these optimal distributions give fixed point estimates for the parameters and hyper-parameters which are denoted as $\bar{\mathbf{w}}_j$, $\bar{\mathbf{v}}_j$, $\bar{\alpha}_j$, $\bar{\mu}_j$ and $\bar{\beta}_j$. The expressions for each of these estimates are given in the following sections.

Optimal distributions for expert networks

For the *parameters* of the experts the optimal distribution is given by:

$$\log Q^*(\mathbf{w}) = \sum_{j=1}^{J} \log Q^*(\mathbf{w}_j) = \sum_{j=1}^{J} \left[-\frac{\bar{\alpha}_j}{2} \mathbf{w}_j^T \mathbf{w}_j - \sum_n \bar{\zeta}_j^{(n)} \frac{\bar{\beta}_j}{2} \left(y^{(n)} - \hat{y}_j^{(n)} \right)^2 \right] + C_1. \tag{7.31}$$

Comparing this expression to the auxiliary function for the maximum likelihood EM algorithm (Equation 4.13) we notice that Equation 7.31 is given by the sum of the auxiliary function and the log of the prior $P(\mathbf{w}_j|\alpha_j)$. Since the auxiliary function is the expected value of the complete data log-likelihood, we can interpret Equation 7.31 as the sum of log-prior and log-likelihood terms.

We may note that Equation 7.31 is a set of J Gaussian distributions with means $\{\bar{\mathbf{w}}_j\}$, and covariance matrices $\{\mathrm{A}_{\mathbf{w}_j}\}$. For node $\mathcal{E}_j$ the covariance matrix is given by the negative of the inverse of its Hessian matrix which is a $(d+1) \times (d+1)$ matrix [140, pp. 71], [173]:

$$\mathrm{A}_{\mathbf{w}_j} = \left(-\frac{\partial^2 \log Q^*(\mathbf{w}_j)}{\partial \mathbf{w}_j \partial \mathbf{w}_j^T} \right)^{-1}, \tag{7.32}$$

where

$$\frac{\partial^2 \log Q^*(\mathbf{w}_j)}{\partial \mathbf{w}_j \partial \mathbf{w}_j^T} = -\sum_n \bar{\zeta}_j^{(n)} \bar{\beta}_j (\boldsymbol{x}^{(n)})(\boldsymbol{x}^{(n)})^T - \bar{\alpha}_j . I, \tag{7.33}$$

where I is an identity matrix.

To obtain new values of the parameters $\mathbf{w}_j$ we optimise Equation 7.31 with respect to the parameters. The derivatives of $\log Q^*(\mathbf{w}_j)$ take a similar form to the derivatives of the auxiliary function (Equations 4.14 and 4.15) but with the additional derivative of the log-prior term. The new values $\bar{\mathbf{w}}_j$ are found by solving:

$$\sum_n \bar{\zeta}_j^{(n)} \bar{\beta}_j (y^{(n)} - \hat{\boldsymbol{y}}_j^{(n)}) \boldsymbol{x}^{(n)} - \bar{\alpha}_j \mathbf{w}_j = 0 \tag{7.34}$$

For the expert complexity hyper-parameters, the integration of Equation 7.27 takes the following form:

$$\log Q^*(\alpha_j) = \frac{d}{2}\log\alpha_j - \frac{\alpha_j}{2}\int Q(\mathbf{w}_j)\mathbf{w}_j^T\mathbf{w}_j d\mathbf{w}_j. \tag{7.35}$$

The integral of $\mathbf{w}_j^T\mathbf{w}_j$ may be rewritten as:

$$\int Q(\mathbf{w}_j)\mathbf{w}_j^T\mathbf{w}_j\, d\mathbf{w}_j = \int Q(\mathbf{w}_j)(\mathbf{w}_j-\bar{\mathbf{w}}_j)^T(\mathbf{w}_j-\bar{\mathbf{w}}_j)\, d\mathbf{w}_j + \bar{\mathbf{w}}_j^T\bar{\mathbf{w}}_j \tag{7.36}$$

where $\bar{\mathbf{w}}_j = \int \mathbf{w}_j Q(\mathbf{w}_j)\, d\mathbf{w}_j$. If we now observe that the inner product in the integral may be expressed as the trace of an outer product, we may write:

$$\int Q(\mathbf{w}_j)\mathbf{w}_j^T\mathbf{w}_j\, d\mathbf{w}_j = \text{Trace}\left(\int Q(\mathbf{w}_j)(\mathbf{w}_j-\bar{\mathbf{w}}_j)(\mathbf{w}_j-\bar{\mathbf{w}}_j)^T\, d\mathbf{w}_j\right) + \bar{\mathbf{w}}_j^T\bar{\mathbf{w}}_j. \tag{7.37}$$

The term inside the trace operator is the simply the covariance of $\mathbf{w}_j$ over $Q(\mathbf{w}_j)$, which is given by Equation 7.32. This allows the log of the optimal distribution $Q^*(\alpha_j)$ to be written as:

$$\log Q^*(\alpha_j) = \frac{d}{2}\log\alpha_j - \frac{\alpha_j}{2}\left(\bar{\mathbf{w}}_j^T\bar{\mathbf{w}}_j + \text{Trace}(\mathrm{A}_{\mathbf{w}_j})\right). \tag{7.38}$$

This is the log of a Gamma distribution with mean $\bar{\alpha}_j$ given by:

$$\frac{1}{\bar{\alpha}_j} = \frac{\mathbf{w}_j^T\mathbf{w}_j + \text{Trace}(\mathrm{A}_{\mathbf{w}_j})}{d}. \tag{7.39}$$

This gives a new estimate of the hyper-parameters α_j. Note that this is in the same form as the update for hyper-parameters in a generalised linear regression model by the ensemble learning method of MacKay [136], except that MacKay also uses a Gamma distribution for α which introduces extra terms to the update.

For the noise hyper-parameter of expert $\mathcal{E}_j$, the required integral is given by Equation 7.29. Expanding the integral and performing the integration with respect to $Q(Z)$ gives:

$$\log Q^*(\beta_i) = -\frac{\beta_i}{2}\sum_n \zeta_i^{(n)}\int Q(\mathbf{w}_i)(y^{(n)}-\mathbf{w}_i^T\boldsymbol{x}^{(n)})^2\, d\mathbf{w}_i + \tfrac{1}{2}\textstyle\sum_n \zeta_i^{(n)}\log\frac{\beta_i}{2\pi} + \rho\log\left(\frac{\beta_i}{v}\right) - \frac{\beta_i}{v}. \tag{7.40}$$

The last two terms of this expression come from the Gamma prior on $\log\beta_i$. To integrate the first term of the expression with respect to $\mathbf{w}_i$, we make the following expansion:

$$A = \int Q(\mathbf{w}_i)(y^{(n)}-\mathbf{w}_i^T\boldsymbol{x}^{(n)})^2\, d\mathbf{w}_i = \int Q(\mathbf{w}_i)\left[(y^{(n)}-\bar{\mathbf{w}}_i^T\boldsymbol{x}^{(n)}) - (\mathbf{w}_i-\bar{\mathbf{w}}_i)^T\boldsymbol{x}^{(n)}\right]^2 d\mathbf{w}_i, \tag{7.41}$$

where

$$\bar{\mathbf{w}}_i = \int \mathbf{w}_i Q(\mathbf{w}_i)\, d\mathbf{w}_i. \tag{7.42}$$

This may be rearranged to give:

$$A = (y^{(n)} - \bar{\mathbf{w}}_i^T \boldsymbol{x}^{(n)})^2 + \int Q(\mathbf{w}_i) \boldsymbol{x}^{(n)^T} (\mathbf{w}_i - \bar{\mathbf{w}}_i)(\mathbf{w}_i - \bar{\mathbf{w}}_i)^T \boldsymbol{x}^{(n)} \, d\mathbf{w}_i, \tag{7.43}$$

in which the term $\int Q(\mathbf{w}_i)(\mathbf{w}_i - \bar{\mathbf{w}}_i)(\mathbf{w} - \bar{\mathbf{w}}_i)^T \, d\mathbf{w}_i$ may be recognised as the covariance matrix $\mathrm{A}_{\mathbf{w_i}}$ (7.32). This allows the log of the optimal distribution to be written as:

$$\begin{aligned} \log Q^*(\beta_i) = &- \frac{\beta_i}{2} \left(\sum_n \zeta_i^{(n)} (y^{(n)} - \bar{\mathbf{w}}_i^T \boldsymbol{x}^{(n)})^2 - \sum_n \zeta_i^{(n)} \boldsymbol{x}^{(n)^T} \mathrm{A}_{\mathbf{w_i}} \boldsymbol{x}^{(n)} \right) \\ &+ \tfrac{1}{2} \textstyle\sum_n \zeta_i^{(n)} \log \frac{\beta_i}{2\pi} + \rho \log \left(\frac{\beta_i}{\upsilon} \right) - \frac{\beta_i}{\upsilon}. \end{aligned} \tag{7.44}$$

This is the log of a Gamma distribution whose mean $\int \beta_i Q^*(\beta_i) \, d\beta_i$ is given by:

$$\frac{1}{\bar{\beta}_i} = \frac{\sum_n \bar{\zeta}_j^{(n)} \left[(y^{(n)} - \bar{\mathbf{w}}_i^T \boldsymbol{x}^{(n)})^2 + \boldsymbol{x}^{(n)^T} \mathrm{A}_{\mathbf{w_j}} \boldsymbol{x}^{(n)} \right] + 2/\upsilon}{\sum_n \bar{\zeta}_j^{(n)} + 2\rho}. \tag{7.45}$$

$1/\beta_i$ is the estimate of the noise variance for expert $\mathcal{E}_j$. In this framework it is modified to include a dependence on the hyper-hyper-parameters υ and ρ, which is expected due the use of the Gamma prior on $\log \beta$. In addition, the noise variance is modified by the term $\sum_n \bar{\zeta}_j^{(n)} (\boldsymbol{x}^{(n)^T} \mathrm{A}_{\mathbf{w_j}} \boldsymbol{x}^{(n)})$ which represents the sum of the uncertainties of the expert's predictions.

Optimal distributions for gate networks

The optimal distribution for the gate parameters $Q^*(\mathbf{v})$ is obtained in a similar fashion to the other optimal distributions, and its log is given by:

$$\log Q^*(\mathbf{v}) = \sum_j \log Q^*(\mathbf{v}_j) = - \sum_j \frac{\bar{\mu}_j}{2} \mathbf{v}_j^T \mathbf{v}_j + \sum_n \bar{\varphi}_j^{(n)} \log g_j^{(n)} + C. \tag{7.46}$$

Each optimal distribution for the gates $Q^*(\mathbf{v}_j)$ is approximated by a Gaussian distribution fitted at its maximum $\mathbf{v}_j = \bar{\mathbf{v}}_j$ with covariance matrix $\mathrm{A}_{\mathbf{v_j}}$:

$$\log Q(\mathbf{v}_j) \approx \log Q(\bar{\mathbf{v}}_j) - \tfrac{1}{2} \Delta \mathbf{v}_j^T A_{\mathbf{w_j}}^{-1} \Delta \mathbf{v}_j, \tag{7.47}$$

where $\Delta \mathbf{v}_j = \mathbf{v}_j - \bar{\mathbf{v}}_j$, and the covariance matrix of $\mathbf{v}_j$ is given by the negative of the inverse of the second derivative of $\log Q^*(\mathbf{v}_j)$ [140, pp. 119]:

$$\frac{\partial^2 \log Q^*(\mathbf{v})}{\partial \mathbf{v}_j \partial \mathbf{v}_j^T} = - \sum_{n=1}^{N} g_j^{(n)} (1 - g_j^{(n)}) (\boldsymbol{x}^{(n)}) (\boldsymbol{x}^{(n)})^T - \bar{\mu}_j . I. \tag{7.48}$$

The parameters $\bar{\mathbf{v}}_j$ are found by solving the following equations:

$$\sum_{n=1}^{N} (\varphi_j^{(n)} - g_j^{(n)}) \boldsymbol{x}^{(n)} - \bar{\mu}_j \mathbf{v}_j = 0. \tag{7.49}$$

Using this approximation, the re-estimation equations for the hyper-parameters μ are given by:

$$\frac{1}{\bar{\mu}_j} = \frac{\mathbf{v}_j^T \mathbf{v}_j + \text{Trace}(\text{A}_{\mathbf{v}_\text{j}})}{d}. \tag{7.50}$$

Optimal distribution of Z

Finally, for the optimal distribution of missing data, the following integral is required:

$$\begin{aligned}\log Q^*(Z) = \sum_{n=1}^{N}\sum_{j=1}^{I} \int Q(\mathbf{w}_j)Q(\beta_j)\log[P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, z_j^{(n)}=1, \beta_j, \mathbf{w}_j)]\, d\mathbf{w}_j\, d\beta_j \\ + \int Q(\mathbf{v}_j)\log[P(z_j^{(n)}=1|\boldsymbol{x}^{(n)}, \mathbf{v}_j)]\, d\mathbf{v}_j + C,\end{aligned} \tag{7.51}$$

where the sum is over I experts. The integrals over $\mathbf{w}_j$ and β_j are straightforward using the techniques of the previous section:

$$\begin{aligned}\log Q^*(Z) = \sum_{n=1}^{N}\sum_{j=1}^{I} \left(\log[P(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}, z_j^{(n)}=1, \bar{\beta}_j, \bar{\mathbf{w}}_j)] - \frac{\beta_j}{2}\boldsymbol{x}^{(n)^T} A_{\mathbf{w}_\text{j}} \boldsymbol{x}^{(n)}\right) \\ + \int Q(\mathbf{v}_j)\log[P(z_j^{(n)}=1|\boldsymbol{x}^{(n)}, \mathbf{v}_j)]\, d\mathbf{v}_j + C.\end{aligned} \tag{7.52}$$

The integral over $\mathbf{v}_j$ is less easy, however. If we write $a_j = \mathbf{v}_j^T \boldsymbol{x}^{(n)}$, we may write the softmax function as:

$$\begin{aligned}P(z_j^{(n)}=1|\boldsymbol{x}^{(n)}, \mathbf{v}_j) &= \frac{\exp(a_j)}{\sum_{i=1}^{I}\exp(a_i)} \\ &= \frac{1}{1+\sum_{i\neq j}\exp(-(a_j-a_i))},\end{aligned} \tag{7.53}$$

so that the integral is given by:

$$\int Q(\mathbf{v}_j)\log P(z_j^{(n)}=1|\boldsymbol{x}^{(n)}, \mathbf{v}_j) d\mathbf{v}_j = -\int Q(\mathbf{v}_j)\log(1+\sum_{i\neq j}\exp(-(a_j-a_i))) \tag{7.54}$$

Since we have made a Gaussian approximation to $Q(\mathbf{v}_j)$, a local linear approximation to a_j means $Q(a_j)$ will also be a Gaussian [14, pp.405]. This integral does not have an analytic solution, however. Potentially, an approximation such as that that used by MacKay [132] for evaluating the integral of a Gaussian times a sigmoid could be used, although in this case we have the integral of a Gaussian times the *log* of a sigmoid. Alternatively, a variational method [95] could be used to bound the integral with a simpler expression. Another possibility is the use of various approximation suggested by Gales [65].

In this work I simply approximate the integral by its value at $\bar{\mathbf{v}}_j$:

$$\int Q(\mathbf{v}_j)\log[P(z_j^{(n)}=1|\boldsymbol{x}^{(n)}, \mathbf{v}_j)]\, d\mathbf{v}_j = \log\frac{\exp(\bar{\mathbf{v}}_j^T\boldsymbol{x}^{(n)})}{\sum_i \exp(\bar{\mathbf{v}}_i^T\boldsymbol{x}^{(n)})}, \tag{7.55}$$

where $\bar{\mathbf{v}}_j$ is the value of the gate parameter vector given by the mean $\int \mathbf{v}_j Q^*(\mathbf{v}_j) d\mathbf{v}_j$. This is an expedient approximation and we recognise that it could be improved.

After normalising, the optimal distribution $Q^*(Z)$ is then given by:

$$\begin{aligned} Q^*(Z) &= \prod_n \prod_j \left\{ \exp(s_j^{(n)}) \Big/ \sum_{i=1}^{I} \exp(s_i^{(n)}) \right\} \\ \text{where} \quad s_j^{(n)} &= \log[P(z_j^{(n)} = 1 | \bar{\mathbf{v}}_j, \boldsymbol{x}^{(n)})] + \log[P(\boldsymbol{y}^{(n)} | \boldsymbol{x}^{(n)}, \bar{\mathbf{w}}_j, z_j^{(n)} = 1, \bar{\beta}_j)] \\ &\quad - \frac{\bar{\beta}_j}{2} \boldsymbol{x}^{(n)^T} A_{\mathbf{w}_j} \boldsymbol{x}^{(n)} \end{aligned} \tag{7.56}$$

Comparing this to the standard EM algorithm, we may observe that the standard E-step computes a distribution of Z given fixed values of parameters and the data. In this case, by finding the optimal $Q(Z)$ the alternative expression of Equation 7.56 is obtained, with dependencies on the uncertainty of the experts' predictions, $\frac{\bar{\beta}_j}{2} \boldsymbol{x}^{(n)^T} A_{\mathbf{w}_j} \boldsymbol{x}^{(n)}$.

Making predictions

In order to make predictions via Bayesian inference of a target $y^{(N+1)}$ given an input vector $\boldsymbol{x}^{(N+1)}$, we use the *predictive* distribution which is given by:

$$P(y^{(n)} | \boldsymbol{x}^{(N+1)}, D) = \int P(y^{(N+1)} | \boldsymbol{x}^{(N+1)}, \theta) P(\theta | D) \, d\theta \tag{7.57}$$

where $P(\theta|D)$ is the posterior distribution of the parameters θ. We use the optimal distributions Q^* to approximate the posterior distribution. In the case of regression over one dimensional targets, the overall mean $\hat{y}^{(N+1)}$ of the predictive distribution is given by:

$$\hat{y}^{(N+1)} \quad = \quad \sum_j P(\mathcal{E}_j | \boldsymbol{x}^{(N+1)}, \bar{\mathbf{v}}_j) \hat{y}_j{}^{(N+1)} \tag{7.58}$$

where $\hat{y}_j{}^{(N+1)}$ is the prediction of $y^{(N+1)}$ by expert $\mathcal{E}_j$.

Multinomial logit experts

In the case of multinomial logit experts the joint probability density is given by:

$$P(\mathbf{w}, \mathbf{v}, \alpha, \mu, Z, X, Y) = \prod_{i=1}^{I} P(\mathbf{v}_i | \mu_i) P(\mathbf{w}_i | \alpha_i) \prod_{n=1}^{N} \left\{ P(z_i^{(n)} = 1 | \boldsymbol{x}^{(n)}, \mathbf{v}_i) \prod_{c=1}^{k} (\hat{\boldsymbol{y}}_{ic}^{(n)})^{\boldsymbol{y}_c^{(n)}} \right\}, \tag{7.59}$$

where $\hat{\boldsymbol{y}}_{ic}^{(n)} = P(\boldsymbol{y}_c^{(n)} | \boldsymbol{x}^{(n)}, z_i^{(n)} = 1, \mathbf{w}_i)$.

Once again we specify an approximating distribution which we assume to be separable:

$$Q(\mathbf{w}, \mathbf{v}, \alpha, \mu, Z) = Q(\mathbf{w}) Q(\mathbf{v}) Q(\alpha) Q(\mu) Q(Z). \tag{7.60}$$

In common with the gate parameters for linear regression, we make a Gaussian approximation to $Q(\mathbf{w})$ and $Q(\mathbf{v})$:

$$\log Q(\mathbf{w}_j) \approx \log Q(\bar{\mathbf{w}}_j) - \tfrac{1}{2}\Delta\mathbf{w}_j^T A_{\mathbf{w_j}}^{-1}\Delta\mathbf{w}_j, \tag{7.61}$$

where $A_{\mathbf{w_j}}$ is the covariance matrix of $\mathbf{w}_j$, and $\Delta\mathbf{w}_j = \mathbf{w}_j - \bar{\mathbf{w}}_j$. Using this approximation, the re-estimation equations for the hyper-parameters α are given by:

$$\frac{1}{\bar{\alpha}_j} = \frac{\mathbf{w}_j^T\mathbf{w}_j + \text{Trace}(\text{A}_{\mathbf{w_j}})}{d}. \tag{7.62}$$

For the gate hyper-parameters the re-estimation equations are the same as for the linear regression experts (Equation 7.50). This leaves the optimal distributions of expert parameters and missing data:

$$\begin{aligned}
\log Q^*(\mathbf{w}) &= \int Q(Z)Q(\alpha)Z\log[P(Y|X,Z,\mathbf{w})P(\mathbf{w}|\alpha)]\;dZ\;d\alpha + C_1; \\
\log Q^*(Z) &= \int Q(\mathbf{w})\log[P(Y|X,Z,\mathbf{w})]\;d\mathbf{w} + \int Q(\mathbf{v})\log[P(Z|X,\mathbf{v})]\;d\mathbf{v} + C_2.
\end{aligned} \tag{7.63}$$

In the case of the expert parameters $\mathbf{w}_j$, the integral is straightforward, and the log of the optimal distribution is given by: given by:

$$\log Q^*(\mathbf{w}_j) \quad = \quad \sum_{n=1}^{N}\zeta_j^{(n)}\sum_{c=1}^{k} y_c\log\hat{y}_c - \sum_{c=1}^{k}\frac{\alpha_{jc}}{2}\mathbf{w}_{jc}^T\mathbf{w}_{jc}. \tag{7.64}$$

The derivatives of this expression are similar to the derivatives of the gate parameters. For the missing data, the integrals over the expert parameters and gate parameters are approximated by the values of the integrands at $\bar{\mathbf{w}}$ and $\bar{\mathbf{v}}$ using the same assumptions as for the missing data in the case of linear regression experts. This gives the distribution of missing data in the same form as in Chapter 4 with the difference that $\bar{\zeta}_j^{(n)}$ and $\bar{\varphi}_j^{(n)}$ are computed using the mean values $\bar{\mathbf{w}}_j$ and $\bar{\mathbf{v}}_j$ for the parameters.

7.3 Method

In the previous section I have outlined the theory of ensemble learning as applied to mixtures-of-experts. This framework extends naturally from the EM learning algorithm covered in Chapter 2. In this section I describe how I implement this theory in two learning methods. The first of these methods uses a fixed width mixtures-of-experts, whilst the second uses a fixed depth binary hierarchy of experts. I denote this methods `me-el-1` and `hme-el-1` respectively. The code `-el-` denotes the use of the ensemble learning framework. These methods are similar to the early stopping methods with fixed structure, `me-ese-1` and `hme-ese-1` described in Chapter 5, but use the ensemble learning framework to infer their parameters rather than maximum likelihood plus early stopping.

1. The hyper-parameters are initialised to small values $\alpha_{jk} = 0.00001$, for each output k of expert $\mathcal{E}_j$ and $\mu_j = 0.0001$ for each output of the gate. The expert and gate parameters are initialised using the randomised segmentation algorithm of Section 4.5.
2. Compute the log posterior probability $L^{(t)} = \log[P(\mathbf{w}^{(t)}, \mathbf{v}^{(t)}, \beta^{(t)}, \mu^{(t)}, \alpha^{(t)}|D)]$.
3. Keeping the values of the hyper-parameters fixed, the EM algorithm is used to train the expert and gate parameters to convergence. The E step of the EM algorithm computes the optimal distribution $Q^*(Z)$ rather than the standard joint posteriors. In the case of regression, β_j for each expert $\mathcal{E}_j$ is updated during each M step.
4. The hyper-parameters are re-estimated using Equations 7.39, 7.45 and 7.50.
5. Compute log posterior probability $L^{(t+1)} = \log[P(\mathbf{w}^{(t+1)}, \mathbf{v}^{(t+1)}, \beta^{(t+1)}, \mu^{(t+1)}, \alpha^{(t+1)}|D)]$.
6. If $\delta L^{(t)} = L^{(t+1)} - L^{(t)} > 10^{-4}$ goto 3, else terminate.

Figure 7.2: **Exact search algorithm for ensemble learning in mixtures-of-experts**

A search method for ensemble learning in mixtures-of-experts models

This section is organised as follows. First, I outline a method for searching through the various possible parameter and hyper-parameter configurations defined by the ensemble learning optimal distributions. I show that although a training algorithm can be derived which implements the ensemble learning framework exactly, this is typically computationally demanding, and therefore it is preferable to use an approximate search method which I describe. Second, I describe the use of committees of models which, in common with Chapters 5 and 6, I used in the methods `me-el-1` and `hme-el-1`.

The ensemble learning framework outlined in Section 7.2 specifies that the free energy F should be minimised with respect to each variable in turn. Given the set of optimal distributions derived in Section 7.2, this minimisation becomes the alternate maximisation of the optimal distributions with respect to each variable. The scheduling of these maximisations can potentially have an effect on the speed of the search, *i.e.,* the path through parameters, hyper-parameters and missing data to a converged model, and the final configuration of this model. For example, if the hyper-parameters $\{\mu\}$ are updated too early in the search the model will typically under-fit the data, with the complexity priors $P(\mathbf{v}|\mu)$ dominating. This point was also discussed in the context of the evidence framework for MLPs by MacKay [133] and Thodberg [221]. The solution to this particular problem in MLPs is typically to train only the parameters of the model using fixed initial small values for the complexity hyper-parameters by maximising the posterior probability of the parameters. After this the hyper-parameter and parameter updates are alternated in turn until convergence of the posterior probability. This scheme is called the *exact* method for search in this chapter, and is described in Figure 7.2.

The exact search algorithm of Figure 7.2 proved successful at producing well trained models on small artificial data sets, and also on the Boston data. For larger problems, however, the algorithm takes a large amount of time. In order to speed up the algorithm, two approaches could be taken: speed up

1. Initialise the hyper-parameters and parameters as in Figure 7.2.
2. Compute $L^{(t)}$.
3. Train the parameters for 100 iterations.
4. Update the hyper-parameters as in Figure 7.2.
5. Compute $\delta L^{(t)}$.
6. If $\delta L^{(t)} > 10^{-4}$ and elapsed time $< 1.8765 * N/4$ seconds, goto 3, else terminate.

Figure 7.3: **An approximate search algorithm for ensemble learning in mixtures-of-experts. This is the basis of the methods `me-el-1` and `hme-el-1`.**

the EM algorithm itself using methods such as those suggested by Jordan and Xu [113]; or speed up the search method by *increasing* the values of the optimal distributions rather than maximising them in a similar fashion to generalised EM algorithms. Although speeding up the EM algorithm would be preferred, current research into speed ups has been inconclusive. I therefore choose the latter option and seek to only *increase* the values of $\log Q^*(\mathbf{w})$ and $\log Q^*(\mathbf{v})$ rather than to maximise them. This scheme is similar to, and inspired by, the one outline by Thodberg [221] for implementation of the evidence framework [133] in MLPs. It is anticipated that the search will still converge to a well trained model, but with fewer computational demands. The modified search algorithm is shown in Figure 7.3.

In order to investigate the effect of the restricted search algorithm, I compared their performance on the artificial data set described in Chapter 3 and on the Boston data. The evolution of the log posterior probability for the two algorithms on the Boston data is compared in Figure 7.4. One interesting point of these curves is that the log posterior probability typically drops after a hyper-parameter update. This is a function of the initial over-fitting used to start the search. After each hyper-parameter update the error on the training set increases until it has "taken up the slack" in the model.

Committees

In common with the early stopping method of `me-ese-1` and `hme-ese-1` of Chapter 4, I used committees of models in the methods `me-el-1` and `hme-el-1`. Each committee was trained on all of the training data, but initialised with different random segmentations, by choosing different seeds for the random number generators. The outputs of the committee members were averaged for out of sample prediction as before.

As in the methods `me-ese-1` and `hme-ese-1`, I allowed a minimum of 3 committee members, and a maximum of 50. The number of committee members was also a function of time, with the algorithm terminating after $1.8765 \times N$ seconds had elapsed. In practice, since the methods `me-el-1` and `hme-el-1` took longer to train than the early stopping methods, only 3 committee members were typically trained. The extra computation is caused by the cycling through parameter and hyper-parameter training, and

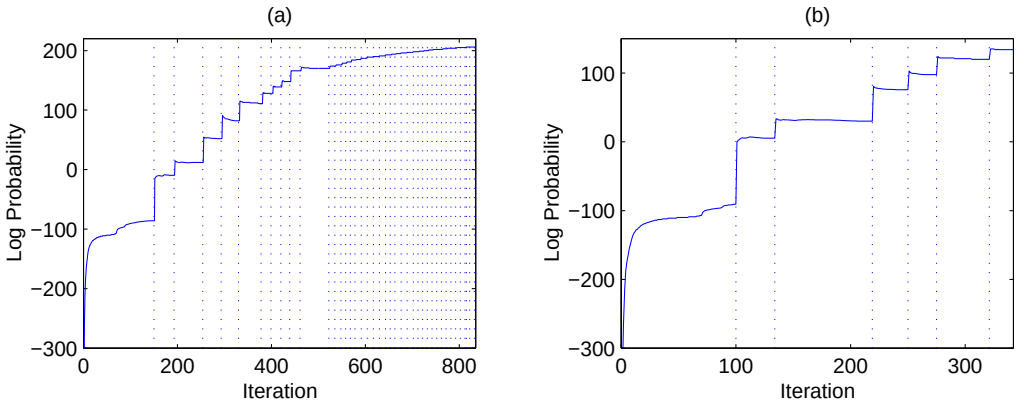

Figure 7.4: **Comparison of exact and approximate search methods for ensemble learning.** The figure shows a comparison of the exact (a) and approximate (b) ensemble learning procedures. The vertical axes show the log probability of the two methods for initially identical models (using the same random seed in the initialisation). The horizontal axis shows the iterations of the algorithm. The vertical dashed lines indicate where a hyper-parameter update took place. Note that the frequency of hyper-parameter updates increases towards the end of the algorithms.

also because in the early stopping methods the parameter training is not iterated to convergence on the training data. Indeed, as described in Chapter 5, the early stopping point often occurs within very few iterations of the training algorithm. A comparison of the speed of the different learning methods is given in Chapter 8. Finally, in order to choose the number of experts for `me-el-1` and the depth of `hme-el-1` I used the algorithm described in Section 5.3.

7.4 Related Work

There are a number of possible methods that could be used for Bayesian inference of mixtures-of-experts. Firstly, it should be noted that since the mixtures-of-experts model can be considered as a belief network, structure as well as parameters could be estimated in a Bayesian framework. This approach is not taken in this chapter, although a data-driven cross validation method for learning structure is given in Chapter 6. In addition, via the belief network formulation of the mixtures-of-experts, a multitude of inference techniques are possible *e.g.*, Bayesian inference using Gibbs sampling techniques [222] or Markov Chain Monte Carlo (MCMC) [153]. A similar technique was used by Peng et al. [172] to infer parameters of a mixture of experts. This method was applied to the Peterson Barney vowel data set. One drawback of the Gibbs sampling approach used by Peng et al. [172] is the potentially large amount of computation required to make a parameter update. In other related work, Neal has applied the MCMC technique to unsupervised mixture models [152] and has shown how mixture models with infinite components can be inferred.

Many other priors such as entropic priors are discussed in Buntine and Weigend [29]. Many practitioners of neural networks have used the concept of weight decay, often to good effect. In many cases,

however, only a single hyper-parameter is used, which corresponds to the assumption that all the parameters of the network are distributed according to a single common Gaussian. Nowlan and Hinton [161] address this assumption by using a mixture of Gaussians as a prior on the network parameters.

7.5 Summary

In this chapter I have described the ensemble learning framework for mixtures-of-experts which generalises the EM algorithm along an empirical Bayesian direction. This framework was used to develop two learning methods `me-el-1` and `hme-el-1` whose implementation was also described in this chapter. These methods will be compared in Chapter 8 with the other ME methods of Chapter 5 and Chapter 6.

Chapter 8

Results on DELVE

8.1 Introduction

Chapters 5 to 7 have introduced five learning methods based on mixtures-of-experts models. These methods use different techniques to control complexity in their models. Although it may be of interest to compare these methods on theoretical grounds alone, empirical comparisons are necessary to get a true feel for their advantages and disadvantages. As described in Chapter 3, the DELVE framework has been used in this thesis to compare learning methods. In this chapter the results of the five ME methods are compared with various other learning methods on the regression and classification tasks introduced in Chapter 3. The results indicate that mixtures-of-experts are competitive with other state of the art learning methods. In particular, a selective superiority effect is noticed, *i.e.*, the relative performance of each of the methods changes according to the data set used for comparison.

The mixtures-of-experts based methods, or *ME family* of methods, studied in this chapter are as follows:

`me-ese-1`: a committee of mixtures-of-experts trained by early stopping;

`hme-ese-1`: a committee of hierarchical mixtures-of-experts trained by early stopping;

`hme-grow-1`: a committee of hierarchical mixtures-of-experts grown via early stopping;

`me-el-1`: a committee of mixtures-of-experts trained by ensemble learning;

`hme-el-1`: a committee of hierarchical mixtures-of-experts trained by ensemble learning.

In this chapter the ME family of methods are compared with seven other methods for regression used by Rasmussen [189] and contained in the DELVE archive. The methods used by Rasmussen were:

`gp-map-1`: a Gaussian process model trained using a maximum a-posteriori method;

`gp-mc-1`: a Gaussian process model trained using the hybrid Markov Chain Monte Carlo technique of Neal [153];

`knn-cv-1`: a k nearest neighbour model with k selected by leave-one-out cross validation;

`lin-1`: Linear regression, identical to Rasmussen [189];

`mars3.6-bag-1`: a bagged version of Friedman's [58] FORTRAN implementation of MARS (version 3.6);

`mlp-ese-1`: a committee of multi-layer perceptrons trained with early stopping;

`mlp-mc-1`: a multi-layer perceptron trained using the hybrid Markov Chain Monte Carlo technique of Neal [153].

