Distributed Machine Learning

PIERRE LATOUCHE

MSc by Research in Pattern Analysis and Neural Networks

ASTON UNIVERSITY

September 2007

Acknowledgment

I would like to express my deep sense of gratitude to **professor Ian T. Nabney**, for his help and guidance during the course of this project. I am highly indebted to him for constantly encouraging me by giving his critics on my work. I am grateful to him for having given me the support and confidence I needed.

Pierre Latouche
September 2007
Aston University, Birmingham (UK)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last ten years, there has been an ever increasing use of databases, to store information, and Machine Learning methods to manipulate, extract, and analyse data. More and more problems are being tackled in science, health and engineering. As a consequence, there has been a concurrent increase in the use of highly distributed computing to store and manipulate data.

In this thesis, we work on regression problems that consist of approximating underlying processes that map input variables to target variables. We introduce the concept of *distributed learning environment* where local agents use distributed data to train and we show that two critical applications can be tackled using such architectures. First, in Chapter 3, we consider a situation where data is originally physically distributed on nodes. The agents do not agree to share their data for privacy and security reasons but do agree to share their models. In this environment, the issue is to combine the learned information in order to build a more accurate preditive model. For our experiments, we consider multilayer perceptrons and radial basis function networks. We test some model combination methods using a toy dataset and some scatterometry data.

Then, in Chapter 4, we tackle Gaussian processes that are known to have a poor scaling with large data sets since they require matrix inversions of which the computational cost and memory requirement are of order $O(N)^3$ and $O(N^2)$ respectively where N is the number of training data points. We investigate techniques that consist of splitting and then distributing the data on nodes. Thus, we show that the Bayesian committee machine can be applied to estimate Gaussian process predictions whereas a factorized hyperposterior can lead to optimization procedures over the whole training data set even if N is large. We experiment with these approximations using the scatterometry data.

Chapter 2

Problem Statement

In this Chapter, we define some concepts and techniques that are used in Chapter 3 and 4. More precisely, in Section 2.1, we review regression problems and we focus on neural networks, linear regression models, and Gaussian processes. Then, we describe some distributed learning environments and two critical applications that can be tackled using such architectures. Finally, we present the two data sets that we used to carry out our experiments.

2.1 A Review of Regression Problems

In statistics, data related to a specific problem is usually described by many variables $x^1, x^2, \ldots, x^p, t^1, t^2, \ldots, t^q$. Throughout this thesis, x^i will represent a one dimensional real variable and *not* to the function $x \rightarrow g(x) = x^i$. Moreover, we define $x_i = (x_i^1, x_i^2, \ldots, x_i^p)'$ as the ith p-dimensional vector in a given data set.

The aim of regression methods is to approximate the underlying process that maps input variables to target (output) variables (Bishop, 2006 [1]). It can be modelled by the general equation

$$t = f(x) + \varepsilon, \quad x = (x^1, x^2, \ldots, x^p)' \in \mathbb{R}^p,\ t = (t^1, t^2, \ldots, t^q)' \in \mathbb{R}^q.$$

The unknown function f governs the deterministic part of the mapping whereas ε is an additive zero mean random variable with unknown σ^2 variance called *noise*. For consistency with the notation used in other statistical domains, the variance can be defined as $\sigma^2 = \frac{1}{\beta}$. Moreover, we will assume that the noise is drawn from a Gaussian distribution. When several input points $\{x_i\}$ are considered, such that $t_i = f(x_i) + \varepsilon_i$, $\forall i$, we will assume that the different noise variables are independent. In other words, we wil have $\text{cov}(t_i, t_j) = 0$, $\forall i \neq j$.

Given a training data set $\{(x_1, t_1); (x_2, t_2); \ldots; (x_N, t_N)\}$, the goal is to predict the value of t for a new value of x. In the literature, this data set is sometimes separated between $\chi = (x_1, x_2, \ldots, x_N)'$ and $T = (t_1, t_2, \ldots, t_N)'$.

The functional form of $f(.)$ is usually unknown (Pena and Redondas, 2004 [13]). To approximate the mapping, *linear regression models* and *neural networks* use parametric approaches and define $f(.) = f(w, .)$. Conversely, as in the *Gaussian Process* (GP) framework, other types of model make predictions without giving the unknown function an explicit parameterization (Mackay, 1998 [8]).

2.1.1 Linear regression models

If we consider a *single* target variable, the general form of linear regression models is

$$t = f(w, x) + \varepsilon = w'\phi(x) + \varepsilon.$$

In this equation, the vector $\phi(x) = (\phi_0(x), \phi_1(x), \ldots, \phi_d(x))'$ is defined with a set of fixed non-linear basis functions. To get a bias term, it is often convenient to consider $\phi_0(x) = 1$. Functions of the form $f(w, x) = w'\phi(x)$ are called linear models since they are linear with respect to the weight vector $w = (w_0, w_1, \ldots, w_d)'$.

The simplest choice for the basis functions is $\phi_i(x) = x^i$, $\forall i$. Thus, we obtain $f(w, x) = w_0 + w'x$. This regression method is often called *Multivariate Linear Regression* in the literature.

If we define the model such that each basis function $\phi_i(.)$ depends only on the radial distance from a centre x_i, we obtain a *Radial Basis Function* (RBF) network $f(w, x) = \sum_{i=1}^{m} w_i \phi(||x - x_i||)$. Clustering techniques such as K-means and the Expectation Maximization algorithm (Dempster et al., 1977 [3]) have been commonly used to fix the centres (Nabney, 2003 [10]).

After having chosen the model, the aim of training is to build an estimate $\hat{t}(x)$, called a *predictive model*, of the underlying process $t = f(w, x) + \varepsilon$ using a given data set called the *training data set*.

Maximum likelihood

The most common training algorithm is based on maximum likelihood theory.

Since we assume that the additive noise ε is drawn from a zero mean $\frac{1}{\beta}$ variance Gaussian distribution $N\left(\varepsilon; 0, \frac{1}{\beta}\right)$ and $t = f(w,x) + \varepsilon$, the conditional probability density $p(t|x,w,\beta)$ is given by

$$p(t|x,w,\beta) = N\left(t; f(w,x), \frac{1}{\beta}\right).$$

Moreover, if we assume that the observations of the training data set are i.i.d[1], the complete log-likelihood is given by

$$\ln p(T|\chi,w,\beta) = \sum_{n=1}^{N} \ln N\left(t_n; f(w,x_n), \frac{1}{\beta}\right) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta E(w),$$

where $E(w)$ is the *sum-of-squares error function* defined by

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - f(w,x_n)\}^2.$$

Maximizing the complete log-likelihood function with respect to w is equivalent to minimizing $E(w)$. Since the model considered is linear, $\nabla E(w)$ has a simple form

$$\nabla E(w) = \sum_{n=1}^{N}\{t_n - w'\phi(x_n)\}\phi(x_n)',$$

and $\nabla E(w) = 0$ admits a unique solution $w_{ML} = (\Phi'\Phi)^{-1}\Phi'T$ containing all the hidden parameters of the model. The matrix Φ is usually called the *design matrix* and is defined such that $\Phi_{ni} = \phi_i(x_n)$. If we now differentiate the complete log-likelihood function with respect to β, we can create a noise variance estimator

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{t_n - w'_{ML}\phi(x_n)\}^2.$$

Finally, our predictive model is $\hat{t}(x) \sim p(t|x,w_{ML},\beta_{ML},\chi,T) = N\left(t; f(w_{ML},x), \frac{1}{\beta_{ML}}\right)$. In the literature, the distribution of $\hat{t}(x)$ is often called the *predictive distribution.*

[1]Independently and identically distributed.

Regularized maximum likelihood

We now introduce a prior probability over the weights $p(w)$. If we assume that this distribution is an isotropic Gaussian distribution governed by an *hyperparameter* α, we have

$$p(w|\alpha) = N\left(w; 0, \frac{I}{\alpha}\right).$$

Using Bayes' theorem we can compute the posterior distribution over the weights

$$p(w|\chi, T, \alpha, \beta) \propto p(T|\chi, w, \beta)p(w|\alpha). \tag{2.1}$$

Since the model considered is linear, this distribution is also Gaussian and thus its mode corresponds to its mean.

Maximizing (2.1) with respect to w is equivalent to minimizing the *regularized sum-of-squares error function*

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - f(w, x_n)\}^2 + \frac{\lambda}{2}w'w,$$

where $\lambda = \frac{\alpha}{\beta}$. As before, we set the gradient of $E(w)$ to zero. The maximum *a posteriori* solution is given by $w_{MP} = (\lambda I + \Phi'\Phi)^{-1}\Phi'T$.

The evidence procedure

We now consider the usual case where the hyperparameter α which governs the prior over the weights is not given. Moreover, contrary to the previous methods where we defined a predictive model by using a single value for w (w_{ML}, w_{MP}), we now integrate over all the hidden parameters in order to take into account our uncertainty as to which one that is. The predictive distribution of $\hat{t}(x)$ is then given by

$$p(t|x, \chi, T) = \int\int\int p(t|x, w, \beta)p(w|\chi, T, \alpha, \beta)p(\alpha, \beta|\chi, T)\, dw\, d\alpha\, d\beta.$$

However, this complete marginalization is analytically intractable. In order to approximate this integral, a technique called the *evidence approximation* was developed in the machine learning community (MacKay, 1992 [7]). If we assume that $p(\alpha, \beta|\chi, T)$ is sharply peaked around $\hat{\alpha}$ and $\hat{\beta}$

$$p(\alpha, \beta|\chi, T) = \delta_{\hat{\alpha},\hat{\beta}}(\alpha, \beta),$$

then the predictive distribution is simply obtained by marginalizing over the weight vector w

$$p(t|x,\chi,T) \simeq p(t|x,\hat{\alpha},\hat{\beta},\chi,T) = \int p(t|x,w,\hat{\beta})p(w|\chi,T,\hat{\alpha},\hat{\beta})\,dw. \tag{2.2}$$

The posterior distribution $p(\alpha,\beta|\chi,T)$ is given by

$$p(\alpha,\beta|\chi,T) \propto p(T|\chi,\alpha,\beta)p(\alpha,\beta).$$

Thus, if we assume that the *hyperprior* $p(\alpha,\beta)$ is flat, maximizing $p(\alpha,\beta|\chi,T)$ to find $\hat{\alpha}$ and $\hat{\beta}$ is equivalent to maximizing the *marginal likelihood*[2] $p(T|\chi,\alpha,\beta)$.

Since the model is linear, an analytical equation can be obtained (Bishop, 2006 [1])

$$\ln p(T|\chi,\alpha,\beta) = \frac{M}{2}\ln\alpha + \frac{N}{2}\ln\beta - E(m_N) - \frac{1}{2}\ln\det A - \frac{N}{2}\ln(2\pi), \tag{2.3}$$

where M is the dimensionality of w, $A = \alpha I + \beta\Phi'\Phi$, $m_N = \beta A^{-1}\Phi' T$ is the mean of the posterior distribution over the weights, and

$$E(m_N) = \frac{\beta}{2}\|T - \Phi m_N\|^2 + \frac{\alpha}{2}m_N' m_N.$$

In order to maximize the marginal likelihood, we consider the eigenvalues λ_i of the matrix $\beta\Phi'\Phi$. From its definition, we know that the eigenvalues of A have got the form $v_i = \alpha + \lambda_i$. Thus, $\ln\det A$ can be written as

$$\ln\det A = \ln\prod_i(\lambda_i+\alpha) = \sum_i \ln(\lambda_i+\alpha).$$

Then, by setting the derivatives of (2.3), with respect to α and β, to zero, we obtain the solutions $\alpha = \frac{\gamma}{m_N' m_N}$ and $\frac{1}{\beta} = \frac{1}{N-\gamma}\sum_{n=1}^{N}\{t_n - m_N'\phi(x_n)\}^2$ where $\gamma = \sum_i \frac{\lambda_i}{\alpha+\lambda_i}$. It must be noted that both terms depend on α and β through γ and the eigenvalue decomposition of A. By first initializing α and β, we can create an iterative algorithm. The vector m_N and the parameter γ are calculated and then used to optimize α and β. This procedure is repeated until convergence.

Finally, after a few iterations, we obtain solutions $\hat{\alpha}$ and $\hat{\beta}$. Since the model is linear, the integral in (2.2) has got a simple form and the predictive model is given by

$$\hat{t}(x) \sim p(t|x,\chi,T) = N\left(t; m_N'\phi(x), \sigma_N^2(x)\right),$$

[2]The marginal likelihood is also called the model evidence.

where $\sigma_N^2(x) = \hat{\beta}^{-1} + \phi(x)' A^{-1} \phi(x)$ and since m_N is the mean of the posterior distribution, we have $m_N = w_{MP}$.

2.1.2 Neural networks

Contrary to linear models, neural networks are based on non-linear basis functions, usually called *hidden units*, which depend on parameters that can be adjusted. Through this thesis, we only consider multilayer perceptrons. In regression, the global network model can then be written as (Nabney, 2003 [10])

$$t = f(w, x) + \varepsilon = \sum_{j=0}^{M} \left(w_j^{(2)} \phi_j (\sum_{i=0}^{D} w_{ji}^{(1)} x^i) \right) + \varepsilon.$$

Optimization of w and β

Unlike linear regression models, the posterior over the weights of multilayer perceptrons is *not* Gaussian and they do not have analytical solutions for the maximum likelihood estimates $\{w_{ML}, \beta_{ML}\}$ and the regularized maximum likelihood estimates $\{w_{MP}, \beta_{MP}\}$. Thus, for the training process, these models require to use an optimization algorithm such as scale conjugate gradient.

Bayesian neural networks

Multilayer perceptrons are commonly called Bayesian neural networks when they use the evidence procedure. Since the posterior over the weights is not Gaussian, the integral in (2.2) is analytically intractable. Thus, we use a Laplace approximation that estimates the posterior with

$$p(w|\chi, T, \hat{\alpha}, \hat{\beta}) \approx N\left(w; w_{MP}, H(w_{MP})^{-1}\right),$$

where H is the hessian matrix of $-\ln p(w|\chi, T, \hat{\alpha}, \hat{\beta})$. Moreover, we linearize the model using a first order Taylor expansion

$$t \approx f(w_{MP}, x) + \nabla f(w_{MP}, x)'(w - w_{MP}) + \varepsilon.$$

Thus, t is defined as a linear combination of Gaussian random variables and the corresponding predictive distribution is given by

$$p(t|x, \chi_j, T_j) \approx N\left(t; f(w_{MP}, x), \nabla f(w_{MP})' H(w_{MP})^{-1} \nabla f(w_{MP}) + \beta^{-1}\right).$$

This is a good approximation only if the number of training data points considered is sufficiently large such that $p(w|\chi_j, T_j, \hat{\alpha}, \hat{\beta})$ is sharply peaked around w_{MP}.