In addition, the mixtures-of-experts family methods are compared with three other methods for classification implemented by Mike Revow of the University of Toronto[1].

`cart-1`: a classification and regression tree using the Gini index splitting criterion;

`knn-class-cv-1`: a k nearest neighbour model for classification, with k selected by leave-one-out cross validation;

`1nn-1`: a one nearest neighbour model for classification.

Finally, two further methods for classification and regression were designed to show up details of particular aspects of the implementation of the ME family of methods. These are:

`lin-2`: a generalised linear model for classification or regression (with the link function chosen according to the task) in which the parameters were trained using the `macopt` conjugate gradient algorithm of Section 4.4 - this clearly has a unique solution and is included for comparison purposes only;

`lin-ese-1`: a committee of GLIMs of the form of `lin-2` trained with `macopt`, in which the training of each committee member was stopped early according to the performance on a set of validation data; this model is equivalent to the early stopped committee of ME models of Chapter 5 when one expert only is used rather than a mixture of experts.

All these methods are summarised for convenience in Table 8.1. The rest of this chapter is organised as follows. The results of these methods on different data sets are described in two sections. Section 8.2 describes and discuss the results on the regression tasks while Section 8.3 describes the results on the classification tasks. Section 8.4 concludes this chapter and this part of the thesis and gives suggestions for future work.

[1]Details of these methods and others can be found at the DELVE web site: http://www.cs.toronto.edu/∼delve/methods.

Name	Usage (C/R)	Details
gp-mc-1	R	Gaussian processes trained with MCMC methods
gp-map-1	R	Gaussian processes trained with MAP methods
knn-cv-1	R	k-nearest neighbour with k selected by leave-one-out CV
lin-1	R	Linear regression
mars3.6-bag-1	R	Multivariate Adaptive Regression Splines with bagging
mlp-ese-1	R	Multi-Layer Perceptrons trained using early stopping committees
mlp-mc-1	R	Multi-Layer Perceptron trained using MCMC methods
cart-1	C	Classification and Regression Trees
knn-class-cv-1	C	k-nearest neighbour for classification
1nn-1	C	One-nearest neighbour for classification
me-ese-1	C/R	Mixtures of Experts trained using early stopping committees
hme-ese-1	C/R	Hierarchical Mixtures of Experts trained using early stopping committees
me-el-1	C/R	Mixtures of Experts trained using ensemble learning
hme-el-1	C/R	Hierarchical Mixtures of Experts trained using ensemble learning
hme-grow-1	C/R	Hierarchical Mixtures of Experts trained via early stopped growing
lin-2	C/R	Generalised linear model trained using conjugate gradients
lin-ese-1	C/R	Committee of generalised linear models trained using conjugate gradients and early stopping (equivalent to me-ese-1 with one expert only)

Table 8.1: **Summary of the methods considered in this chapter.** Each method is shown with its name, usage and description. The usage is denoted *R* if the method is used for regression, and *C* if it is used for classification. Methods which may be used for regression or classification are given the notation *C/R*. The link function in the GLIMs of the mixtures-of-experts models is changed according to whether the models are used for regression or classification.

8.2 Results on Regression Tasks

I considered two types of data in my study of methods for regression: real and realistic. The real data was the Boston housing data [78] (see Chapter 3). The realistic data was taken from two families of data sets: the Kin family and the Pumadyn family, which are both simulations of robot arms. In each of the Kin and Pumadyn families there are eight data sets which are split into two sets of four data sets of 8 inputs and 32 inputs respectively. The simulations also have varying degrees of noise and nonlinearity in them, as described in Chapter 3. When combined with the Boston housing data this gives 17 regression tasks in total. For convenience the full set of numerical and graphical results are given in Appendix B. In the following sections I discuss and summarise the major points from each data set and draw conclusions

from them. In these studies I consider only squared error loss which is standardised by dividing by the variance of the targets, so that a model which predicts the mean of the targets will achieve a loss of 1 in the standardised domain. DELVE also allows the use of absolute error loss and log probability loss, but because of time and space constraints I do not consider these measures here.

Boston

Task	gp-map-1	knn-cv-1	lin-1	mars3.6-bag-1	mlp-ese-1	me-ese-1	hme-ese-1	me-el-1	hme-el-1	hme-grow-1
Boston/price/std.32	7	9	10	3	7	4	6	5	1	1
Boston/price/std.64	7	9	8	10	6	4	2	5	1	3
Boston/price/std.128	7	10	9	1	8	3	5	2	4	6
mean	7	9.3	9	4.7	7	3.7	4.3	4	2	3.3

Table 8.2: **Averages ranks on the Boston data**

The results of five of the original methods plus the five ME methods are shown in Figure B.1 and Table B.1. Two of the original methods are omitted (`gp-mc-1` and `mlp-mc-1`) since results for these methods were not available from the DELVE archive. Table B.1 shows the standardised squared error losses for the three tasks containing 32, 64 and 128 examples. The winning method (*i.e.*, lowest error) is shown in bold in this table. This information is additionally shown graphically in the plot of Figure B.1. In this plot the standardised squared error loss for each method is shown in a bar chart form, with standard errors shown above each bar. The order of the methods in this plot is shown at the left of the plot. In addition, below the plot is a matrix of p-values obtained from paired F-tests (or t-tests in hierarchical testing schemes). The lowest p-value shown is 1, and the highest is 5. If a p-value is greater than 5 it is not considered significant and not shown in the matrix.

For a particular task size, reading along the matrix for a method gives the p-values at which all the other methods outperformed it. If the method was not outperformed by another method no number is written in the matrix. Therefore if a method has no entries in its row it has not been outperformed significantly by any other method. Conversely, reading down the matrix from each method's bar gives the p-values at which it outperformed another method.

As noted by Rasmussen [189], the results on the Boston data are somewhat hard to interpret due to the large error bars and lack of significant p-values in the matrices. Many of the error bars in the plots overlap for example `me-el-1` and `hme-el-1` for the task with 64 examples. In this case, however, the result of the F-test suggests that `hme-el-1` significantly outperforms `me-el-1` at the 99% level. Rasmussen suggests that caution be used when interpreting the results on the Boston data set, mostly due to its small size.

Although in this case it may be hard to draw significant conclusions from the results, they do indicate that the ME family of methods is *competitive* with the best of the other methods. A comparison of the methods in terms of average rank is given in Table 8.2. The best performing methods are `hme-el-1` and

hme-grow-1. mars3.6-bag-1 also does well on the tasks with 32 and 128 examples but very badly on the task with 64 examples. Neither knn-cv-1 nor lin-1 perform particularly well on this data set. mlp-ese-1 and gp-map-1 do moderately well but not as well as the best ME methods.

The results for lin-2 are marginally better than those of lin-1, indicating that the conjugate gradient algorithm works well for linear regression experts. Over all the regression tasks, the performances of lin-2 and lin-1 were not significantly different. The early stopping plus committees variant of linear regression, lin-ese-1 gives mixed results. For the smaller data sets it performs better than lin-1 and lin-2 but for the larger data set it performs marginally worse. Once again, these differences were not significant for the regression tasks. Within the ME methods the ensemble learning methods do marginally better than the early stopping methods, although this improvement is not consistent over different sized training sets.

Results on the Kin and Pumadyn Families

An overview of the results of the twelve regression methods (not including lin-2 and lin-ese-1) on the Kin and Pumadyn datasets is given in Tables 8.3 and 8.4. These show the average ranks of each method computed over the different sized tasks of each data set. The final row in each table gives the average ranks over all the data sets.

An alternative viewpoint is given in Figures 8.1 to 8.8. These show the geometric means of the standardised squared error loss averaged over different data sets in the Kin and Pumadyn families. The data sets were designed to have different characteristics such as degree of linearity, amount of noise and number of inputs, as described in Chapter 3. The geometric mean was also used by Rasmussen [189] to compare DELVE learning methods. It is used in preference to the arithmetic mean since the relative size of the different losses on different data sets varies considerably. The geometric mean gives a measure of the *relative* differences between the methods. Given a set of losses $\{l_i\}_1^m$, the geometric mean is given by $\exp(\sum_i \log(l_i)/m)$.

Each figure shows the geometric mean averaged over similar data sets on each of the different sized tasks. The method number is shown along the horizontal axis of each plot, with the geometric mean on the vertical axis. A legend is given for each figure listing the mapping from method number to name. A method which did not contain results for a particular task is shown with the symbol $\emptyset$. Various comparisons are shown in each of the pairs of figures:

Figures 8.1 and 8.2 : these figures show the dependence of losses on data set family - either Kin (8.1) or Pumadyn (8.2).

Figures 8.3 and 8.4 : these figures show the dependence on the degree of linearity in the tasks on both the Kin and Pumadyn families - fairly linear tasks (8.3) or non-linear tasks (8.4).

Figures 8.5 and 8.6 : these figures show the dependence on the amount of noise in the tasks on both the Kin and Pumadyn families - medium noise tasks (8.5) or high noise tasks (8.6).

Figures 8.7 and 8.8 : these figures show the dependence on the number of inputs in the tasks on both the Kin and Pumadyn families - 8 input tasks (8.5) or 32 input tasks (8.6).

Overall, the Gaussian process and MLP methods perform best on the Kin and Pumadyn data sets. The MCMC methods for both GP and MLP methods are the most successful on average. However, `gp-map-1` performs well on the larger data sets, with its relative performance improving as the size of the data set is increased. The MLP with early stopping method, `mlp-ese-1`, performs almost as well as the MLP with MCMC, `mlp-mc-1`, on the Kin data sets but then performs much worse on the Pumadyn data sets.

The k nearest neighbour method, `knn-cv-1` performs poorly on average over all tasks in the Kin and Pumadyn tasks, ranking last on most of the tasks. It performs better than the `lin-1` method, however, on the non-linear tasks (Figure 8.4), in particular, the non-linear tasks with 8 inputs.

The linear regression method, `lin-1`, performs better on the Kin data sets than on the Pumadyn data sets. It performs well on the fairly linear tasks, as might be expected.

The performance of MARS with bagging, `mars3.6-bag-1`, is particularly dependent on the data set family. It ranks 11^{th} on the Kin data sets, worse on average than `lin-1`, yet ranks 4^{th} on average on the Pumadyn data sets. The method also performs better on the 32 input tasks than on the 8 input tasks.

For the mixtures-of-experts methods, the performance lies somewhere between the linear method and the MLP with early stopping. On the Kin data sets they perform better than MARS whilst on the Pumadyn data sets they perform worse than MARS. Their best performance is on kin-8fh in which they perform best out of all the methods. Within the ME methods the best performing method, on average, is `me-ese-1`. However, for the small data sets `hme-grow-1` performs best, whilst for the larger data sets `me-el-1` performs best. The ensemble learning methods, `me-el-1` and `hme-el-1` work better on the non-linear data sets than on the linear data sets but surprisingly work best when there is more data available. This was not anticipated before these empirical results since it is generally thought that early stopping via cross validation will not work as well as Bayesian methods when there is a small amount of data. Possibly the use of committees for the early stopping methods is averaging effectively over the uncertainty in the validation samples for small data sets. There appears to be little change in the overall behaviour of the ME methods according to the amount of noise. The number of inputs, however, seems to affect their performance. In comparison to the early stopping methods, the ensemble learning methods perform better on the 8 dimensional data sets than on the 32 dimensional data sets. In contrast, the growing method, `hme-grow-1`, performs better on the 32 dimensional data sets than on the 8 dimensional data sets. This dependence of number of inputs on the growing algorithm can partly be explained by the performance of the linear regression method, `lin-1`, which also performs better on the 32 than on the 8 dimensional data sets. Since the growing algorithm can select a single expert as well as hierarchies of experts, its performance might be expected to be correlated with that of the linear regression method. Finally, the mixture models seem to be better overall than the binary hierarchical models, apart from the hierarchies made by growing.

The most spectacular differences between the methods may be seen on the non-linear Pumadyn data sets, Pumadyn-32nm, Pumadyn-32nh, Pumadyn-8nm and Pumadyn-8nh. As noted by Rasmussen

[189], these data sets have only a few important inputs. The methods which can take account of this fact perform the best on these data sets - the GP methods which use the Bayesian automatic relevance detection approach [153], and MARS (with bagging).

Data set	gp-mc-1	gp-map-1	knn-cv-1	lin-1	mars3.6-bag-1	mlp-ese-1	mlp-mc-1	me-ese-1	hme-ese-1	me-el-1	hme-el-1	hme-grow-1
Kin-32fm	2.0	5.0	11.8	5.4	11.2	3.0	2.2	7.6	8.6	9.4	8.0	4.0
Kin-32fh	3.8	6.6	11.8	2.0	11.2	3.6	4.0	7.2	7.0	9.8	8.2	3.4
Kin-32nm	1.3	6.0	9.6	10.8	9.0	4.4	2.8	4.8	5.8	8.2	7.4	8.0
Kin-32nh	1.8	5.0	8.8	10.0	8.6	3.8	2.6	5.4	6.8	9.4	8.0	8.0
Kin-8fm	2.3	3.2	12.0	10.8	10.2	4.8	1.2	4.6	5.8	6.6	8.6	8.2
Kin-8fh	5.0	6.6	12.0	8.8	10.2	7.2	5.0	1.8	2.8	8.2	7.0	4.2
Kin-8nm	1.8	3.6	10.0	11.6	11.0	2.8	2.4	6.0	6.2	5.4	8.4	9.0
Kin-8nh	4.0	4.2	10.4	10.6	10.6	2.6	2.6	6.4	6.2	4.6	7.0	8.8
mean	2.7	5.0	10.8	8.7	10.3	3.9	2.8	5.5	6.1	7.7	7.7	7.1

Table 8.3: **Average ranks on the Kin data sets**

Data set	gp-mc-1	gp-map-1	knn-cv-1	lin-1	mars3.6-bag-1	mlp-ese-1	mlp-mc-1	me-ese-1	hme-ese-1	me-el-1	hme-el-1	hme-grow-1
Pumadyn-32fm	3.3	2.4	12.0	6.0	2.0	5.4	2.8	7.8	8.2	10.0	10.2	8.4
Pumadyn-32fh	3.0	2.8	11.8	8.6	3.2	5.2	1.4	7.2	7.6	9.2	10.2	8.2
Pumadyn-32nm	1.5	1.6	9.6	9.8	3.2	5.0	3.8	8.4	9.6	8.4	8.6	8.2
Pumadyn-32nh	2.0	3.6	8.2	9.4	1.8	4.8	3.2	8.4	8.6	9.8	9.8	8.6
Pumadyn-8fm	2.0	2.4	12.0	9.6	3.0	5.0	2.8	7.0	8.6	7.8	8.8	9.0
Pumadyn-8fh	4.0	3.2	12.0	6.4	6.4	4.8	1.8	6.2	8.6	10.2	9.6	5.8
Pumadyn-8nm	1.5	2.4	11.4	11.6	5.0	4.4	2.2	8.0	9.0	6.0	6.6	10.0
Pumadyn-8nh	1.8	2.4	11.4	11.2	3.8	5.0	2.2	7.6	9.2	7.4	7.4	8.8
mean	2.4	2.6	11.1	9.1	3.6	4.9	2.5	7.6	8.7	8.6	8.9	8.4

Table 8.4: **Average ranks on the Pumadyn data sets.**

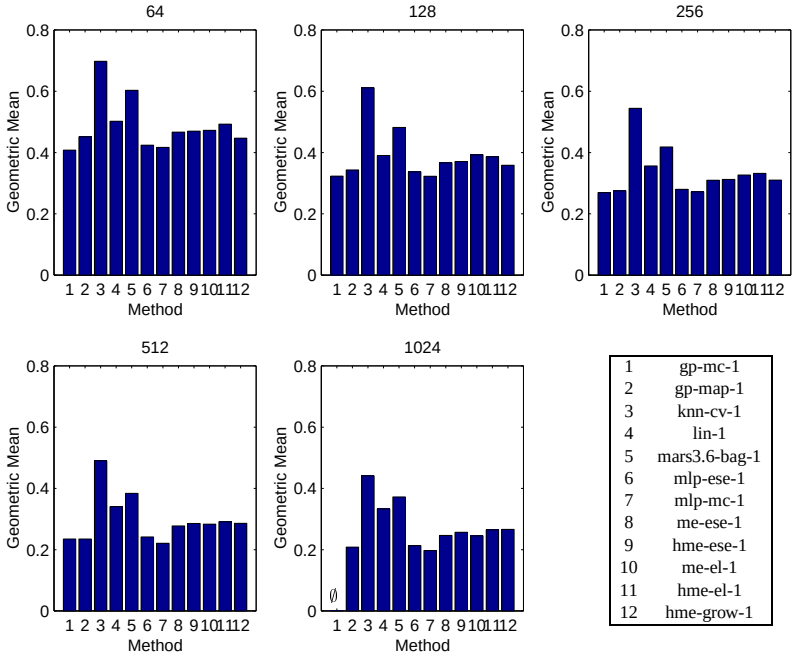

Figure 8.1: **Geometric mean of squared error loss averaged over all tasks in the Kin family**

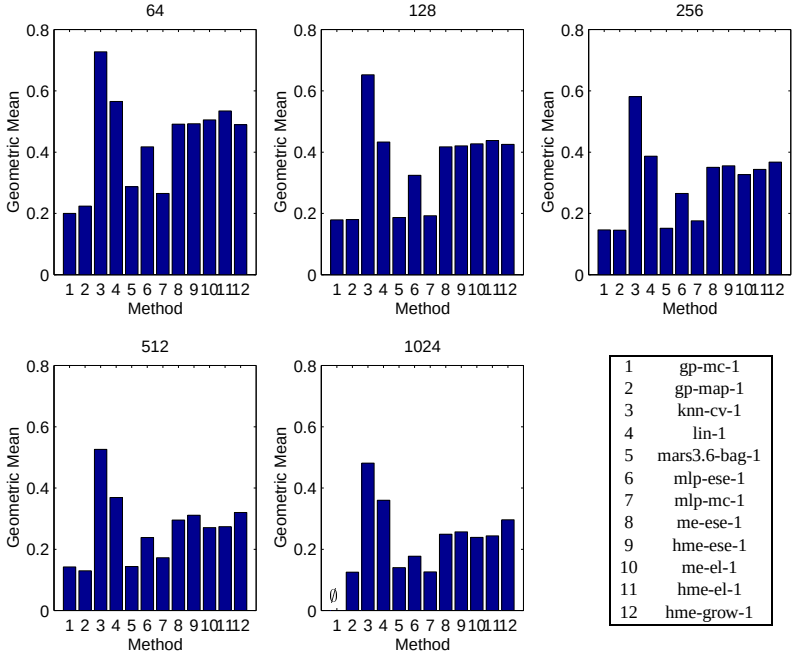

Figure 8.2: **Geometric mean of squared error loss averaged over all tasks in the Pumadyn family**

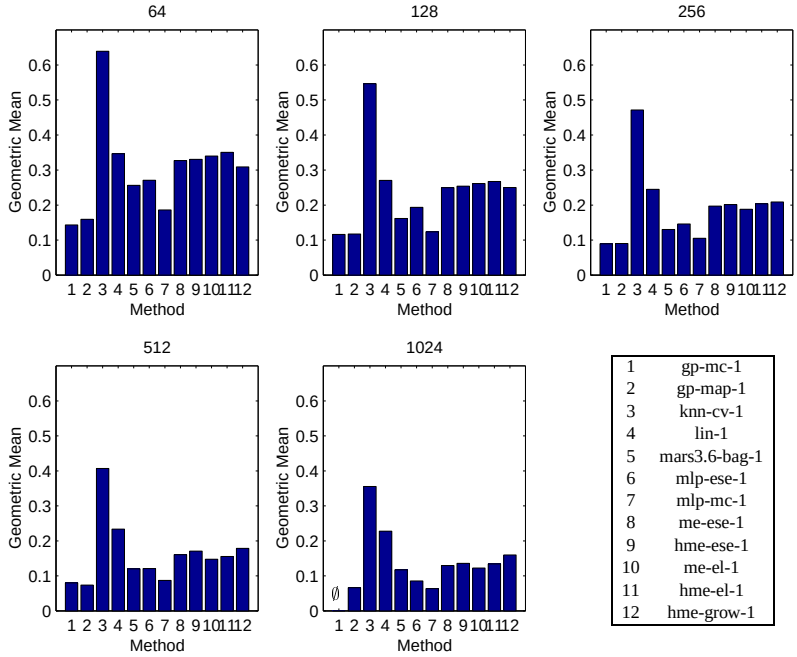

Figure 8.3: **Geometric mean of squared error loss averaged over all fairly linear tasks in the Kin and Pumadyn families**

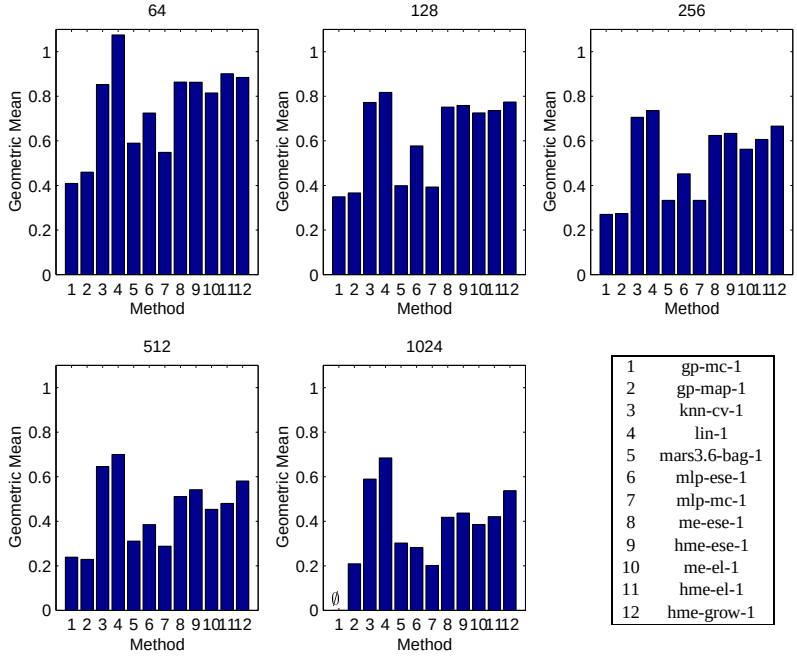

Figure 8.4: **Geometric mean of squared error loss averaged over all non-linear tasks in the Kin and Pumadyn families**

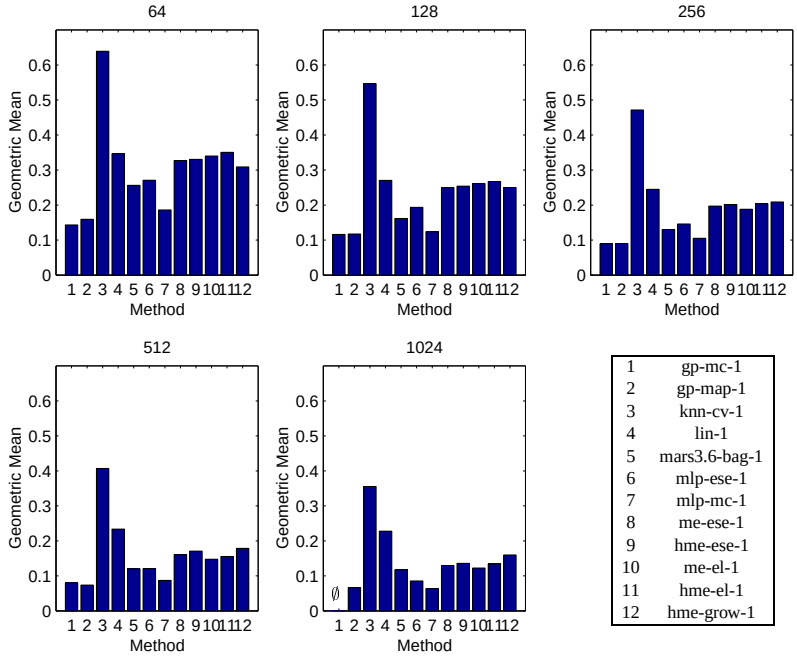

Figure 8.5: **Geometric mean of squared error loss averaged over all medium noise tasks in the Kin and Pumadyn families**

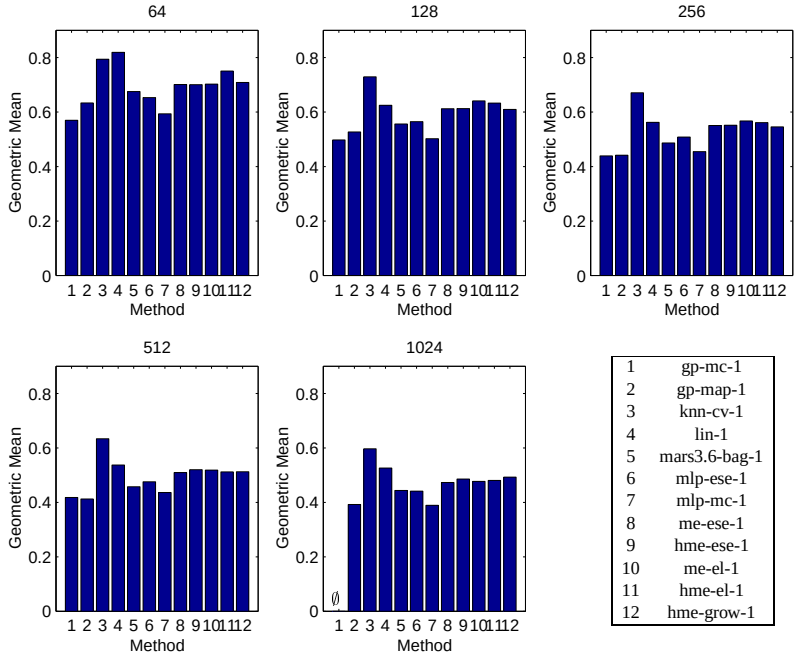

Figure 8.6: **Geometric mean of squared error loss averaged over all high noise tasks in the Kin and Pumadyn families**

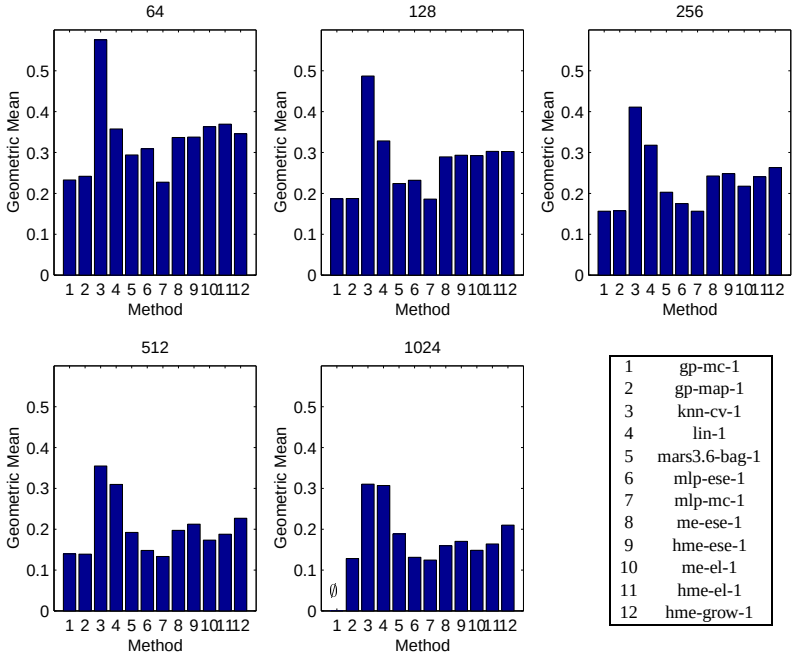

Figure 8.7: **Geometric mean of squared error loss averaged over all 8 dimensional input tasks in the Kin and Pumadyn families**

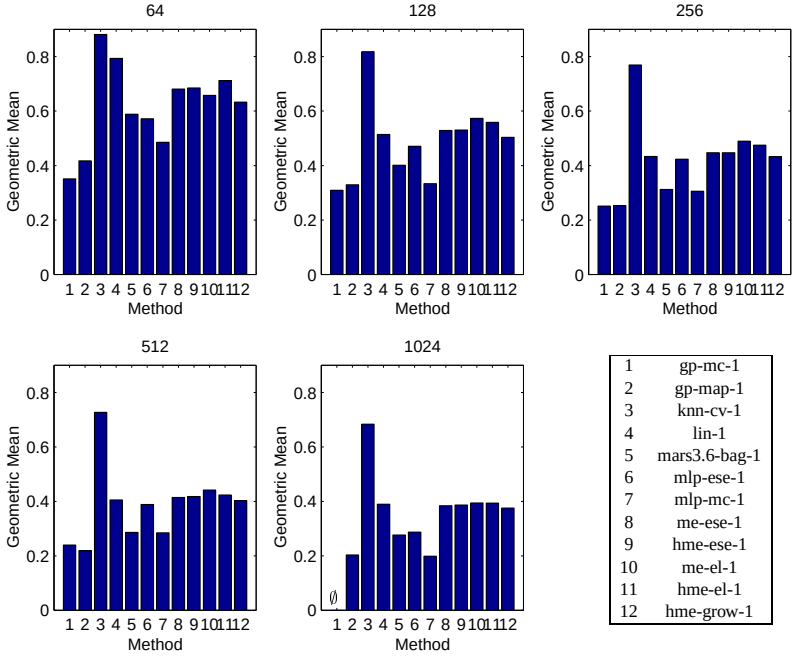

Figure 8.8: **Geometric mean of squared error loss averaged over all 32 dimensional input tasks in the Kin and Pumadyn families**

8.3 Results on Classification Tasks

In addition to the regression tasks described in the last section, three classification tasks were used in this thesis. These were described in Chapter 3 and are Image, Letter and Splice. The full results for these data sets are given in Appendix C. Each of the sections C.2 to C.3 contain a figure and table showing the $0/1$ losses standardised by the baseline of guessing the majority class.

Image

The results on the Image data set are shown in Section C.1. There are large error bars evident in Figure C.1 which contribute to the small number of significant performances in the p-value matrices. Few conclusions can be drawn from this data set except that the ME family of methods perform better (but not always significantly better) than either of CART, nearest neighbours or logistic regression. For the largest training set the performance of `cart-1` approaches the ME methods and is not significantly outperformed on this training set. Within the ME family, the ensemble learning methods perform best on the training sets of size 70 and 140 with the growing method performing best on the 280 example training sets. None of the ME methods significantly outperform any of the other ME methods, making conclusions difficult in this case.

Letter

The results on the Letter data are shown in Section C.2. In contrast to the Image data the error bars are much smaller in magnitude, and there are many more significant values in the matrices. Overall `cart-1` is significantly outperformed by all methods on all tasks. The nearest neighbour methods are significantly outperformed by some of the ME methods on the first two tasks but are not outperformed on the largest task. The ME methods are on average the most successful on this data, and the early stopping methods are the most effective within these. Together the early stopping methods significantly outperform all other methods on all tasks except for the nearest neighbour methods on the largest task sizes.

Splice

The results on the Splice data are shown in Section C.3. The error bars are moderate in Figure C.3 and the results of comparisons appear significant. Overall, the nearest neighbour methods do poorly on all three task sizes. CART does better, particularly on the largest tasks. However, all three methods are significantly outperformed by the ME methods across all three task sizes. Within the ME methods there is little space to choose a best method. On the smallest task size the ensemble learning methods have the smallest loss whilst on the larger two tasks the early stopping methods perform best. There are, however, only a few significant out performances amongst the ME methods. However, one of the best performing methods on this data set is `lin-ese-1`, which is equivalent to the `me-ese-1` method with only one expert. This suggests that the mixtures are not adding much benefit in this data set.

8.4 Conclusions

The results on the classification tasks indicate that the ME methods are competitive with the best classifiers in the study on the three tasks chosen. Indeed on most of the classification tasks the ME methods are the best overall by a significant margin. These conclusions should be taken with care, however, firstly as a result of the small number of data sets and secondly as a result of the lack of many comparative classification methods in DELVE. However, the methods that were compared span most of the popular methods for classification, *i.e.*, nearest neighbour, decision trees and logistic regression, so there is at least some indication that the ME methods are useful within this range.

On the regression tasks, the ME methods perform well on the Boston data but overall on the Kin and Pumadyn data sets the Gaussian process and MLP based methods perform best. Early stopping works well for the Kin data but less well for the Pumadyn data for both MLP and ME methods. MARS with bagging performs poorly on the Kin data but well on the Pumadyn data and overall is on par with the early stopped MLP. The nearest neighbour method is the worst performing method by a considerable margin whilst the linear regression method performs poorly on average, but is sometimes the best method for the Kin data. The ME family of methods lie between the linear regression, bagged MARS and early stopped MLPs and perform better on the Kin data than on the Pumadyn data.

Within the ME family of methods the mixture models perform better than the hierarchical models. Overall, there seems to be little to choose between early stopping and Bayesian methods for controlling complexity, although the Bayesian methods perform better on the Pumadyn data sets and on non-linear data sets. The growing method performs the least well on average, although it is the best method on small data sets. However, each of the methods in the ME family has at least one data set on which it performs best out of the ME family methods.

Viewed as a whole, the regression tasks indicate that Gaussian processes and MLPs are the best methods on average, whilst MARS is competitive on some of the tasks as are the ME methods. On average linear regression and nearest neighbour methods are not competitive with these more sophisticated methods.

These conclusions must be taken with caution, however. Whilst the regression tasks considered are fairly wide ranging in that they contain different noise and non-linear effects, they are still from a restricted domain, *i.e.*, robot arm simulations. It seems likely that the winning methods in this domain will perform well on other robot arm data sets, either *realistic*, as in this study, or hopefully *real* ones. Whether they will also perform well on other domains remains to be seen. Hopefully, the flexibility of the DELVE framework will encourage other researchers to contribute new methods and data sets which will allow more conclusions to be drawn in the future.