To optimize the hyperparameters, we use a Laplace approximation and we compute

$$\ln p(T|\chi, \alpha, \beta) \approx -E(w_{MP}) - \frac{1}{2} \ln \det A + \frac{W}{2} \ln \alpha + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi),$$

where W is the total number of parameters in w and

$$E(w_{MP}) = \frac{\beta}{2} \sum_{n=1}^{N} \{f(w_{MP}, x_n) - t_n\}^2 + \frac{\alpha}{2} w'_{MP} w_{MP}.$$

Using an iterative algorithm very similar to the one presented in previous Section, we can obtain solutions $\hat{\alpha}$ and $\hat{\beta}$.

2.1.3 Gaussian processes

Gaussian processes are based on a full probabilistic framework. They are said to be non-parametric since they characterize a prior over functions $p(F)$ directly instead of giving the unknown function f an explicit parameterization (Mackay, 1998 [8]). Here we have used $F = (f(x_1), f(x_2), \ldots, f(x_N))$ and to be consistent with the wide literature on Gaussian processes we deliberately ignore the implicit dependency on the inputs χ. The prior is chosen to be a zero mean multivariate Gaussian distribution with covariance matrix K. Since the model is $t = f(x) + \varepsilon$, the prior $p(T)$ is also Gaussian and given by

$$p(T) = N(T; 0, C), \tag{2.4}$$

where $C = K + \beta^{-1} I$.

From parametric models to Gaussian processes

Lets assume that we are given a linear regression model with M basis functions. Using matrix notation, we can write $T = \Phi w + \varepsilon$. As before, we consider an isotropic Gaussian

distribution $p(w|\alpha) = N(w; 0, \alpha^{-1}I)$ governed by an hyperparameter α. Since $F = \Phi w$ is based on a linear combination of Gaussian random variables w_i, F is also Gaussian with mean

$$E[F] = \Phi E[w] = 0,$$

and covariance matrix

$$\text{cov}(F) = \Phi E[ww']\Phi' = \alpha^{-1}\Phi\Phi'.$$

Thus, if we define $C = \frac{\Phi\Phi'}{\alpha} + \frac{I}{\beta}$, we obtain a Gaussian process with

$$p(T) = N(T; 0, C).$$

Moreover, in his work, Neal (1996 [11]) showed that for specific choices of priors $p(w)$, the prior over functions $p(F)$ of a one hidden layer neural network converges to well known Gaussian process priors as the number of hidden neurons tends to infinity.

Kernel functions

In the Gaussian process framework, the covariance matrix C is generally given directly without parameterizing the unknown function f. Its elements are given by

$$C(x_i, x_j) = k(x_i, x_j) + \beta^{-1}\delta(i - j),$$

where k is a function called *kernel* that depends on variables called *hyperparameters.*

A list of kernel functions for Gaussian process regression can be found in the book written by Rasmussen and Williams (2006 [15]). Two of the most widely used are the *squared exponential*

$$k(x_i, x_j) = v_0 \exp\left(-\frac{1}{2}\sum_{l=1}^{p} a_l(x_i^l - x_j^l)^2\right) + b,$$

and the *rational quadratic*

$$k(x_i, x_j) = v_0\left(1 + \sum_{l=1}^{p} a_l(x_i^l - x_j^l)^2\right)^{-\nu} + b.$$

The optimization of the parameters a_l allows the relative importance of the corresponding inputs to be inferred from the data. Since C is a covariance matrix, all its terms have to be positive and so must the kernel hyperparameters. We define θ as the vector containing all the hyperparameters with dimension d.

Predictions

In the previous Section, we saw that the covariance matrix C was governed by some specific hyperparameters depending on the choice of kernel function. We will describe how to optimize those parameters in the next Section but for now, we assume that their values are given. Our goal is to predict the value of t_{N+1} for a new input x_{N+1}. The joint distribution $p(T, t_{N+1})$ is given by

$$p(T, t_{N+1}) = N\left((T, t_{N+1}); 0, C_{N+1}\right),$$

where C_{N+1} is a matrix partitioned as follows

$$\begin{pmatrix} C & k \\ k' & c \end{pmatrix}.$$

Here C is the covariance matrix of T (2.4), k has elements $k(x_n, x_{N+1})$ for $n = 1, \ldots, N$, and $c = k(x_{N+1}, x_{N+1}) + \frac{1}{\beta}$.

Since the joint distribution is Gaussian, the predictive distribution has a simple form

$$\hat{t}(x) \sim p(t_{N+1}|T) = N\left(t; k'C^{-1}T, c - k'C^{-1}k\right).$$

Training of the hyperparameters

Given a training data set $\{\chi, T\}$, the training task consists of maximizing the log-likelihood function[3] $\ln p(T|\chi, \theta)$. As before, we ignore the implicit dependency on the inputs χ and we can write

$$\ln p(T|\theta) = -\frac{1}{2}\ln\det C - \frac{1}{2}T'C^{-1}T - \frac{N}{2}\ln(2\pi). \tag{2.5}$$

Since the kernel functions are only defined for positive hyperparameters, we constrain θ by setting $\theta = \exp(\phi)$. It is then possible to use a non-linear optimization algorithm such as scaled conjugate gradient to optimize (2.5) with respect to ϕ.

It is also possible to introduce a prior probability distribution to constraint the hyperparameters defined in the log-space (Nabney, 2003 [10]). In the literature, this

[3]In the Gaussian process framework, this function is sometimes called the log-marginal-likelihood function.

prior is usually called *hyperprior*. A common choice consists of using an isotropic Gaussian distribution $p(\phi|\sigma^2_{prior}) = N\left(\phi; 0, \sigma^2_{prior} I\right)$. Using Bayes' rule, the posterior distribution is given by

$$p(\phi|T, \sigma^2_{prior}) = \frac{p(T|\phi)p(\phi|\sigma^2_{prior})}{Z},$$

where Z is a normalizing constant. This gives rise to a regularized log-likelihood

$$\ln p(\phi|T, \sigma^2_{prior}) = -\frac{1}{2}\ln\det C - \frac{1}{2}T'C^{-1}T - \frac{\phi'\phi}{2\sigma^2_{prior}} - \frac{N}{2}\ln(2\pi) - \frac{d}{2}\ln(2\pi) - \frac{d}{2}\ln(\sigma^2_{prior}) - \ln(Z).$$

2.2 Distributed Learning Environment

So far, we have seen that regression methods can be used to approximate the underlying process that maps input variables to target variables of a specific data set. First, we saw that we had to choose a model (linear regression model, neural network, Gaussian process). Then, using a *single* training data set and a learning algorithm, we noted that we could learn a predictive model $\hat{t}(x)$.

We now consider the case that we are given m training data sets $\{\chi_j, T_j\}$ that are distributed (stored) on disjoint *nodes*. Each node is an autonomous program called an *agent*. All of them can be run on a single machine or on distinct computers. As we are going to see throughout this thesis, the nodes may have different properties. In particular, we are going to study agents that train independently using their own data set and others that can share some learnt information with one another. Those two training methods are both *local learning strategies* (Grossman et al., 1999 [4]) since local agents work on distributed data. Conversely, we do not consider *centralized learning strategies* that move the data to one central location for model building. Indeed, if the quantity of information sent by each agent is significant, the time or bandwith for the central node to recieve all the examples can be prohibitive.

Systems that are based on agents that can learn models and potentially share some information are usually called *multi-agent systems* in the computer science community and the architecture of such applications is known as a *distributed learning environment*

(Figure 2.1). In Statistics, this domain of research is called the *Distributed Data Mining* (DDM). Park and Kargupta (2002, [12]) wrote a paper that describes algorithms, systems, and applications based on DDM. According to them, lots of work has been done on classifiers. Conversely, except using tree-based models, regression problems have not really been tackled so far and we did not find in this paper any method that we could use in our framework.

Our main assumptions are that the data is *homogeneous* and that one *Data Generative Model* exists. In other words, we assume that all the training data sets have been generated by the same underlying process and that they are described by the same variables. Thus, predictive models $\hat{t}_j(x)$ built at each node can be seen as estimates of the same process $t = f(x) + \varepsilon$.

The fusion of models for data privacy and security preservation, and the speed up of some regression models are two critical applications that can be described by a distributed learning environment.

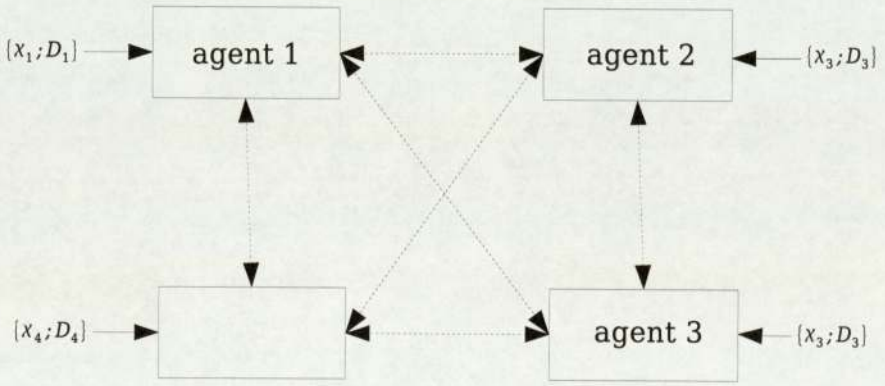


Figure 2.1: A distributed learning environment.

2.2.1 Privacy/Security of data

First, we assume a situation where data is originally physically distributed on multiple nodes. For instance, in the biomedical domain, we can consider m research units or private companies working independently and storing the results of their experiments in their own databases. Moreover, we assume that they work on the same type of data

describing the same biological phenomenon. All the nodes have got models to predict the value of t for a new input x.

Let us assume now that they do not agree to share their data for privacy and security reasons but do agree to share their models. In other words, receiving an input x sent by another model MO, they consent to predict the value of t and to send it back to MO.

In this environment, it could be particularly interesting for each node to use a predictive model based on the fusion of all the models trained at each node. The issue of privacy preserving and model fusion methods was adressed by Wright and Yang (2004, [24]).

2.2.2 Limited learning systems

Tresp (2000, [21]) noted that "for reasons typically associated with their architectures and their learning algorithms, some learning systems are limited in their capability to handle large data sets". For instance, in the Gaussian process framework, both the training and inference task require matrix inversions of which the computational cost is $O(N^3)$, where N is the number of training data points.

In this situation, we can split the *single* training data set on m different data sets that are then stored on nodes. Local agents can use the distributed data and by exchanging some information with one another they can build an approximation of the predictive distribution that a model would have obtained by having kept all the data in one data set.

The issue of this structure is to develop very fast regression models. In particular, as detailed above, since the nodes work locally, they can be run, in parallel, on different computers.

2.3 Data Sets

For the experiments presented through this thesis, we use two data sets.

2.3.1 Toy data set

We generate a toy data set $A = \{(x_i, t_i)\}\ \forall i$, where x is drawn from a uniform distribution $U(0, 2)$, ε from a zero mean Gaussian distribution with 0.2 standard deviation and $t = sin(2\pi x) + \varepsilon$.

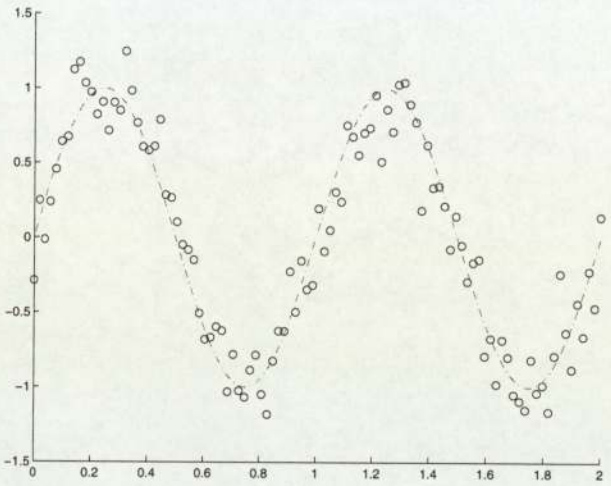


Figure 2.2: Toy data set. Function (*dashdot line*), Training data points (*circles*).

2.3.2 Scatterometry data

We also use some scatterometry data, given by Dr. Dan Cornford, to test our models. More precisely, we are given 180000 data points described by three inputs *ur*, *vr*, *inc* and one target variable *sig*. Scatterometers are radars that aim at transmitting pulses of microwave energy towards the Earth's surface and measuring the reflected energy (Stoffelen, 1998 [19]). By changing the angle *inc* of the signals, measuring the corresponding reflected energy *inc*, and using some geophysical model functions, it is possible to estimate the positions *ur* and *vr* of wind vectors over the ocean. In our case, we are interested in an inverse problem. Given *ur*, *vr*, and *inc*, we want to predict the reflected energy *sig*.

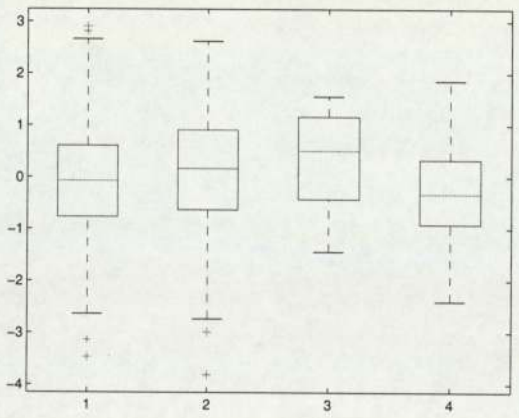


Figure 2.3: Box plot of the scatterometry data. Variables 1, 2, and 3 correspond to the inputs *ur*, *vr*, and *inc*. Variable 4 represents the target *sig*.

Chapter 3

Fusion of Physically Distributed Regression Models

In this Chapter, we address the issue of fusing models that are trained on originally physically distributed data. We define a distributed learning environment where all the agents are identical, for example radial basis function networks or multilayer perceptrons, with exactly the same form of output function $f(w, x)$ and depending on weight vectors w of the same size. The problem of considering an architecture based on different types of regression models is not tackled and remains for future work.

As we showed in Section 2.1.1, the parameters of a radial basis function network are obtained thanks to a two-step learning process. The first step consists of using a clustering technique to fix the centres x_i. Since we only consider agents with the same form of output function $f(w, x)$ and depending only on weight parameters w, we assume that all the radial basis function networks share the same centres which have already been computed.

Moreover, some of the methods described in this Chapter require the agents to be trained using Bayesian regression techniques. Thus, for our experiments, we chose the evidence procedure for the learning algorithm. We recall that it estimates the hyperparameters $\hat{\alpha}$, $\hat{\beta}$, and w_{MP}. For simplicity and consistency of notation, we define

the predictive distribution of node j as

$$\hat{t}_j(x) \sim p(t|x, M_j) = N\left(t; f(w_j, x), \sigma_j^2(x)\right),$$

where w_j is the maximum of the posterior distribution over the weights $p(w|\chi_j, T_j, \hat{\alpha}_j, \hat{\beta}_j)$.

In the first three Sections of this Chapter, we describe the techniques that we used to fuse models trained on distributed data and how we had to modify or extend some of them in order to tackle our problem. More precisely, in Section 3.1, we identify methods that fuse models by working in the space of weight parameters. Then, we investigate approaches, such as model combination methods and the Bayesian model averaging algorithm, that deal with single predictions. In Section 3.3, we describe the Bayesian committe machine and we show how it can be applied to neural networks. Finally, we present the experiments that we carried out and we draw some conclusions.

3.1 Fusion of Weight Parameters

3.1.1 Distributed cooperative Bayesian learning strategies

We first consider Yamanishi's *distributed cooperative Bayesian learning strategies* (1999, [25]). It was originally developed for *density estimation* problems. In other words, it aimed at inferring hidden parameters w of probability distributions $p(x|w)$.

In this model, a *distributed learning system* consists of a number, m, of agents, called the *agent learners*, and an entity, called the *population learner* (p-learner). Each agent learner independently observes a sequence of examples $\chi_j = \{x_{j1}; x_{j2}; \ldots; x_{jn_j}\}$. Here, n_j defines the number of training data points that are used to train the jth agent. Having a prior $p(w)$, each agent computes independently the posterior probability distribution

$$p(w|\chi_j) = \frac{p(w)p(\chi_j|w)}{\int p(w)p(\chi_j|w)\, dw} = \frac{p(w)\prod_{i=1}^{n_j} p(x_{ji}|w)}{\int p(w)\prod_{i=1}^{n_j} p(x_{ji}|w)\, dw}.$$

Then, estimates $\hat{w}_j$ of the parameter w, specifying the target distribution $p(x|w)$, are chosen randomly according to the corresponding distribution $p(w|\chi_j)$.

In this model, the p-learner does not have access to the different training data sets, but only to the parameters output by the agent learners. It combines the estimates using a simple average

$$\hat{w}_{fus} = \frac{\sum_{j=1}^{m} \hat{w}_j}{m}. \tag{3.1}$$

We now extend these methods to a distributed regression problem. Thus, we consider m data sets $\{\chi_j, T_j\}$ and m regression models. Using its own data, each agent learner computes $\hat{\alpha}_j$ and $\hat{\beta}_j$ using the evidence procedure. Then, the p-learner calculates (3.1) by sampling estimates from the different posterior probability distributions $p(w|\chi_j, T_j, \hat{\alpha}_j, \hat{\beta}_j)$. In order to get some errors bars on our predictions, we chose to use similar techniques to (3.1) and we defined $\frac{1}{\beta_{fus}} = \frac{1}{m}\sum_{j=1}^{m} \frac{1}{\hat{\beta}_j}$. Finally, the predictive model is given by

$$\hat{t}_{fus}(x) \sim N\left(t; f(\hat{w}_{fus}, x), \frac{1}{\beta_{fus}}\right).$$

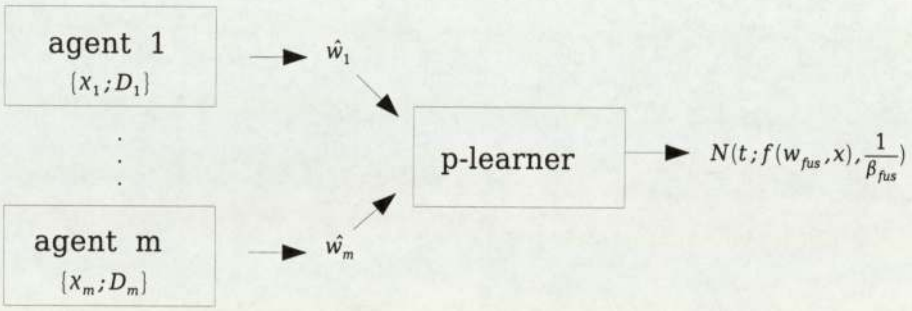


Figure 3.1: Distributed learning System.

3.1.2 Hierarchical Bayesian modelling

In the previous Section, we considered agents training independently using their own data. An entity called the p-learner was used to average some estimated parameters to build a predictive model.

We now consider a situation where agents can potentially learn from one another during the training process by exchanging some learned knowledge. The Bayesian approach that we study here is called the *hierarchical Bayesian Modelling* and was

described as a technique for multi-agent learning by Tresp and Yu (2004, [22]). The basic idea is that information can be exchanged between models via common hyperparameters.

Assume that we are given a prior probability distribution over the weight parameters $p(w|h_1)$ governed by an hyperparameter h_1. For instance, this distribution can be Gaussian with $h_1 = \{\mu_1, \Sigma_1\}$. Having a training data set $\{\chi_1, T_1\}$, the posterior is computed using Bayes' rule

$$p(w|\chi_1, T_1, h_1) \propto p(T_1|\chi_1, h_1)p(w|h_1).$$

If $p(w|h_1)$ and $p(w|\chi_1, T_1, h_1)$ are conjugate then they have got the same functional form. When additional data becomes available, the posterior becomes the new prior and we define $p(w|h_2) = p(w|\chi_1, T_1, h_1)$. This is an iterative process and the more data is used for the training, the more these distributions become sharply peaked around w_{ML}, the maximum of the likelihood function.

In our distributed learning environment, we want to use similar concepts in order to approximate the posterior $p(w|\{\chi_j, T_j\}_{j=1}^m)$. Thus, when a new agent with its own data set $\{\chi_{m+1}, T_{m+1}\}$ is considered, it would use this posterior as a prior to build its own predictive model. Since this Bayesian approach for multi-agent learning does not specify how to combine the different noise variance estimates $\frac{1}{\beta_j}$, as we did in the previous Section, we chose to define $\frac{1}{\beta_{fus}} = \frac{1}{m}\sum_{j=1}^m \frac{1}{\beta_j}$.

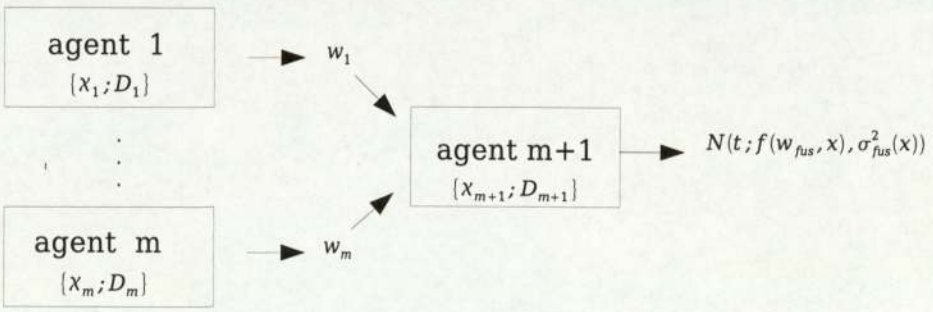


Figure 3.2: Hierarchical Bayesian modelling for multi-agent learning.

Parametric approaches

When a new agent wants to learn a regression model, it receives the learned maximum *a posteriori* weight vectors w_j of all the other nodes. It then fits a specific distribution through them to get an approximation of $p(w|\{\chi_j, T_j\}_{j=1}^m)$. In our experiments, we considered the Gaussian distribution and we estimated its mean and covariance matrix with $\mu = \frac{1}{m}\sum_{j=1}^m w_j$ and $\Sigma = \frac{1}{m}\sum_{j=1}^m (w_j - \mu)(w_j - \mu)'$ respectively. The posterior is then given by

$$p(w|\{\chi_j, T_j\}_{j=1}^m) \approx N(w; \mu, \Sigma).$$

Having its own data set, the new agent uses this distribution as a prior and computes

$$p(w|\{\chi_j, T_j\}_{j=1}^{m+1}, \beta_{fus}) \propto p(T_{m+1}|\chi_{m+1}, \beta_{fus}) p(w|\{\chi_j, T_j\}_{j=1}^m). \tag{3.2}$$

The predictive distribution is given by

$$p(t|x, M_{m+1}) = \int p(t|x, w, \hat{\beta}_{fus}) p(w|\{\chi_j, T_j\}_{j=1}^{m+1}, \beta_{fus})\, dw.$$

Techniques described in Section 2.1.1 can then be used to compute this integral. Unlike radial basis function networks, multilayer perceptrons, as non-linear regression models, require some approximations such as Taylor expansions and Gaussian approximations.

Finally, the predictive model is

$$p(t|x, M_{m+1}) = N\left(t; f(w_{fus}, x), \sigma_{fus}^2(x)\right),$$

where w_{fus} is the mode of (3.2).

Non-parametric approaches

Non-parametric approaches can be used when the empirical distribution of the maximum *a posteriori* weight vectors does not fall into a well known class of distributions. The posterior is approximated with a sum of delta functions

$$p(w|\{\chi_j, T_j\}_{j=1}^m) = \sum_{j=1}^m \delta_{w_j}(w).$$

The predictive model is then given by

$$p(t|x, M_{m+1}) = \frac{1}{C} \int p(t|x, w, \hat{\beta}_{fus}) p(T_{m+1}|\chi_{m+1}, \beta_{fus}) \sum_{j=1}^m \delta_{w_j}(w)\, dw,$$

where C is a normalizing constant. Finally,

$$p(t|x, M_{m+1}) = \frac{1}{C} \sum_{j=1}^{m} p(t|x, w_j, \hat{\beta}_{fus}) p(T_{m+1}|\chi_{m+1}, w_j, \beta_{fus}),$$

and $C = \sum_{j=1}^{m} p(T_{m+1}|\chi_{m+1}, w_j, \beta_{fus})$.

3.2 Fusion of Single Predictions

In the previous Section, we described how models trained on distributed data could be fused by working in the space of weight parameters. In particular, we saw that different estimates $\hat{w}_j$ could be combined using a simple average $\hat{w}_{fus} = \frac{\sum_{j=1}^{m} \hat{w}_j}{m}$. We now investigate techniques that were originally developed to fuse models all trained using the same data. One characteristic of such methods is that they combine directly the different model predictions $\hat{t}_j(x)$ for a *single* given input x. We will see in Section 3.3 how Bayesian techniques can be used when several inputs are considered at the same time.

In this part, our work consisted of investigating if and how these approaches could be extended to the multiple data set case.

3.2.1 Error averaging

The simplest method consists of averaging the different model predictions

$$\hat{t}_{fus}(x) = \frac{\sum_{j=1}^{m} \hat{t}_j(x)}{m}.$$

The model obtained is called a *committee*. It can directly be applied to our problem by simply considering that the nodes are now trained using different data sets.

The mean of the predictive distribution is then given by

$$E[\hat{t}_{fus}(x)] = \frac{\sum_{j=1}^{m} E[\hat{t}_j(x)]}{m}.$$

However, it is more complex to compute the variance since we need to estimate the covariances $\mathrm{cov}(\hat{t}_i(x), \hat{t}_j(x))$ between the models

$$var(\hat{t}_{fus}(x)) = \frac{1}{m^2} \left\{ \sum_{j=1}^{m} var(\hat{t}_j(x)) + 2 \sum_{i,j:i<j}^{m} \mathrm{cov}(\hat{t}_i(x), \hat{t}_j(x)) \right\}.$$

One technique to approximate those terms is to use the maximum likelihood estimator

$$\text{cov}(Z) = \frac{1}{N}\sum_{i=1}^{n}(Z_i - \bar{Z})(Z_i - \bar{Z})', \tag{3.3}$$

where $Z_i = (\hat{t}_1(x_i), \hat{t}_2(x_i), \ldots, \hat{t}_m(x_i))'$ and $\bar{Z} = \frac{1}{N}\sum_{i=1}^{m} Z_i$. For our distributed environment, that means that we need to consider a new N point training data set $\chi = (x_1, x_2, \ldots, x_N)'$. The nodes produce a prediction $\hat{t}_j(x_i)$ for each input x_i and by applying (3.3) we estimate the covariances between the regression models.

3.2.2 Weighted error averaging

We still use a simple linear averaging but in the form

$$\hat{t}_{fus}(x) = \sum_{j=1}^{m}\alpha_j\hat{t}_j(x) = \frac{\sum_{j=1}^{m}(C^{-1})_{jk}}{\sum_{j=1}^{m}\sum_{k=1}^{m}(C^{-1})_{jk}}\hat{t}_j(x),$$

where C^{-1} is the inverse covariance matrix between the different predictive models. Thus, we need to use methods described for the simple averaging case and we compute $C^{-1} = \text{cov}(Z)^{-1}$. As before, a new training data set has to be considered to evaluate the terms in $\text{cov}(Z)^{-1}$.

According to Lowe (2001, [6]), since the weighted committee networks exploit covariance knowledge, they generally give better results than the simple averaging of committee members.

3.2.3 Conditional mixture models

In a mixture of regression models, the fused predictive distribution is given by

$$p_{fus}(t|x) = \sum_{j=1}^{m}\pi_j p(t|x, M_j) = \sum_{j=1}^{m}\pi_j N\left(t; f(w_j, x), \sigma_j^2(x)\right).$$

We cannot see this method as a solution to our problem since they are based on training procedures, such as the EM algorithm, for which the extension to the multiple data set case is not obvious. More important, to compute the π_j, they need to have all the training data points at first hand, to fuse the models. Thus, in our case, even if we assume that the conditional mixture models could be extended to the multiple

training data set case, after having learnt independently, the agents would have to send all their data to one central node. If the quantity of information sent by each agent is significant, the time or bandwith for the central node to receive all the examples can be prohibitive.