This concludes the description of the results on the DELVE data sets. Although the results give some indication as to which method we might want to use for a particular task, there are many other issues in choosing a learning method. These issues include the ability to add expert knowledge, the speed of the method and how interpretable it is. These issues are discussed in the following sections. Before this, we discuss the issue of selective superiority and how one might choose between different methods for a particular task.

Selective superiority

For both the overall results and the ME family results, we therefore see two levels of *selective superiority*, both at a model bias level and at a search bias level. As noted by Brodley:

> *The results of empirical comparisons of existing learning algorithms illustrate that each algorithm has a selective superiority; it is best for some but not all tasks.*
> Brodley [24]

Given any particular data set there exists a certain model class (*e.g.*, Gaussian processes, MLPs, MEs) and a certain search method for that model class (*e.g.*, MCMC, MAP, early stopping) which will perform best. This begs a question at a *meta* level - how do we *choose* between different models and search methods for a new, unseen data set ? This question was considered by Michie et al. [144, pp.197-211] in their study of different classifiers. They used a series of characteristic statistics of the data sets in their study and examined the relationship between these statistics and the performance of the different classifiers by using a decision tree and derived rules built by C4.5 [186]. Their conclusions were that it is possible to infer certain relationships between data set characteristics and the performance of different classifiers on that data set, although they stress that care must be taken with direct interpretation of their rules since the space of classifiers and data sets was relatively sparse. Another approach taken by Brodley [24] is to use a number of simple classifiers (Brodley used k-nearest neighbour, linear discriminants and decision trees) and *recursively* apply them to a data set. The recursive process works by first applying simple rules to choose the initial classifier. For regions of the data where the behaviour of the first classifier follows certain rules, a new classifier is then chosen. This process is iterated and the classifiers combined in a piecewise manner. Brodley shows that the resulting hybrid classifiers have performance equal to or greater than the best single classifiers on all data sets in the study. Such an approach is attractive for simple classifiers such as the ones studied by Brodley, but is less applicable to the sophisticated methods used here which take much longer to train. A final possibility is the use of *simple* models to predict the performance of more *complex* models. This *yardstick* approach was been used by Michie et al. [144, pp.210-211] and also by Holte [92]. Holte considered the use of a simple decision "stump" (a decision tree with only one rule), which he coined the "1 rule" (1R) method. In Holte's study he found that the 1R method was capable of predicting quite accurately where more sophisticated methods, in particular C4.5, would do well.

Using expert knowledge

The results from DELVE represent the best that can be achieved with an *automatic* learning method. As noted by Michie et al. [144] the behaviour of many methods will be markedly different in the hands of an expert:

> *In the hands of an expert the performance of an algorithm can be radically different, and of course there is always the possibility of transforming or otherwise preprocessing the data. These considerations will often outweigh any choice of algorithm.*
> Michie et al. [144, pp.225]

This point is further emphasised in Bayesian methods in which choice of appropriate priors can have a major effect. For example, Zhu and Rohwer [259] describe results on a number of the UCI data sets which show that appropriate choice of priors based on performance on some data sets can help improve performance on unseen data sets of the same type. However, automatic learning methods are the only *practical* manner in which to perform comparisons since otherwise extra variability such as the performance of the expert must be considered. The former point, however, means that the failure of an algorithm on a data set in DELVE does not mean that in the hands of an expert it might not work much better. It is interesting to note that the winning methods, Gaussian processes and MLPs, have many adjustable parameters but few that can be set easily using prior knowledge, or readily interpreted.

One additional point to consider is that the comparisons made here are between particular instances of learning methods, *i.e.*, we are comparing software programs not abstract concepts. Hopefully these programs represent good implementations of the state of the art in each method but it should be remembered that this does not mean that a poorly performing method means that another implementation of that method will also do badly, per se. Given the many design choices and restrictions imposed by finite computing resources (and patience) it is likely that new implementations will have improved performance either as a result of being more efficient, or by using more effective search strategies, *e.g.*, Markov chain Monte Carlo. Only after a number of implementations of a method have been compared will it be possible to say which model class or search method gives the best performance on average. In particular, the results do not mean that ME or HME model classes are not as good as other model classes. There is a possibility that for the regression tasks the training and architecture selection methods do not find the optimal model. Future research will be needed to validate these conclusions on general data sets.

Speed of algorithms

An interesting discussion of the issues of searching for the best model is given by Dietterich [46], who describes the apparent trade off between computational time spent during search and generalisation of models. This is in contrast to computationally intensive methods such as MCMC trained MLPs [153]. Rasmussen [189] performed a study of varying the time allowed for different Gaussian process and MLP methods which suggest that even when the MCMC methods are allowed a small amount of time (and will therefore not usually have converged) they still perform significantly better that other methods on average.

One of the original motivations for mixtures-of-experts was their fast convergence rates. As pointed out in Chapter 2 most of these speed comparisons were in terms of epochs (*i.e.*, presentations of the training data) rather than actual time taken. In this study I have not attempted to compare ME methods with MLPs in terms of speed, except that the `me-ese-1` and `mlp-ese-1` methods are allowed the same overall time to build committees. In general, the `me-ese-1` method takes of the order of $M \times P \times T_{lin}$ where M is the number of experts and gates, P is the number of iterations (typically about 10) and T_{lin} is the time taken by linear regression. In comparison, Figure 8.9 shows the average time taken by a single model in each of the five ME methods on the Kin and Pumadyn data sets. Note that this is only the time taken by *one* model in the committees and that the overall time allowed for each method is the same. One obvious point is that the ensemble learning methods take on average around 20 times longer (23.6 times longer for `me-el-1` than `me-ese-1` and 18.7 times longer for `hme-el-1` than `me-el-1`) than the early stopping methods. This is as a result of the parameter/hyper-parameter iterations which take a long time to converge. One encouraging aspect however is that the growing algorithm `hme-grow-1` takes a similar amount of time as the fixed architecture early stopping methods.

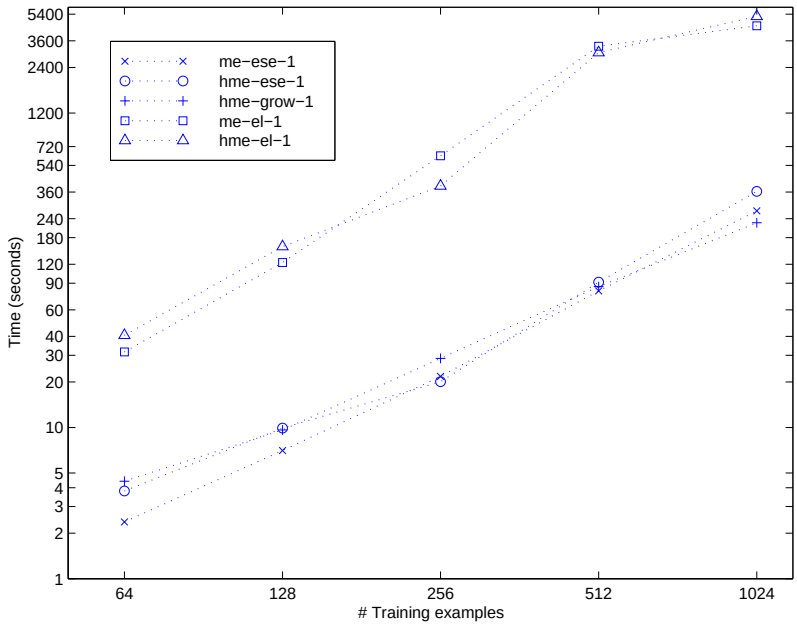

Figure 8.9: **Speed of mixtures-of-experts methods.**
The figure shows the time taken in seconds by a single model in each of the ME methods on the vertical axis, against the number of training cases on the horizontal axis. Dashed lines are shown between the symbols to guide the eye. The computer used in these experiments was a Sun Sparc-station 20.

Interpretability

The mixtures-of-experts based learning methods studied here all use generalised linear models as experts and gates. These models can be given some interpretation. For example, in logistic regression [140, pp.110] we may rewrite the model as:

$$\log\left(\frac{y}{1-y}\right) = \boldsymbol{\theta}^T \boldsymbol{x}. \tag{8.1}$$

The term on the left is the *log odds* of the category y. If all other covariates are kept fixed, a unit change in one of the covariates x_i increases the log odds by an amount given by its parameter θ_i.

In comparison to other tree structured predictors, however, mixtures-of-experts models are less interpretable. Univariate classification and regression trees use only one covariate per split, and use constant functions as predictions at their leaves. These models lend themselves readily to interpretation, a property which has led to their widespread usage in some fields. Although mixtures-of-experts have similarities with standard classification and regression trees, they use all the covariates to make each split and to make each prediction at the leaves. It may be possible, however, to make mixtures-of-experts more interpretable. For example, by using less covariates to make each split and each prediction, more interpretation might be possible. Similar techniques for building multivariate trees which use a restricted set of covariates have been described by Brodley and Utgoff [26].

A final point to note about interpretation was made by Breiman [20] who notes that there is an apparent trade off between predictive accuracy of a model and its interpretability. For example, in bagging, multiple copies of a model are trained on different bootstrap samples from a training set. The resulting predictions, when combined in a committee, are more accurate than a single model, but the committee is less interpretable than a single model. This trade-off is expressed neatly by Breiman [20] in his anecdotal *Heisenberg uncertainty principle for statistics*:

$$IA \geq b \tag{8.2}$$

where I is interpretability, A is predictive accuracy and b is "Breiman's constant". It is interesting to note that although mixtures-of-experts have less interpretability than CART, they have been shown in this work to be more accurate than CART on classification tasks. Whether this result holds true across a range of tasks will only be determined after more extensive comparisons have been performed.

8.5 Summary

The results presented in this chapter indicate that mixtures-of-experts models are competitive with other state of the art learning methods. Although they perform poorly on some tasks they perform well on others. The results on the three classification tasks are particularly encouraging, with mixtures-of-experts models outperforming CART, k nearest neighbour and logistic regression in the majority of the tasks. As described previously, it is difficult to say in general what kind of tasks mixtures-of-experts or any other kind of model will perform well on in general. Further comparisons will be necessary to determine this.

Since the results of the comparisons studied here are available via the DELVE framework, it is hoped that other researchers will be able to extend the work done here when more data sets and methods are incorporated into DELVE.

This concludes the first part of the thesis. In this part of the thesis the aim was to place mixtures-of-experts in context with other learning methods, and to investigate new methods for controlling complexity of mixtures-of-experts models. In the second part of the thesis the mixtures-of-experts framework is applied to speech recognition tasks. The aim in Part II is to investigate how successfully the mixtures-of-experts framework can be applied to a large scale, real world problem. The data sets studied in Part II are much larger than the ones studied in Part I of the thesis. For example the largest training set studied in Part I was 1560 points, whereas in Part II the largest training set used has over 15 million examples. Due in part to the difficulties of training models in such large databases, the classification models described in Part I are not used in Part II. Instead, mixtures of standard neural networks, used successfully in speech recognition in the past, are investigated. As will be shown in Part II, the use of mixtures gives a significant decrease in error rates for a state of the art speech recognition system on a large speech database.

Part II

Extensions of Mixtures-of-Experts and their Evaluation on Speech Recognition Tasks

Chapter 9

Introduction to Part II

9.1 Automatic Speech Recognition

Automatic speech recognition (ASR) is the transcription of speech into text. It is one of the most exciting areas of machine learning, since its solution promises many applications and innovations in everyday life. It is, however, one of the most difficult aspects of machine learning, involving very large data sets and issues such as noisy environments and variation in speakers. In the last decade a large amount of research effort has gone into ASR and current state of the art systems are capable of word error rates around 10% in controlled environments, with speech recorded in quiet conditions with a known microphone [165]. This chapter gives a basic introduction to one of state of the art speech recognition systems: the ABBOT system of Robinson et al. [199].

The dominant technique for speech recognition is based on statistical methods [*e.g.,* 6]. In this framework, the goal of a speech recogniser is to find the word sequence $\hat{W} = w_1 w_2 \dots w_N$ with the maximum *a posteriori* probability given the sequence of acoustic observations $X = x_1 x_2 \dots x_T$:

$$\hat{W} = \arg\max_W P(W|X) \tag{9.1}$$

$$= \arg\max_W \left(\frac{P(X|W)P(W)}{P(X)} \right). \tag{9.2}$$

While some of the original recognition systems attempted to model $P(W|X)$ directly it is generally acknowledged that this is an impractical approach. Standard systems now break the problem down into the modelling of $P(W)$ via a *language model* and the modelling of $P(X|W)$ via an *acoustic model*. Ideally we would like to model $P(X|W)$ directly but this is impractical since W is the set of all possible word sequences in the vocabulary. Instead we could choose to model $P(X|\{w_i\})$ where w_i is a particular word, and then use the language model to provide probabilities of each sequence of words. This is a practical approach only for small vocabularies, less than say 100 words, but is impractical for larger vocabularies. This approach also blurs the fact that words occur in different contexts. For large vocabulary modelling, therefore, the following approach is used. We model *sub-word* units $\{q_i\}$ via an acoustic model $P(X|q_i)$. These sub-word units are mapped to words using a *lexicon*. Possible word

sequences are subsequently evaluated using the language model. The most common sub-word unit used is the *phone*. As an example, the word "speech" is represented by a sequence of phones as $/s/p/iy/ch/$.

The most popular speech recognition systems are based on hidden Markov models (HMM) [eg 130, 94]. Individual HMMs are used to model the probability $P(X|q_i)$ of the sequence of acoustic data X given each phone q_i. These phone HMMs are concatenated together to form a larger HMM which models the possible phone sequences that made up the acoustic data. The language model allows any word sequence but gives low probability to sequences not observed in the training data. A lexicon is used to map from words to phone sequences. Most language models consist of counts of the frequency of each token (phone or word) given the previous set of N tokens. These are called N-gram language models and common examples in use are *trigrams* and *bigrams* although some more recent systems have used *four-grams*.

There are two major approaches to modelling the probability of the acoustics given each phone $P(X|q_i)$ in HMMs for speech recognition. The oldest, and most popular, is to use parametric densities to model each state in the HMM for each phone. The most popular choice of density is a Gaussian mixture with mean $\boldsymbol{\mu}$ and diagonal covariance matrix Σ (denoted collectively as parameters θ). For example, for state j of an HMM for phone q_i the density is given by:

$$P(X|s_j, \theta) = \sum_{k=1}^{K} \alpha_k P(X|s_j, \boldsymbol{\mu}_{jk}, \Sigma_{jk}) \tag{9.3}$$

where the sum is over the K mixture components and α_k is the weight for each component. These systems are known as *continuous density hidden Markov models* (CDHMM) speech recognition systems.

Training of the HMM is performed using the Baum-Welch (BW) algorithm [8] which is a specific instance of the EM algorithm [45]. The E step of the BW algorithm involves calculating posterior probabilities of state occupation using the *forward-backward* algorithm. The M step then computes the updated state transition probabilities and parameters of the Gaussians. The updates of the Gaussian distributions are particularly simple, consisting of the accumulation of sufficient statistics [49] weighted by the posterior probabilities of state occupation. CHDMM systems remain the most popular and successful method for speech recognition. A good overview of state of the art techniques in CDHMM systems is given by Knill and Young [120].

An alternative acoustic modelling approach is to model the *posterior* $P(q_i|X)$ using a classifier such as a neural network, and subsequently use Bayes' rule to compute the probability density $P(X|q_i)$ for each state in a HMM. Since neural networks are used to model $P(q_i|X)$ these systems are known as *connectionist-HMM* hybrids. Both multi-layer perceptrons [17] and recurrent networks [198] have been used in these systems. The hybrid systems generally have similar performance to CDHMM systems and use many fewer parameters [84]. However, the best systems on benchmark trials (described in Section 9.3) remain the CDHMMs [165].

Before describing the speech recognition framework further, let use introduce some commonly used terms in ASR. State of the art speech recognition systems (either CDHMMs or hybrids) are either speaker independent (SI) in that they are designed to work equally well across a set of speakers or speaker dependent (SD) in that they are designed to work for one speaker only. In order to handle new speak-

ers from outside the corpus in SI systems *speaker adaptation* is used. Note that this is different from speaker *recognition* or *identification* which is the problem of classifying the identity of different speakers. Speaker adaptation is considered further in Chapter 11. Finally a related research area is speech *coding* which is concerned with the efficient compression of speech and decompression of these codes to naturally sounding speech (see Atal et al. [5] for a review). Speech coding is not considered further in this thesis.

In this thesis I focus on a hybrid connectionist speech recognition system, specifically the ABBOT system of Robinson et al. [199]. In this part of the thesis I describe two extensions to the ABBOT system based on mixtures-of-experts models. The aim is firstly to demonstrate how the mixtures-of-experts framework can be integrated into an existing hybrid ASR system and secondly to investigate whether mixtures-of-experts can give a significant improvement in performance of this system. In Chapter 10 I describe the use of mixtures of multi-layer perceptrons within ABBOT and demonstrate improved recognition accuracy. In Chapter 11 I extend the speaker adaptation method of ABBOT via a mixtures-of-experts scheme and demonstrate improved performance on supervised speaker adaptation. One of the main issues in this part of the thesis is the scaling up of the mixtures-of-experts framework to large data sets. In speech there are often many millions of examples in training sets. Efficient search through parameter space is therefore vital for training a speech recognition system.

In this chapter I introduce the ABBOT system of Robinson et al. [199] and important aspects of speech recognition that relate to this system. This chapter is structured as follows. In Section 9.2 I describe the ABBOT system. I outline methods for acoustic preprocessing in Section 9.2 and alternative acoustic models in Section 9.2. Finally Section 9.2 describes the language modelling scheme used in ABBOT. I conclude this chapter with a discussion of evaluation schemes for speech recognition systems and a description of the data sets used.

9.2 The ABBOT System

The ABBOT system of Cambridge University Engineering Department (UK) was one of the first hybrid connectionist-HMM speech recognition systems and evolved from research into the use of recurrent neural networks for phone recognition by Robinson [197], [see also 198] at Cambridge University Engineering Department (CUED). Another influence on the development of ABBOT was the work of Bourlard and Morgan [17] at the International Computer Science Institute (ICSI) (Berkeley, CA) on the use of multi-layer perceptrons for speech recognition with hybrid systems.

A schematic diagram of the ABBOT system is shown in Figure 9.1. The individual components of this figure will be explained further in the following sections. In summary, speech is pre-processed into *frames* every $16ms$. Each frame is a vector of 13 features in length. These frames are presented as inputs to a connectionist model which estimates the probability of 54 phones. The resulting sequence of phone probabilities are converted to likelihoods and used as observation probabilities in a set of hidden Markov models. The most likely word sequence or *utterance hypothesis* is then computed by the combination of a pronunciation lexicon, language model and the NOWAY stack decoder [191].

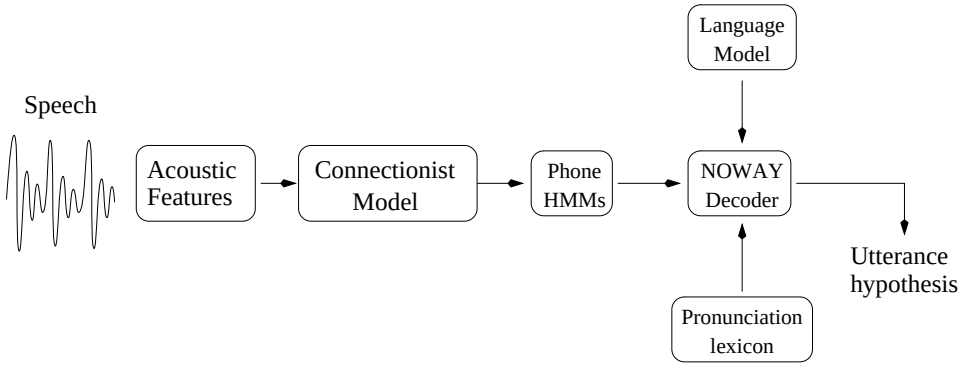

Figure 9.1: **The ABBOT system.** See Section 9.2 for more details.

Acoustic Preprocessing

Although speech is a non-stationary process, it is generally assumed to be "quasi-stationary" over short time periods. The the original speech is sampled at between 8 and 16 kHz depending on the quality of the speech (8 kHz is used for telephone quality speech while 16 kHz is used for high quality speech). Features are extracted into *frames* every $16ms$. The quasi-stationarity assumption is that over the period of these frames the statistics of the speech do not differ considerably.

A number of acoustic feature formats have been used in the past, including spectral coefficients and linear predictive coefficients [138]. Two of the predominant feature vectors in use in modern ASR systems are mel-frequency cepstral coefficients (MFCC) [42] and perceptually-based linear prediction (PLP) [85]. MFCC coefficients are use predominantly in CDHMM systems [250] whilst PLP coefficients are used in the ABBOT system. Perceptual linear prediction is similar in form to standard linear prediction [138]. In PLP, however, several well known auditory properties are simulated, and the resulting spectrum approximated by an all-pole model.

To improve the temporal modelling across frames, difference coefficients [64] or *deltas* of acoustic features are used to augment the feature vectors. If the original PLP feature vector is given by $\boldsymbol{x}^{(n)}$, the resulting vector after deltas are added is given by $[\boldsymbol{x}^{(n)^T} \boldsymbol{\Delta}^{(n)^T}]^T$. The delta coefficients are computed over a symmetric window of $2P$ frames and take the following form:

$$\boldsymbol{\Delta}^{(n)} = \frac{\sum_{p=1}^{P} \left(\boldsymbol{x}^{(n+p)} - \boldsymbol{x}^{(n-p)}\right) p}{2 \sum_{p=1}^{P} p^2}. \tag{9.4}$$

where p is the index over the window of frames. P is typically set to 5.

An additional method of modelling context is a *sliding window* which was first proposed by [212] in the NETtalk system. A sliding window pads each feature vector (of length d) with a set of its adjacent frames *e.g.*, from time $(n - P)$ to $(n + P)$ for feature vector $\boldsymbol{x}^{(n)}$, yielding an overall feature vector of

length $(2P + 1)d$. Sliding windows may be applied to either the acoustic feature vectors alone or to the vectors after they have been augmented with delta coefficients.

Acoustic Modelling

The acoustic model of a connectionist-HMM hybrid system computes the probabilities of a set phones given an acoustic vector. Two types of acoustic model are used in ABBOT: recurrent neural networks [198] (RNN) and multi-layer perceptrons (MLP) [16]. The recurrent neural network is the most commonly used model for large vocabulary evaluations. However, recent research into efficient methods of using MLPs in ABBOT by Cook [38] has suggested that these may be competitive with RNNs. In this work I have used mixtures of both the MLP and RNN models within ABBOT. I describe these models further in Chapters 10 and 11 respectively.

Although the models developed in Part I of this thesis could potentially be used as acoustic models within ABBOT, I have not pursued this line of research. My aim was to demonstrate how an existing model, either RNN or MLP, could be augmented by a mixture-of-experts framework, rather than starting from scratch with a completely new model. Indeed considerable research has been done at both CUED and ICSI on methods of training these models effectively. Complexity control is performed in both of these models by early stopping via cross-validation. Bayesian methods for the MLP acoustic model of ABBOT have also been investigated by Renals and MacKay [192]. They used the evidence framework of MacKay [133] to train MLPs with up to 256 hidden units on the Resource Management task. They also made an approximation that the number of well determined parameters was equal to the number of weights for each model (the "cheap" approximation discussed by MacKay [135]). Although they used the evidence framework to set the hyper-parameters of the priors, they still used cross-validation to set the learning rates of the MLPs in a similar scheme to that described in Section 10.2. In their evaluations they found that the use of the Bayesian methods did not improve the performance of the ABBOT system and led to poorer results when the automatic relevance determination (ARD) method [153] was used. For this reason, I do not consider the use of Bayesian methods for acoustic models in this thesis. I also believe that Bayesian methods would require much more computation than cross validation training methods, as was the case for the Bayesian methods of Chapter 7. Given that training times for the RNN and MLPs are already around 3 weeks for the Wall Street Journal corpus (see Section 9.3) on a dedicated parallel computer, the issue of fast computation is clearly paramount.

Decoding

The decoder component of ABBOT is responsible for finding the most probable word sequence $\hat{W}$ given the estimates of phone probabilities from the acoustic model. The decoder used is NOWAY [191]. The search procedure in this decoder is based on stack decoding [169] and is an approximated A^* search. NOWAY combines information from the pronunciation dictionary, the language model and the acoustic model to compute W^*. The pronunciation dictionary contains a database of the legal word pronunciations and the relative frequencies. The language model contains the probabilities of each word in the domain given the previous words. Generally a bigram or trigram language model is used in ABBOT.

The decoding process requires the likelihoods of each phone $P(X|q_i)$ to compute the optimal word sequence. Since the acoustic model gives the posterior probability $P(q_i|X)$ the following approximate scaling procedure is used [199]. The posterior probabilities are divided by the prior probabilities of the phones $\mathrm{P}(q_i)$ to give *scaled likelihoods*:

$$P(X|q_i) \propto \frac{P(q_i|X)}{P(q_i)}. \tag{9.5}$$

9.3 Evaluating Speech Recognition Systems

The majority of speech recognition evaluations are conducted by the US Defence Advanced Research Projects Agency (ARPA)[1]. DARPA invites research sites to take part in yearly benchmark tests with data drawn from a corpus of speech data. The first such corpus was the Resource Management (RM) database [181] which consisted of 1000 words from 109 speakers with a total of 4290 sentences. A slightly larger database is the TIMIT database [124] which consists of 630 speakers and a total of 6300 sentences. The TIMIT database is also phonetically hand labelled and is thus commonly used for comparisons of phone classification accuracy [*e.g.,* 198]. The RM database, on the other hand, is not phonetically labelled, and only a transcript of the words spoken with start and end times is given for each sentence. In order to use the RM database for training or recognition, therefore, a process of *forced alignment* is used. In this process a reference transcript of words is used in conjunction with a pronunciation dictionary to produce a Viterbi alignment of the speech data in terms of phones. The method of forced alignment is also used on all the modern databases which will now be described.

Although the TIMIT and RM databases were used for a number of years and are still used for development of new models, they are restrictive in their domain and speaker sets. For example, the domain for RM is US military terminology and the speaker set is US military personnel. In order to develop speech recognition systems for everyday use a wider domain and speaker set are required. The Wall Street Journal (WSJ) corpus [170] was therefore gathered to achieve these aims. The WSJ corpus was the first general purpose English large vocabulary database and contains recorded data from a large number of north American speakers. The domain is north American business news, specifically sentences from the Wall Street Journal newspaper. The database contains both speaker independent (SI) and speaker dependent (SD) portions. The SI portions are intended for the development of SI speech recognition systems and consist of the following sets:

SI84 This was released as the pilot database, and is also known as WSJ0. SI84 contains 84 speakers, each uttering either 50 or 100 sentences. There are 7240 sentences in total giving 15.3 hours of speech.

SI12 This was released as another pilot database. SI12 is the same size as SI84 but only contains 12 speakers with consequently more sentences per speaker.

[1] DARPA was formerly known as the Advanced Research Projects Agency (ARPA).

SI284 This was released as a full training set corpus and contains both the data of SI84 and an additional 200 speakers. There are 284 speakers in total each uttering between 50 and 150 sentences. There are $36,000$ sentences in total lasting over 80 hours.

SI37 This was another full training set corpus which is the same length in total hours as SI284 but only contains 37 speakers.

A number of tests have been constructed from the WSJ corpus. These are specified according to size of vocabulary in words ($5k$, $20k$ or unlimited), the type of vocabulary (either open or closed) and the type of language model (either bigram, trigram or any). Tables 9.1, 9.2, and 9.3 lists all the DARPA large vocabulary development and evaluation test sets used throughout this work. The constraints (if any) placed on these tests are also summarised in the table. Further information on all these test sets (*i.e.*, data collection practices, speaker information etc.) can be found in [166, 165, 167].

The phone set used in ABBOT for the WSJ corpus is the ICSI-LIMSI 54 phone set. This is shown in Table 9.4.

1993: WSJ tasks; standard bigram & trigram LMs provided						
Test Set Name	**Number of Words**	**Evaluation Constraints**				
		Training Data	**Vocabulary**		**Language Model**	**Adaptation Allowable**
			Size	**Type**		
dt_s5	3343	SI84 or SI12	5k	closed	bigram	none
dt_s6	3318	SI84 or SI12	5k	closed	bigram	none
et_h2_c1	3851	SI84 or SI12	5k	closed	bigram	none
et_h2_p0	3851	any	5k	closed	any	incremental

Table 9.1: **The 1993 DARPA large vocabulary continuous read speech tasks**

1994 H1 North American Business News; standard 20k trigram provided						
Test Set Name	**Number of Words**	**Evaluation Constraints**				
		Training Data	**Vocabulary**		**Language Model**	**Adaptation Allowable**
			Size	**Type**		
et_h1_c0	8186	SI284 or SI37	20k	open	trigram	none
et_h1_p0	8186	any	unlimited	open	any	incremental

Table 9.2: **The 1994 DARPA large vocabulary continuous read speech tasks**

1995 H3 Multiple Unknown Microphone; c0 Sennheiser, p0 unknown microphone						
Test Set Name	**Number of Words**	**Evaluation Constraints**				
		Training Data	**Vocabulary**		**Language Model**	**Adaptation Allowable**
			Size	**Type**		
et_h3_p0	6026	any	unlimited	open	any	block
et_h3_c0	6026	any	unlimited	open	any	block

Table 9.3: **The 1995 DARPA large vocabulary continuous read speech tasks**

Vowels		Semi-vowels		Nasals	
eh	b**e**t	l	**l**at	m	**m**um
ih	b**i**t	el	bott**le**	em	bott**om**
ao	b**ough**t	r	**r**ay	n	**n**oon
ae	b**a**t	w	**w**ay	en	butt**on**
aa	b**o**tt	y	**y**acht	ng	si**ng**
ah	b**u**t	hh	**h**ay		
uw	b**oo**t	hv	a**h**ead		
uh	b**oo**k				
er	b**ir**d	Fricatives		Stops	
ay	b**i**te	s	**s**ea	p	**p**ea
oy	b**oy**	sh	**sh**e	b	**b**ee
ey	b**ai**t	z	**z**one	t	**t**ea
iy	b**ee**t	zh	a**z**ure	d	**d**ay
aw	b**ou**t	th	**th**in	k	**k**ey
ow	b**oa**t	dh	**th**en	g	**g**ay
ax	**a**bout	f	**f**in	dx	mu**dd**y
axr	butt**er**	v	**v**an		
ix	deb**i**t				
		Affricates		Closures	
		ch	**ch**oke	pcl	ha**pp**y
		jh	**j**oke	bcl	ho**bb**it
Silence				tcl	ke**tt**le
h#				dcl	da**dd**y
				kcl	ho**ck**ey
				gcl	ha**gg**le

Table 9.4: **The ICSI-LIMSI phone set.** There are 54 phones in total, separated in the table into vowels, semi-vowels, nasals, stops, fricatives, affricates, closures and silence. Each phone symbol on the left of each column is accompanied by an example of its usage in a word on the right of each column. The location of the phone in each word is signified by the use of bold face.

Scoring and significance tests

The first step in scoring a speech recognition system requires determining an alignment between a hypothetical word string that is recognised and the true reference transcript. This is performed by a Viterbi dynamic programming algorithm. From this alignment the number of substitutions, deletions and insertions with respect to the true transcript can be determined. These are defined as follows:

substitutions : words which are substituted for other words, *e.g., is* for *as*;

deletions : words which are completely deleted from the transcript;

insertions : words which are inserted into the hypothesised transcript that were not present in the true transcript.

The word error rate is proportional to the sum of these quantities:

$$\text{Word error rate (WER)} = \frac{S + D + I}{N} \times 100 \tag{9.6}$$

where N is the number of words in the reference transcription, S the number of substitutions, D the number of deletions and I the number of insertions.

In most cases word error rate is used rather than sentence error rate. Sentence error rates are typically very high, at around 70%, but in most systems little effort is actually made to reduce this, with the emphasis begin on word error rates. The use of more sophisticated sentence level language models or syntactical constraints would likely reduce this sentence error rate considerably.

To test the whether one recogniser has performed significantly better than another, NIST have designed a number of tests. In this section only the most powerful of these tests is used, the *matched pairs sentence segment word error test* (MAPSSWE) suggested by Gillick and Cox [74]. A description of this and other NIST significance tests is given by Martin [139]. In the MAPSSWE the recognisers are tested on the same set of speech. This test set is then divided into a series of segments which are at most one word long and at most one sentence long. Each of the segments are bounded by at least two words correctly recognised by both systems or by the start or end of a sentence. The total number of word errors in both systems is then computed for each segment. Let z_i be the difference between the word errors of the two systems for segment i. The mean and variance of such differences for n segments is given by:

$$\bar{z} = \frac{1}{n}\sum_{i=1}^{n} z_i, \quad \text{and} \quad \bar{\sigma}_z^2 = \frac{1}{n-1}\sum_{i=1}^{n}(z_i - \bar{z})^2. \tag{9.7}$$

The test statistic W may now be written as:

$$W = \frac{\bar{z}}{(\bar{\sigma}_z/\sqrt{n})}. \tag{9.8}$$

We reject the null hypothesis that the two systems are the same if the measured value, w, of W is such that $2 \times Pr(W \geq w) \leq 0.05$.

All the significance tests used in Chapters 10 and 11 were performed using the NIST software package `score` v3.6.2, and the matched pair sentence segment (word error) test[2].

[2]The NIST testing software is available via anonymous ftp from ftp://jaguar.ncsl.nist.gov/pub/score_3.6.2.tar.Z.