3.2.4 Product of predictive distributions

These techniques are based on properties of the Gaussian distribution. Thus, we build m predictive models $\hat{t}_j(x)$ that can be fused by computing the product of their corresponding distributions

$$\hat{t}_{fus}(x) \sim \prod_{j=1}^{m} N\left(t; f(w_j, x), \sigma_j^2(x)\right) \propto N\left(t; a_{fus}(x), b_{fus}^2(x)\right),$$

where $(b_{fus}^2(x))^{-1} = \sum_{j=1}^{m} (\sigma_j^2(x))^{-1}$ and $a_{fus}(x) = b_{fus}^2(x) \sum_{j=1}^{m} (\sigma_j^2(x))^{-1} f(w_j, x)$.

Similar techniques were used in the work of Sudderth et al. (2003, [20]) to compute message products for the Nonparametric Belief Propagation.

The extension to the multiple data set case is straightforward.

3.2.5 Bayesian model averaging

The Bayesian Model Averaging (BMA) algorithm was studied in detail by Hoeting et al. (1999, [5]). Practical applications of BMA can be seen in the work of A. E. Raftery et al., (2003, [14]) to calibrate forecast ensembles.

Minka (2000, [9]) noted that, unlike techniques, such as conditional mixture models, BMA is not a model combination method. Indeed, it assumes that the whole training data set that is given has been generated by only one model and it uses the distribution $p(M_j|\chi, T)$ to reflect our uncertainty as to which one that is. Conversely, in model combination methods, the points of the data set can have been generated by different models. Thus, even if these techniques look very similar, the algorithms that they use are different.

Given m predictive models $\hat{t}_j(x)$ trained using the data set $\{\chi, T\}$, the fused pre-

dictive distribution is given by

$$p_{fus}(t|x) = \sum_{j=1}^{m} p(M_j|\chi, T)p(t|x, M_j) = \sum_{j=1}^{m} p(M_j|\chi, T)N\left(t; f(w_j, x), \sigma_j(x)^2\right).$$

We now consider different data sets $\{\chi_j, T_j\}$ distributed on nodes. If we assume that the models have got the same prior, such that $p(M_j) = p(M)$, using Bayes' rule we find

$$p_{fus}(t|x) = \sum_{j=1}^{m} \alpha_j p(t|x, M_j) = \sum_{j=1}^{m} \alpha_j N\left(t; f(w_j, x), \sigma_j(x)^2\right),$$

where $\alpha_j = \frac{p(T_j|\chi_j, T_j)}{\sum_{i=1}^{m} p(T_i|\chi_i, T_i)}$.

3.3 The Bayesian Committee Machine

In the previous Section, we described methods that could be used to combine m model predictions $\hat{t}_j(x)$ corresponding to a given input x. We now introduce the Bayesian Committee Machine (BCM) that was developed by Tresp (2000, [21]) and that fuses models by considering several input points at the same time.

3.3.1 Fusion of Bayesian estimators

The BCM originally aimed at decreasing the computational cost of some learning systems, such as Gaussian processes. We will use this aspect of the algorithm in Chapter 4, but for now, we see the BCM as a general technique that can be applied to the combination of any Bayesian regression estimator.

As before, we consider m training data sets $\{\chi_j, T_j\}$ and, following Tresp, we use the notation $X_q = (x_1, x_2, \ldots, x_q)'$ and $F_q = (f(x_1), f(x_2), \ldots, f(x_q))'$ for the query set. It is important to distinguish between χ_j, T_j, X_q, and F_q. For the first two data sets that correspond to training data, j specifies the index of the training data set. Conversely, for the query set, q defines the number of points for which we want to obtain a prediction.

The Bayesian committee machine formulae are defined in the space of functions and are obtained using Bayes' rule. For simplicity, as we did in Section 2.1.3 when we

described the Gaussian process framework, we deliberately ignore the implicit dependency on the inputs X_q.

$$p(F_q|\{\chi_j, T_j\}_{j=1}^m) \propto p(F_q)p(\{\chi_j, T_j\}_{j=1}^m|F_q).$$

If we assume that the likelihood function can be factorized, the posterior predictive distribution is given by

$$p(F_q|\{\chi_j, T_j\}_{j=1}^m) \propto p(F_q)\prod_{j=1}^m p(\chi_j, T_j|F_q) \propto \frac{1}{p(F_q)^{m-1}}\prod_{j=1}^m p(F_q|\chi_j, T_j).$$

Although Tresp considered the distributions over F_q, we found that it was also possible to consider the underlying additive Gaussian noise ε. Indeed, $T_q = F_q + \varepsilon$ where $\varepsilon \sim N\left(\varepsilon; 0, \frac{I}{\beta}\right)$ and we have

$$p(T_q|\{\chi_j, T_j\}_{j=1}^m) \propto \frac{1}{p(T_q)^{m-1}}\prod_{j=1}^m p(T_q|\chi_j, T_j). \tag{3.4}$$

In our framework, the distributions $p(T_q|\chi_j, T_j)$ are computed by the agents and (3.4) is used to calculate the fused model.

If those distributions are Gaussian, (3.4) takes a very simple form and its mean is given by

$$E[T_q] = \mathrm{cov}(T_q)\sum_{j=1}^m (\mathrm{cov}(T_q)_j)^{-1}E[T_q]_j, \tag{3.5}$$

with

$$\mathrm{cov}(T_q)^{-1} = -(m-1)\Sigma^{-1} + \sum_{j=1}^m (\mathrm{cov}(T_q)_j)^{-1},$$

where $E[T_q]_j$ and $\mathrm{cov}(T_q)_j$ are the mean and covariance matrix of $p(T_q|\chi_j, T_j)$. Σ is the covariance matrix of the prior $p(T_q)$.

Tresp showed that if the data sets $\{\chi_j, T_j\}$ are unconditionally independent, then the factorization of the likelihood function is exact. A good approximation can be achieved by first using some clustering techniques and assigning the data of each cluster to a separate node. We will use this approach in Chapter 4 but, for now, since we assume that the data is originally physically distributed on nodes, we can not apply such preprocessing methods. It is also possible to obtain good approximations when

the number of query points is sufficiently large. Thus, T_q determines T everywhere and the data sets become independent conditioned on T_q.

We are now going to see how the posterior predictive distributions $p(T_q|\chi_j, T_j)$ and the prior $p(T_q)$ can be determined when considering different types of regression models. In his work, Tresp showed that analytical solutions could be obtained for Gaussian processes and linear regression models.

3.3.2 Gaussian processes

In the Gaussian process framework, the posterior distribution at node j is Gaussian and it is straightforward to compute its mean and covariance matrix. Indeed, we recall that when a unique input point x_{N+1} is considered, the predictive distribution is given by

$$N\left(t; k'C^{-1}T_j, c - k'C^{-1}k\right),$$

where k has elements $k(x_n, x_{N+1})$ for $n = 1, \ldots, N$, and $c = k(x_{N+1}, x_{N+1}) + \frac{1}{\beta}$. If we are now given several input points, this distribution becomes

$$N\left(T_q; A'C^{-1}T_j, B - A'C^{-1}A\right), \tag{3.6}$$

where A and B are now matrices such that

$$A_{ij} = k(x_i, (X_q)_j),$$

and

$$B_{ij} = k((X_q)_i, (X_q)_j) + \frac{\delta(i-j)}{\beta}.$$

Moreover, the prior $p(T_q)$ is also Gaussian and is given by

$$p(T_q) = N\left(T_q; 0, \Sigma\right),$$

where $\Sigma_{ij} = k((X_q)_i, (X_q)_j) + \delta(i-j)$.

Although it is worth describing, as an introduction to the two next Sections, how the prior and the posterior distributions can be determined in the Gaussian process framework, we recall that in this Chapter, we only consider distributed learning environments based on radial basis function networks and multilayer perceptrons. Gaussian processes will be studied in Chapter 4.

3.3.3 Linear regression models

In Section 2.1.3, we showed, by working in the space of functions rather than the space of weight parameters, that linear regression models could be seen as specific instances of Gaussian processes. Indeed, we demonstrated that the prior $p(T)$ was given by

$$p(T) = N(T; 0, C),$$

where $C = \frac{\Phi\Phi'}{\alpha} + \frac{I}{\beta}$. Using the results presented in Section 2.1.3 and in 3.3.2, we compute

$$N\left(T_q; \beta\Phi_q A^{-1}\Phi' T_j, \Phi_q A^{-1}\Phi_q' + \frac{I}{\beta}\right), \tag{3.7}$$

where $\Phi_{ni} = \phi_i(x_n)$, $(\Phi_q)_{ni} = \phi_i((X_q)_n)$, and $A = \alpha I + \beta\Phi'\Phi$.

For our experiments, since radial basis function networks are generalized linear regression models, we used (3.7) to obtain the predictive distribution at node j. The prior $p(T_q)$ is given by

$$p(T_q) = N(T_q; 0, \Sigma),$$

where $\Sigma = \frac{\Phi_q\Phi_q'}{\alpha} + \frac{I}{\beta}$.

3.3.4 The neural network case

So far, we have seen that the Bayesian committe machine was applicable to Gaussian processes and linear regression models, such that radial basis function networks. We showed that the posteriors at each node and the prior $p(T_q)$ were all Gaussians and could be determined analytically. In his work, Tresp derived equations (3.6) and (3.7) but to our knowledge there is no paper defining how the BCM can be applied to neural networks, such as multilayer perceptrons. Indeed, since such models are non-linear with respect to the weight parameters w, their priors and posteriors over the functions are non-Gaussian (Williams and Rasmussen, 1996 [23]).

In this part, our work consisted in deriving approximations to the relevant quantities in order to use the BCM to combine multilayer perceptrons.

Posterior over functions

First, we tackle the problem of approximating the posterior distribution $p(T_q|\chi_j, T_j)$. We first recall that when an input point x is considered, a first order Taylor expansion can be used to linearize the model

$$t \approx f(w_{MP}, x) + \nabla f(w_{MP})'(w - w_{MP}) + \varepsilon,$$

where w_{MP} is the maximum *a posteriori* of $p(w|\chi_j, T_j, \hat{\alpha}, \hat{\beta})$. Then, Laplace's method can be applied to approximate this distribution with a Gaussian centred around w_{MP} such that

$$p(w|\chi_j, T_j, \hat{\alpha}, \hat{\beta}) \approx N\left(w; w_{MP}, H(w_{MP})^{-1}\right),$$

where $H_{ij} = -\frac{\partial^2}{\partial w_i \partial w_j} \ln p(w|\chi_j, T_j, \hat{\alpha}, \hat{\beta})$.

Thus, t is defined as a linear combination of Gaussian random variables and the corresponding predictive distribution is given by

$$p(t|x, \chi_j, T_j) \approx N\left(t; f(w_{MP}, x), \nabla f(w_{MP})' H(w_{MP})^{-1} \nabla f(w_{MP}) + \beta^{-1}\right).$$

This is a good approximation only if the number of training data points considered is sufficiently large such that $p(w|\chi_j, T_j, \hat{\alpha}, \hat{\beta})$ is sharply peaked around w_{MP}.

We now consider a query set X_q. We found that the model could still be linearized by using the Jacobian of $F_q = (f(w, x_1), f(w, x_2), \ldots, f(w, x_q))'$

$$J = \begin{pmatrix} \frac{\partial f(w,x_1)}{\partial w_1} & \cdots & \frac{\partial f(w,x_1)}{\partial w_d} \\ \cdots & \cdots & \cdots \\ \frac{\partial f(w,x_q)}{\partial w_1} & \cdots & \frac{\partial f(w,x_q)}{\partial w_d} \end{pmatrix}.$$

Thus, we have

$$T_q = F_q(w_{MP}) + J(w_{MP})(w - w_{MP}) + \varepsilon,$$

and the corresponding posterior distribution is given by

$$p(T_q|\chi_j, T_j) = N\left(T_q; w_{MP}, J(w_{MP})H(w_{MP})^{-1}J(w_{MP})' + \beta^{-1}I\right). \tag{3.8}$$

We ignore the implicit dependency on X_q to be consistent with the Bayesian committe machine notation.

Prior over functions

Through our experiments, we investigated two approximations for the prior over functions.

The first one is based on similar techniques as in the previous section. Indeed, we use (3.8) to linearize the model. Then, since the prior over the weights $p(w|\alpha) = N\left(w; 0, \frac{I}{\alpha}\right)$ is Gaussian, we obtain directly

$$p(T_q) = N\left(T_q; 0, \frac{J(w_{MP})J(w_{MP})'}{\alpha} + \frac{I}{\beta}\right).$$

Our second approach is inspired by Neal's work on bayesian neural networks (1996 [11]). Since the prior can be written as $p(T_q) = \int p(T_q, w)p(w)\, dw$, it can be approximated by a Gaussian using Markov chain Monte Carlo techniques.

3.4 Experiments

Through the experiments presented in this Section, our goal is twofold. Indeed, we want to compare the results obtained when using model fusion techniques on multilayer perceptrons and on radial basis function networks. Moreover, we aim at analysing what happens when we vary the number of training data points at each node. Thus, we consider three situations. In the first two series of experiments, we assume that all the nodes have got the same number N of points where N is small or large. Then, we simulate a distributed learning environment where the nodes have got data sets of different size.

For the toy data, we consider a training set and a test set both having 1000 data points. We use multilayer perceptrons with three hidden units and networks having 10 radial basis functions since these models give rise to similar predictions when trained on the same data set. In the first situation, we fix $N = 25$ and the number of models is given by $m = 40$. Then, we experiment $N = 100$ and $m = 10$. Finally, we distribute randomly the data into data sets of different size.

For the scatterometry data, we are given 180000 data points. We split the data into two data sets of the same size. The first one is used for the training whereas the second

node	number of data points
1	86
2	369
3	249
4	126
5	72
6	45
7	32
8	21

Table 3.1: Toy data set. Distribution of the training data points into the nodes.

aims at testing the model fusion techniques. We use multilayer perceptrons with 10 hidden units and networks having 50 radial basis functions. In the first situation, we fix $N = 100$ and $m = 900$. Then, we experiment $N = 1000$ and $m = 90$. Finally, we distribute randomly the data into data sets of different size.