9.4 Previous Applications of Mixtures-of-Experts to Speech Recognition

There have been a number of applications of modular architectures, of which the mixture of experts is an example, to speech recognition. One motivation for the use of the mixture of experts in speech recognition is the inherent modularity of speech. Speech is composed of a number of different sub-processes such as speaker dialect region or sex and at lower levels, broad phonetic category. It is generally more difficult for a monolithic architecture to learn the effect and interplay of all these sub-processes. Modular architectures provide an alternative solution to this problem.

One of the first applications of modular networks to speech recognition was Waibel et al. [231]. The Meta-Pi network [77] was a generalisation of these ideas. The Meta-Pi network is similar to the mixtures-of-experts model and consists of a series of time-delay neural networks (TDNN) whose outputs are combined by a gating network. Its training algorithm was very different however, with each expert trained individually on a specific speaker or broad class of phones, and the gating network trained with these broad labels as targets.

Zhao et al. [258] used an HME in combination with a hidden Markov model for continuous speech recognition. The HME was used in a connectionist-HMM hybrid framework in which the HME estimates the probability densities for each of the HMM states. Zhao et al. [258] reported good results on a subset of the Wall Street Journal corpus. Fritsch [60] considered a related connectionist hybrid system (JANUS) in which HME models were used again. In this work however, both gating and expert networks were based on normalised Gaussian radial basis functions. This method was successfully applied to the Switchboard corpus, a notoriously difficult spontaneous speech database [61]. The method has also been extended via a growing algorithm for HME models [62], similar to that described in Chapter 6, and to allow the incorporation of context via "polyphone trees" [63]. Within the ABBOT hybrid system described in this chapter, Hochberg et al. [89] have investigated a variety of combination methods for pre-trained connectionist models in the ABBOT system, including mixtures-of-experts, although they found that simple averaging of outputs often worked as well as more complex schemes.

Speaker identification has also emerged as a popular area of application of modular networks. Bennani [11] used a modular network with TDNNs as experts, whose outputs are combined via a "typology detector". The overall framework is identical to the mixtures-of-experts, although the training algorithm is quite different. The input data is first clustered via k-means into distinct typologies. The data from each typology is used to train each expert network. The typology detector is trained to predict which typology is active. Bennani reported perfect identification on a 102 speaker task. Other work on HME models in speaker identification has also yielded good results [35, 34].

9.5 Outline of Part II

The next two chapters describe two applications of the mixtures-of-experts framework within the ABBOT system. The aim is to demonstrate that the ME framework can be extended to a large, real world problem. The results of both applications indicate that the ME framework can significantly improve the performance of the ABBOT speech recogniser. In Chapter 10 the use of mixtures of MLPs is com-

pared to the use of single MLPs as acoustic models in ABBOT. The single MLPs contained a similar number of parameters as the mixture models. Results on the Wall Street Journal database indicate a significant improvement for the mixture models over the single MLP models. In Chapter 11 mixtures of recurrent networks are used for speaker adaptation within the ABBOT system. The performance of these mixture models is compared with the use of a single recurrent network on an adaptation task from the WSJ database. Once again, significant improvements are obtained for the mixture models over the single models. However, it is noted in Chapter 11 that the mixture models give better improvement for supervised adaptation than for unsupervised adaptation - an effect which is investigated further in that chapter.

Chapter 10

Speech Recognition using Mixtures of Multi-layer Perceptrons

10.1 Introduction

In this chapter I describe the application of the mixtures-of-experts framework to automatic speech recognition. I use the ABBOT system which was introduced in Chapter 9. Conventionally the acoustic model in ABBOT is either a recurrent network [198] or a multi-layer perceptron [17]. In this chapter I extend the MLP model to a mixture of MLPs gated by an MLP. I also describe the use of a boosting procedure to provide good initialisations for the mixture components. Use of the resulting models within ABBOT give word error rates which are significantly better than naive strategies for combining models and a single MLP with an equivalent number of parameters.

This chapter is structured as follows. Section 10.2 introduces the techniques used in acoustic modelling via MLPs. The extension of these techniques to mixtures of MLPs is then discussed in Section 10.3. In Section 10.4 results are presented for the ABBOT system using a mixture of MLPs on the WSJ corpus.

10.2 Multi-Layer Perceptrons in ABBOT

The MLP has been used in a number of speech recognition systems and has proved successful on a number of small to medium sized problems (see Bourlard and Morgan [16] or [17] for a review). Recently Cook [38] has described the use of the MLP as an acoustic model in the ABBOT system as an alternative to the standard RNN. The framework of the MLP used in Cook's work was based on the ICSI QuickNet system of Wawrzynek et al. [239]. QuickNet is a highly optimised MLP training and testing toolkit which is designed for large speech recognition problems. In this chapter I extend the QuickNet system to work with mixtures of MLPs. I describe this extension in the next section. Before this, I discuss some of the more important issues of QuickNet.

In QuickNet, training of the MLP on speech data is performed using the standard back-propagation algorithm. Optimisation is done using gradient descent, either in an on-line fashion or using *partial*

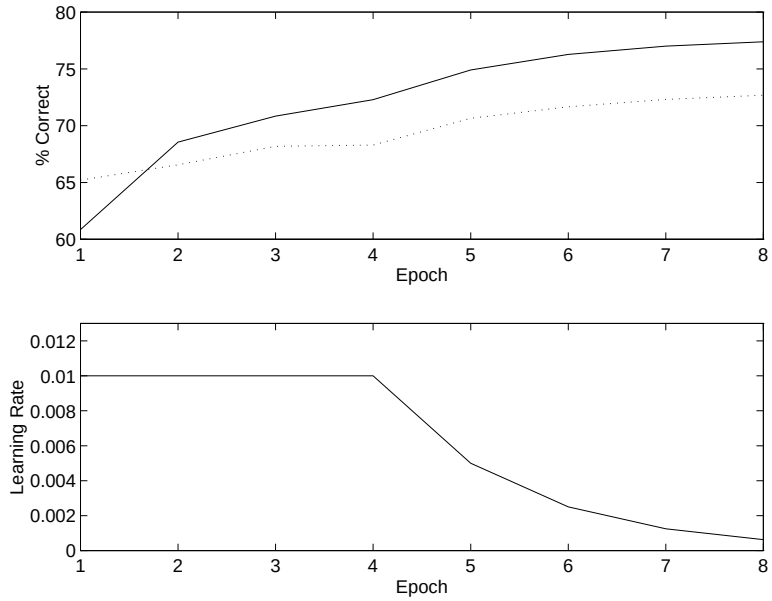

Figure 10.1: **Learning rate adjustment for MLPs**. The top figure shows the evolution of the training (solid line) and validation (dashed line) frame rate accuracy on the Wall Street Journal SI284 database for a MLP with 500 hidden units. The lower figure shows the adjustment of the learning rate for the MLP.

batches. In the on-line scheme updates are made after every acoustic frame has been observed. Partial batch or *bunch* mode training involves making updates to the parameters after a set of frames has been observed. Bunch mode training typically gives faster convergence if up to 200 frames per bunch are used and is also computationally more efficient than the on-line method. In both schemes the frames are presented in a random order. This is to avoid the sequential presentation of frames from the same phone categories or environments which could cause large changes in the parameters to model these frames. Subsequent presentation of a different type of frames could then cause another swing towards these type of frames, resulting in an alternating behaviour of parameters during training. Randomisation smooths this alternating behaviour and leads to more rapid convergence.

In order to avoid over-training of the MLP on speech data, early stopping via cross validation is used. In practice, only one validation set is used, due to the long training times of the MLP on large speech databases. In addition to early stopping, the validation set is also used to control the learning rate of the gradient descent algorithm, as described by Morgan and Bourlard [147]. Specifically, after the error on the validation set begins to increase, the learning rate is reduced by a constant factor on subsequent epochs. This strategy is illustrated in Figure 10.1 which shows the learning rate and evolution of error on the training and validation sets for an MLP on the WSJ SI284 database.

The acoustic preprocessing used for the MLP is as follows. Each input consists of 9 frames of acoustic feature vectors; the frame on which the network is currently performing classification, plus 4 frames of left context and 4 frames of right context. Each acoustic feature vector consists of 12^{th} order PLP plus log energy coefficients computed with an analysis window of $32ms$, every $16ms$ at a sampling

rate of $8kHz$ together with the dynamic delta coefficients of these features.

Speech recognition databases are typically very large. For example the WSJ SI284 database consists of around 15 million frames each containing 12 PLP features and labelled with 54 phones. With such a large number of examples it is imperative that efficient methods are used in the MLPs. In addition to the fast training algorithms already described, considerable effort has been made both at ICSI in the development of QuickNet, and in this work in its extension to mixtures, in making the code efficient. This includes the use of fast matrix and vector operations which use varying degrees of blocking of arrays and unrolling of loops.

10.3 Mixtures of MLPs as Acoustic Models

This section describes the application of an alternative connectionist model to speech recognition as part of the ABBOT system. This model is a mixture of experts, in which the experts are multi-layer perceptrons (MLP) and the gate is also an MLP. The mixture of experts estimates the posterior probabilities $P(q_i|\boldsymbol{x}^{(n)})$ of phonetic classes q_i given the current speech frame $\boldsymbol{x}^{(n)}$. The output of the mixture is obtained through the convex sum of the expert outputs weighted by the gate outputs,

$$P(q_i|\boldsymbol{x}^{(n)}) = \sum_{i=1}^{I} P(\mathcal{E}_i|\boldsymbol{x}^{(n)}, \mathbf{v}) P(q_i|\boldsymbol{x}^{(n)}, \mathcal{E}_i, \mathbf{w}_i) \tag{10.1}$$

where $\mathcal{E}_i$ represents expert i, $\mathbf{w}_i$ the parameters for expert $\mathcal{E}_i$ and $\mathbf{v}$ represents the gate parameters.

The training algorithm for the mixture of MLPs is based on the EM algorithm, as outlined in Section 2.3. Due to the large size of the speech training databases, batch mode training, in which all the data is observed before making an update of the parameters, is impractical. Instead the following training methods are used: on-line, in which an update is made after observing one pattern, or bunch mode, in which an update is made after a group of patterns (typically 100 to 400) have been observed. The on-line and batch versions of the EM algorithm have recently been given a theoretical justification by Neal and Hinton [154]. In the mixture of MLPs the M step consists of a single update of gradient ascent in which the gradient is computed over the bunch of patterns (or over a single pattern in on-line). By using bunch mode training, an additional degree of control over the EM algorithm can be gained. A copy of the parameters of the mixture are held after an update, and are used to compute the posterior probabilities of missing data while the parameters of the mixture are updated via the M step. This allows multiple M steps to be performed inside one E step. In practice, however the use of multiple M steps has little effect on training speed or accuracy of the solutions and I perform one M step per bunch.

In the next section, the results of applying mixtures of MLPs to speech recognition are described. These are also compared with an alternative method for combining models, based on *boosting*. The boosting procedure was developed and implemented by Gary Cook (see [38] for more details) as a fast way of training acoustic models for speech recognition and will now be described in more detail. Boosting generates a set of experts by recursive filtering of the data and whose outputs are averaged in a

committee. In addition to comparing boosting with the mixtures-of-experts, the next section also investigates using the experts resulting from boosting as initialisations for the mixtures-of-experts models. This boosting procedure is based on the work of Drucker et al. [48] but has been generalised to multiple classes by Cook [38]. Cook and Robinson [40] have also adapted the boosting procedure at a word level to the RNN acoustic model ABBOT system and achieved a 20% reduction in word error rate relative to a baseline model with an equivalent number of parameters. In the case of boosted MLPs the algorithm is as follows. The first network is trained on $1/10$ of the training data randomly selected from the available training data. This network is then used to filter the unseen training data to select frames for training the second network. The first and second networks are then used to select data for the third network. Specifically if the first two networks agree on the classification of a frame, the frame is discarded otherwise the frame was added to the third data set. This method is efficient in the amount of training data used and typically uses only $1/3$ of the available training data. Although Drucker et al. [48] suggest the use of a voting procedure for the boosted networks, Cook [38] reports that a simple average over the probabilities of the 3 networks works more effectively.

10.4 Results

In this section I describe results obtained with mixtures of MLPs as acoustic models in the ABBOT system. The acoustic models were trained on the WSJ SI284 corpus and evaluated on four different tests from the WSJ corpus. These were described briefly in Chapter 9 and are as follows:

dt_s5_93
: November 1993 Spoke 5 development test set. This is a 5000 word, closed vocabulary test. The system uses the standard ARPA bigram language model. The results reported here are for the close-talking Sennheiser microphone only.

dt_s6_93
: November 1993 Spoke 6 development test set. This is also a 5000 word, closed vocabulary test, and the system uses the standard ARPA bigram language model.

et_h2_93
: November 1993 Hub 2 evaluation test set. This is also a 5000 word closed vocabulary task. The system uses the standard ARPA trigram language model.

et_h1_93
: November 1993 Hub 1 evaluation test set. This is a large vocabulary task, the prompting texts for which are from the Wall Street Journal 64k word texts pools. The system uses a 20k-word vocabulary, and a trigram language model built using 35 million words of text from the WSJ0 corpus.

et_h1_94
: November 1994 Hub 1 evaluation test set. This is an open-vocabulary task, which includes prompts

from various newspaper sources, including the Wall Street Journal, the Los Angeles Times, the Washington Post, and the New York Times. The system uses a 64k-word vocabulary and a language model built using 237 million words from the CSR-LM-1 corpus [122].

Two different hypotheses were investigated in this evaluation. Firstly, does a mixture of MLPs give better accuracy than a single MLP? Secondly, is it possible to use an intelligent initialisation scheme to improve the performance of the mixture of MLPs? The mixture of MLPs I used consisted of 3 MLPs of the same size, each with 500 hidden units, and an MLP gate with 100 hidden units. As a comparison 2 MLPs were trained, one with 500 hidden units (**500 MLP**) and one with 1500 hidden units (**1500 MLP**). The training of the mixture of MLPs took around 4 weeks on a Sun Ultra workstation with a clock speed of $167MHz$ and required 200 megabytes of RAM. The training times and requirements for the large MLP (1500 hidden units) were similar to the mixtures. Faster training is also available using a dedicate fixed point processor known as SPERT II [239]. Training a 1500 hidden unit MLP on SPERT II takes 171 hours.

I investigated three methods for initialising the mixture of experts, firstly by randomising the parameters of each expert and the parameters of the gate to different values (**Mixed1**). Secondly I used experts which had been initially trained independently on three different regions of the training set selected sequentially, each of size $1/10$ of the total training set. These experts were used in a mixture of experts in which the gate parameters were initialised to random values and the overall mixture retrained on the remainder of the training set (**Mixed2**). Finally I used a set of three MLPs which had been created from the boosting procedure (**Boost**) as initial experts in conjunction with a randomised gate and retrained as before (**Boost+Mixed**).

Test Set	Word Error Rate					
	500 MLP	**1500 MLP**	**Boost**	**Mixed1**	**Mixed2**	**Boost+Mixed**
dt_s5_93	20.4%	17.5 %	16.5%	17.3%	17.2%	14.2%
dt_s6_93	17.7%	14.1 %	14.8%	14.5%	14.4%	11.5%
et_h2_93	25.2%	12.8 %	12.9%	13.1%	12.9 %	10.9%
et_h1_94	24.7%	20.5 %	20.7%	20.5%	20.4 %	16.7%

Table 10.1: **Evaluation of the performance of different acoustic models on the Wall Street Journal database.** 500 **MLP** and 1500 **MLP** are single MLP models with 500 and 1500 hidden units respectively. **Boost** is the set of 3 MLPs resulting from the boosting procedure. **Mixed1** is a mixture of 3 MLPs trained from random initial parameters. **Mixed2** is a mixture of 3 MLPs in which each expert is initialised on $1/10$ of the training data. **Boost+Mixed** is a mixture of 3 MLPs initialised with the 3 MLPs from the **Boost** method. See Section 10.4 for more details of the test sets used.

Table 10.1 shows the results obtained on the 4 test sets using 5 different acoustic models. In addition to the results shown, an additional model was investigated. This model was a mixture of MLPs with the same initialisations as the **Boost+Mixed** model but in which only the gate was retrained on the training set. This model gave no significant improvement over the **Boost** method. A number of additional methods for training this mixture were investigated including a *winner take all* (WTA) procedure in which

the posterior probabilities were converted to hard assignments (*i.e.,* the probability of the winning expert was set to 1 and the probabilities of the losing experts set to 0). However, none of these additional methods gave significant improvements over the methods shown.

The best method overall on all the test sets is **Boost+Mixed**. This significantly outperforms all other methods at the 5% level. The **Mixed1** and **Mixed2** methods are not significantly better or worse than either the **1500 MLP** or **Boost** methods. The **Boost** method is significantly better than the **500 MLP** which is an interesting result since the Boosting method uses only $1/3$ of the training data (but does have 3 times as many parameters in total as the **500 MLP**).

The improved performance of the **Boost+Mixed** method suggests firstly that boosting is a good initialisation method for mixtures of experts models and secondly that retraining a set of boosted models via the mixtures of experts framework can give significantly better performance. One reason for the success of the boosting initialisation is that the resulting models from boosting are specialising in different parts of the data set. The ME training algorithm is then able to increase this specialisation and so end up with a more specialised model than if a flat start is used as in the **Mixed1** or **Mixed2** methods.

10.5 Conclusions

This chapter has described a mixtures-of-experts framework for acoustic modelling in the ABBOT speech recognition system. Both the experts and gates are MLPs in this framework. Results on the Wall Street Journal database show that the ME model is competitive with a single MLP based on a comparable number of parameters. However, it is found that significantly better performance can be obtained by using boosted models as initial experts in the mixture. These results have a number of implications for the ABBOT system. Firstly they suggest that mixtures of smaller MLPs can be used in place of one large MLP. This may be more efficient in practice since many small models can be trained in parallel on different machines and subsequently combined and retrained in a mixture. In addition, the use of prior knowledge may also be a good initialisation rather than the boosting one, *e.g.,* different speakers or groups of speakers could be used to initialise each expert before combining. A possible approach would therefore be to apply clustering to the speech, as done by Cook and Robinson [39], and then train experts on each cluster.

Chapter 11

Speaker Adaptation using Mixtures of Recurrent Neural Networks

11.1 Introduction

In this chapter I describe the application of the mixtures-of-experts framework to speaker adaptation within the ABBOT speech recognition system using a recurrent neural network as the acoustic. Speaker adaptation is performed by learning a linear transform of the input data based on the performance of the acoustic model. I extend the linear input transformation method for speaker adaptation of Neto et al. [155] to a non-linear one. The new transform consists of a mixture of linear transforms gated by a multinomial logit model. This new transform is shown to have desirable properties and gives improved performance at the frame level. Tests on supervised adaptation of speakers also indicate significant improvements over the linear transform.

The chapter is structured as follows. Section 11.3 introduces the recurrent network and describes its architecture and training algorithm. The linear transform method of adaptation is then introduced in Section 11.4. The extension to mixtures of linear transforms is then described in Section 11.5 and results presented.

11.2 Speaker Adaptation

The accuracy of automatic speech recognition systems may be affected by two major factors: external noise in the recording environment and speaker variability. The external noise in the recording environment may be caused for example by differences in the microphones used to record the speech or by extraneous signals such as other speakers or fan noise in the background. There are two major approaches to handling such noise: model compensation [65] and channel adaptation [128]. In this thesis only speaker variability is considered, although channel adaptation can be performed in a similar framework. Speaker variability may be caused by *inter*-speaker differences or by *intra*-speaker differences. Inter-speaker differences are caused by anatomical differences between speakers or by differences in speaking habit caused, for example, by the nationality or dialect of the speaker. Intra-speaker differ-

ences, on the other hand, are caused by factors such as the health and mood of the speaker. Ideally, an ASR system should be robust to both inter and intra speaker variability. One approach to achieving this aim is *speaker adaptation* which will now be described.

ASR systems are either speaker dependent (SD) or speaker independent (SI). For SD systems the training corpus contains data from just one speaker, whereas for an SI system the corpus contains a range of different speakers. As a result an SD system is typically more accurate for its specific speaker than an SI system would be, but the SI system is more flexible. However, it is generally infeasible to collect enough data to train large SD systems reliably for a single speaker from scratch and so some form of mapping of the speech or models is used. In an attempt to bridge the gap between SD and SI systems, two approaches have been used: *speaker normalisation* and *speaker adaptation*. Speaker normalisation attempts to the mimic the supposed process that occurs within the human auditory system. Humans are able to recognise many different speakers in different contexts and environments, suggesting that a process of normalisation of the speech occurs. Speaker normalisation attempts to learn a mapping of the speech for any speaker to a space of normalised speech. The recogniser then is trained on this normalised speech. In spite of the attractiveness of this approach, there is a problem in the amount of variation between speakers such that finding a good general mapping is difficult. In contrast, speaker adaptation attempts to learn a separate mapping for each speaker. The specific mapping is produced by using a sample of speech for the speaker. All further recognition for this speaker then uses this mapping. Speaker adaptation has emerged as the more successful of the two approaches and will be considered further in the remained of this chapter. Speaker normalisation is not considered further here.

There are two main approaches to speaker adaptation: transformation of the speech or transformation of the model parameters. Speaker adaptation can also be performed in several different ways depending on the purpose of the adaptation system. In speaker adaptation, a speaker first provides an adaptation sample of speech. If a reference transcription for this sample is available, the adaptation is known as *supervised*; otherwise, it is known as *unsupervised*. In addition, the adaptation sample may be used in two ways: block adaptation or incremental adaptation. In block adaptation, all the adaptation data is presented to the adaptation system before any adaptation is performed. In incremental adaptation, the adaptation begins to adapt itself as soon as data is presented and subsequently refines its adaptation as more data is seen.

Previous approaches to speaker adaptation can be separated into [127]:

Speaker Clustering : In this approach the speakers are clustered into different groups. The data in these groups are used to train different models. According to the group which a new speaker falls into, the appropriate model is selected for recognition. Although some good results have been shown by Kosaka and Sagayama [121], speaker clustering suffers from similar problems as speaker normalisation, namely that the variability within each cluster may be too great and also that the new speaker may not be represented within the set of clusters.

Spectral Transformations : In this approach the aim is to map the speech for a new speaker to the speech that the recogniser was trained on. Typically a linear transform is used to map the speech.

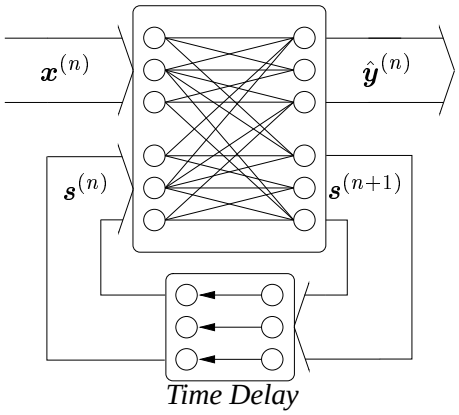

Figure 11.1: **The recurrent network architecture used in ABBOT**. See Section 11.3 for more details.

Spectral transformations have been shown to be successful by Zhao [257]. The linear input network transform of Neto et al. [155] considered in this chapter is an example of this type of approach.

Bayesian Adaptation : In this approach, which was proposed by Gauvian and Lee [66], the parameters of hidden Markov models are adapted using Bayesian inference. In the simplest case the priors are derived from the parameters of the SI system and used in conjunction with the adaptation data to maximise the a-posteriori probability of the models.

Model Parameter Transformations : In this approach the parameters of hidden Markov models are transformed, typically by a linear transform which is learnt from the adaptation data. These are the most popular adaptation approaches in current state of the art systems. The most prominent model parameter transformation technique is the maximum likelihood linear regression (MLLR) method of Leggetter and Woodland [128]. MLLR was used extensively in the state of the art HMM system of Woodland et al. [247].

11.3 Recurrent Neural Networks for Speech Recognition

The recurrent neural network (RNN) is a natural method for modelling time varying signals. A number of different variants of RNNs have been proposed in the neural network literature [*e.g.*, 50, 107]. In the RNN of Robinson [198] which is used in ABBOT, the input consists of two components: the standard input $\boldsymbol{x}^{(n)}$ and a state input $\boldsymbol{s}^{(n)}$. The network consists of fully connected links from these input units to the output units. This is shown schematically in Figure 11.1. The output of the RNN consists of two components: the prediction of the target $\hat{\boldsymbol{y}}^{(n)}$ and the output state vector $\boldsymbol{s}^{(n+1)}$. The output state vector is fed back to the input state units after a unit time delay. The prediction component $\hat{\boldsymbol{y}}^{(n)}$ is the set of probabilities for each of the k phones:

$$\hat{y}_i^{(n)} = P(q_i|\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(n)}, \boldsymbol{\theta}, \boldsymbol{\omega}) \tag{11.1}$$

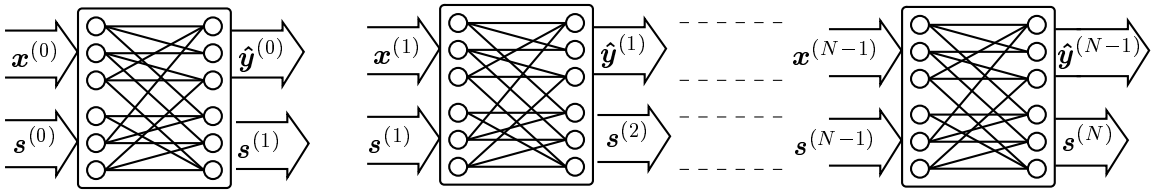

Figure 11.2: **An expanded recurrent network**. The input $\boldsymbol{x}^{(0)}$ and state $\boldsymbol{s}^{(0)}$ at time 0 feed through the network to produce the output $\hat{\boldsymbol{y}}^{(0)}$ and the next state $\boldsymbol{s}^{(1)}$.

where $\boldsymbol{\theta}$ and $\boldsymbol{\omega}$ are matrices of parameters. $\boldsymbol{\theta}$ connects the input and state input units to the output units. $\boldsymbol{\omega}$ connects the input and state input units to the state output units. If we represent the combined input $\boldsymbol{x}^{(n)}$ and $\boldsymbol{s}^{(n)}$ by:

$$\boldsymbol{u}^{(n)} = \left[1\ \boldsymbol{x}^{(n)^T}\boldsymbol{s}^{(n)^T}\right]^T \tag{11.2}$$

we get the following expression for the prediction $\hat{y}_i^{(n)}$:

$$\hat{y}_i^{(n)} = \frac{\exp(\boldsymbol{\theta}_i^T \boldsymbol{u}^{(n)})}{\sum_j \exp(\boldsymbol{\theta}_j^T \boldsymbol{u}^{(n)})} \tag{11.3}$$

where the sum is over all the k classes. The state output $s_j^{(n+1)}$ is given by:

$$s_j^{(n+1)} = \frac{1}{1+\exp(-\boldsymbol{\omega}_j^T \boldsymbol{u}^{(n)})}. \tag{11.4}$$

The RNN used in the standard ABBOT system uses 12^{th} order PLP feature vectors without any delta coefficients or windowing (unlike the MLP of Chapter 10). The RNN contains 256 state units giving a total of $83,700$ parameters.

Training the RNN

The problems of training recurrent networks are similar to those of static multi-layer perceptrons. Although there exist targets $\boldsymbol{y}^{(n)}$ for the outputs $\hat{\boldsymbol{y}}^{(n)}$, there are no targets for the state variables $\boldsymbol{s}^{(n)}$. This is a problem of credit assignment, in that it is not obvious what values of the state variables lead to the desired output $\boldsymbol{y}^{(n)}$. The method used to solve this problem is back-propagation through time (BPTT) [245]. Back-propagation through time (BPTT) involves expanding the recurrent network out over N time intervals so that it resembles an MLP with N layers. This idea is shown schematically in Figure 11.2. Training of the expanded RNN may then be carried out via back-propagation with the additional constraint that the parameters of adjacent hidden layers are tied. Optimisation may then be carried out using any standard gradient based scheme.

The actual optimisation scheme used in the RNN of ABBOT is similar to the RProp algorithm of Schiffmann et al. [211] and delta bar delta algorithm of Jacobs [97] but was developed independently by

Robinson [198]. For a parameter vector $\boldsymbol{\theta}$ the algorithm takes the following form. The objective function E is the cross entropy between the targets and the predictions of the network. For each iteration of the algorithm a local gradient of the objective function is computed, $\partial E^{(t)}/\partial\theta_i^{(t)}$, from the training data in the t^{th} subset of the training data. This local gradient is smoothed using a momentum term:

$$\frac{\partial\tilde{E}^{(t)}}{\partial\theta_i^{(t)}} = \alpha^{(t)}\frac{\partial\tilde{E}^{(t-1)}}{\partial\theta_i^{(t-1)}} + (1-\alpha^{(t)})\frac{\partial E^{(t)}}{\partial\theta_i^{(t)}}. \tag{11.5}$$

where the smoothing parameter, $\alpha^{(t)}$, is increased from $\alpha^{(0)} = 1/2$ to $\alpha^{(\infty)} = 1 - 1/N$ by

$$\alpha^{(t)} = \alpha^{(\infty)} - (\alpha^{(\infty)} - \alpha^{(0)})e^{-t/2T}, \tag{11.6}$$

where T is the number of parameter updates made in each pass through the training data.

Given this smoothed local gradient, a step size $\Delta\theta_i^{(t)}$, is computed for every parameter. Each parameter is updated by its step size in the direction of the smoothed local gradient,

$$\theta_i^{(t+1)} = \begin{cases} \theta_i^{(t)} + \Delta\theta_i^{(t)} & \text{if } \frac{\partial E^{(t)}}{\partial\theta_i^{(t)}} > 0, \\ \theta_i^{(t)} - \Delta\theta_i^{(t)} & \text{otherwise.} \end{cases} \tag{11.7}$$

If the direction of the local gradient is in agreement with the smoothed gradient, the step size is geometrically increased by a factor of ϕ otherwise it is geometrically decreased by a factor of $1/\phi$,

$$\Delta\theta_i^{(t+1)} = \begin{cases} \phi\Delta\theta_i^{(t)} & \text{if } \frac{\partial\tilde{E}^{(t-1)}}{\partial\theta_i^{(t-1)}}\frac{\partial E^{(t)}}{\partial\theta_i^{(t)}} > 0 \\ \frac{1}{\phi}\Delta\theta_i^{(t)} & \text{otherwise.} \end{cases} \tag{11.8}$$

The factor ϕ is set to $1/0.9$ which ensures that random gradients produce little overall change to the search. The size of each block of data that is observed before an update is made is 2880 frames.

11.4 Speaker Adaptation within ABBOT

Speaker adaptation is the process of adapting a speaker independent (SI) recogniser to a new speaker. Another use of adaptation is to adapt a recogniser to new environments such as a new microphone or more noisy recording environment. This is known as *channel* adaptation. In order to perform speaker and channel adaptation, ABBOT uses the linear input network (LIN) adaptation technique of [155]. The LIN makes a linear transform of the acoustic vector which is then fed to the standard acoustic model, as shown in Figure 11.3. The aim is to learn a transform without retraining the original acoustic model. To train the LIN for a new speaker, the LIN parameters are initialised to an identity matrix; this guarantees that the initial starting point is the speaker independent model. The input is propagated forward to the output of the connectionist model. At this point the error is back-propagated through the connectionist model keeping the RNN parameters fixed, and only updating the LIN's parameters.

Neto et al. [155] compared a number of methods for adapting connectionist models (both MLPs and RNNs) and showed that the LIN method has two benefits. Firstly the LIN transform can be learnt more

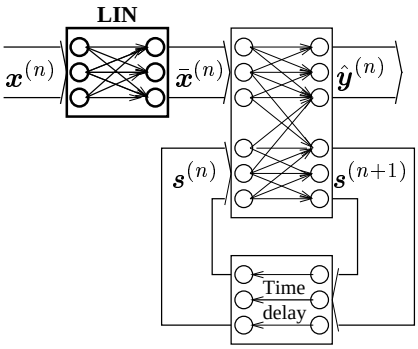

Figure 11.3: **Speaker adaptation via the linear input network (LIN)**. The original acoustic vector $\boldsymbol{x}^{(n)}$ is transformed by the LIN to the vector $\bar{\boldsymbol{x}}^{(n)}$ which is used as input to the RNN.

quickly than retraining the original acoustic model and secondly is actually more accurate on average. The LIN is relatively efficient in parameters using $(d+1)^2$ parameters to transform a d dimensional acoustic vector. On a supervised adaptation task with 100 adaptation sentences from the RM database, Neto et al. [155] demonstrated an average reduction of 35% in word error rate over a baseline MLP acoustic model and 24% over a baseline RNN acoustic model.

In this chapter I consider the use of the LIN for speaker adaptation of the RNN acoustic model in ABBOT. This builds on the work of Kershaw et al. [118] who used the LIN successfully in the 1995 ARPA evaluations for channel adaptation to unknown microphone and noise environments.

11.5 Mixtures of Local Linear Transforms

Although the LIN is effective at adapting to new speakers or a new environment (*e.g.*, a different microphone), the transform is global over the input space. In this section I describe a method in which the transform is *locally* linear over different regions of the input space. The local linear transforms are combined by an additional network using a non-linear transform. This scheme falls naturally into the mixtures-of-experts framework. I call this method the *mixture of linear input networks* or MLIN.

One motivation for the use of the MLIN is the success of local adaptation techniques in hidden Markov models [128]. In HMMs the local regions are selected according to sets of similar phone classes which are clustered using a set of phonetic rules. The selection of local regions based on similar phone classes is not possible with the recurrent network, because the RNN is a dynamical system requiring a continuous input stream.