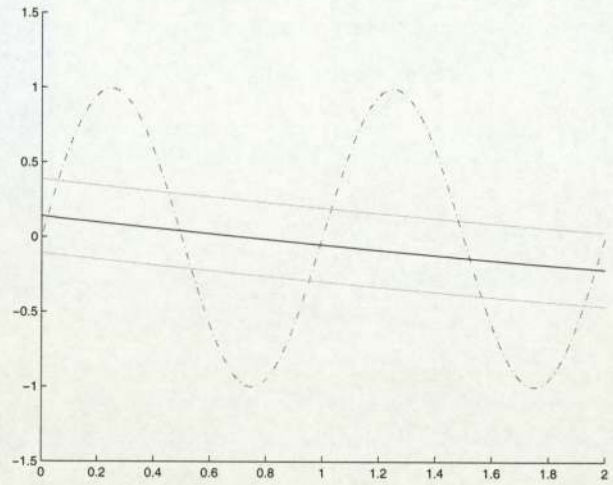
In order to draw the following plots, error bars were computed. Each of those terms corresponds to one standard deviation around the corresponding prediction. Indeed, we recall that all the predictive distributions we consider are Gaussian and so 68.2 percent of the targets are expected to be in these intervals.

node	number of data points
1	12126
2	16210
3	37482
4	15269
5	3365
6	3233
7	1046
8	149
9	126
10	363
11	106
12	242
13	188
14	95

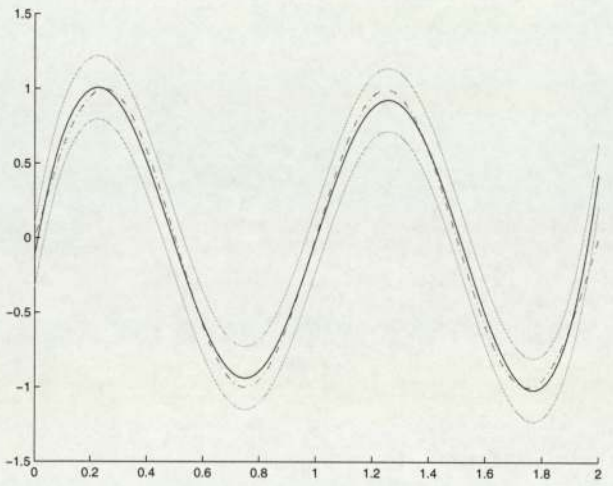
Table 3.2: Scatterometry data. Distribution of the training data points into the nodes.

3.4.1 Distributed cooperative Bayesian learning strategies

Few data at each node



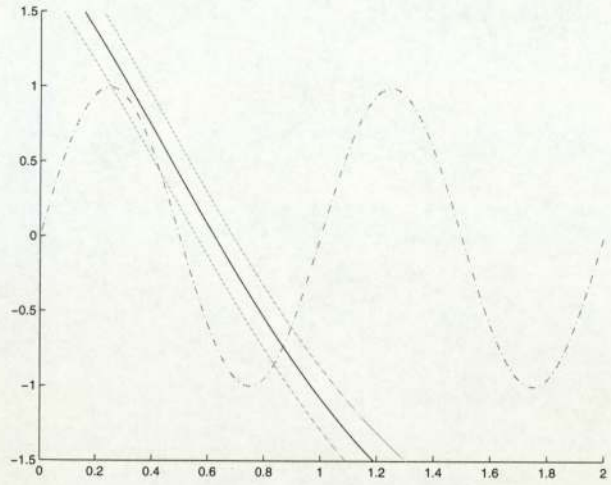
(a)



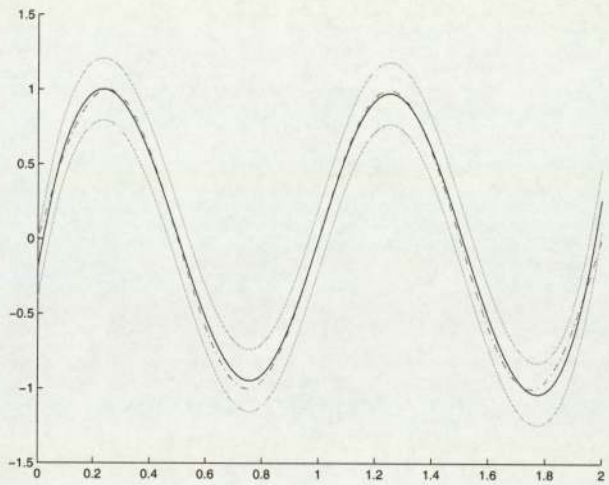
(b)

Figure 3.3: Distributed cooperative Bayesian learning strategies experimented on the toy data set. (a) Fusion of 40 MLP networks. (b) Fusion of 40 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Lots of data at each node



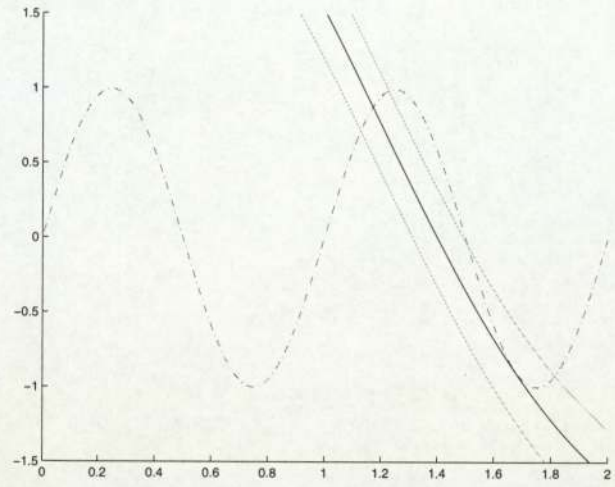
(a)



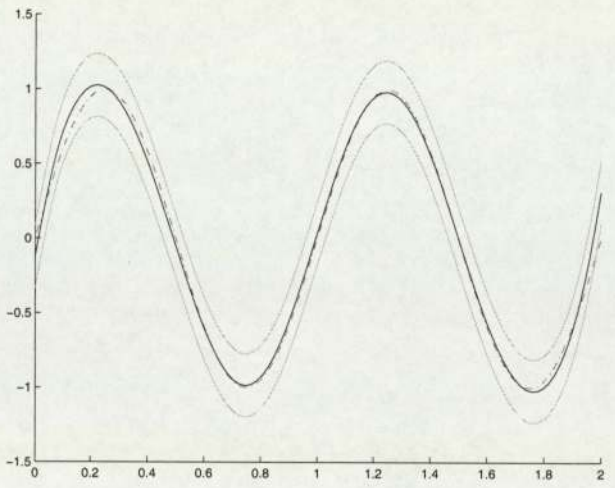
(b)

Figure 3.4: Distributed cooperative Bayesian learning strategies experimented on the toy data set. (a) Fusion of 10 MLP networks. (b) Fusion of 10 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Data sets of different size at each node



(a)



(b)

Figure 3.5: Distributed cooperative Bayesian learning strategies experimented on the toy data set. (a) Fusion of 8 MLP networks. (b) Fusion of 8 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

quantity	regression	data set	NRMSE tr	NRMSE test
Few	MLP	toy	0.8902	0.9124
-	-	scatterometry	1.2442	1.2764
-	RBF	toy	0.0859	0.1033
-	-	scatterometry	0.1189	0.1435
Lot	MLP	toy	4.0660	4.0929
-	-	scatterometry	5.6587	5.7225
-	RBF	toy	0.0806	0.0806
-	-	scatterometry	0.1129	0.1135
Different	MLP	toy	7.9856	8.4628
-	-	scatterometry	11.1789	11.7854
-	RBF	toy	0.0762	0.0863
-	-	scatterometry	0.1065	0.1232

Table 3.3: Distributed cooperative Bayesian learning strategies experimented on the toy data set and on the scatterometry data. Training normalized root mean square error (*NRMSE tr*). Test normalized root mean square error (*NRMSE test*).

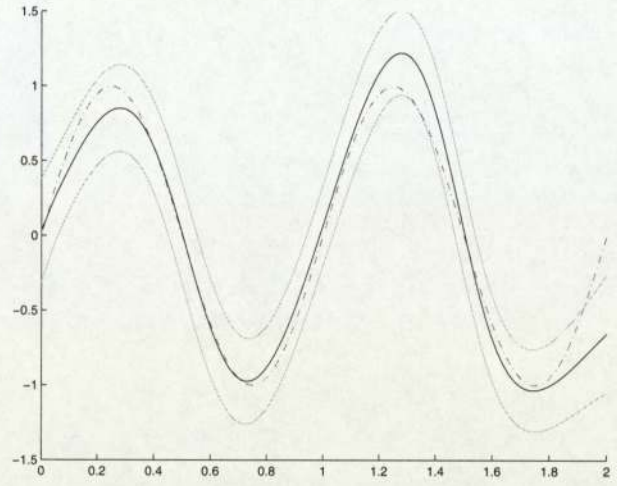
Partial conclusion

First of all, we can observe that the distributed Bayesian learning strategies give poor results when applied to multilayer perceptrons. According to Bishop (2006, [1]), one property of such models is that different weight vectors $\hat{w}_j$ can all give rise to the same mapping function from inputs to outpouts. If those weights vectors are far from one another, there is no reason to believe that a simple average $\frac{\sum_{j=1}^{m} \hat{w}_j}{m}$ will create a better predictive model.

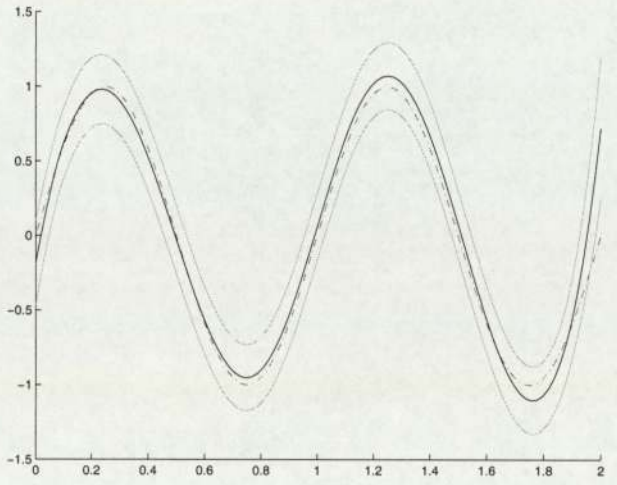
Conversely, we obtained very stable and accurate results using the distributed Bayesian learning strategies on radial basis function networks. In particular, we observed that the more data points there are at each node, the better the predictions of the fused model are.

3.4.2 Parametric hierarchical Bayesian modelling

Few data at each node



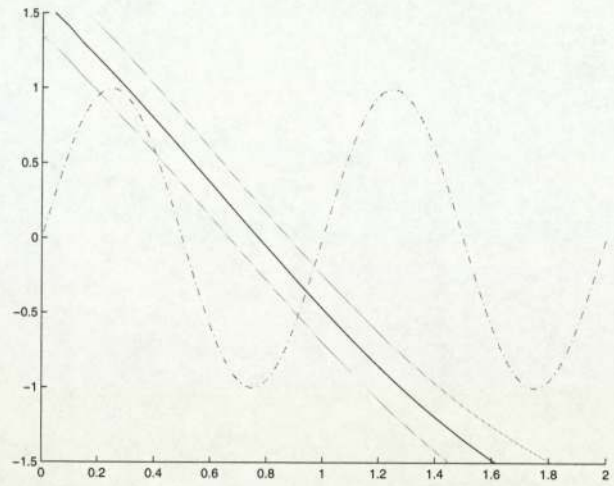
(a)



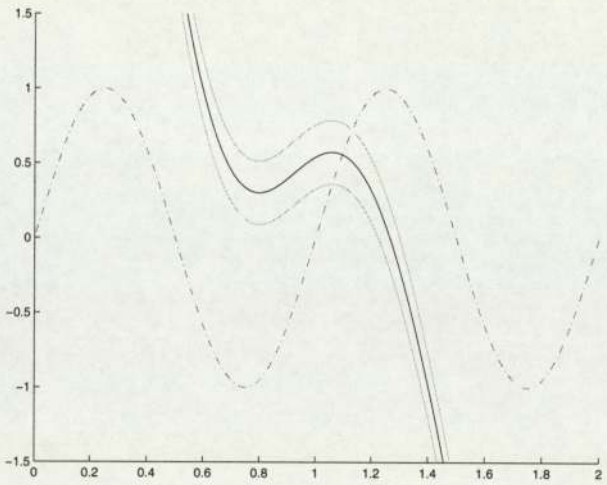
(b)

Figure 3.6: Parametric hierarchical Bayesian modelling experimented on the toy data set. (a) Fusion of 40 MLP networks. (b) Fusion of 40 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Lots of data at each node



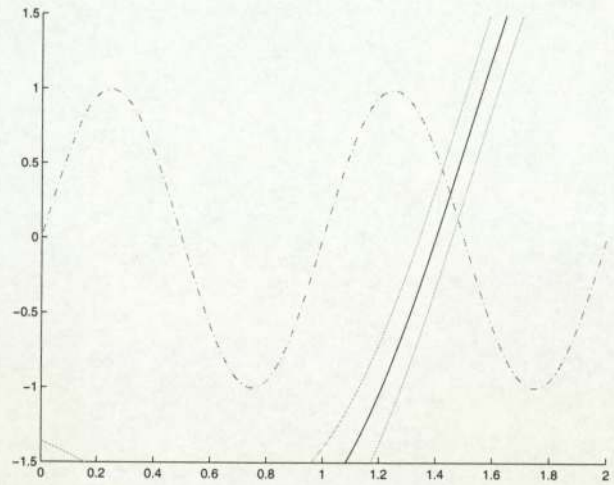
(a)



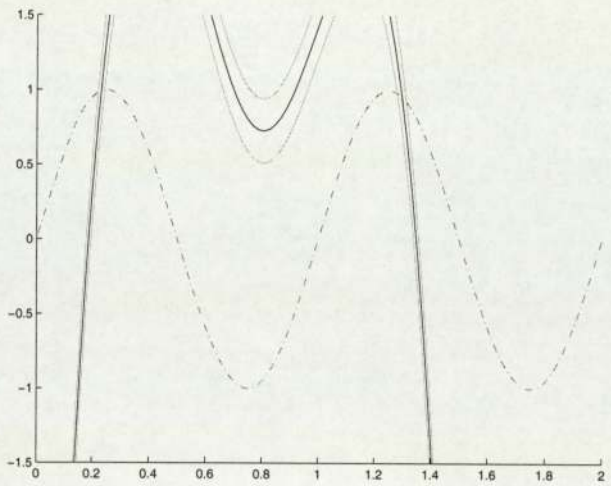
(b)

Figure 3.7: Parametric hierarchical Bayesian modelling experimented on the toy data set. (a) Fusion of 10 MLP networks. (b) Fusion of 10 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Data sets of different size at each node



(a)



(b)

Figure 3.8: Parametric hierarchical Bayesian modelling experimented on the toy data set. (a) Fusion of 8 MLP networks. (b) Fusion of 8 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

quantity	regression	data set	NRMSE tr	NRMSE test
Few	MLP	toy	0.3254	0.3389
-	-	scatterometry	0.4554	0.4737
-	RBF	toy	0.2815	0.3012
-	-	scatterometry	0.3949	0.4209
Lot	MLP	toy	1.5121	1.5356
-	-	scatterometry	2.1156	2.1463
-	RBF	toy	6.8136	6.8243
-	-	scatterometry	9.5229	9.5393
Different	MLP	toy	3.0158	3.0192
-	-	scatterometry	4.2159	4.2209
-	RBF	toy	23.4616	23.4900
-	-	scatterometry	32.7990	32.8353

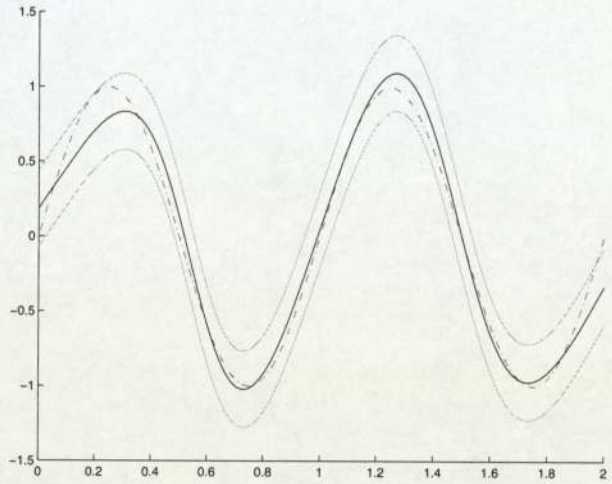
Table 3.4: Parametric hierarchical Bayesian modelling experimented on the toy data set and on the scatterometry data. Training normalized root mean square error (*NRMSE tr*). Test normalized root mean square error (*NRMSE test*).

Partial conclusion

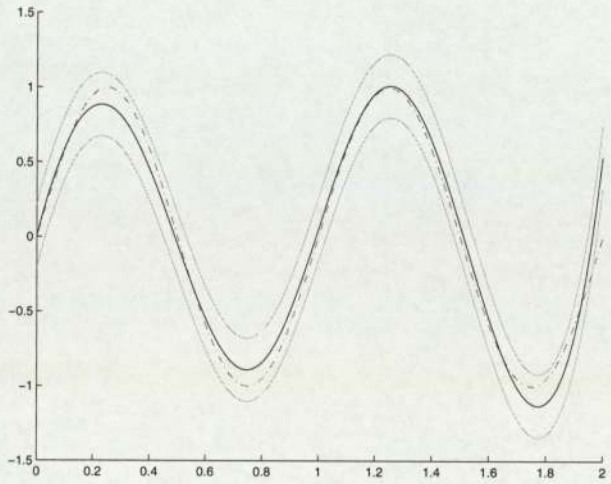
When there are only few training data points at each node, we found that parametric hierarchical Bayesian modelling gives quite accurate predictions for both multilayer perceptrons and radial basis function networks. When the models are given some more data, we observed that the empirical distribution of the maximum *a posteriori* weight vectors cannot be fitted with a Gaussian and so the fused predictive model is far from the underlying process studied. In the neural network case, since very different weight vectors can give rise to similar output functions (symmetry), the more data is given to the agents, the more they specialize during the training process, and the more different the estimated weight vectors potentially are. In this situation, there is no reason to believe that those parameters can be fitted with a Gaussian distribution. However, radial basis function networks do not have such a property and we expected to obtain better results.

3.4.3 Non-parametric hierarchical Bayesian modelling

Few data at each node



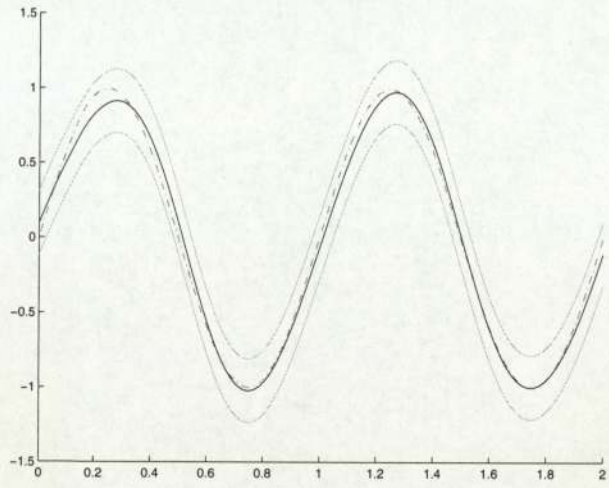
(a)



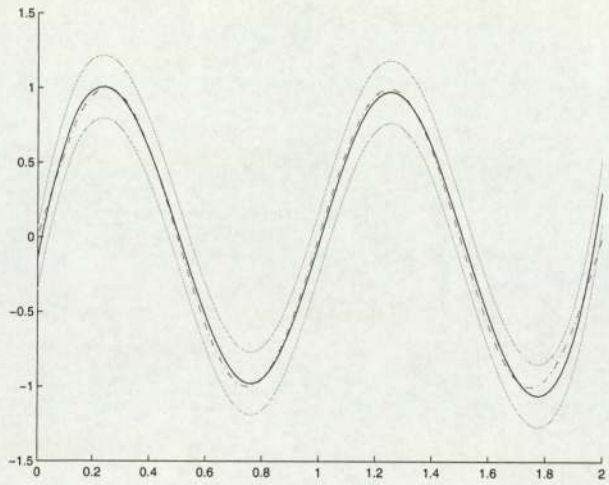
(b)

Figure 3.9: Non-parametric hierarchical Bayesian modelling experimented on the toy data set. (a) Fusion of 40 MLP networks. (b) Fusion of 40 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Lots of data at each node



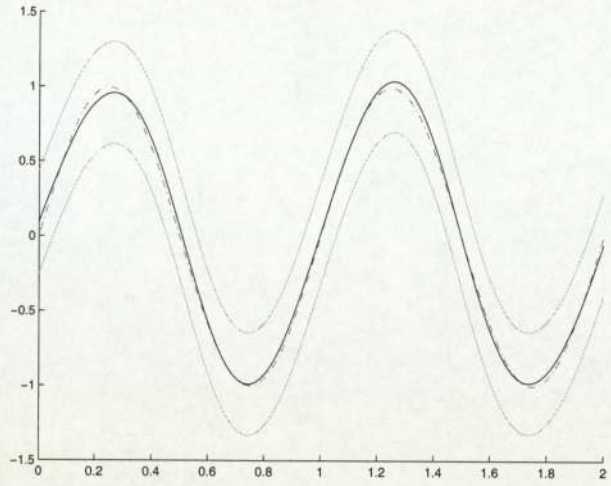
(a)



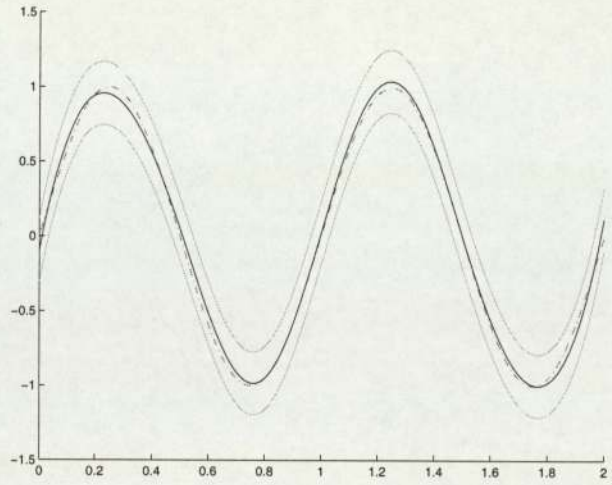
(b)

Figure 3.10: Non-parametric hierarchical Bayesian modelling experimented on the toy data set. (a) Fusion of 10 MLP networks. (b) Fusion of 10 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Data sets of different size at each node



(a)



(b)

Figure 3.11: Non-parametric hierarchical Bayesian modelling experimented on the toy data set. (a) Fusion of 8 MLP networks. (b) Fusion of 8 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

quantity	regression	data set	NRMSE tr	NRMSE test
Few	MLP	toy	0.3407	0.3413
-	-	scatterometry	0.4762	0.4782
-	RBF	toy	0.2981	0.3021
-	-	scatterometry	0.4165	0.4227
Lot	MLP	toy	0.3217	0.3335
-	-	scatterometry	0.4497	0.4664
-	RBF	toy	0.2557	0.2622
-	-	scatterometry	0.3572	0.3679
Different	MLP	toy	0.2458	0.2775
-	-	scatterometry	0.3437	0.3869
-	RBF	toy	0.2967	0.2983
-	-	scatterometry	0.4149	0.4173

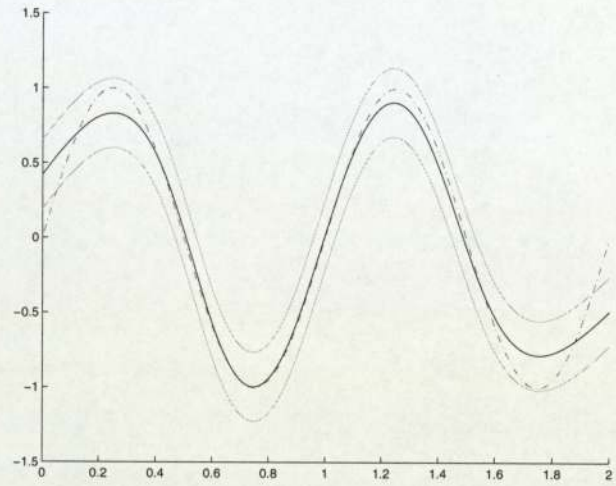
Table 3.5: Non-parametric hierarchical Bayesian modelling experimented on the toy data set and on the scatterometry data. Training normalized root mean square error (*NRMSE tr*). Test normalized root mean square error (*NRMSE test*).

Partial conclusion

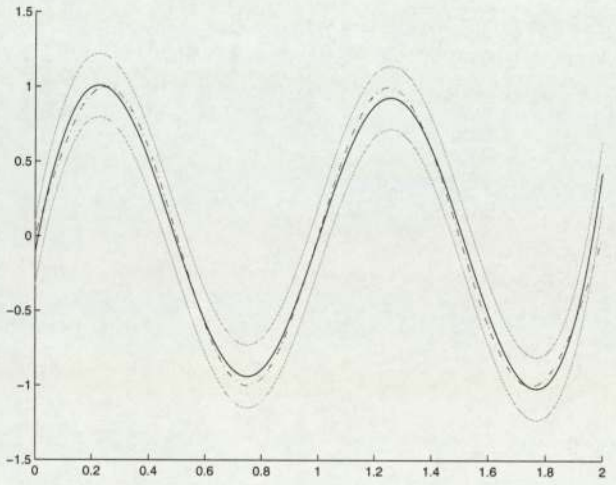
Non-parametric hierarchical Bayesian modelling is one of the most stable technique that we used. It gives similar results for both multilayer perceptrons and radial basis function networks. Moreover, it is very robust regarding the number of points stored on each node. Contrary to the method we saw in the previous Section, this technique can be used when the empirical distribution of the maximum *a posteriori* weight vectors cannot be fitted with a Gaussian.

3.4.4 Error averaging

Few data at each node



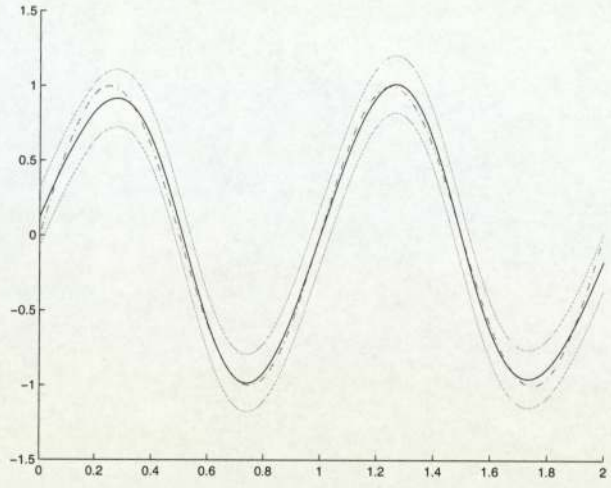
(a)



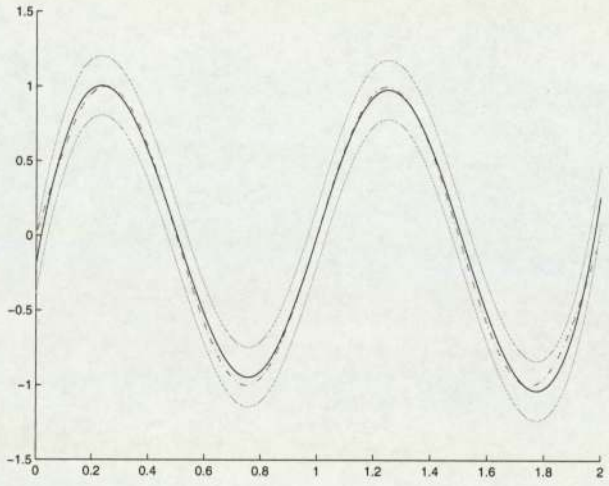
(b)

Figure 3.12: Error averaging experimented on the toy data set. (a) Fusion of 40 MLP networks. (b) Fusion of 40 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Lots of data at each node



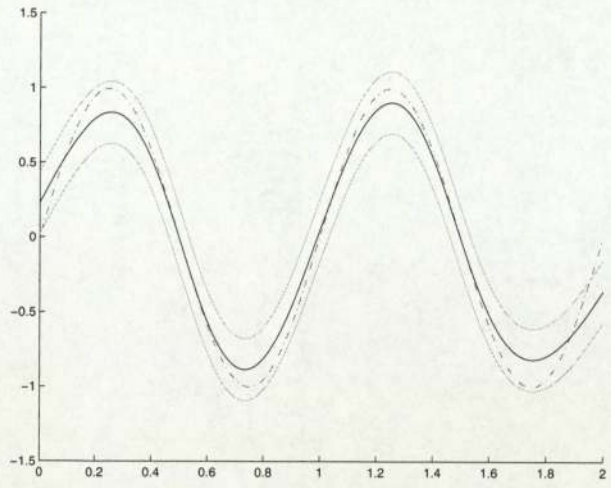
(a)



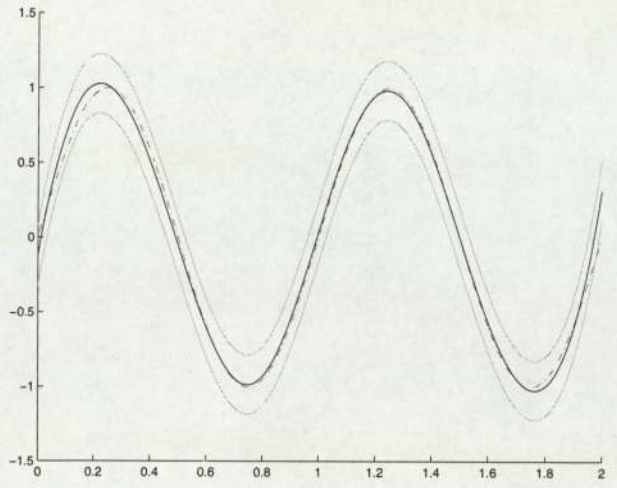
(b)

Figure 3.13: Error averaging experimented on the toy data set. (a) Fusion of 10 MLP networks. (b) Fusion of 10 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Data sets of different size at each node



(a)



(b)

Figure 3.14: Error averaging experimented on the toy data set. (a) Fusion of 8 MLP networks. (b) Fusion of 8 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

quantity	regression	data set	NRMSE tr	NRMSE test
Few	MLP	toy	0.3214	0.3479
-	-	scatterometry	0.4209	0.4876
-	RBF	toy	0.2631	0.2939
-	-	scatterometry	0.3634	0.4120
Lot	MLP	toy	0.2349	0.2790
-	-	scatterometry	0.3284	0.3900
-	RBF	toy	0.2442	0.2843
-	-	scatterometry	0.3487	0.4012
Different	MLP	toy	0.3175	0.3315
-	-	scatterometry	0.4399	0.4621
-	RBF	toy	0.2909	0.2917
-	-	scatterometry	0.4066	0.4078

Table 3.6: Error averaging experimented on the toy data set and on the scatterometry data. Training normalized root mean square error (*NRMSE tr*). Test normalized root mean square error (*NRMSE test*).