Figure 11.4 shows the architecture of this mixture of linear input networks (MLIN). The gate consists of a single layer network with softmax activation function. Each expert consists of a linear input network followed by a recurrent network. The recurrent network parameters are shared across all experts and are not adapted during training. Each LIN is distinct however, so that individual linear transforms can be performed by each of the experts. The gate combines the outputs of each of the experts (after the recurrent network portion) which are class conditional probability estimates. The overall output is given

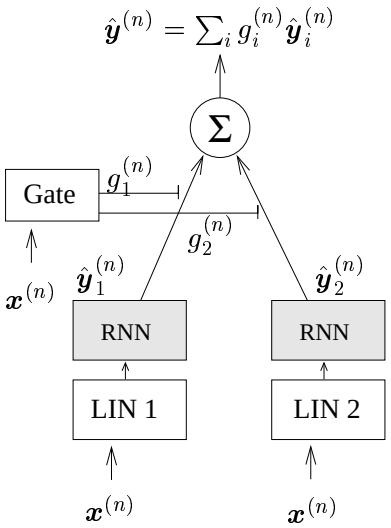

Figure 11.4: **A mixture of 2 linear input networks (MLIN) for adapting the recurrent network**

by:

$$\hat{\boldsymbol{y}}^{(n)} = \sum_i g_i^{(n)} \hat{\boldsymbol{y}}_i^{(n)} \tag{11.9}$$

where $\hat{\boldsymbol{y}}_i^{(n)}$ is the output of each expert and $g_i^{(n)}$ is the i^{th} output of the gate. If the output of the RNN is represented as a function of the input, $f(\boldsymbol{x}^{(n)})$, this can be rewritten as:

$$\hat{\boldsymbol{y}}^{(n)} = \sum_i g_i^{(n)} f(\mathbf{w}_i^T \boldsymbol{x}^{(n)}) \tag{11.10}$$

where $\mathbf{w}_i$ is the parameter matrix of expert i.

Before training, the parameters of the LINs in each expert are initialised to diagonal matrices with random parameters between 0.95 and 1.05. The gate parameter matrix is initialised using random values between -0.5 and 0.5. This randomisation ensures symmetry breaking and prevents the experts all learning the same transform.

During training of the MLIN the current input $\boldsymbol{x}^{(n)}$ is propagated forward through each LIN and RNN expert to give an error term at the output of each expert $e_{ic} = y_c - \hat{y}_{ic}$. This term is then weighted by the posterior probability of each expert:

$$\varphi_i^{(n)} = \frac{g_i^{(n)} \exp\left(\sum_{c=1}^k y_c^{(n)} \log \hat{y}_{ic}^{(n)}\right)}{\sum_j g_j^{(n)} \exp\left(\sum_{c=1}^k y_c^{(n)} \log \hat{y}_{jc}^{(n)}\right)} \tag{11.11}$$

where $y_c^{(n)}$ is the target for class c at time n and $\hat{y}_{ic}^{(n)}$ is the conditional probability of class c given by expert i. This gives an error $\varphi_i^{(n)} e_i$ which is then back-propagated through the RNN using the scheme described in Section 11.3.

The gate error terms are based on the difference between the posterior probability of each expert $\varphi_i^{(n)}$ and the gate's estimate of this probability $g_i^{(n)}$:

$$\frac{\partial E}{\partial \mathbf{v}_i} = \sum_n (\varphi_i^{(n)} - g_i^{(n)}) \boldsymbol{x}^{(n)}. \tag{11.12}$$

The expert and gate parameters are optimised using these gradients and the optimisation algorithm described in Section 11.3.

Note that the MLIN does not increase the number of parameters significantly, since the RNN is effectively tied across the different experts. The computational cost of training is increased slightly however, due to the need to back-propagate the error terms through each copy of the RNN.

11.6 Results

In this section I describe results of evaluations using the LIN and MLIN adaptation models on the ARPA 1995 H3 multiple unknown microphones (MUM) Task. I chose the et_h3_c0 subset of the H3 task described in Table 9.3 of Chapter 9. This is the 1995 H3 MUM unlimited vocabulary test recorded using a Sennheiser microphone. I considered two tasks: supervised adaptation using enrolment sentences, and unsupervised adaptation.

For supervised adaptation 10 speakers were chosen from the H3 database which had 40 enrolment sentences each. For each speaker the recogniser is adapted to the enrolment sentences using the transcriptions of these sentences as targets. After adaptation the recogniser was then tested on 15 unseen sentences for each speaker.

For the unsupervised adaptation task 20 speakers were chosen from the H3 database, with 15 sentences per speaker. The task in unsupervised adaptation is to improve the accuracy of the recogniser for new speakers *without* seeing the transcription of the speech. Although this is called unsupervised adaptation, it is similar to supervised adaptation with the difference that the reference transcript is replaced by a transcript guessed by the recogniser. In order to get labels for each frame in the adaptation sentences a Viterbi alignment is performed initially using the untrained MLIN, whose performance is similar to the baseline recogniser. After adapting the MLIN model to this set of labels a new alignment is performed using the MLIN model. The MLIN model is then adapted to this new alignment. This process is iterated until there is no change in the alignment hypotheses from one adaptation pass to the next. This typically converges in 2 to 3 iterations. this process is known as unsupervised *block* adaptation since the whole block of data is seen before adaptation is performed. This is distinct from *on-line* adaptation in which adaptation must be done as soon as data arrives.

Supervised adaptation

For each speaker, the recogniser was adapted to the enrolment sentences using 30 sentences for adaptation and 10 for cross validation. Early stopping (based on the frame accuracy) on the validation set was used to control the learning of the system in the following way. After each pass through the adaptation

Model	# Sub^n.	Del^n.	Ins^n.	WER
RNN	12.2 %	3.3 %	2.4 %	17.8 %
LIN	11.4 %	3.0 %	2.1 %	16.6 %
MLIN_2	10.7 %	2.9 %	2.1 %	15.7 %
MLIN_3	10.5 %	2.8 %	2.0 %	15.3 %
MLIN_4	10.6 %	2.9 %	1.8 %	15.3 %
MLIN_5	10.2 %	2.7 %	1.9 %	14.9 %

Table 11.1: **Word error rates of different acoustic models for supervised adaptation** The table shows the decoding performance on the supervised adaptation task using a single LIN, and MLINs with $2, 3, 4$ and 5 experts. The improvements of each model are significant over the simpler models at a 5% level using the NIST scoring software (see Section 9.3 for details) except for MLIN_4 and MLIN_3 whose results are not significantly different.

sentences the frame accuracy of the recogniser on the validation sentences was evaluated. A minimum of 7 passes through the data were made and the best accuracy chosen out of these 7 models. If the best model was found on the 7^{th} pass, a further 7 passes were made until the error on the validation set reached a minimum. Typically the best model was found after 5 passes.

The results of the baseline system (RNN), single linear input network (LIN) and mixtures of 2 to 5 experts (MLIN_2 to MLIN_5) are shown in Table 11.1. These results suggest that the MLIN method gives a significant improvement over the simple LIN adaptation method. The largest model considered here, MLIN_5 gives a relative improvement over the LIN method of 10%.

Unsupervised Adaptation

In unsupervised adaptation the MLIN models were adapted to the hypothesised transcriptions obtained from the baseline model. In the first instance this was the RNN model. After subsequent adaptation passes, the resulting adapted models were used to align the speech. These alignments were then used as hypothesised transcriptions in the next adaptation pass. In each adaptation pass the models were trained using the same cross validation scheme described for the supervised adaptation task, with 10 sentences chosen for adaptation and 5 for validation.

Table 11.2 shows the error rates using different models and different numbers of adaptation passes. Whilst the MLIN models give an overall decrease in word error rate (MLIN_5 gives a 6.7 % relative improvement over LIN), the improvement is not as uniform as for the supervised adaptation. In addition, in some cases, *e.g.*, MLIN_2 and LIN after 2 passes of adaptation, the performance degrades as the number of parameters in the models is increased. In order to investigate this behaviour in more detail I examined the phone error rates at a frame level. These are relative to the hypothesised transcription in each case and are shown in Table 11.3. As this table shows the more complex models have lower frame error rates *relative* to the hypothesised transcription. As the word error rates suggest, however, this hypothesised transcription is sometimes in error.

This effect can be likened to a game of Chinese whispers. If the final person in the chain listens

Model	# Passes	Sub^{n}.	Del^{n}.	Ins^{n}.	WER
RNN	0	13.1 %	2.9 %	2.1 %	18.1 %
LIN	1	11.7 %	2.7 %	2.1 %	16.5 %
LIN	2	11.2 %	2.6 %	2.1 %	15.9 %
LIN	3	11.1 %	2.6 %	2.1 %	15.8 %
MLIN_2	1	11.7 %	2.7 %	2.1 %	16.5 %
MLIN_2	2	11.4 %	2.6 %	2.1 %	16.1 %
MLIN_2	3	11.3 %	2.6 %	2.1 %	16.0 %
MLIN_4	1	11.2 %	2.6 %	2.1 %	15.9 %
MLIN_4	2	11.1 %	2.6 %	2.1 %	15.8 %
MLIN_4	3	10.9 %	2.6 %	2.1 %	15.6 %
MLIN_5	1	11.0 %	2.5 %	2.1 %	15.6 %
MLIN_5	2	10.9 %	2.5 %	2.1 %	15.5 %
MLIN_5	3	10.8 %	2.5 %	2.1 %	15.4 %

Table 11.2: **Word error rates of different acoustic models for unsupervised adaptation**. The table shows the decoding performance on the H3 task using a single LIN, MLIN with between 2 and 5 experts (MLIN_2 - MLIN_5). The number of passes of adaptation for each model is shown, along with the percentage of substitutions (Sub^{n}.), deletions (Del^{n}), insertions (Ins^{n}) and overall word error rate (WER). Only the results for 3 passes are shown since there are no further significant improvements for any of the models after this number.

Model	# Passes	Frame Error Rate
RNN	0	22.8 %
LIN	3	19.9 %
MLIN_2	3	18.9 %
MLIN_4	3	18.0 %
MLIN_5	3	17.8 %

Table 11.3: **Frame error rates of different acoustic models for unsupervised adaptation**. The table shows the phone error rate at the frame level relative to the hypothesised transcriptions on the H3 task using a single LIN model and MLIN models with 2 and 4 experts (MLIN_2, MLIN_4).

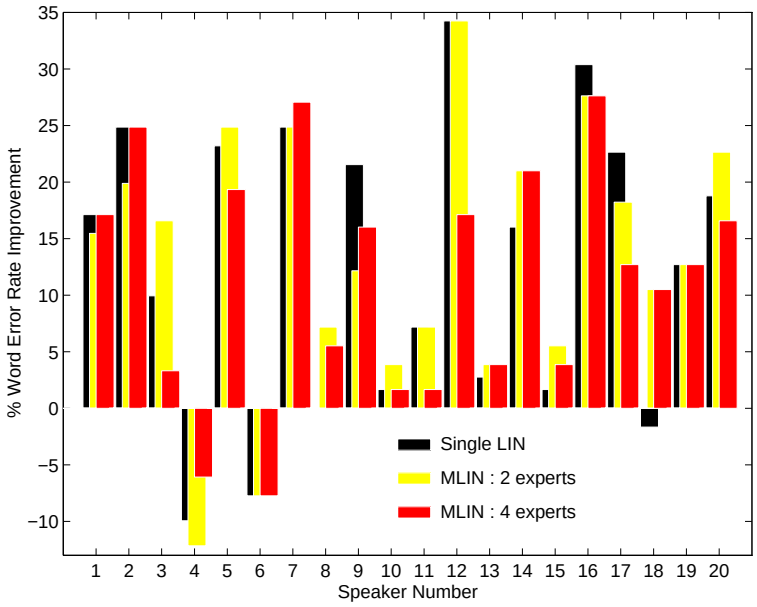

Figure 11.5: **Change in word error rate per speaker for unsupervised learning**. The figure shows the improvement of word error rate per speaker using a single LIN, a mixture of 2 LINs, and a mixture of 4 LINs, after a third pass of adaptation.

harder then the accuracy of their guess may be improved, but this may still be the wrong guess that they are learning. By improving the transform used in adaptation, significant improvements in error rate for some speakers are achieved at the expense of degradations in others. The MLIN makes a better job of transforming the input for both the correctly and incorrectly transcribed frames than the LIN does. This effect is demonstrated in Figure 11.5 which shows the improvement or degradation per speaker after different adaptation schemes. The speakers for which adaptation degraded word error rates correspond to those for which the speaker independent system (RNN) gave very crude initial alignments.

11.7 Conclusions

This chapter has shown that the mixtures-of-experts framework can be effectively incorporated into the ABBOT system using recurrent networks. I have described how the linear input network adaptation scheme can be augmented using mixtures of linear input networks and recurrent networks (MLIN). I have demonstrated significant improvements on both supervised and unsupervised adaptation tasks from the WSJ database. These results suggest that the MLIN is an effective method for performing adaptation in connectionist HMM hybrids. The results must be taken with caution, however, in the case of unsupervised adaptation since it seems that more complex models such as the MLIN can cause degradation in the per speaker error rates.

There are a number of possible future directions for this work. Instead of performing speaker adaptation with the MLIN, channel adaptation is also possible. Prior knowledge could then be used to initialise

each LIN to a particular speaker before training the MLIN. This could enable faster training and a more effective adaptation overall. Alternatively, the mixtures of RNNs could be used for recognition alone rather than adaptation. A more attractive architecture could be a mixture of multinomial logit experts gated by a recurrent network which is a further generalisation of the work presented here. This structure is then akin to the IOHMM of Bengio and Frasconi [10] and the HMDT of Jordan et al. [110].

Part III

Conclusions, Appendices and References

Chapter 12

Conclusions and Future Work

This thesis has described the mixtures-of-experts framework and investigated its application to small, medium and large real world problems. A mixtures-of-experts model using generalised linear models as experts and gates was described in Part I of the thesis. Two extensions of this model were proposed - one based on a constructive algorithm and one based on Bayesian methods. These extensions were compared empirically with the standard model and with other statistical models on data sets from the DELVE archive. The results of these comparisons indicate that whilst mixtures-of-experts are not always the *best* model to choose for a task, they are *competitive* with other models on regression tasks, and amongst the best studied on classification tasks. In Part II a larger scale task was investigated - automatic speech recognition using the mixtures-of-experts as an acoustic model within the ABBOT system. The mixtures-of-experts framework was used to motivate two new acoustic models based on mixtures of multi-layer perceptrons and mixtures of recurrent neural networks. These models were compared empirically with existing acoustic models on speech recognition and speaker adaptation tasks, and shown to give significant improvements. These studies indicate first that the mixtures-of-experts framework can be applied successfully to large, real world problems such as speech recognition, and second, that significant performance improvements can be realised. Overall the thesis has presented mixtures-of-experts in context with other statistical models and shown empirically that they can be applied successfully to problems in classification and regression.

Future Work

A number of possibilities for future work into mixtures-of-experts have been described in each chapter of this thesis. These, and other suggestions are described below.

Empirical comparisons

The results presented in Part I of the thesis on the DELVE archive give a number of insights into the performance of mixtures-of-experts models in comparison with other models. Further research in this area should focus on the extension of these comparisons to more models and more data sets. In particular, more comparisons on medium to large real world problems would allow

us to discover the types of tasks that mixtures-of-experts models will, in general, perform well on. Many of the applications of mixtures-of-experts have so far been on time series prediction or speech recognition, as discussed in Chapter 2 and Chapter 9. Comparisons on these types of data sets would therefore be potentially enlightening.

Constructive models

The constructive algorithm for hierarchical mixtures-of-experts described in Chapter 6 was shown to be successful on regression and classification tasks in Chapter 8. Future work in this direction should focus on further connections with existing tree growing literature. In particular, there should be further investigation into making pruning successful in these algorithms. In addition, the incorporation of Bayesian methods might allow some indication of the usefulness of a particular node in the model, as was done by Jacobs et al. [105].

Bayesian methods

The Bayesian methods described in Chapter 7 were motivated by an empirical Bayesian perspective, which uses approximations to the posterior distributions in the model. An alternative approach would be to use Monte Carlo techniques to sample from these posterior distributions. Since the mixtures-of-experts has an interpretation as a belief network, the framework of Thomas et al. [222] could be used to perform Gibbs sampling, in a similar way to that done by Peng et al. [172].

Speech Recognition

The results presented in Part II of the thesis indicate that the mixtures-of-experts framework is an effective tool for acoustic modelling in connectionist speech recognition. The approach taken in this thesis was to use mixtures of existing, previously successful, networks. Alternatively, much larger mixtures of small networks could be used, for example mixtures of generalised linear models such as those described in Part I. Other researchers have used a similar approach, as described in Chapter 9, including Zhao et al. [258] and Fritsch [60]. Potentially the most effective acoustic model, given the effectiveness of the recurrent networks used in ABBOT, might be a recurrent mixtures-of-experts model such as the IOHMM of Bengio and Frasconi [10]. Given the results presented in this thesis, there is now clear scope to extend the recurrent network of ABBOT in this direction.

Alternative models

In Chapter 2 some of the connections between mixtures-of-experts and other statistical models were described. I believe further exploration of these connections would be of benefit in designing alternative models based on the mixtures-of-experts framework. For example, one possible line of research is to consider reducing the number of input attributes in the experts and gates in the spirit of a multivariate decision tree [26]. Such models would be more interpretable, and potentially more efficient in parameters and training times. An alternative approach would be to use *different* types of models for each expert, *e.g.*, using multi-layer perceptrons and MARS models as experts in the same mixture. The mixture of such models could be successful when neither model alone is successful, or, as is often the case, when the best model for a task is unknown. This approach

is reminiscent of Brodley's model class selection system [24], with the difference that Brodley used simple models such as logistic regression and k nearest neighbours as leaves in a decision tree. Potentially the smoothing of the boundaries between models, as performed in the mixtures-of-experts, might provide an added advantage over Brodley's approach.

Appendix A

Availability of Source Code and Results

To generate the results in Part I of this thesis I implemented five learning methods based on mixtures-of-experts. In order to make future work possible on these methods I have released the source code for these methods. The code is written in C++ and contains additional documentation with its distribution.

The source code the five mixtures-of-experts based learning methods described in Part I of the thesis may be obtained from http://www.cs.toronto.edu/∼delve. This site also contains code for the other learning methods investigated by Rasmussen [189]. In addition the guess and loss files for all the methods on the DELVE datasets described in Part I may also be obtained from this site.

Both David MacKay's original C code and my adapted C++ version of `macopt` are available from http://wol.ra.phy.cam.ac.uk/mackay/c/macopt.html. The C++ version is also available as part of the mixtures-of-experts family of methods source code at http://www.cs.toronto.edu/∼delve/methods/meese-1/.

Appendix B

Results on DELVE Regression Tasks

This appendix contains the results of applying all the learning methods described in Part I of the thesis to the DELVE regression tasks. There are 17 data sets in total: the Boston data set plus 8 data sets in each of the Kin and Pumadyn families. For brevity, detailed legends are not included for each of the tables and figures and only a short caption is used. Each figure and table shows the standardised squared error loss for each learning method for different sized training sets. For example, the results on the Boston data set are shown in Figure B.1 and Table B.1. Each column of the figure and table shows the squared error loss for the different learning methods for training sets with different numbers of examples. The key to each column in the figures is given at the bottom left of the figure. Reading from top to bottom of the list of methods corresponds to the bars from left to right of each column. In the table the losses are shown in numeric form. The results for the best performing method, *i.e.*, the one with smallest loss, is shown in bold face. In the figure the losses are shown as horizontal lines with the associated standard error shown as bars on either side of the loss. Note that for the Kin and Pumadyn families, the method `gp-mc-1` was not run on the largest data sets of 1024 points. In these cases the results for `gp-mc-1` are marked as with the symbol $\emptyset$ in the tables and figures. Also, for brevity, only the results for the 7 original methods and the 5 ME family methods are shown in the figures. The excluded methods are `lin-2` and `lin-ese-1`, whose results are shown instead in the tables.

Each figure also includes a matrix of p-values for each of the methods. These are the p-values obtained from paired t-tests or F-tests between each of the methods. The minimum p-value shown is 1 and the maximum is 5. A result is considered significant if the p value is 5 or less. Non-significant comparisons are shown with a dot.

B.1 Boston

	method	32	64	128
Boston/price	gp-map-1	0.407	0.280	0.194
	knn-cv-1	0.522	0.425	0.344
	lin-1	0.528	0.327	0.281
	mars3.6-bag-1	0.325	0.494	**0.157**
	mlp-ese-1	0.407	0.258	0.210
	me-ese-1	0.366	0.234	0.160
	hme-ese-1	0.386	0.229	0.173
	me-el-1	0.370	0.243	0.159
	hme-el-1	**0.322**	**0.210**	0.162
	hme-grow-1	**0.322**	0.233	0.176
	lin-2	0.525	0.326	0.281
	lin-ese-1	0.446	0.312	0.285

Table B.1: **Results on the Boston data set**

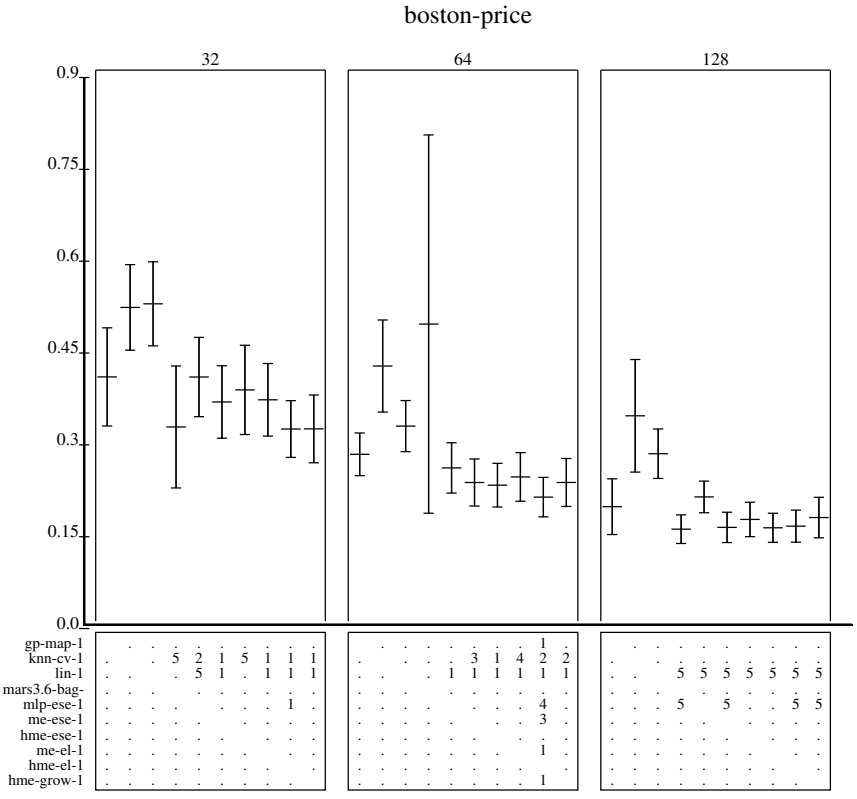

Figure B.1: **Results on the Boston data set**

B.2 The Kin and Pumadyn families

Kin-32fm/dist

	Method	Data set size				
		64	128	256	512	1024
Kin-32fm/dist	gp-mc-1	**0.172**	**0.122**	0.106	0.094	∅
	gp-map-1	0.202	0.144	0.113	0.091	0.081
	knn-cv-1	0.705	0.618	0.544	0.496	0.440
	lin-1	0.182	0.124	0.107	0.101	0.097
	mars3.6-bag-1	0.765	0.517	0.364	0.284	0.270
	mlp-ese-1	0.187	0.124	**0.105**	0.092	0.076
	mlp-mc-1	0.181	0.124	0.106	**0.080**	**0.071**
	me-ese-1	0.318	0.183	0.124	0.111	0.092
	hme-ese-1	0.330	0.187	0.125	0.114	0.093
	me-el-1	0.292	0.205	0.145	0.118	0.099
	hme-el-1	0.272	0.181	0.136	0.107	0.090
	hme-grow-1	0.181	0.125	0.107	0.099	0.080
	lin-2	0.182	0.124	0.107	0.101	0.097
	lin-ese-1	0.186	0.127	0.107	0.102	0.098

Table B.2: **Results on the Kin-32fm data set.**

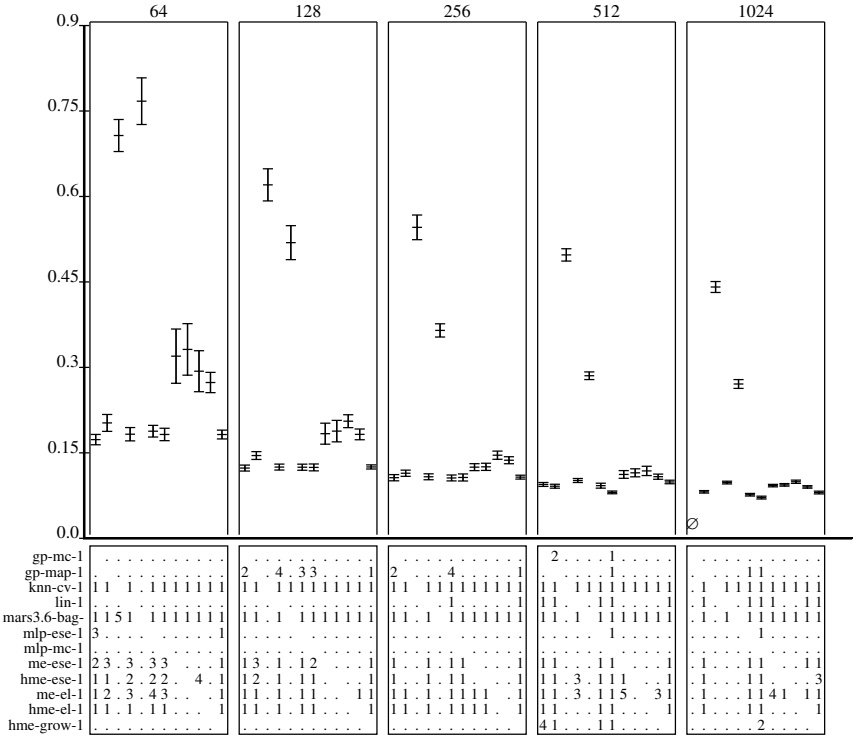

Figure B.2: **Results on the Kin-32fm data set**

Kin-32fh/dist

	Method	Data set size				
		64	128	256	512	1024
Kin-32fh/dist	gp-mc-1	**0.584**	0.389	0.349	0.325	∅
	gp-map-1	0.656	0.439	0.350	0.335	0.310
	knn-cv-1	0.835	0.730	0.682	0.637	0.594
	lin-1	0.603	**0.383**	**0.341**	**0.319**	0.307
	mars3.6-bag-1	0.851	0.716	0.565	0.498	0.458
	mlp-ese-1	0.643	0.386	0.343	**0.319**	0.306
	mlp-mc-1	0.649	0.391	**0.341**	0.320	**0.305**
	me-ese-1	0.637	0.444	0.390	0.360	0.325
	hme-ese-1	0.643	0.444	0.389	0.364	0.325
	me-el-1	0.666	0.504	0.445	0.403	0.365
	hme-el-1	0.627	0.520	0.401	0.376	0.337
	hme-grow-1	0.602	0.386	0.345	0.322	0.309
	lin-2	0.604	0.384	**0.341**	**0.319**	0.307
	lin-ese-1	0.599	0.388	0.346	0.322	0.309

Table B.3: **Results on the Kin-32fh data set.**

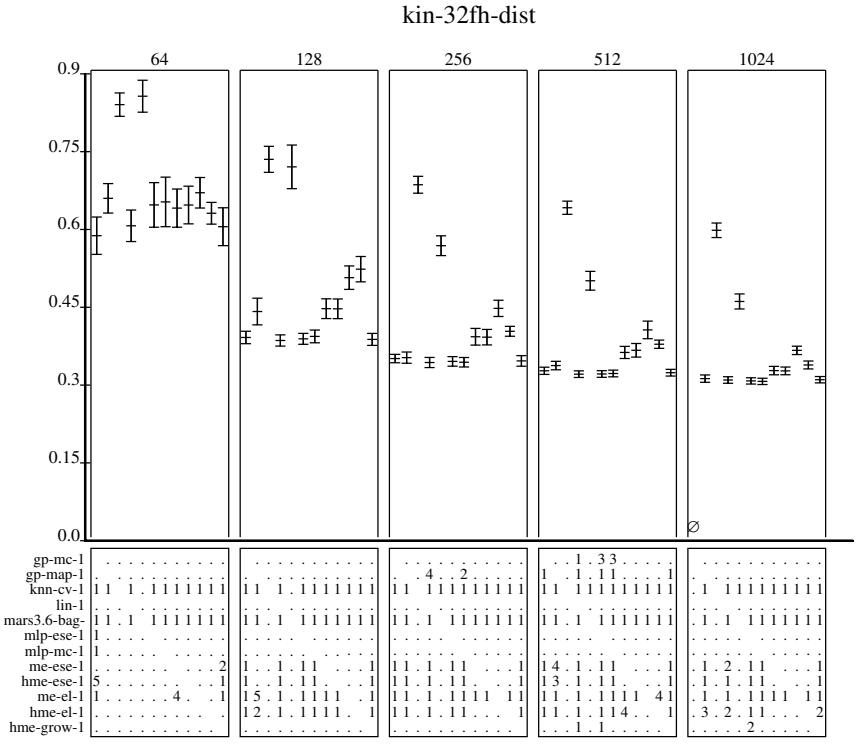

Figure B.3: **Results on the Kin-32fh data set**

Kin-32nm/dist

	Method	Data set size				
		64	128	256	512	1024
Kin-32nm/dist	gp-mc-1	**0.930**	**0.867**	**0.803**	0.677	$\emptyset$
	gp-map-1	1.054	0.941	0.831	0.683	0.610
	knn-cv-1	0.965	0.926	0.910	0.843	0.811
	lin-1	1.446	1.001	0.847	0.788	0.765
	mars3.6-bag-1	0.985	0.922	0.873	0.826	0.804
	mlp-ese-1	0.936	0.886	0.813	0.778	0.729
	mlp-mc-1	0.941	0.897	0.823	**0.675**	**0.582**
	me-ese-1	1.031	0.915	0.822	0.767	0.708
	hme-ese-1	1.029	0.923	0.822	0.771	0.722
	me-el-1	0.969	0.958	0.843	0.780	0.728
	hme-el-1	1.115	0.919	0.846	0.768	0.723
	hme-grow-1	1.062	0.947	0.836	0.779	0.719
	lin-2	1.444	1.001	0.847	0.788	0.765
	lin-ese-1	1.064	0.947	0.835	0.791	0.766

Table B.4: **Results on the Kin-32nm data set**

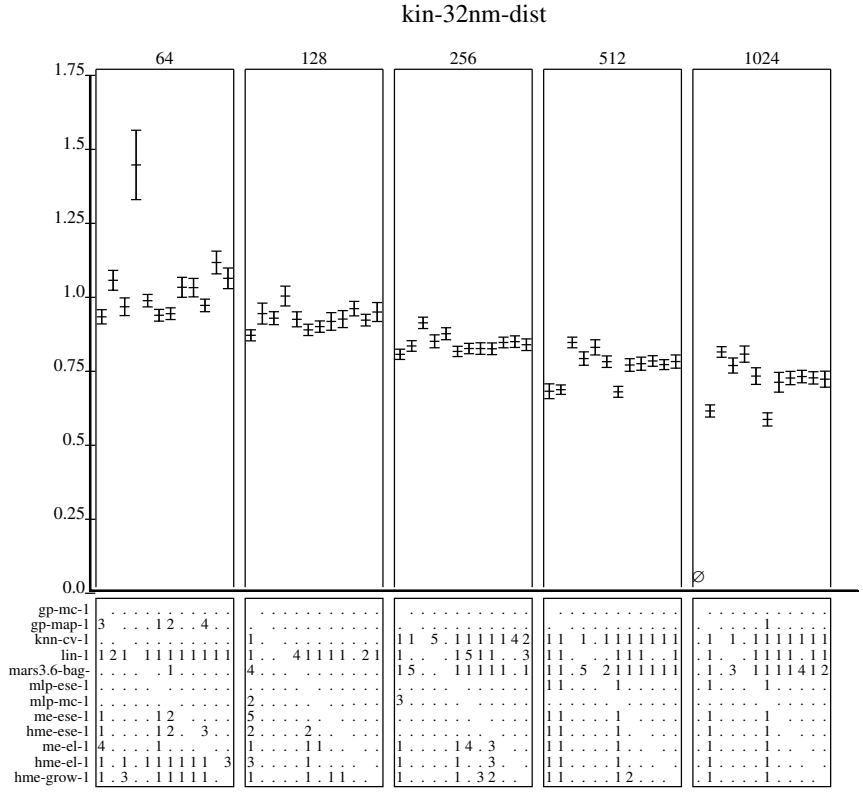

Figure B.4: **Results on the Kin-32nm data set**

Kin-32nh/dist

	Method	Data set size				
		64	128	256	512	1024
Kin-32nh/dist	gp-mc-1	0.972	**0.942**	**0.862**	0.802	∅
	gp-map-1	1.018	1.038	0.890	**0.796**	0.707
	knn-cv-1	1.009	0.971	0.937	0.914	0.866
	lin-1	1.806	1.079	0.898	0.850	0.815
	mars3.6-bag-1	1.085	0.979	0.908	0.880	0.844
	mlp-ese-1	**0.960**	0.944	0.868	0.844	0.796
	mlp-mc-1	0.969	0.951	0.876	0.824	**0.694**
	me-ese-1	1.183	1.002	0.887	0.839	0.783
	hme-ese-1	1.183	1.004	0.889	0.848	0.783
	me-el-1	1.078	1.093	0.913	0.856	0.805
	hme-el-1	1.344	1.003	0.917	0.835	0.793
	hme-grow-1	1.241	1.026	0.892	0.849	0.791
	lin-2	1.804	1.079	0.898	0.850	0.815
	lin-ese-1	1.24	1.028	0.891	0.852	0.819