Partial conclusion

Error averaging gives similar results for both multilayer perceptrons and radial basis function networks. It is a simple, widely used, and accurate technique. However, one of its weakness is that if there is a node that outputs a prediction far from all the others, since the mean is an estimator very sensitive to outliers, then the fused predictive model risks to be far from the underlying process.

3.4.5 Weighted error averaging

Few data at each node

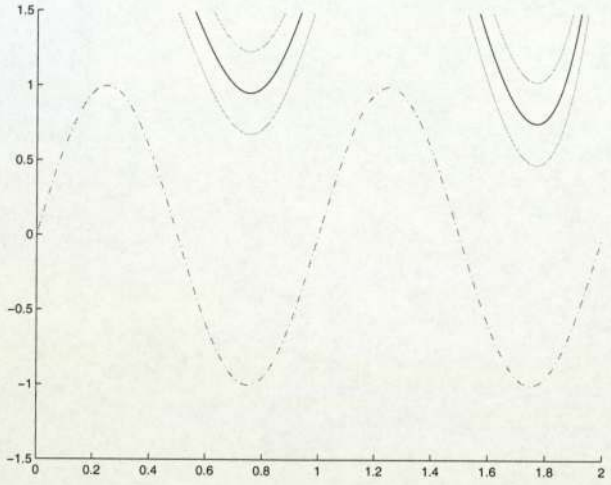
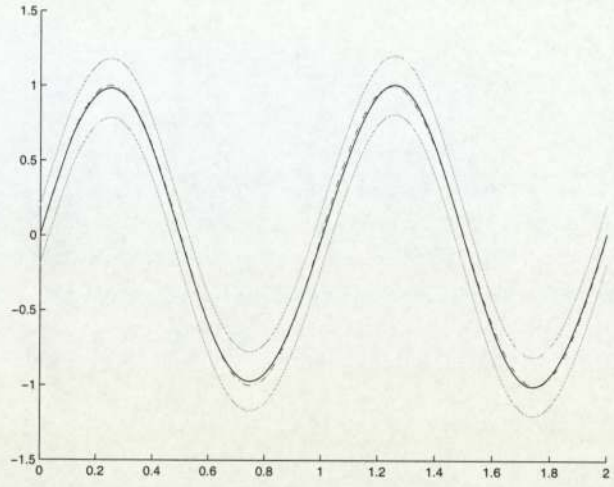
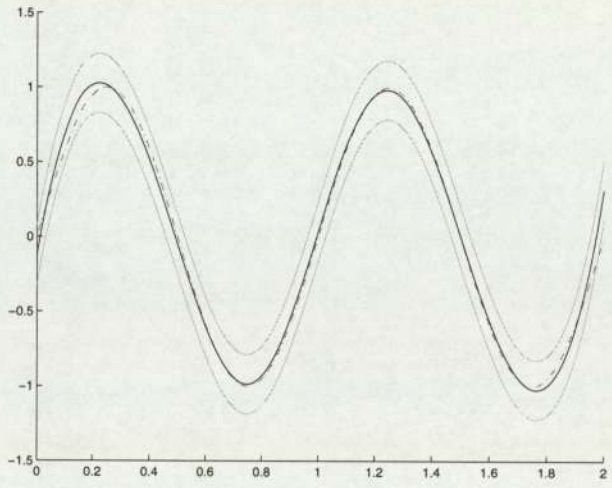


Figure 3.15: Weighted error averaging experimented on the toy data set. Fusion of 40 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Lots of data at each node



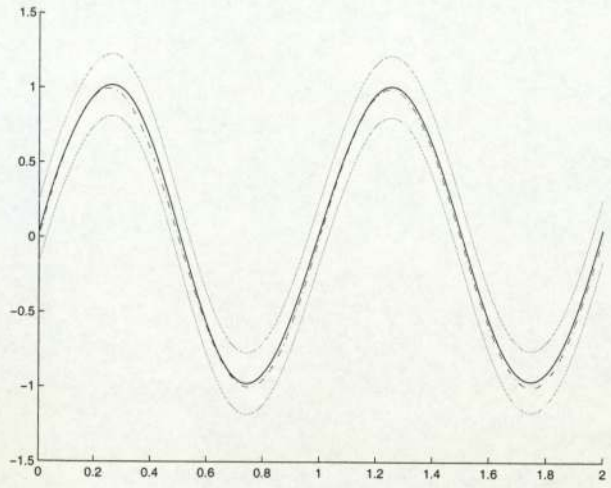
(a)



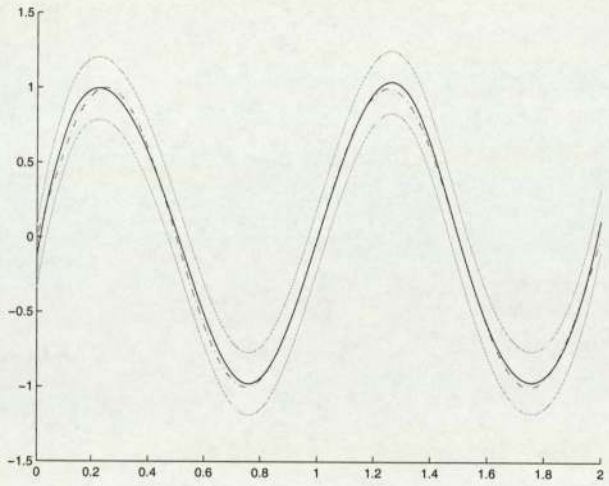
(b)

Figure 3.16: Weighted error averaging experimented on the toy data set. (a) Fusion of 10 MLP networks. (b) Fusion of 10 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Data sets of different size at each node



(a)



(b)

Figure 3.17: Weighted error averaging experimented on the toy data set. (a) Fusion of 8 MLP networks. (b) Fusion of 8 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

quantity	regression	data set	NRMSE tr	NRMSE test
Few	MLP	toy	9.9525e+04	1.1283e+05
-	-	scatterometry	1.3926e+05	1.5732e+05
-	RBF	toy	2.2535	2.5550
-	-	scatterometry	3.1501	3.5715
Lot	MLP	toy	0.2340	0.2700
-	-	scatterometry	0.3409	0.3807
-	RBF	toy	0.2374	0.2719
-	-	scatterometry	0.3610	0.3812
Different	MLP	toy	0.2848	0.2862
-	-	scatterometry	0.3982	0.4000
-	RBF	toy	0.2832	0.2850
-	-	scatterometry	0.3960	0.3984

Table 3.7: Weighted error averaging experimented on the toy data set and on the scatterometry data. Training normalized root mean square error (*NRMSE tr*). Test normalized root mean square error (*NRMSE test*).

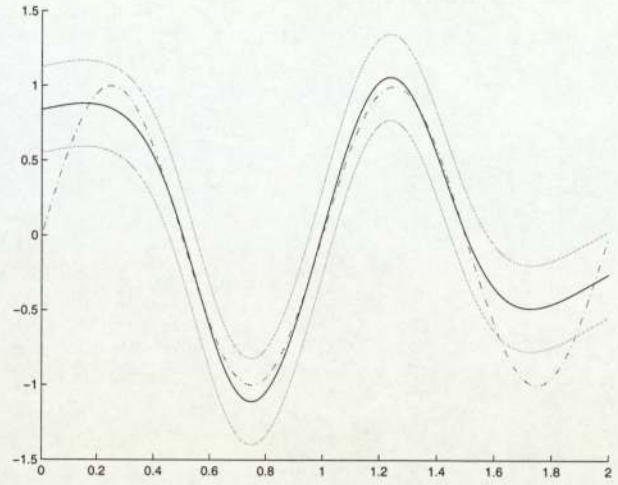
Partial conclusion

We found that weighted error averaging does not work when there are few points at each node. Indeed, if we examine the results obtained in detail, we can observe that the values of the predictions have become very high (we could not even plot the results obtained with multilayer perceptrons). This is mainly due to the fact that the algorithm is based on the inversion of the covariance matrix C between the noises of the models. Thus if the correlations are small, the values of C^{-1} become significant.

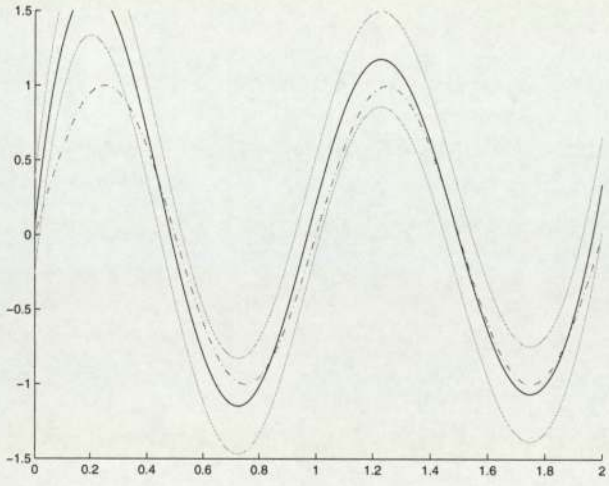
Otherwise, weighted error averaging gives more accurate predictions and is more stable than simple error averaging. It also gives similar results for both multilayer perceptrons and radial basis function networks.

3.4.6 Bayesian model averaging

Few data at each node



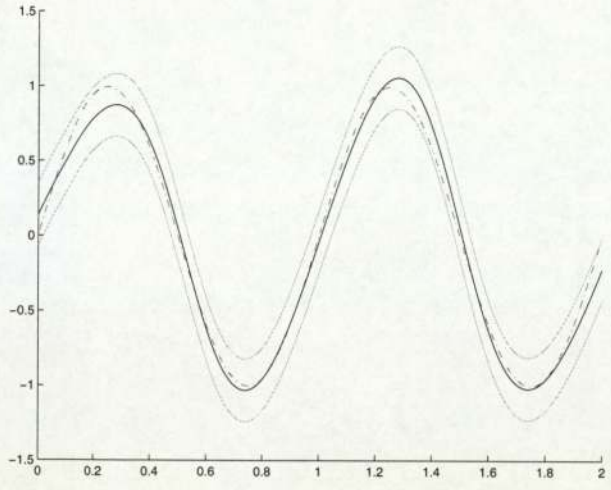
(a)



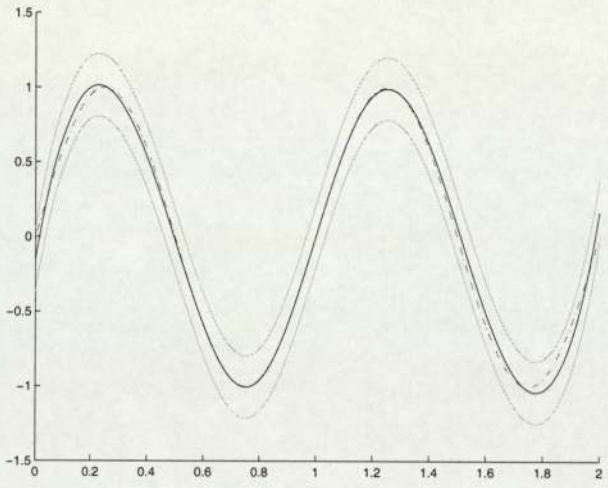
(b)

Figure 3.18: Bayesian model averaging experimented on the toy data set. (a) Fusion of 40 MLP networks. (b) Fusion of 40 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Lots of data at each node



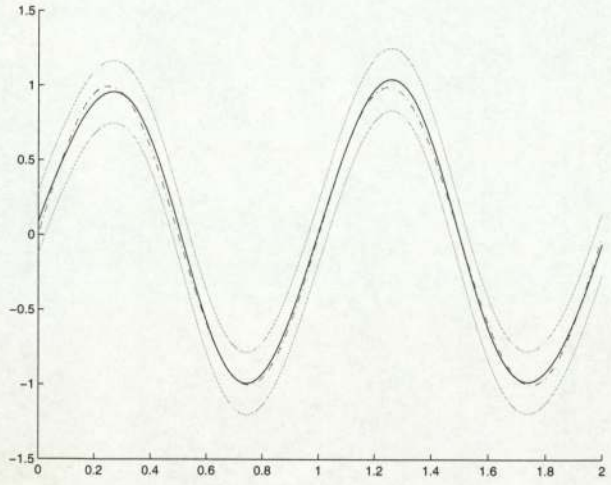
(a)