Table B.5: **Results on the Kin-32nh data set**

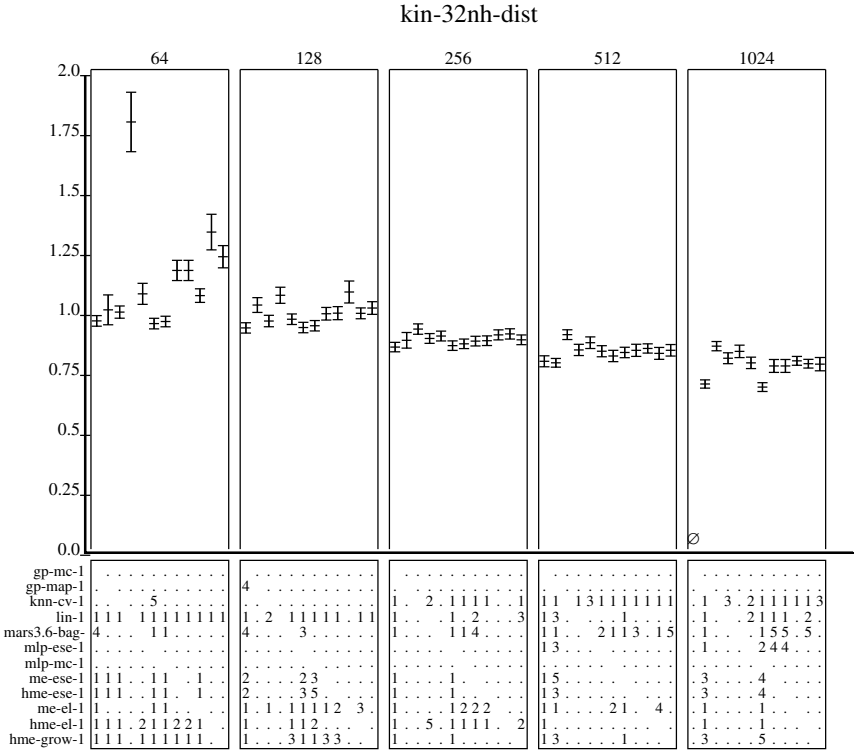

Figure B.5: **Results on the Kin-32nh data set**

Kin-8fm/dist

	Method	Data set size				
		64	128	256	512	1024
Kin-8fm/dist	gp-mc-1	0.061	0.043	0.033	**0.028**	∅
	gp-map-1	0.067	0.044	0.033	**0.028**	**0.026**
	knn-cv-1	0.347	0.276	0.207	0.161	0.134
	lin-1	0.088	0.083	0.080	0.079	0.078
	mars3.6-bag-1	0.123	0.068	0.059	0.057	0.061
	mlp-ese-1	0.072	0.052	0.038	0.031	0.028
	mlp-mc-1	**0.060**	**0.040**	**0.031**	**0.028**	**0.026**
	me-ese-1	0.064	0.047	0.038	0.033	0.030
	hme-ese-1	0.066	0.049	0.040	0.035	0.032
	me-el-1	0.078	0.056	0.040	0.033	0.030
	hme-el-1	0.085	0.058	0.045	0.036	0.033
	hme-grow-1	0.076	0.059	0.043	0.037	0.034
	lin-2	0.088	0.083	0.080	0.079	0.079
	lin-ese-1	0.088	0.083	0.081	0.079	0.079

Table B.6: **Results on the Kin-8fm data set**

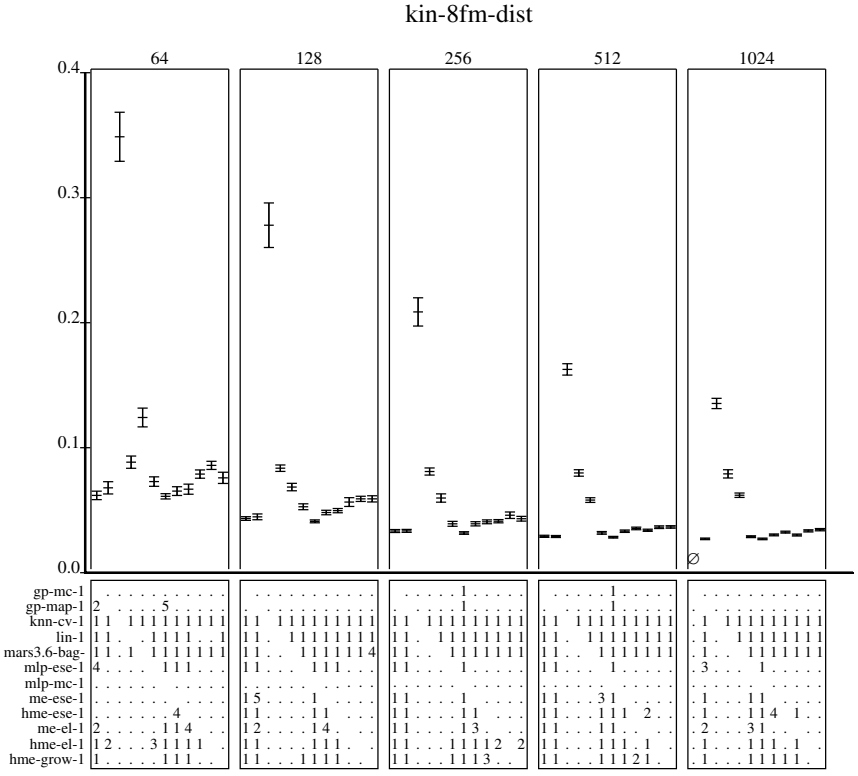

Figure B.6: **Results on the Kin-8fm data set**

Kin-8fh/dist

	Method	Data set size				
		64	128	256	512	1024
Kin-8fh/dist	gp-mc-1	0.316	0.307	0.280	0.262	∅
	gp-map-1	0.364	0.322	0.286	0.263	**0.250**
	knn-cv-1	0.524	0.452	0.404	0.376	0.346
	lin-1	0.315	0.308	0.296	0.287	0.285
	mars3.6-bag-1	0.404	0.330	0.289	0.272	0.265
	mlp-ese-1	0.317	0.305	0.286	0.270	0.259
	mlp-mc-1	0.320	0.304	0.284	0.262	0.253
	me-ese-1	0.311	**0.291**	**0.270**	**0.260**	0.253
	hme-ese-1	**0.308**	0.292	0.273	0.261	0.254
	me-el-1	0.354	0.314	0.298	0.271	0.251
	hme-el-1	0.348	0.300	0.287	0.270	0.258
	hme-grow-1	0.314	0.297	0.276	0.264	0.255
	lin-2	0.315	0.308	0.296	0.287	0.285
	lin-ese-1	0.318	0.309	0.297	0.287	0.286

Table B.7: **Results on the Kin-8fh data set**

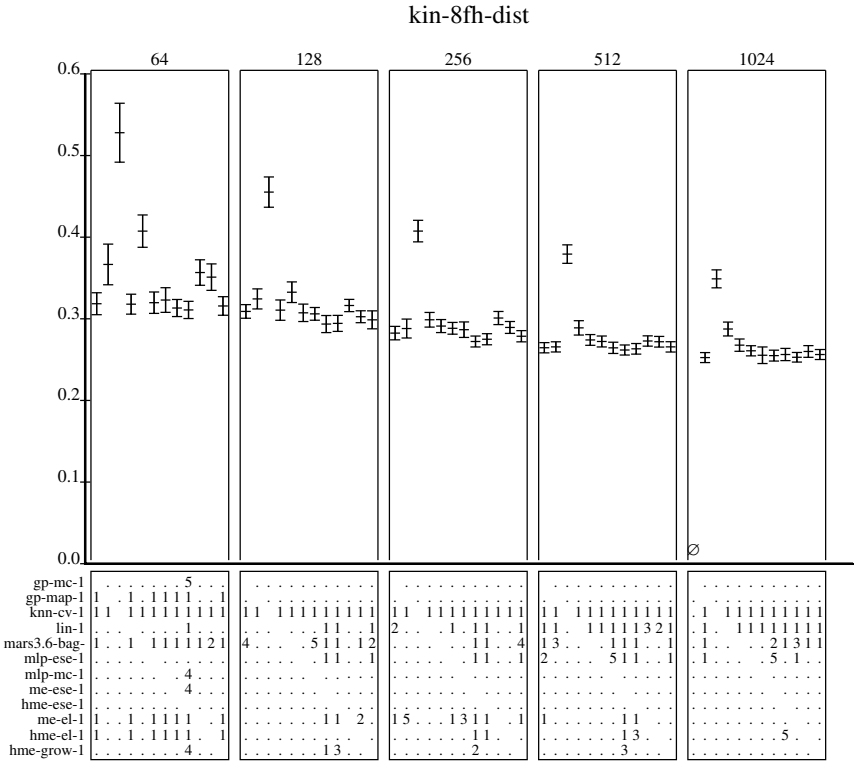

Figure B.7: **Results on the Kin-8fh data set**

Kin-8nm/dist

	Method	Data set size				
		64	128	256	512	1024
Kin-8nm/dist	gp-mc-1	0.586	**0.353**	**0.231**	0.162	∅
	gp-map-1	0.595	0.361	**0.231**	0.162	0.116
	knn-cv-1	0.693	0.557	0.479	0.394	0.317
	lin-1	0.655	0.600	0.590	0.574	0.569
	mars3.6-bag-1	0.655	0.566	0.532	0.487	0.460
	mlp-ese-1	**0.581**	0.408	0.248	0.154	0.104
	mlp-mc-1	0.594	0.359	0.234	**0.122**	**0.094**
	me-ese-1	0.596	0.488	0.387	0.300	0.229
	hme-ese-1	0.592	0.496	0.395	0.325	0.262
	me-el-1	0.598	0.463	0.370	0.274	0.182
	hme-el-1	0.626	0.497	0.445	0.360	0.287
	hme-grow-1	0.615	0.516	0.438	0.395	0.378
	lin-2	0.655	0.600	0.591	0.574	0.569
	lin-ese-1	0.652	0.599	0.592	0.575	0.569

Table B.8: **Results on the Kin-8nm data set**

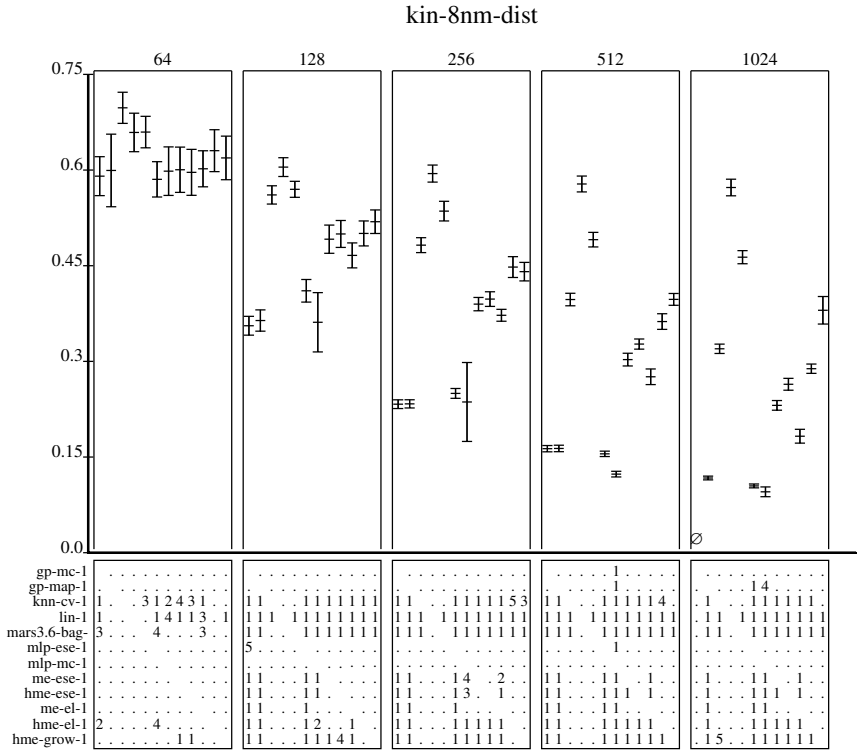

Figure B.8: **Results on the Kin-8nm data set**

Kin-8nh/dist

	Method	Data set size				
		64	128	256	512	1024
Kin-8nh/dist	gp-mc-1	0.735	0.643	0.501	0.449	∅
	gp-map-1	0.827	**0.596**	**0.497**	0.449	0.413
	knn-cv-1	0.775	0.686	0.604	0.567	0.522
	lin-1	0.773	0.676	0.655	0.632	0.629
	mars3.6-bag-1	0.763	0.678	0.627	0.593	0.579
	mlp-ese-1	**0.715**	0.630	0.531	0.453	0.405
	mlp-mc-1	0.726	0.632	0.536	**0.434**	**0.396**
	me-ese-1	0.753	0.639	0.582	0.519	0.464
	hme-ese-1	0.753	0.638	0.579	0.538	0.508
	me-el-1	0.733	0.637	0.571	0.517	0.451
	hme-el-1	0.722	0.647	0.594	0.556	0.521
	hme-grow-1	0.751	0.644	0.599	0.567	0.559
	lin-2	0.773	0.676	0.654	0.632	0.629
	lin-ese-1	0.771	0.678	0.656	0.632	0.630

Table B.9: **Results on the Kin-8nh data set**

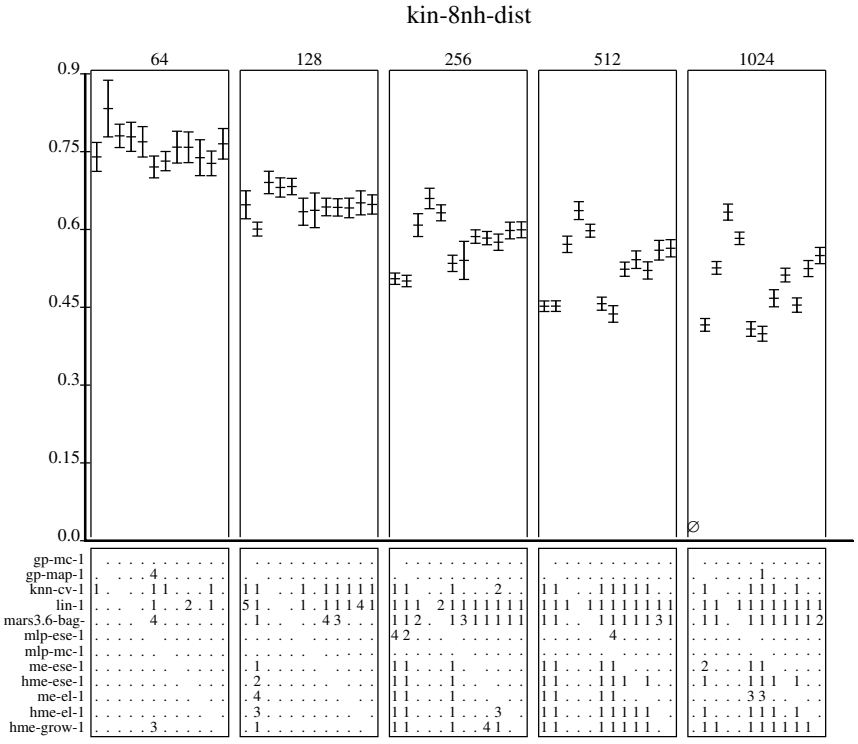

Figure B.9: **Results on the Kin-8nh data set**

Pumadyn-32fm/accel

	Method	Data set size				
		64	128	256	512	1024
Pumadyn-32fm/accel	gp-mc-1	**0.060**	0.057	0.046	0.050	∅
	gp-map-1	0.086	0.047	0.041	**0.036**	**0.031**
	knn-cv-1	0.713	0.598	0.543	0.497	0.443
	lin-1	0.153	0.097	0.082	0.076	0.072
	mars3.6-bag-1	0.068	**0.045**	**0.037**	**0.036**	0.035
	mlp-ese-1	0.141	0.097	0.082	0.072	0.049
	mlp-mc-1	0.070	0.046	0.044	0.041	**0.031**
	me-ese-1	0.169	0.099	0.083	0.076	0.072
	hme-ese-1	0.172	0.098	0.083	0.076	0.072
	me-el-1	0.215	0.142	0.109	0.093	0.071
	hme-el-1	0.185	0.139	0.101	0.085	0.076
	hme-grow-1	0.147	0.098	0.083	0.076	0.072
	lin-2	0.153	0.098	0.083	0.076	0.073
	lin-ese-1	0.144	0.098	0.083	0.076	0.072

Table B.10: **Results on the Pumadyn-32fm data set**

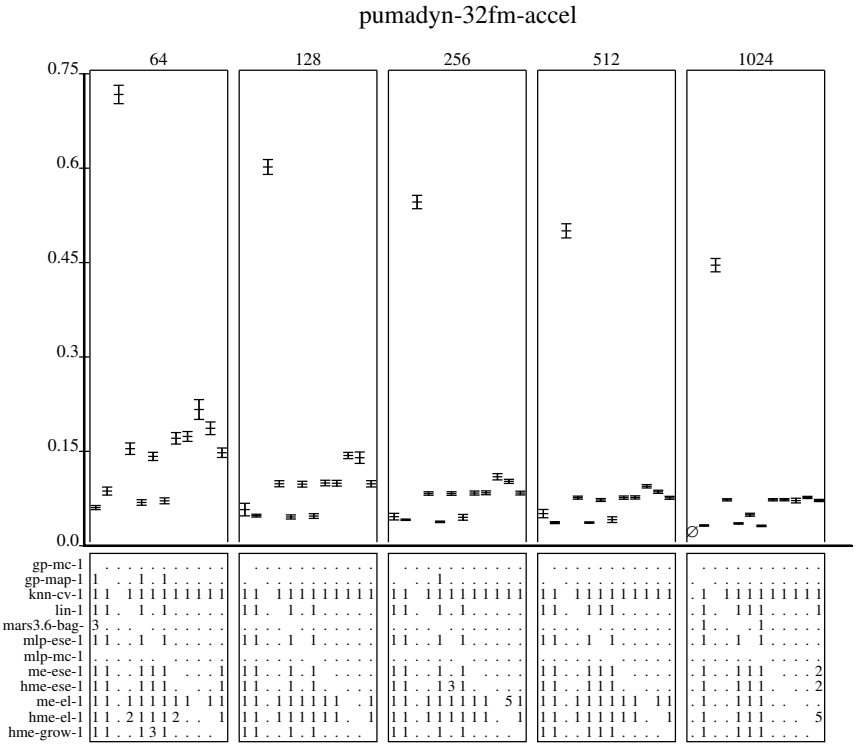

Figure B.10: **Results on the Pumadyn-32fm data set**

Pumadyn-32fh/accel

	Method	Data set size				
		64	128	256	512	1024
Pumadyn-32fh/accel	gp-mc-1	0.649	0.612	0.585	0.597	∅
	gp-map-1	0.602	0.601	0.589	0.581	0.566
	knn-cv-1	0.895	0.851	0.824	0.783	0.762
	lin-1	1.143	0.759	0.637	0.603	0.582
	mars3.6-bag-1	0.664	0.628	0.590	0.569	**0.560**
	mlp-ese-1	0.772	0.698	0.627	0.600	0.582
	mlp-mc-1	**0.593**	**0.585**	**0.570**	**0.568**	0.565
	me-ese-1	0.796	0.724	0.635	0.606	0.583
	hme-ese-1	0.797	0.724	0.634	0.605	0.587
	me-el-1	0.772	0.743	0.668	0.651	0.632
	hme-el-1	0.890	0.757	0.685	0.632	0.618
	hme-grow-1	0.836	0.744	0.636	0.604	0.584
	lin-2	1.141	0.760	0.637	0.604	0.582
	lin-ese-1	0.845	0.738	0.636	0.605	0.584

Table B.11: **Results on the Pumadyn-32fh data set**

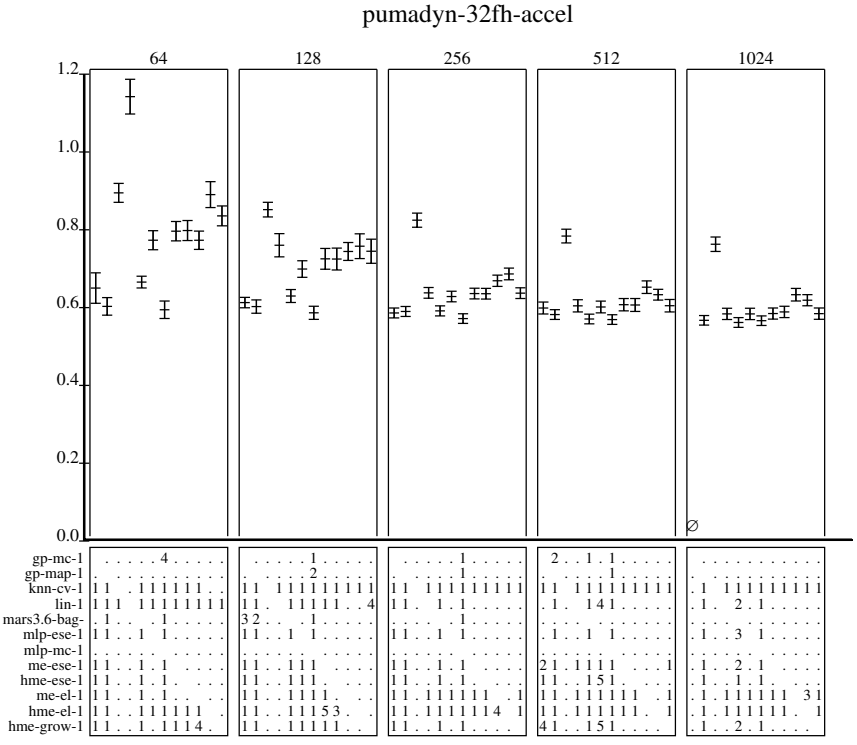

Figure B.11: **Results on the Pumadyn-32fh data set**

Pumadyn-32nm/accel

	Method	Data set size				
		64	128	256	512	1024
Pumadyn-32nm/accel	gp-mc-1	**0.108**	0.117	**0.058**	0.057	$\emptyset$
	gp-map-1	0.123	**0.113**	0.062	**0.043**	**0.044**
	knn-cv-1	0.993	0.961	0.912	0.884	0.849
	lin-1	1.573	1.070	0.885	0.821	0.788
	mars3.6-bag-1	0.494	0.135	0.068	0.062	0.061
	mlp-ese-1	0.977	0.922	0.854	0.704	0.170
	mlp-mc-1	0.788	0.236	0.231	0.220	0.046
	me-ese-1	1.103	1.013	0.889	0.819	0.784
	hme-ese-1	1.110	1.019	0.885	0.824	0.789
	me-el-1	0.992	0.963	0.918	0.825	0.701
	hme-el-1	1.173	0.985	0.898	0.819	0.735
	hme-grow-1	1.117	1.047	0.879	0.814	0.779
	lin-2	1.572	1.070	0.886	0.822	0.789
	lin-ese-1	1.162	1.038	0.885	0.821	0.790

Table B.12: **Results on the Pumadyn-32nm data set**

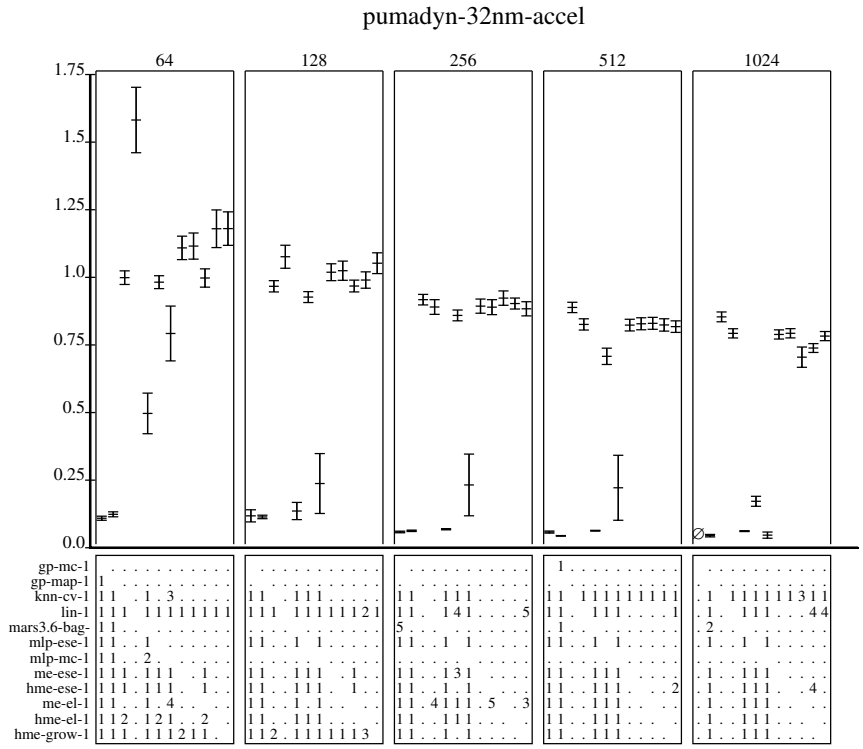

Figure B.12: **Results on the Pumadyn-32nm data set**

Pumadyn-32nh/accel

	Method	Data set size				
		64	128	256	512	1024
Pumadyn-32nh/accel	gp-mc-1	**0.601**	**0.526**	0.398	0.384	$\emptyset$
	gp-map-1	1.018	0.700	0.384	0.356	0.347
	knn-cv-1	1.002	1.010	0.948	0.929	0.908
	lin-1	1.985	1.203	0.965	0.894	0.866
	mars3.6-bag-1	0.926	0.528	**0.376**	**0.344**	**0.342**
	mlp-ese-1	0.993	0.959	0.922	0.885	0.708
	mlp-mc-1	0.879	0.582	0.502	0.592	0.345
	me-ese-1	1.253	1.127	0.962	0.896	0.867
	hme-ese-1	1.229	1.125	0.963	0.898	0.871
	me-el-1	1.041	1.060	0.990	0.917	0.882
	hme-el-1	1.332	1.050	0.976	0.906	0.880
	hme-grow-1	1.300	1.156	0.970	0.889	0.859
	lin-2	1.978	1.203	0.965	0.894	0.866
	lin-ese-1	1.32	1.15	0.966	0.895	0.865

Table B.13: **Results on the Pumadyn-32nh data set**

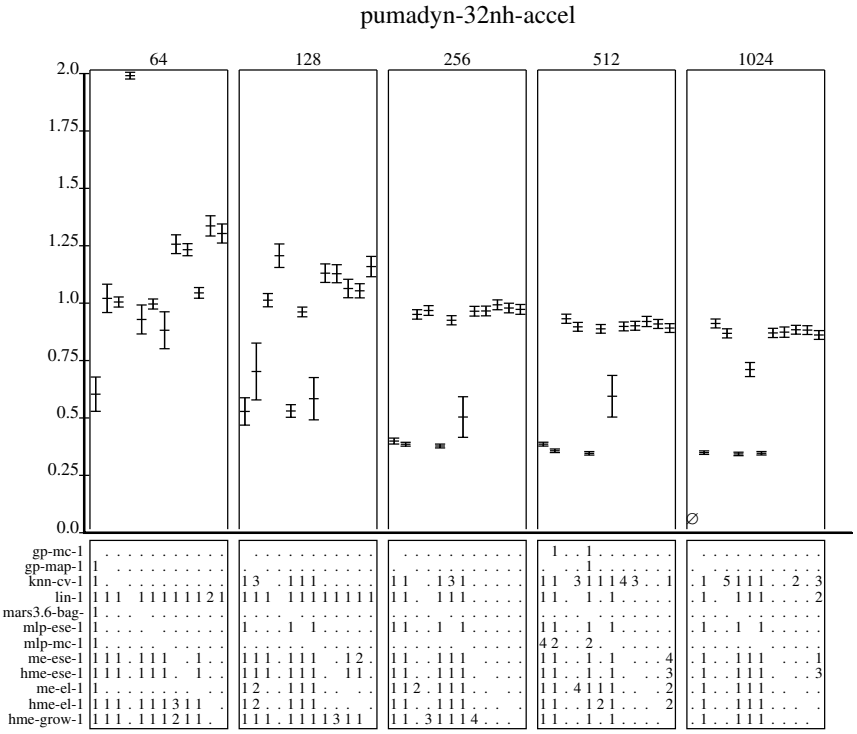

Figure B.13: **Results on the Pumadyn-32nh data set**

Pumadyn-8fm/accel

	Method	Data set size				
		64	128	256	512	1024
Pumadyn-8fm/accel	gp-mc-1	0.071	0.059	0.054	**0.052**	∅
	gp-map-1	0.069	0.059	0.054	**0.052**	**0.050**
	knn-cv-1	0.352	0.292	0.221	0.176	0.146
	lin-1	0.097	0.085	0.079	0.077	0.076
	mars3.6-bag-1	0.077	0.061	**0.053**	**0.052**	**0.050**
	mlp-ese-1	0.090	0.073	0.061	0.056	0.052
	mlp-mc-1	**0.064**	**0.057**	0.054	**0.052**	0.050
	me-ese-1	0.097	0.083	0.074	0.066	0.055
	hme-ese-1	0.097	0.084	0.075	0.071	0.062
	me-el-1	0.113	0.086	0.066	0.059	0.057
	hme-el-1	0.117	0.092	0.073	0.065	0.057
	hme-grow-1	0.097	0.084	0.076	0.066	0.059
	lin-2	0.097	0.085	0.079	0.077	0.077
	lin-ese-1	0.098	0.085	0.079	0.077	0.077

Table B.14: **Results on the Pumadyn-8fm data set**

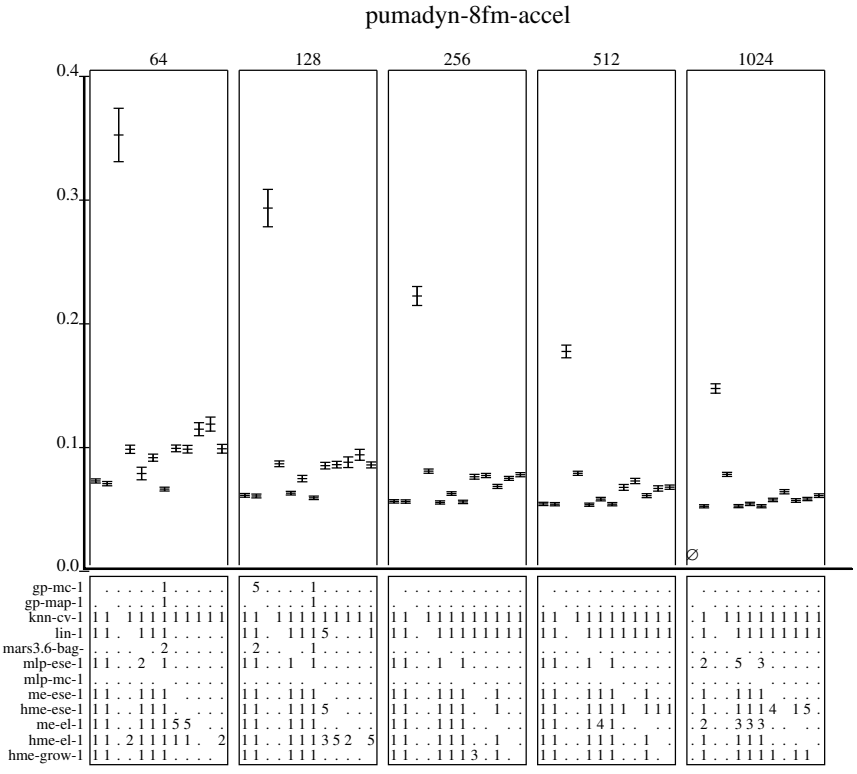

Figure B.14: **Results on the Pumadyn-8fm data set**

Pumadyn-8fh/accel

	Method	Data set size				
		64	128	256	512	1024
Pumadyn-8fh/accel	gp-mc-1	0.467	0.431	0.407	0.405	$\emptyset$
	gp-map-1	0.470	0.429	0.411	**0.395**	**0.390**
	knn-cv-1	0.652	0.598	0.529	0.493	0.470
	lin-1	0.456	0.434	0.418	0.413	0.410
	mars3.6-bag-1	0.488	0.462	0.418	0.403	0.393
	mlp-ese-1	0.461	0.432	0.416	0.408	0.399
	mlp-mc-1	**0.445**	**0.420**	**0.406**	0.400	0.395
	me-ese-1	0.460	0.436	0.416	0.412	0.407
	hme-ese-1	0.461	0.437	0.418	0.414	0.410
	me-el-1	0.533	0.491	0.436	0.427	0.403
	hme-el-1	0.538	0.482	0.435	0.414	0.402
	hme-grow-1	0.456	0.437	0.416	0.408	0.400
	lin-2	0.456	0.435	0.419	0.414	0.410
	lin-ese-1	0.458	0.435	0.419	0.414	0.410

Table B.15: **Results on the Pumadyn-8fh data set**

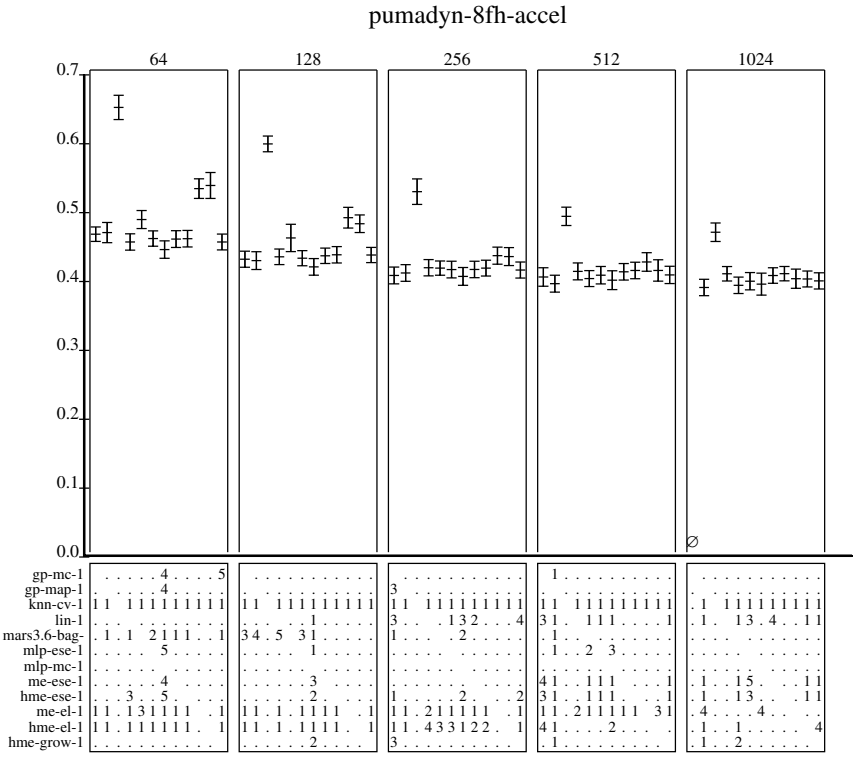

Figure B.15: **Results on the Pumadyn-8fh data set**

Pumadyn-8nm/accel

	Method	Data set size				
		64	128	256	512	1024
Pumadyn-8nm/accel	gp-mc-1	0.067	**0.052**	0.044	**0.038**	∅
	gp-map-1	0.067	**0.052**	0.044	**0.038**	**0.036**
	knn-cv-1	0.683	0.536	0.452	0.365	0.303
	lin-1	0.587	0.519	0.522	0.507	0.500
	mars3.6-bag-1	0.118	0.068	0.062	0.057	0.055
	mlp-ese-1	0.315	0.128	0.058	0.046	0.040
	mlp-mc-1	**0.066**	0.054	**0.042**	**0.038**	**0.036**
	me-ese-1	0.569	0.465	0.269	0.127	0.057
	hme-ese-1	0.572	0.485	0.296	0.160	0.057
	me-el-1	0.555	0.363	0.127	0.059	0.046
	hme-el-1	0.553	0.423	0.189	0.070	0.049
	hme-grow-1	0.579	0.492	0.388	0.229	0.174
	lin-2	0.588	0.519	0.522	0.508	0.500
	lin-ese-1	0.588	0.520	0.523	0.507	0.501

Table B.16: **Results on the Pumadyn-8nm data set**

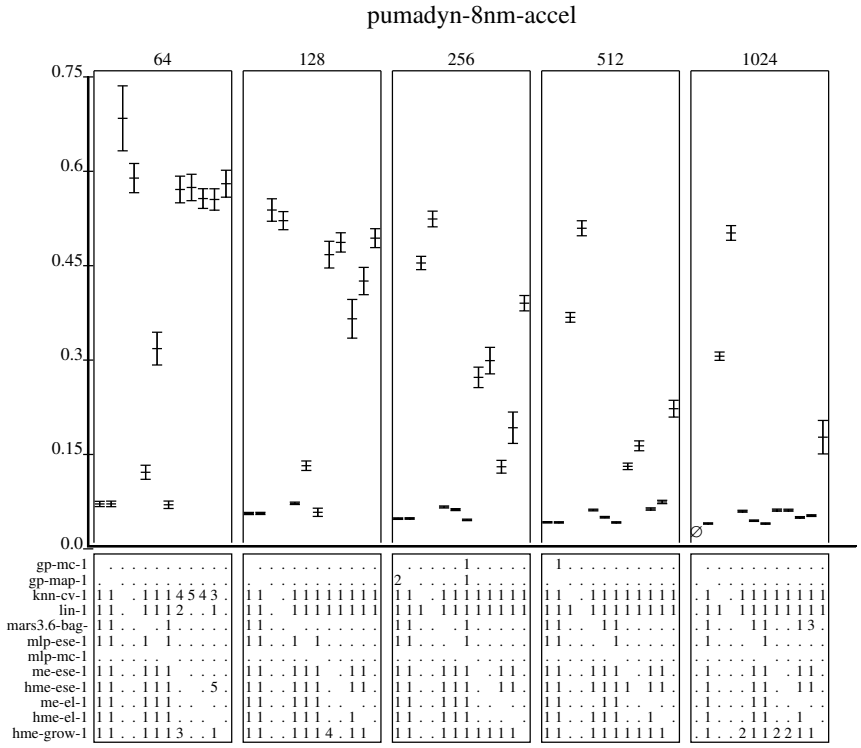

Figure B.16: **Results on the Pumadyn-8nm data set**

Pumadyn-8nh/accel

	Method	Data set size				
		64	128	256	512	1024
Pumadyn-8nh/accel	gp-mc-1	0.461	**0.373**	**0.345**	0.325	∅
	gp-map-1	**0.447**	0.376	0.351	0.325	0.323
	knn-cv-1	0.786	0.709	0.635	0.582	0.533
	lin-1	0.735	0.680	0.654	0.635	0.631
	mars3.6-bag-1	0.505	0.382	0.363	0.344	0.336
	mlp-ese-1	0.668	0.509	0.409	0.366	0.342
	mlp-mc-1	0.452	0.394	**0.345**	**0.323**	**0.321**
	me-ese-1	0.719	0.667	0.608	0.497	0.406
	hme-ese-1	0.724	0.672	0.613	0.549	0.448
	me-el-1	0.741	0.669	0.536	0.425	0.360
	hme-el-1	0.742	0.663	0.537	0.423	0.364
	hme-grow-1	0.723	0.672	0.603	0.537	0.512
	lin-2	0.735	0.681	0.655	0.635	0.631
	lin-ese-1	0.730	0.679	0.655	0.635	0.632

Table B.17: **Results on the Pumadyn-8nh data set**

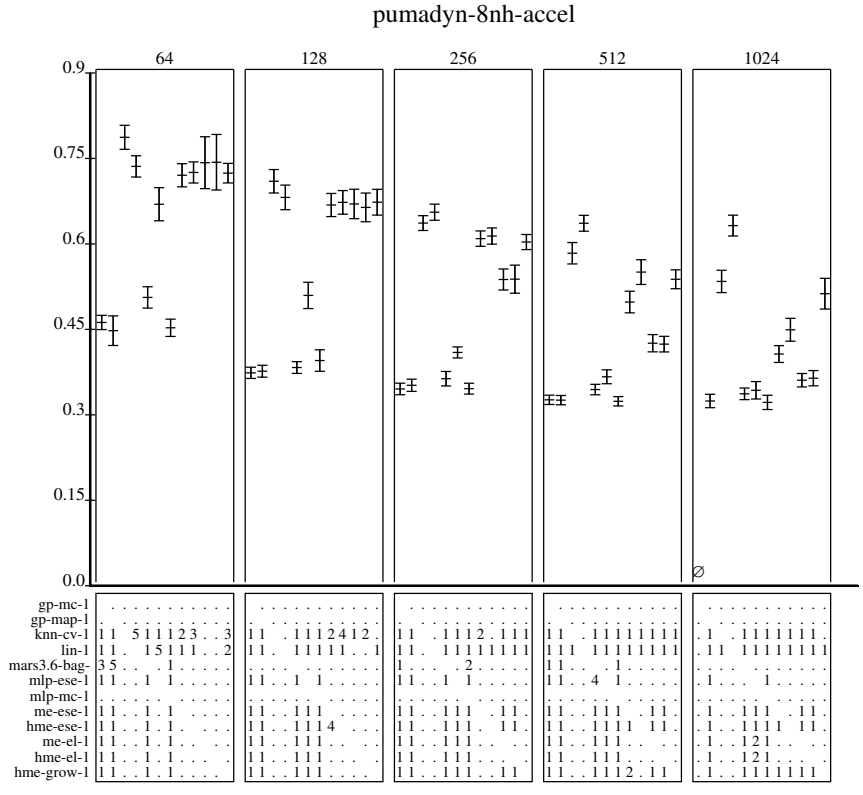

Figure B.17: **Results on the Pumadyn-8nh data set**

Appendix C

Results on DELVE Classification Tasks

This chapter contains the results of applying all the learning methods described in Part I of the thesis to the DELVE classification tasks. There are four data sets in total: the Image, Letter and Splice data sets. Once again, for brevity, detailed legends are not included for each of the tables and figures and only a short caption is used. Each figure and table shows the standardised zero:one losses for each learning method for different sized training sets. For example, the results on the Image data set are shown in Figure C.1 and Table C.1. Each column of the figure and table shows the squared error loss for the different learning methods for training sets with different numbers of examples. The key to each column in the figures is given at the bottom left of the figure. Reading from top to bottom of the list of methods corresponds to the bars from left to right of each column. In the table the losses are shown in numeric form. The results for the best performing method, *i.e.*, the one with smallest loss, is shown in bold face. In the figure the losses are shown as horizontal lines with the associated standard error shown as bars on either side of the loss.

Each figure also includes a matrix of p-values for each of the methods. These are the p-values obtained from paired t-tests or F-tests between each of the methods. The minimum p-value shown is 1 and the maximum is 5. A result is considered significant if the p value is 5 or less. Non-significant comparisons are shown with a dot.

C.1 Image

	Method	Data set size		
		70	140	280
Image-seg/seg	cart-1	0.422	0.406	0.336
	knn-class-1	0.412	0.389	0.376
	1nn-1	0.395	0.387	0.376
	me-ese-1	0.374	0.394	**0.328**
	hme-ese-1	0.372	0.364	0.330
	me-el-1	**0.360**	0.370	0.334
	hme-el-1	0.363	**0.357**	0.330
	hme-grow-1	0.369	0.374	0.332
	lin-2	0.366	0.385	0.334
	lin-ese-1	0.371	0.380	0.368

Table C.1: **Results on the Image data set**

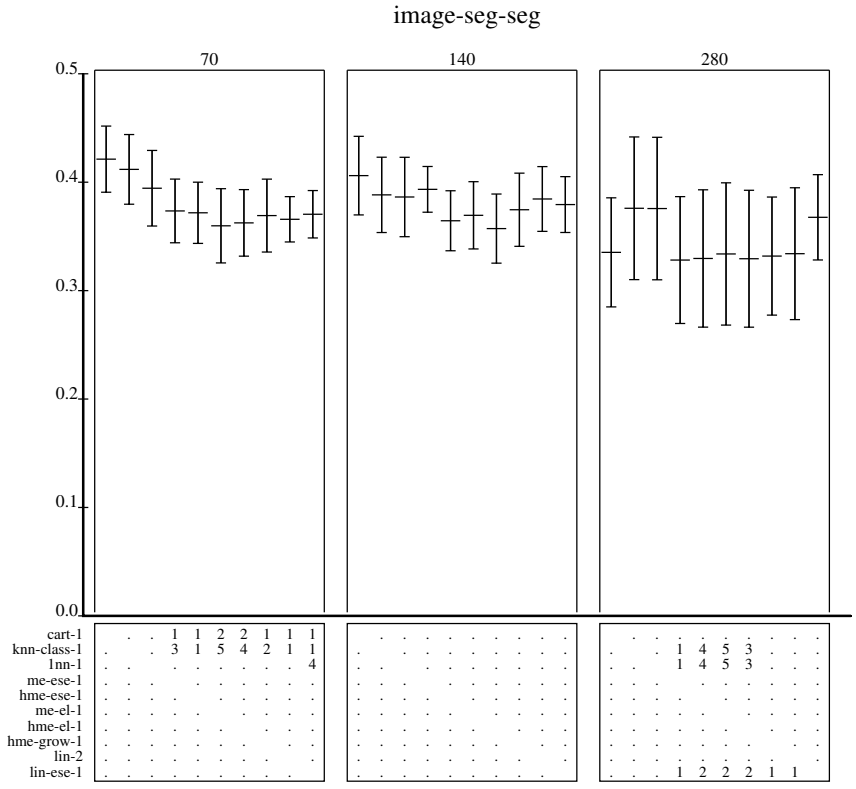

Figure C.1: **Results on the Image data set**

C.2 Letter

	Method	Data set size		
		390	790	1560
Letter/recog	cart-1	0.522	0.429	0.341
	knn-class-1	0.366	0.274	0.194
	1nn-1	0.366	0.274	0.194
	me-ese-1	**0.316**	0.263	**0.193**
	hme-ese-1	0.326	**0.250**	0.198
	me-el-1	0.337	0.274	0.203
	hme-el-1	0.328	0.257	0.204
	hme-grow-1	0.360	0.289	0.227
	lin-2	0.419	0.326	0.277
	lin-ese-1	0.344	0.298	0.271

Table C.2: **Results on the Letter data set**

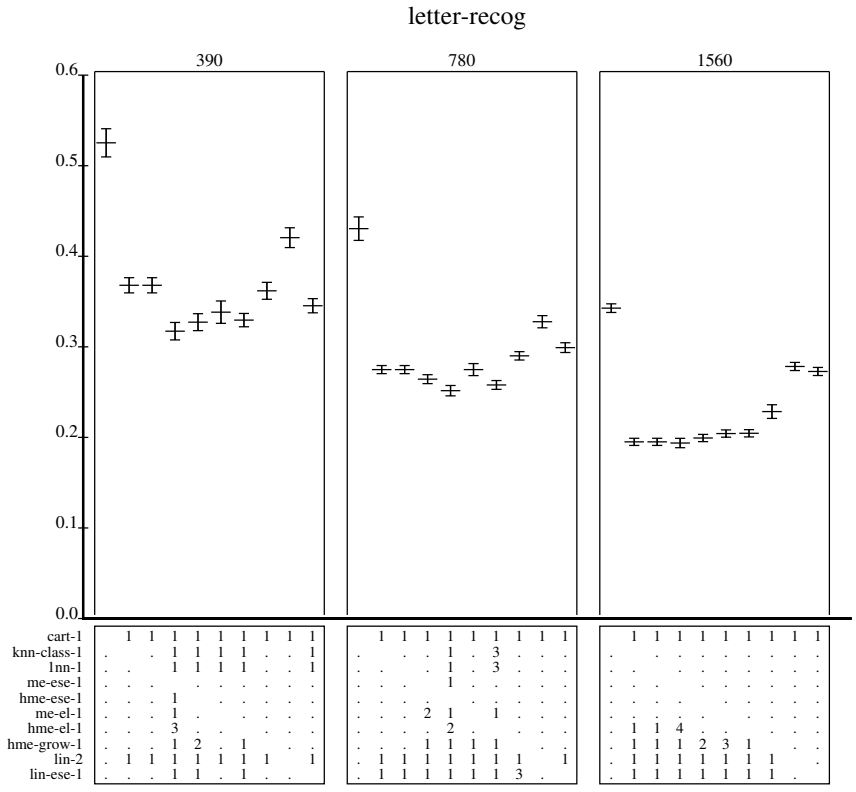

Figure C.2: **Results on the Letter data set**

C.3 Splice

	Method	Data set size		
		100	200	400
Splice/bound	cart-1	0.451	0.293	0.210
	knn-class-1	0.710	0.614	0.551
	1nn-1	0.963	0.898	0.823
	me-ese-1	0.317	0.203	**0.142**
	hme-ese-1	0.321	0.207	0.146
	me-el-1	0.303	0.210	0.160
	hme-el-1	**0.301**	0.207	0.146
	hme-grow-1	0.321	0.221	0.151
	lin-2	0.303	0.216	0.167
	lin-ese-1	0.304	**0.196**	0.144

Table C.3: **Results on the Splice data set**

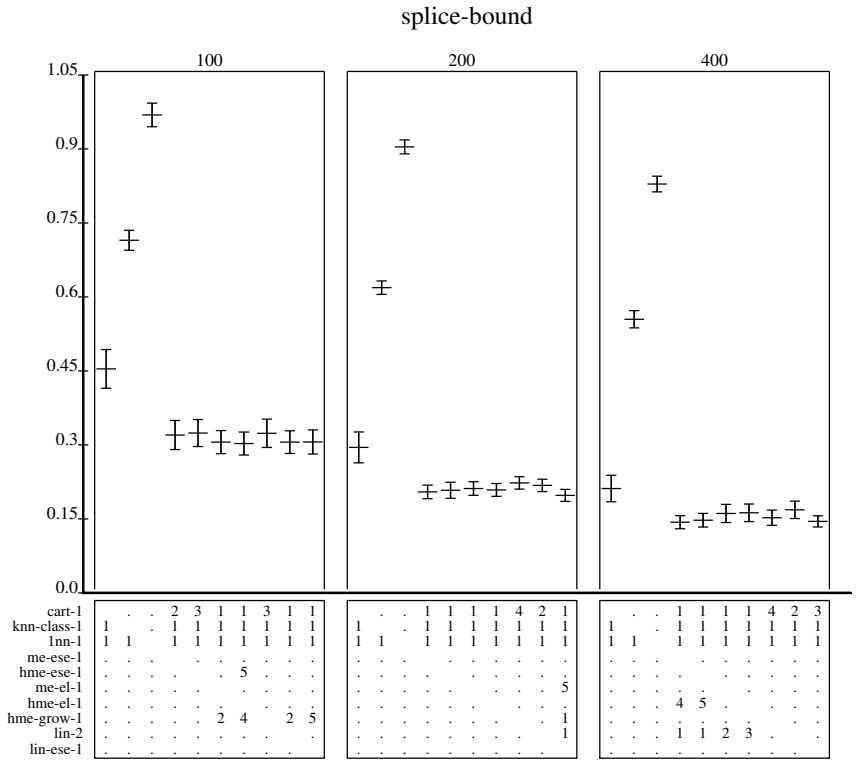

Figure C.3: **Results on the Splice data set**

Bibliography

[1] Alpaydin, E. and Jordan, M. I. [1996], 'Local linear perceptrons for classification', *IEEE Transactions on Neural Networks* **7**(3), 788–792.

[2] Amari, S. [1995*a*], 'The EM algorithm and information geometry in neural-network learning', *Neural Computation* **7**(1), 13–18.

[3] Amari, S. I. [1995*b*], 'Information geometry of the EM and *em* algorithms for neural networks', *Neural Networks* **8**(9), 1379–1408.

[4] Anand, R., Mehrotra, K., Mohan, C. K. and Ranka, S. [1995], 'Efficient classification for multiclass problems using modular neural networks', *IEEE Transactions on Neural Networks* **6**(1), 117–124.

[5] Atal, B. S., Cuperman, V. and Gersho, A. [1991], *Advances in Speech Coding*, Kluwer Academic Publishers.

[6] Bahl, L. R., Jelinek, F. and Mercer, R. L. [1983], 'A maximum-likelihood approach to continuous speech recognition', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **5**(2), 179–190.

[7] Barlow, T. W. [1995], 'Feedforward neural networks for secondary structure prediction', *Journal of Molecular Graphics* **13**(3), 175–183.

[8] Baum, L. E. [1972], 'An inequality and associated maximization in statistical estimation for probabilistic functions of Markov processes', *Inequalities* **3**, 1–8.

[9] Bengio, S., Fessant, F. and Collobert, D. [1996], 'Use of modular architectures for time-series prediction', *Neural Processing Letters* **3**(2), 101–106.

[10] Bengio, Y. and Frasconi, P. [1996], 'Input-output HMMs for sequence processing', *IEEE Transactions on Neural Networks* **7**(5), 1231–1249.

[11] Bennani, Y. [1995], 'A modular and hybrid connectionist system for speaker identification', *Neural Computation* **7**(4), 791–798.

[12] Berger, J. [1985], *Statistical Decision theory and Bayesian Analysis*, Springer.

[13] Bernardo, J. M. and Smith, A. F. M. [1994], *Bayesian Theory*, Wiley.

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

[15] Bishop, C. M. and Nabney, I. T. [1996], 'Modelling conditional-probability distributions for periodic variables', *Neural Computation* **8**(5), 1123–1133.

[16] Bourlard, H. A. and Morgan, N. [1993], 'Continuous speech recognition by connectionist statistical-methods', *IEEE Transactions on Neural Networks* **4**(6), 893–909.

[17] Bourlard, H. A. and Morgan, N. [1994], *Connectionist Speech Recognition: A Hybrid Approach*, Kluwer Academic Publishers.

[18] Box, G. E. P. and Tiao, G. C. [1973], *Bayesian inference in statistical analysis*, Addison–Wesley.

[19] Bregler, C. and Malik, J. [1997], Learning appearance based models: mixtures of second moment experts, *in* Mozer et al. [148], pp. 845–851.

[20] Breiman, L. [1997], 'The uncertainty principle in statistics'. Lecture notes from the Generalisation in Neural Networks and Machine Learning NATO ASI, at the Isaac Newton Institue for Mathematical Sciences, Cambridge University.

[21] Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. [1984], *Classification and Regression Trees*, Wadsworth and Brooks/Cole, Monterey, CA.

[22] Breiman, L. and Meisel, W. S. [1976], 'General estimates of the intrinsic variability of data in nonlinear regression models', *Journal of the American Statistical Association* **71**(354), 301–307.

[23] Bridle, J. S. [1989], Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition, *in* F. Fougelman-Soulie and J. Hérault, eds, 'Neurocomputing: Algorithms, Architectures and Applications', Springer-Verlag, pp. 227–236.

[24] Brodley, C. E. [1995], 'Recursive automatic bias selection for classifier construction', *Machine Learning* **20**(1-2), 63–94.

[25] Brodley, C. E. and Smyth, P. [1997], 'Applying classification algorithms in practice', *Statistics and Computing* **7**(1), 45–56.

[26] Brodley, C. E. and Utgoff, P. E. [1995], 'Multivariate decision trees', *Machine Learning* **19**(1), 45–77.

[27] Buntine, W. L. [1994], 'Operations for learning with graphical models', *Journal of Artificial Intelligence Research* **2**, 159–225.

[28] Buntine, W. L. [1995], Graphical models for discovering knowledge, *in* Fayyad, Piatetsky-Shapiro, Smyth and Uthurusamy [52], chapter 3, pp. 59–83.
*http://www.ultimode.com/wray/

[29] Buntine, W. L. and Weigend, A. S. [1991], 'Bayesian back–propagation', *Complex Systems* **5**, 603–643.

[30] Cacciatore, T. W. and Nowlan, S. J. [1994], Mixtures of controllers for jump linear and nonlinear plants, *in* J. D. Cowan, G. Tesauro and J. Alspector, eds, 'Advances in Neural Information Processing Systems 6', Morgan Kaufmann, San Francisco, CA, pp. 719–726.

[31] Carter, C. and Catlett, J. [1987], 'Assessing credit card applications using machine learning', *IEEE Expert* **2**(3), 71–79.

[32] Chaer, W. S., Bishop, R. H. and Ghosh, J. [1997], 'A mixture-of-experts framework for adaptive Kalman filtering', *IEEE Transactions on Systems, Man, and Cybernetics, Part B* **27**(3), 452–464.

[33] Cheeseman, P. and Stutz, J. [1995], Bayesian classification (AutoClass): theory and results, *in* Fayyad, Piatetsky-Shapiro, Smyth and Uthurusamy [52], chapter 6, pp. 153–180.

[34] Chen, K., Xie, D. H. and Chi, H. S. [1996*a*], 'A modified HME architecture for text-dependent speaker identification', *IEEE Transactions on Neural Networks* **7**(5), 1309–1313.

[35] Chen, K., Xie, D. H. and Chi, H. S. [1996*b*], 'Speaker identification using time-delay HMEs', *International Journal of Neural Systems* **7**(1), 29–43.

[36] Cheng, B. and Titterington, D. M. [1994], 'Neural networks - a review from a statistical perspective', *Statistical Science* **9**(1), 2–30.

[37] Cohen, P. R. [1995], *Empirical Methods for Artificial Intelligence*, MIT Press.

[38] Cook, G. D. [1997], Data Selection and Model Combination in Connectionist Speech Recognition, PhD thesis, Cambridge University Engineering Department.

[39] Cook, G. D. and Robinson, A. J. [1995], Utterance clustering for large vocabulary continuous speech recognition, *in* 'Proceedings of the European Conference on Speech Technology', Vol. I, pp. 219–22.
*http://svr-www.eng.cam.ac.uk/ gdc/papers/eurospeech95.ps.Z

[40] Cook, G. D. and Robinson, A. J. [1996], Boosting the performance of connectionist large-vocabulary speech recognition, *in* 'Proceedings of the International Conference on Spoken Language Processing'.
*http://svr-www.eng.cam.ac.uk/ gdc/papers/icslp96-preprint.ps.Z

[41] Csiszár, I. and Tusnády, G. [1983], Information geometry and alternating minimization procedures, *in* e. a. E. J. Dudewicz, ed., 'Recent Results in Estimation Theory and Related Topics', Statistics and Decisions, Supplement Issue No. 1, 1984, Oldenburg Verlag, Munich.

[42] Davis, S. B. and Mermelstein, P. [1980], 'Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences', *IEEE Transactions on Acoustics, Speech, and Signal Processing* **28**(4), 357–66.

[43] Dayan, P., Hinton, G. E., Neal, R. M. and Zemel, R. S. [1995], 'The Helmholtz machine', *Neural Computation* **7**(5), 889–904.

[44] Dayan, P. and Zemel, R. S. [1995], 'Competition and multiple cause models', *Neural Computation* **7**(3), 565–579.

[45] Dempster, A. P., Laird, N. M. and Rubin, D. B. [1977], 'Maximum likelihood from incomplete data via the EM algorithm', *Journal of the Royal Statistical Society, Series B* **39**, 1–38.

[46] Dietterich, T. [1995], 'Overfitting and undercomputing in machine learning', *ACM Computing Surveys* **27**(3), 326–327.
*ftp://ftp.cs.orst.edu/pub/tgd/papers/cs95.ps.gz

[47] Dietterich, T. G., Hild, H. and Bakiri, G. [1995], 'A comparison of ID3 and backpropagation for English text-to-speech mapping', *Machine Learning* **18**(1), 51–80.
*ftp://ftp.cs.orst.edu/pub/tgd/papers/mlj-nettalk.ps.gz

[48] Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y. and Vapnik, V. [1994], 'Boosting and other ensemble methods', *Neural Computation* **6**, 1289–1301.

[49] Duda, R. O. and Hart, P. E. [1974], *Pattern Classification and Scene Analysis*, John Wiley and Sons, New York.

[50] Elman, J. L. [1990], 'Finding structure in time', *Cognitive Science* **14**, 179–211.

[51] Fayyad, U. M., Piatetsky-Shapiro, G. and Smyth, P. [1995], From data-mining to knowledge discovery: An overview., *in* Fayyad, Piatetsky-Shapiro, Smyth and Uthurusamy [52], chapter 1, pp. 1–37.

[52] Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, R., eds [1995], *Knowledge Discovery in Data Bases II*, AAAI Press / The MIT Press, Menlo Park, CA.

[53] Fletcher, R. [1987], *Practical Methods of Optimization*, Wiley, New York.

[54] Flexer, A. [1996], Statistical evaluation of neural network experiments: minimum requirements and current practice, *in* T. R., ed., 'Cybernetics and Systems. Proceedings of the 13th European meeting on Cybernetics and Systems Research.', Austrian Society for Cybernetic Studies, Vienna, pp. 1005–1008.

[55] Foley, J. and van Dam A. [1982], *Fundamentals of interactive computer graphics*, Addison-Wesley, London.

[56] Frey, P. W. and Slate, D. J. [1991], 'Letter recognition using Holland-style adaptive classifiers', *Machine Learning* **6**(2), 161–182.

[57] Friedman, J. H. [1979], A tree structured approach to nonparametric multiple regression, *in* T. H. Gasser and M. Rosenblatt, eds, 'Smoothing techniques for curve estimation', number 757 *in* 'Lecture notes in mathematics', Springer-Verlag, chapter 1, pp. 5–22.

[58] Friedman, J. H. [1991], 'Multivariate adaptive regression splines', *Annals of Statistics* **19**(1), 1–67.

[59] Friedman, J. H. and Stuetzle, W. [1981], 'Projection pursuit regression', *Journal of the American Statistical Association* **76**(376), 817–823.

[60] Fritsch, J. [1996], Modular neural networks for speech recognition, Master's thesis, University of Karlsruhe, Germany.
*ftp://reports.adm.cs.cmu.edu/usr/anon/1996/CMU-CS-96-203.ps.gz

[61] Fritsch, J. and Finke, M. [1997], Improving performance on Switchboard by combining hybrid HME/HMM and mixture of Gaussians acoustic models, *in* 'Proceedings of the European Conference on Speech Technology'.
*http://werner.ira.uka.de/∼fritsch/

[62] Fritsch, J., Finke, M. and Waibel, A. [1997*a*], Adaptively growing hierarchical mixtures of experts, *in* Mozer et al. [148], pp. 459–465.
*http://werner.ira.uka.de/∼fritsch/fritsch-nips96.ps.gz

[63] Fritsch, J., Finke, M. and Waibel, A. [1997*b*], Context-dependent hybrid HME/HMM speech recognition using polyphone clustering decision trees, *in* 'Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing'.
*http://werner.ira.uka.de/∼fritsch/fritsch-icassp97.ps

[64] Furui, S. [1986], 'Speaker-independent isolated word recognition using dynamic features of speech spectrum', *IEEE Transactions on Acoustics, Speech, and Signal Processing* **34**(1), 52–59.

[65] Gales, M. J. F. [1995], Model-based techniques for noise robust speech recognition, PhD thesis, Cambridge University Engineering Department.

[66] Gauvian, J.-L. and Lee, C.-H. [1994], 'Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains', *IEEE Transactions on Acoustics, Speech, and Signal Processing* **2**(2), 291–298.

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

[68] Geman, S. and Geman, D. [1984], 'Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**(6), 721–741.

[69] Ghahramani, Z. [1996*a*], 'The kin datasets'.
*http://www.cs.utoronto.ca/ delve/data/kin/kin.ps.gz

[70] Ghahramani, Z. [1996*b*], 'The pumadyn datasets'.
*http://www.cs.utoronto.ca/ delve/data/pumadyn/pumadyn.ps.gz

[71] Ghahramani, Z. and Jordan, M. I. [1994], Learning from incomplete data, Technical Report CBCL 108, Massachusetts Institute of Technology.
*ftp://publications.ai.mit.edu/ai-publications/1500-1999/AIM-1509.ps.Z

[72] Ghahramani, Z. and Jordan, M. I. [1996], Factorial hidden Markov models, *in* Touretzky et al. [226], pp. 472–478.

[73] Ghahramani, Z. and Wolpert, D. M. [1997], 'Modular decomposition in visuomotor learning', *Nature* **386**(6623), 392–395.

[74] Gillick, L. and Cox, S. [1989], Some statistical issues in the comparison of speech recognition algorithms, *in* 'Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing', pp. 532–535.

[75] Godfrey, J. J., Holliman, E. C. and McDaniel, J. [1992], SWITCHBOARD: Telephone speech corpus for research and development, *in* 'Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing', pp. 517–520.

[76] Hamilton, J. D. [1990], 'Analysis of time-series subject to changes in regime', *Journal of Econometrics* **45**(1-2), 39–70.

[77] Hampshire, J. B. and Waibel, A. [1992], 'The meta-Pi network - building distributed knowledge representations for robust multisource pattern-recognition', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**(7), 751–769.

[78] Harrison, D. and Rubinfeld, D. [1978], 'Hedonic prices and the demand for clean air', *Journal of Environmental Economics & Management* **5**, 81–102.

[79] Hartley, M. [1978], 'Comment on Quandt and Ramsey [184]', *Journal of the American Statistical Association* **73**(364), 730–752.

[80] Hastie, T. J. and Tibshirani, R. J. [1990], *Generalised Additive Models*, Chapman and Hall, London.

[81] Hathaway, R. J. [1985], 'A constrained formulation of maximum-likelihood estimation for normal mixture distributions', *Annals of Statistics* **13**(2), 795–800.

[82] Hathaway, R. J. [1986], 'Another interpretation of the EM algorithm for mixture distributions', *Statistics & Probability Letters* **4**(2), 53–56.

[83] Hering, K., Haupt, R. and Villmann, T. [1996], Hierarchical strategy of model partitioning for VLSI-design using an improved mixture of experts approach, *in* 'Tenth Workshop on Parallel and Distributed Simulation - PADS 96, Proceedings', pp. 106–113.
*http://www.informatik.uni-leipzig.de/ifi/abteilungen/ti/pers/PAPER/pads96.ps

[84] Herman J.M. Steeneken, D. A. V. L. [1995], Multi-lingual assessment of speaker independent large vocabulary speech-recognition systems: the SQALE project (speech recognition quality assessment for language engineering), *in* 'Proceedings of the European Conference on Speech Technology'.

[85] Hermansky, H. [1990], 'Perceptual linear predictive (PLP) analysis of speech', *Journal of the Acoustical Society of America* **87**(4), 1738–1752.

[86] Hinton, G. E. and Sejnowski, T. J. [1986], Learning and relearning in Boltzmann machines, *in* D. E. Rumelhart and J. E. McClelland, eds, 'Parallel Distributed Processing', MIT Press, Cambridge Mass., pp. 282–317.

[87] Hinton, G. E. and van Camp, D. [1993], Keeping neural networks simple by minimizing the description length of the weights, *in* 'Proceedings of the 6th Annual conference on Computational Learning Theory', ACM Press, New York, NY, pp. 5–13.

[88] Hinton, G., Revow, M. and Dayan, P. [1995], Recognizing handwritten digits using mixtures of linear models, *in* Tesauro et al. [218].

[89] Hochberg, M., Cook, G., Renals, S. and Robinson, A. J. [1994], Connectionist model combination for large vocabulary speech recognition, *in* 'Proceedings of the IEEE Workshop on Neural Networks for Signal Processing', IEEE Press, Long Beach, CA, pp. 269–278.

[90] Hoerl, A. E. and Kennard, R. W. [1970], 'Ridge regression: biased estimation for nonorthogonal problems', *Technometrics* **12**, 55–67.

[91] Holmstrom, L., Koistinen, P., Laaksonen, J. and Oja, E. [1997], 'Neural and statistical classifiers - taxonomy and two case studies', *IEEE Transactions on Neural Networks* **8**(1), 5–17.

[92] Holte, R. C. [1993], 'Very simple classification rules perform well on most commonly used datasets', *Machine Learning* **11**(1), 63–91.

[93] Hu, Y. H., Palreddy, S. and Tompkins, W. J. [1995], Customized ECG beat classifier using mixture of experts, *in* 'Proceedings of the IEEE Workshop on Neural Networks for Signal Processing', IEEE Press, Long Beach, CA, pp. 459–464.

[94] Huang, X. D., Ariki, Y. and Jack, M. A. [1990], *Hidden Markov models for speech recognition*, Edinburgh University Press, Edinburgh.

[95] Jaakkola, T. S. and Jordan, M. I. [1996*a*], Bayesian logistic regression: a variational approach, Technical report, MIT.
*ftp://psyche.mit.edu/pub/tommi/jaak-bayes.ps.Z

[96] Jaakkola, T. S. and Jordan, M. I. [1996*b*], Computing upper and lower bounds on likelihoods in intractable networks, *in* 'Proceedings of the Twelfth Conference on Uncertainty in AI', Morgan Kaufmann.
*ftp://psyche.mit.edu/pub/tommi/jaak-ul-bounds.ps.Z

[97] Jacobs, R. A. [1988], 'Increased rates of convergence through learning rate adaptation', *Neural Networks* **1**(4), 295–307.

[98] Jacobs, R. A. [1990], Task composition through competition in a modular connectionist architecture, PhD thesis, University of Massachusetts, Department of Computer and Information Science.

[99] Jacobs, R. A. [1995], 'Methods for combining experts probability assessments', *Neural Computation* **7**(5), 867–888.

[100] Jacobs, R. A. [1997], 'Bias/variance analyses of mixtures-of-experts architectures', *Neural Computation* **9**(2), 369–383.

[101] Jacobs, R. A. and Jordan, M. I. [1991], A competitive modular connectionist architecture, *in* Lippmann et al. [131], pp. 767–773.

[102] Jacobs, R. A. and Jordan, M. I. [1993], 'Learning piecewise control strategies in a modular neural-network architecture', *IEEE Transactions on Systems, Man, and Cybernetics* **23**(2), 337–345.

[103] Jacobs, R. A., Jordan, M. I. and Barto, A. G. [1991], 'Task decomposition through competition in a modular connectionist architecture - the What and Where vision tasks', *Cognitive Science* **15**(2), 219–250.

[104] Jacobs, R. A., Jordan, M. I., Nowlan, S. J. and Hinton, G. E. [1991], 'Adaptive mixtures of local experts', *Neural Computation* **3**(1), 79–87.

[105] Jacobs, R. A., Peng, F. C. and Tanner, M. A. [1997], 'A Bayesian approach to model selection in hierarchical mixtures-of-experts architectures', *Neural Networks* **10**(2), 231–241.

[106] Jensen, F. V. [1996], *An introduction to Bayesian networks*, University College London Press.

[107] Jordan, M. I. [1986], Serial order: A parallel distributed processing approach, ICS Report 8604, Institute for Cognitive Science, University of California, San Diego.

[108] Jordan, M. I. [1994], A statistical approach to decision tree modelling, *in* N.Warmuth, ed., 'Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory', ACM Press.

[109] Jordan, M. I. and Bishop, C. M. [1996], 'Neural networks', *ACM Computing Surveys* **28**(1), 73–75.

[110] Jordan, M. I., Ghahramani, Z. and Saul, L. K. [1997], Hidden Markov decision trees, *in* Mozer et al. [148], pp. 501–507.

[111] Jordan, M. I. and Jacobs, R. A. [1992], Hierarchies of adaptive experts, *in* J. E. Moody, S. J. Hanson and R. P. Lippmann, eds, 'Advances in Neural Information Processing Systems 4', Morgan Kaufmann, San Mateo, California, pp. 985–992.

[112] Jordan, M. and Jacobs, R. [1994], 'Hierarchical mixtures of experts and the EM algorithm', *Neural Computation* **6**(2), 181–214.

[113] Jordan, M. and Xu, L. [1995], 'Convergence results for the EM approach to mixtures of experts architectures', *Neural Networks* **8**(9), 1409–1431.

[114] Kang, K. and Oh, J. [1997], Statistical mechanics of the mixture of experts, *in* Mozer et al. [148], pp. 183–189.

[115] Kang, K., Oh, J. H. and Kwon, C. [1997], 'Learning by a population of perceptrons', *Physical Review E* **55**(3PtB), 3257–3261.

[116] Kecman, V. [1996], 'System identification using modular neural network with improved learning', *Proceedings of International Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing, NICROSP* pp. 40–48.

[117] Kehagias, A. and Petridis, V. [1997], 'Predictive modular neural networks for time series classification', *Neural Networks* **10**(1), 31–49.

[118] Kershaw, D. J., Robinson, A. J., Renals, S. J. and Hochberg, M. M. [1996], The 1995 ABBOT LVCSR system, *in* 'Proceedings of the ARPA Speech Recognition Workshop', Morgan Kauffman, 340 Pine St, 6th Floor, San Fransisco, CA, 94104.

[119] King, R. D., Feng, C. and Sutherland, A. [1995], 'STATLOG - comparison of classification algorithms on large real-world problems', *Applied Artificial Intelligence* **9**(3), 289–333.

[120] Knill, K. and Young, S. [1997], Hidden Markov models in speech and language processing, *in* S. Young and G. Bloothooft, eds, 'Corpus-Based Methods in Language and Speech Processing', Kluwer Academic Publishers, chapter 2, pp. 27–68.

[121] Kosaka, T. and Sagayama, S. [1994], Tree structured speaker clustering for fast speaker adaptation, *in* 'Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing', Vol. 1, pp. 245–248.

[122] Kubala, F. [1995], Design of the 1994 CSR benchmark tests, *in* 'Proceedings of the ARPA Speech Recognition Workshop', Morgan Kaufmann, 340 Pine St, 6th Floor, San Fransisco, CA, 94104, pp. 41–6.

[123] Kullback, S. and Liebler, R. A. [1951], 'On information and sufficiency', *Annals of Mathematics and Statistics* **22**, 79–86.

[124] Lamel, L. F., Kasel, R. H. and Seneff, S. [1987], Speech database development: design and analysis of the acoustic-phonetic corpus, *in* 'Proceedings of the DARPA Speech Recognition Workshop', Morgan Kaufmann, 340 Pine St, 6th Floor, San Fransisco, CA, 94104, pp. 26–32.

[125] Lau, K. N., Yang, C. H. and Post, G. V. [1996], 'Stochastic preference modeling within a switching regression framework', *Computers & Operations Research* **23**(12), 1163–1169.

[126] Lauritzen, S. L. [1996], *Graphical Models*, Oxford Statistical Science Series, Clarendon Press, Oxford.

[127] Leggetter, C. J. [1993], Maximum Likelihood Linear Regression for Speaker Adaptation of Continuous Density HMMs, PhD thesis, Cambridge University Engineering Department.

[128] Leggetter, C. J. and Woodland, P. C. [1995], 'Maximum likelihood linear regression for speaker adaptation of continuous density HMMs', *Computer Speech and Language* **9**, 171–186.

[129] Leite, J. A. F. and Hancock, E. R. [1997], 'Iterative curve organisation with the EM algorithm', *Pattern Recognition Letters* **18**(2), 143–155.

[130] Levinson, S. E., Rabiner, L. R. and Sondhi, M. M. [1983], 'An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition', *The Bell System Technical Journal* **62**(4), 1035–74.

[131] Lippmann, R. P., Moody, J. and Touretzky, D. S., eds [1991], *Advances in Neural Information Processing Systems 3*, Morgan Kaufmann, Redwood City, CA.

[132] MacKay, D. J. C. [1992*a*], 'The evidence framework applied to classification networks', *Neural Computation* **4**(5), 720–736.

[133] MacKay, D. J. C. [1992*b*], 'A practical Bayesian framework for backpropagation networks', *Neural Computation* **4**(3), 448–472.

[134] MacKay, D. J. C. [1995*a*], Developments in probabilistic modelling with neural networks—ensemble learning, *in* 'Neural Networks: Artificial Intelligence and Industrial Applications. Proceedings of the 3rd Annual Symposium on Neural Networks, Nijmegen, Netherlands, 14-15 September 1995', Springer, Berlin, pp. 191–198.

[135] MacKay, D. J. C. [1995*b*], 'Probable networks and plausible predictions - a review of practical Bayesian methods for supervised neural networks', *Network-Computation in Neural Systems* **6**(3), 469–505.

[136] MacKay, D. J. C. [1997], 'Comparison of approximate methods for handling hyperparameters', *Neural Computation* . In press.
*ftp://wol.ra.phy.cam.ac.uk/pub/mackay/abstracts/alpha.html

[137] Macphee, K. and MacKay, D. J. C. [1996], 'Ensemble learning for hidden Markov models'.
*ftp://wol.ra.phy.cam.ac.uk/pub/mackay/ensemblePaper.ps.gz

[138] Makhoul, J. [1975], 'Linear prediction. A tutorial review', *Proceedings of the IEEE* **63**(4), 561–580.

[139] Martin, A. [1997], Statistical significance tests for speech recognition benchmark tests. Unpublished draft. (email: alvin.martin@nist.gov).

[140] McCullagh, P. and Nelder, J. A. [1989], *Generalised Linear Models*, Monographs on Statistics and Applied Probability, second edn, Chapman and Hall, London.

[141] Meilă, M. and Jordan, M. I. [1996], Learning fine motion by Markov mixtures of experts, *in* Touretzky et al. [226], pp. 1003–1009.

[142] Meng, X. L. and van Dyk, D. [1997], 'The EM algorithm - an old folk-song sung to a fast new tune', *Journal of the Royal Statistical Society, Series B* **59**(3), 511–540.

[143] Merz, C. and Murphy, P. [1996], 'UCI Repository of machine learning databases, Irvine, CA: University of California, Department of Information and Computer Science.'.
*http://www.ics.uci.edu/ mlearn/MLRepository.html

[144] Michie, D., Spiegelhalter, D. J. and Taylor, C. C., eds [1994], *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, New York.

[145] Miller, D. and Rose, K. [1996], 'Hierarchical unsupervised learning with growing phase transitions', *Neural Computation* **8**, 425–450.

[146] Miller, D. and Uyar, S. [1997], A mixture of experts classifier with learning based on both labelled and unlabelled data, *in* Mozer et al. [148], pp. 571–578.

[147] Morgan, N. and Bourlard, H. A. [1990], Generalization and parameter estimation in feedforward nets: some experiments, *in* D. Touretzky, ed., 'Advances in Neural Information Processing Systems 2', Morgan Kaufmann, San Mateo, CA, pp. 630–637.

[148] Mozer, M. C., Jordan, M. I. and Petsche, T., eds [1997], *Advances in Neural Information Processing Systems 9*, MIT Press, 55 Hayward St., Cambridge, MA, 02142-1399.

[149] Murata, N., Yoshizawa, S. and Amari, S. [1994], 'Network information criterion - determining the number of hidden units for an artificial neural network', *IEEE Transactions on Neural Networks* **5**(6), 865–872.

[150] Murray-Smith, R. and Johansen, T. A. [1997], *Multiple model approaches to modelling and control*, Taylor and Francis, London.
*http://www.itk.ntnu.no/SINTEF/ansatte/Johansen_Tor.Arne/mmamc/mmamc_book.html

[151] Nadas, A. [1995], 'Binary classification by stochastic neural nets', *IEEE Transactions on Neural Networks* **6**(2), 488–491.

[152] Neal, R. M. [1991], Bayesian mixture modelling, *in* C. R. Smith, G. J. Erickson and P. O. Neudorfer, eds, 'Maximum Entropy and Bayesian Methods, Seattle 1991', Kluwer, Dordrecht, pp. 197–211.

[153] Neal, R. M. [1996], *Bayesian learning for neural networks*, Springer-Verlag.

[154] Neal, R. M. and Hinton, G. E. [1993], 'A new view of the EM algorithm that justifies incremental and other variants'.
*http://www.cs.toronto.edu/∼radford/

[155] Neto, J., Almeida, L., Hochberg, M. M., Martins, C., Nunes, L., Renals, S. J. and Robinson, A. J. [1995], Speaker-adaptation for hybrid HMM-ANN continuous speech recognition system, *in* 'Proceedings of the European Conference on Speech Technology', pp. 2171–4.

[156] Nikovski, D. [1995], Adaptive computation techniques for time series prediction, Master's thesis, Department of Computer Science, Southern Illinois University at Carbondale.
*http://www.cs.cmu.edu/∼danieln/thesis.ps.Z

[157] Nikovski, D. and Zargham, M. [1996], Comparison of two learning networks for time series prediction, *in* 'Proceedings of the Ninth International Conference IEA/AIE 96 (Industrial and Engineering Applications of Artificial Intelligence and Expert Systems), Fukuoka, Japan', pp. 531–555.
*http://www.cs.cmu.edu/∼danieln/hmerbf.ps.Z

[158] Noordewier, M. O., Towell, G. G. and Shavlik, J. W. [1991], Training knowledge-based neural networks to recognize genes in DNA sequences, *in* Lippmann et al. [131], pp. 530–536.

[159] Nowlan, S. J. [1991], Soft competitive adaptation: neural network learning algorithms based on fitting statistical mixtures, PhD thesis, Carnegie Mellon University. CS–91–126.

[160] Nowlan, S. J. and Hinton, G. E. [1991], Evaluation of adaptive mixtures of competing experts, *in* Lippmann et al. [131].

[161] Nowlan, S. J. and Hinton, G. E. [1992], 'Simplifying neural networks by soft weight-sharing', *Neural Computation* **4**(4), 473–493.

[162] Ó Ruanaidh, J. J. K. and Fitzgerald, W. J. [1996], *Numerical Bayesian methods applied to signal processing*, Springer Verlag, New York.

[163] Page, E. S. [1955], 'A test for a change in a parameter occurring at an unknown point', *Biometrika* **42**, 523–527.

[164] Page, E. S. [1957], 'On problems in which a change in a parameter occurs at an unknown point', *Biometrika* **44**, 248–52.

[165] Pallett, D. S., Fiscus, J. G., Fisher, W. M., Garofolo, J. S., Lund, B. A., Martin, A. and Przybocki, M. A. [1995], 1994 benchmark test for the ARPA spoken language program, *in* 'Proceedings of the ARPA Spoken Language Systems Technology Workshop', Morgan Kauffman, 340 Pine St, 6th Floor, San Fransisco, CA, 94104.

[166] Pallett, D. S., Fiscus, J. G., Fisher, W. M., Garofolo, J. S., Lund, B. A. and Pryzbocki, M. A. [1994], 1993 benchmark tests for the ARPA spoken language program, *in* 'Proceedings of the ARPA Workshop on Human Language Technology', Morgan Kauffman, pp. 51–73. Merrill Lynch Conference Center, Plainsboro, NJ.

[167] Pallett, D. S., Fiscus, J. G., Fisher, W. M., Garoforo, J. S., Martin, A. F. and Przybocki, M. A. [1996], 1995 Hub-3 NIST multiple microphone corpus benchmark tests, *in* 'Proceedings of the ARPA Speech Recognition Workshop', Morgan Kauffman, 340 Pine St, 6th Floor, San Fransisco, CA, 94104.

[168] Parisi, G. [1988], *Statistical Field Theory*, Addison-Wesley, Redwood City, CA.

[169] Paul, D. B. [1992], An efficient A* stack decoder algorithm for continuous speech recognition with a stochastic language model, *in* 'Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing', Vol. 1, pp. 25–28.

[170] Paul, D. B. and Baker, J. M. [1992], The design for the Wall Street Journal-based CSR corpus, *in* 'Proceedings of the DARPA Speech Recognition Workshop', Morgan Kaufmann, 340 Pine St, 6th Floor, San Fransisco, CA, 94104, pp. 357–62.

[171] Pawelzik, K., Kohlmorgen, J. and Muller, K. R. [1996], 'Annealed competition of experts for a segmentation and classification of switching dynamics', *Neural Computation* **8**(2), 340–356.

[172] Peng, F. C., Jacobs, R. A. and Tanner, M. A. [1996], 'Bayesian-inference in mixtures-of-experts and hierarchical mixtures- of-experts models with an application to speech recognition', *Journal of the American Statistical Association* **91**(435), 953–960.

[173] Penny, W. D. and Roberts, S. J. [1997], Neural network predictions with error bars. Submitted to IEEE Transactions on Neural Networks.

[174] Perrone, M. P. and Cooper, L. N. [1993], When networks disagree: Ensemble methods for hybrid neural networks, *in* 'Neural Networks for Speech and Image Processing', Chapman-Hall.

[175] Peterson, G. E. and Barney, H. L. [1952], 'Control methods used in a study of vowels', *Journal of the Acoustical Society of America* **24**, 175–184.

[176] Petridis, V. and Kehagias, A. [1996], 'Modular neural networks for MAP classification of time-series and the partition algorithm', *IEEE Transactions on Neural Networks* **7**(1), 73–86.

[177] Plate, T., Bert, J., Grace, J. and Band, P. [1997], A comparison between neural networks and other statistical techniques for modelling the relationship between tobacco and alcohol and cancer, *in* M. Mozer, M. Jordan and T. Petsche, eds, 'Advances in Neural Information Processing 9 (NIPS*96)', MIT Press.

[178] Prechelt, L. [1995], 'Some notes on neural learning algorithm benchmarking', *Neurocomputing* **9**(3), 343–347.

[179] Prechelt, L. [1996], 'A quantitative study of experimental evaluations of neural-network learning algorithms - current research practice', *Neural Networks* **9**(3), 457–462.

[180] Press, W. H., Flannery, B. P., Teukolsky, S. A. and Vetterling, W. T. [1988], *Numerical Recipes in C*, Cambridge.

[181] Price, P., Fisher, W. M., Bernstein, J. and Pallett, D. S. [1988], The DARPA 1000-word Resource Management database for continuous speech recognition, *in* 'Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing', Vol. 1, pp. 651–654.

[182] Quandt, R. E. [1958], 'The estimation of the parameters of a linear regression system obeying two separate regimes', *Journal of the American Statistical Association* **53**, 873–80.

[183] Quandt, R. E. [1972], 'A new approach to estimating switching regressions', *Journal of the American Statistical Association* **67**, 306–310.

[184] Quandt, R. E. and Ramsey, J. B. [1978], 'Estimating mixtures of normal distributions and switching regressions', *Journal of the American Statistical Association* **73**(364), 730–752.

[185] Quinlan, J. R. [1986], 'Induction of decision trees', *Machine Learning* **1**, 81–106.

[186] Quinlan, J. R. [1993], *C4. 5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA.

[187] Ramamurti, V. and Ghosh, J. [1997*a*], 'Structural adaptation in mixtures of experts'. *ftp://ftp.lans.ece.utexas.edu/pub/papers/struct_adapt_ICPR96.ps.Z

[188] Ramamurti, V. and Ghosh, J. [1997*b*], 'Structurally adaptive modular networks for non-stationary environments'. Submitted to the IEEE Transactions on Neural Networks. *ftp://ftp.lans.ece.utexas.edu/pub/papers/struct_adapt_IEEETNN.ps.Z

[189] Rasmussen, C. E. [1996], Evaluation of Gaussian Processes and other methods for non-linear regression, PhD thesis, University of Toronto. *http://www.cs.toronto.edu/∼carl/

[190] Rasmussen, C. E., Neal, R. M., Hinton, G. E., van Camp, D., Revow, M., Ghahramani, Z., Kustra, R. and Tibshirani, R. [1996], The DELVE manual, (*version 1.1*), Technical report, University of Toronto.
*http://www.cs.utoronto.ca/∼delve/

[191] Renals, S. and Hochberg, M. [1997], Decoder technology for connectionist large vocabulary speech recognition, Technical Report CS-95-17 (revised), Dept. of Computer Science, University of Sheffield.

[192] Renals, S. J. and MacKay, D. J. C. [1993], Bayesian regularisation methods in a hybrid MLP-HMM system, *in* 'Proceedings of the European Conference on Speech Technology'.

[193] Ripley, B. D. [1993], Statistical aspects of neural networks, *in* O. E. Barndorff-Nielsen, J. L. Jensen and W. S. Kendall, eds, 'Networks and Chaos—Statistical and Probabilistic Aspects', Chapman & Hall, London, pp. 40–123.

[194] Ripley, B. D. [1994], 'Neural networks and related methods for classification', *Journal of the Royal Statistical Society, Series B* **56**(3), 409–437.

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

[196] Rissanen, J. [1983], 'A universal prior for integers and estimation by minimum description length', *Annals of Statistics* **11**(2), 416–431.

[197] Robinson, A. J. [1989], Dynamic Error Propagation Networks, PhD thesis, Cambridge University Engineering Department.

[198] Robinson, A. J. [1994], 'An application of recurrent nets to phone probability estimation', *IEEE Transactions on Neural Networks* **5**(2), 298–305.

[199] Robinson, A. J., Hochberg, M. and Renals, S. [1995], The use of recurrent neural networks in continuous speech recognition, *in* C. H. Lee, K. K. Paliwal and F. K. Soong, eds, 'Automatic Speech and Speaker Recognition – Advanced Topics', Kluwer Academic Publishers, chapter 19, pp. 159–184.

[200] Roy, H. S. [1995], Sharp, reliable predictions using supervised mixture models, PhD thesis, Department of Computer Science, Stanford University. Tech report number CS-95-1547.
*http://forumweb.stanford.edu/Abstracts/CSD/CSD-1547.html

[201] Rumelhart, D. E., Hinton, G. E. and Williams, R. J. [1986], Learning internal representations by error propagation, *in* D. E. Rumelhart and J. L. McClelland, eds, 'Parallel Distributed Processing: Explorations in the Microstructure of Cognition', Vol. I: Foundations, MIT Press/Bradford Books, Cambridge, MA, pp. 318–362.

[202] Russell, S. J. and Norvig, P. [1995], *Artificial Intelligence. A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ.

[203] Sabes, P. N. and Jordan, M. I. [1996], Reinforcement learning by probability matching, *in* Touretzky et al. [226].

[204] Saito, K. and Nakano, R. [1996], A constructive learning algorithm for an HME, *in* 'Proceedings of the IEEE International Conference on Neural Networks', pp. 1268–1273.

[205] Salzberg, S. L. [1997], 'On comparing classifiers: pitfalls to avoid and a recommended approach', *Data Mining and Knowledge Discovery* **1**(3).
*http://www.cs.bham.ac.uk/∼anp/TheDataMine.html

[206] Saul, L. and Jordan, M. I. [1994], 'Learning in Boltzmann trees', *Neural Computation* **6**(6), 1174–1184.

[207] Saul, L. K., Jaakkola, T. and Jordan, M. I. [1996], 'Mean-field theory for sigmoid belief networks', *Journal of Artificial Intelligence Research* **4**, 61–76.

[208] Saul, L. K. and Jordan, M. I. [1995], Boltzmann chains and hidden Markov models, *in* Tesauro et al. [218], pp. 435–442.

[209] Saul, L. K. and Jordan, M. I. [1996], Exploiting tractable substructures in intractable networks, *in* Touretzky et al. [226], pp. 486–492.

[210] Schaal, S. and Atkeson, C. G. [1996], From isolation to cooperation: an alternative view of a system of experts, *in* Touretzky et al. [226], pp. 486–492.

[211] Schiffmann, W., Joost, M. and Werner, R. [1992], Optimization of the backpropagation algorithm for training multilayer perceptrons, Technical report, University of Koblenz.

[212] Sejnowski, T. J. and Rosenberg, C. R. [1986], 'Parallel networks that learns to pronounce English text', *Complex Systems* **1**, 145–168.

[213] Shavlik, J. W., Mooney, R. J. and Towell, G. G. [1991], 'Symbolic and neural learning algorithms - an experimental comparison', *Machine Learning* **6**(2), 111–143.

[214] Singh, S. P. [1992], 'Transfer of learning by composing solutions of elemental sequential tasks', *Machine Learning* **8**(3-4), 323–339.

[215] Spiegelhalter, D. J., Dawid, A. P., Lauritzen, S. L. and Cowell, R. G. [1993], 'Bayesian analysis in expert systems', *Statistical Science* **8**(3), 219–283.

[216] Stone, M. [1974], 'Cross-validatory choice and assessment of statistical predictions (with discussion)', *Journal of the Royal Statistical Society, Series B* **36**, 111–147.

[217] Sutton, R. S. [1992], 'Introduction - the challenge of reinforcement learning', *Machine Learning* **8**(3-4), 225–227.

[218] Tesauro, G., Touretzky, D. S. and Leen, T. K., eds [1995], *Advances in Neural Information Processing Systems 7*, MIT Press, 55 Hayward St., Cambridge, MA, 02142-1399.

[219] Tham, C. K. [1995*a*], On-line learning using hierarchical mixtures of experts, *in* 'IEE Conference on Artificial Neural Networks', pp. 347–351.

[220] Tham, C. K. [1995*b*], 'Reinforcement learning of multiple tasks using a hierarchical CMAC architecture', *Robotics and Autonomous Systems* **15**(4), 247–274.

[221] Thodberg, H. H. [1996], 'Review of Bayesian neural networks with an application to near- infrared spectroscopy', *IEEE Transactions on Neural Networks* **7**(1), 56–72.

[222] Thomas, A., Spiegelhalter, D. J. and Gilks, W. R. [1992], BUGS: A program to perform Bayesian inference using Gibbs sampling, *in* J. M. Bernardo, J. O. Berger, A. P. Dawid and A. F. M. Smith, eds, 'Bayesian Statistics 4', Clarendon Press, Oxford, pp. 837–842.

[223] Tichy, W. F., Lukowicz, P., Prechelt, L. and Heinz, E. A. [1995], 'Experimental evaluation in computer-science - a quantitative study', *Journal of Systems and Software* **28**(1), 9–18.

[224] Titterington, D. M., Smith, A. F. M. and Makov, U. E. [1985], *Statistical Analysis of Finite Mixture Distributions*, John Wiley, New York.

[225] Tong, H. and Lim, K. S. [1980], 'Threshold autoregression, limit cycles and cyclical data', *Journal of the Royal Statistical Society, Series B* **42**, 245–292.

[226] Touretzky, D. S., Mozer, M. C. and Hasselmo, M. E., eds [1996], *Advances in Neural Information Processing Systems 8*, MIT Press, 55 Hayward St., Cambridge, MA, 02142-1399.

[227] Tresp, V., Hollatz, J. and Ahmad, S. [1997], 'Representing probabilistic rules with networks of Gaussian basis functions', *Machine Learning* **27**(2), 173–200.

[228] Utsugi, A. [1996], 'Topology selection for self-organizing maps', *Network-Computation in Neural Systems* **7**(4), 727–740.

[229] Utsugi, A. [1997], 'Hyperparameter selection for self-organizing maps', *Neural Computation* **9**(3), 623–635.

[230] Waibel, A., Finke, M., Gates, D., Gavalda, M., Kemp, T., Lavie, A., Levin, L., Maier, M., L., M., McNair, A., Rogina, I., Shima, A., Sloboda, T., Woszczyna, M., Zeppenfeld, T. and Zhan, P. [1996], JANUS-II - Advances in spontaneous speech translation, *in* 'Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing', pp. 409–412.

[231] Waibel, A., Sawai, H. and Shikano, K. [1989], 'Modularity and scaling in large phonemic neural networks', *IEEE Transactions on Acoustics, Speech, and Signal Processing* **37**(12), 1888–97.

[232] Waterhouse, S. R. [1993], Speech recognition using hierarchical mixtures of experts, Master's thesis, Cambridge University Engineering Department, Trumpington Street, Cambridge CB2 1PZ, UK.

[233] Waterhouse, S. R. and Cook, G. D. [1997], Ensemble methods for phoneme classification, *in* Mozer et al. [148], pp. 800–806.

[234] Waterhouse, S. R., Kershaw, D. J. and Robinson, A. J. [1996], Smoothed local adaptation of connectionist systems, *in* 'Proceedings of the International Conference on Spoken Language Processing'.

[235] Waterhouse, S. R., MacKay, D. J. C. and Robinson, A. J. [1996], Bayesian methods for mixtures of experts, *in* Touretzky et al. [226], pp. 351–357.

[236] Waterhouse, S. R. and Robinson, A. J. [1994], Classification using hierarchical mixtures of experts, *in* 'Proceedings of the IEEE Workshop on Neural Networks for Signal Processing', IEEE Press, Long Beach, CA, pp. 177–186.

[237] Waterhouse, S. R. and Robinson, A. J. [1995], Non-linear prediction of acoustic vectors using hierarchical mixtures of experts, *in* Tesauro et al. [218], pp. 835–842.

[238] Waterhouse, S. R. and Robinson, A. J. [1996], Constructive algorithms for hierarchical mixtures of experts, *in* Touretzky et al. [226], pp. 584–590.

[239] Wawrzynek, J., Asanovic, K., Kingsbury, B., Beck, J., Johnson, D. and Morgan, N. [1996], SPERT-II: A vector microprocessor system and its application to large problems in backpropagation training, *in* Touretzky et al. [226].

[240] Weigend, A. S. [1996], 'Time series analysis and prediction using gated experts with application to energy demand forecasts', *Applied Artificial Intelligence* **10**(6), 583–624.

[241] Weigend, A. S., Huberman, B. A. and Rumelhart, D. E. [1992], Predicting sunspots and exchange rates with connectionist networks, *in* M. Casdagli and S. Eubank, eds, 'Nonlinear Modelling and Forecasting', Addison-Wesley, pp. 395–432.

[242] Weigend, A. S., Mangeas, M. and Srivastava, A. N. [1995], 'Nonlinear gated experts for time-series - discovering regimes and avoiding overfitting', *International Journal of Neural Systems* **6**(4), 373–399.

[243] Weiss, Y. and Adelson, E. H. [1995], 'Motion estimation and segmentation using a recurrent mixture of experts architecture', *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing* pp. 293–302.

[244] Werbos, P. J. [1974], Beyond Regression: New Tools for Regression and Analysis in the Behavioral Sciences, PhD thesis, Harvard University, Division of Engineering and Applied Physics. Also available as "The roots of backpropagation: from ordered derivatives to neural networks and political forecasting", published by Wiley, New York NY.

[245] Werbos, P. J. [1990], 'Backpropagation through time: What it does and how to do it', *Proceedings of the IEEE* **78**(10), 1550–1560.

[246] Wolberg, W. H., Tanner, M. A., Loh, W. Y. and Vanichsetakul, N. [1987], 'Statistical approach to fine needle aspiration diagnosis of breast masses', *Acto Cytologica* **31**, 737–741.

[247] Woodland, P., Leggetter, C., an d V. Valtchev, J. O. and Young, S. [1995], The development of the 1994 HTK large vocabulary speech recognition system, *in* 'Spoken Language Systems Technology Workshop', pp. 104–9.

[248] Xu, L., Hinton, G. and Jordan, M. I. [1995], An alternative model for mixtures of experts, *in* Tesauro et al. [218], pp. 633–640.

[249] Xu, L. and Jordan, M. I. [1993], EM learning of a generalized finite mixture model for combining multiple classifiers, *in* 'Proceedings of the World Congress on Neural Networks', pp. 227–230.

[250] Young, S. J., Jansen, J., Odell, J. J., Ollason, D. and Woodland, P. C. [1995], *The HTK Hidden Markov Model Toolkit Book*, Entropic Cambridge Research Laboratory.
*http://www.entropic.com/htk.html

[251] Yuille, A. L. and Kosowsky, J. J. [1994], 'Statistical physics algorithms that converge', *Neural Computation* **6**(3), 341–356.

[252] Yuille, A., Stolorz, P. and Utans, J. [1994], 'Statistical physics, mixtures of distributions, and the EM algorithm', *Neural Computation* **6**(2), 334–340.

[253] Zangwill, W. [1969], *Nonlinear programming: a unified approach*, Prentice-Hall, Englewood Cliffs, New Jersey.

[254] Zeevi, A. J., Meir, R. and Adler, R. J. [1997*a*], Non-linear models for time series using mixtures of experts, Technical Report CC-150, Faculty of Electrical Engineering, Technion, Haifa 32000, Israel.
*http://www.ee.technion.ac.il/publications/Neural-Networks/tseries.ps

[255] Zeevi, A. J., Meir, R. and Maiorov, V. [1997], Error bounds for functional approximation and estimation using mixtures of experts, Technical Report CC-132, Faculty of Electrical Engineering, Technion, Haifa 32000, Israel.
*http://www.ee.technion.ac.il/publications/Neural-Networks/mix_experts.ps

[256] Zeevi, A., Meir, R. and Adler, R. [1997*b*], Time series prediction using mixtures of experts, *in* Mozer et al. [148], pp. 309–315.

[257] Zhao, Y. [1994], 'An acoustic-phonetic based speaker adaptation technique for improving speaker independent continuous speech recognition', *IEEE Transactions on Acoustics, Speech, and Signal Processing* **2**(3), 380–394.

[258] Zhao, Y., Schwartz, R., Sroka, J. and Makhoul, J. [1995], Hierarchical mixtures of experts methodology applied to continuous speech recognition, *in* Tesauro et al. [218].

[259] Zhu, H. Y. and Rohwer, R. [1996], 'Bayesian regression filters and the issue of priors', *Neural Computing & Applications* **4**(3), 130–142.
*ftp://cs.aston.ac.uk/neural/zhuh/reg_fil_prior.ps.Z