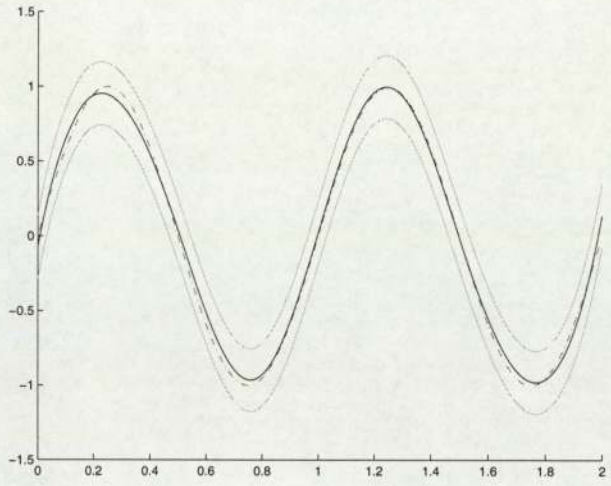
(b)

Figure 3.19: Bayesian model averaging experimented on the toy data set. (a) Fusion of 10 MLP networks. (b) Fusion of 10 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Data sets of different size at each node



(a)



(b)

Figure 3.20: Bayesian model averaging experimented on the toy data set. (a) Fusion of 8 MLP networks. (b) Fusion of 8 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

quantity	regression	data set	NRMSE tr	NRMSE test
Few	MLP	toy	0.4046	0.4672
-	-	scatterometry	0.5690	0.6529
-	RBF	toy	0.3971	0.4659
-	-	scatterometry	0.5562	0.6513
Lot	MLP	toy	0.2262	0.2897
-	-	scatterometry	0.3134	0.4051
-	RBF	toy	0.2392	0.2834
-	-	scatterometry	0.3353	0.3962
Different	MLP	toy	0.2810	0.2861
-	-	scatterometry	0.3940	0.4213
-	RBF	toy	0.2868	0.2869
-	-	scatterometry	0.4010	0.4023

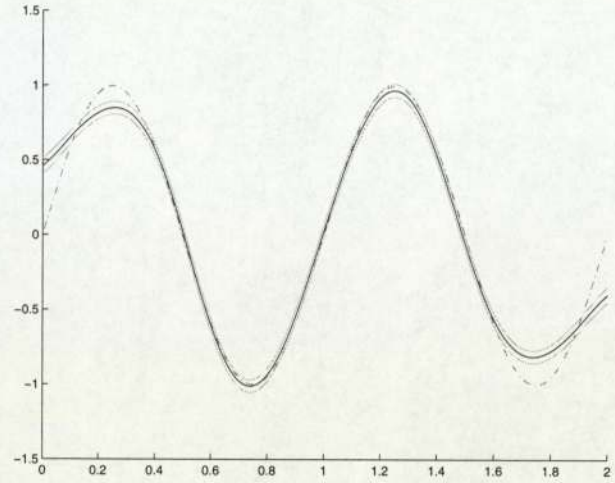
Table 3.8: Bayesian model averaging experimented on the toy data set and on the scatterometry data. Training normalized root mean square error (*NRMSE tr*). Test normalized root mean square error (*NRMSE test*).

Partial conclusion

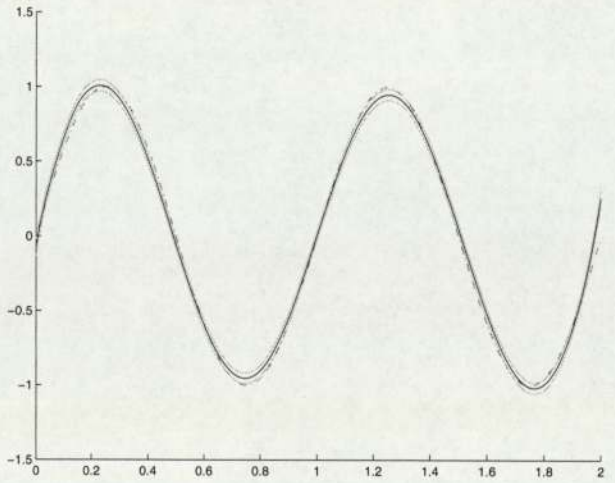
We can observe that Bayesian model averaging gives poor results when there are only few data points at each node. This is due to the fact that the different marginal likelihoods are not well estimated. Otherwise, this method outputs equivalent predictions as weighted error averaging. It gives similar results for both multilayer perceptrons and radial basis function networks.

3.4.7 Product of predictive distributions

Few data at each node



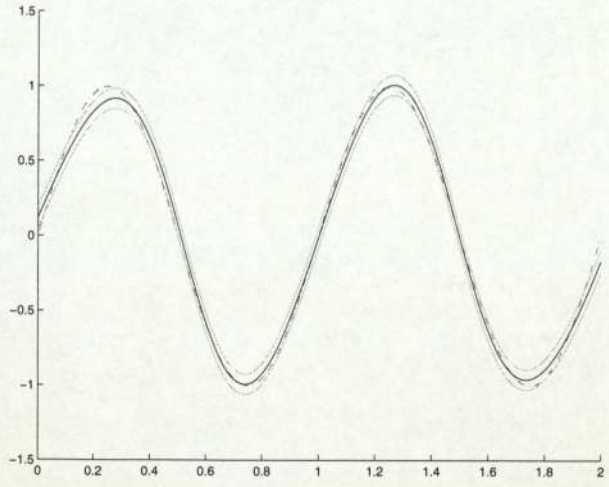
(a)



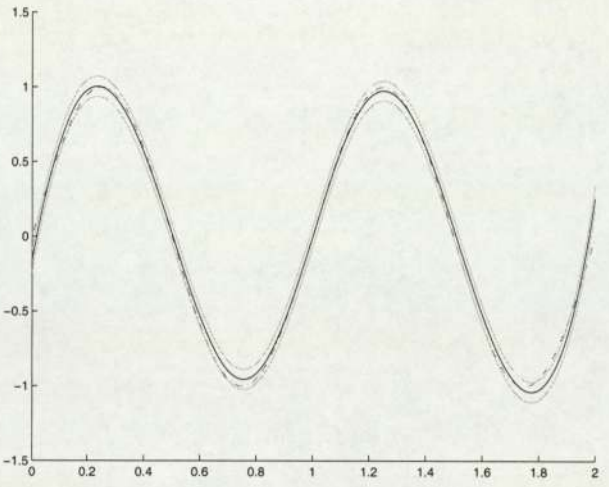
(b)

Figure 3.21: Product of predictive distributions experimented on the toy data set. (a) Fusion of 40 MLP networks. (b) Fusion of 40 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Lots of data at each node



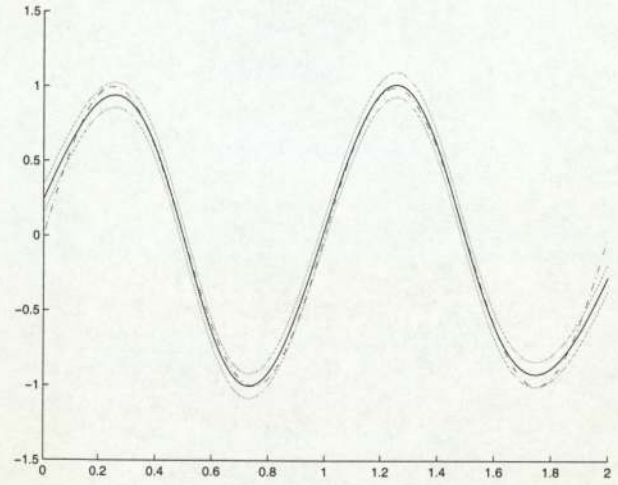
(a)



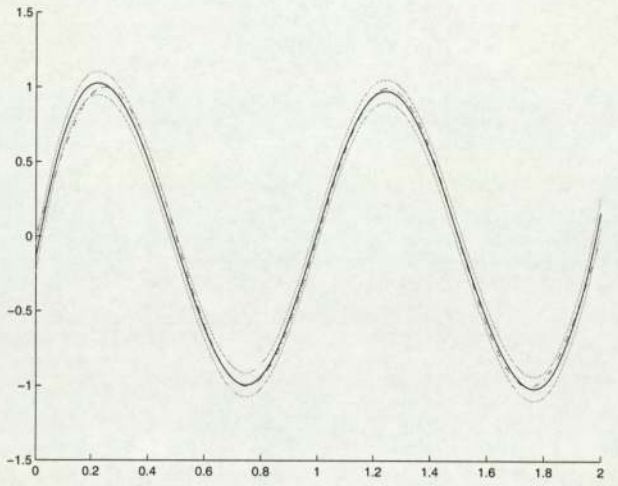
(b)

Figure 3.22: Product of predictive distributions experimented on the toy data set. (a) Fusion of 10 MLP networks. (b) Fusion of 10 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Data sets of different size at each node



(a)



(b)

Figure 3.23: Product of predictive distributions experimented on the toy data set. (a) Fusion of 8 MLP networks. (b) Fusion of 8 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

quantity	regression	data set	NRMSE tr	NRMSE test
Few	MLP	toy	0.3090	0.3356
-	-	scatterometry	0.4312	0.4491
-	RBF	toy	0.2600	0.2867
-	-	scatterometry	0.3623	0.4011
Lot	MLP	toy	0.2341	0.2787
-	-	scatterometry	0.3472	0.3923
-	RBF	toy	0.2432	0.2839
-	-	scatterometry	0.3454	0.3970
Different	MLP	toy	0.2958	0.2993
-	-	scatterometry	0.4126	0.4185
-	RBF	toy	0.2881	0.2881
-	-	scatterometry	0.4028	0.4042

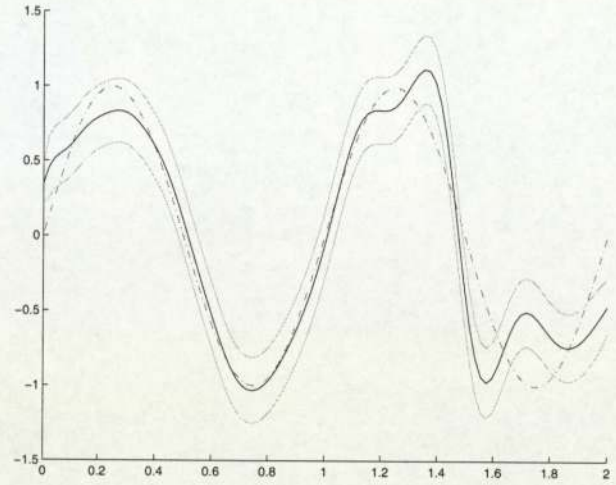
Table 3.9: Product of predictive distributions experimented on the toy data set and on the scatterometry data. Training normalized root mean square error (*NRMSE tr*). Test normalized root mean square error (*NRMSE test*).

Partial conclusion

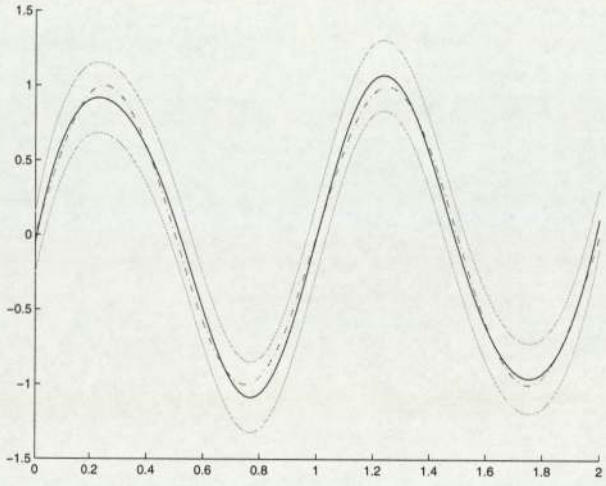
As non-parametric hierarchical Bayesian modelling, the product of predictive distributions is an interesting technique that is very stable regarding the number of points stored on each node. Indeed, in the three series of experiments that we carried out, we obtained very accurate predictions. However, this method underestimates the error bars. More precisely, we found that the more we added nodes to our distributed learning environment, the more the estimated variances decreased to zero. In Section 2.1, we showed that an underlying process can always be decomposed into two terms. The first one represents the deterministic part of the mapping, whereas the second corresponds to a random variable, called noise, with unknown σ^2 variance. Thus, we expect a fused model to ouput both accurate predictions and good estimates of the positive underlying noise variance.

3.4.8 The Bayesian committee machine

Few data at each node



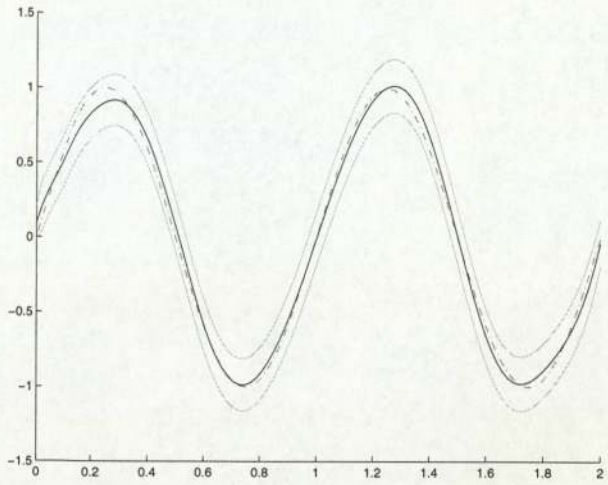
(a)



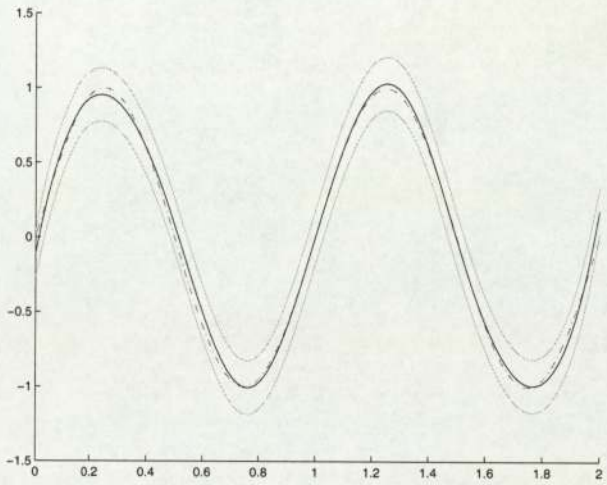
(b)

Figure 3.24: The Bayesian committee machine experimented on the toy data set. (a) Fusion of 40 MLP networks. (b) Fusion of 40 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Lots of data at each node



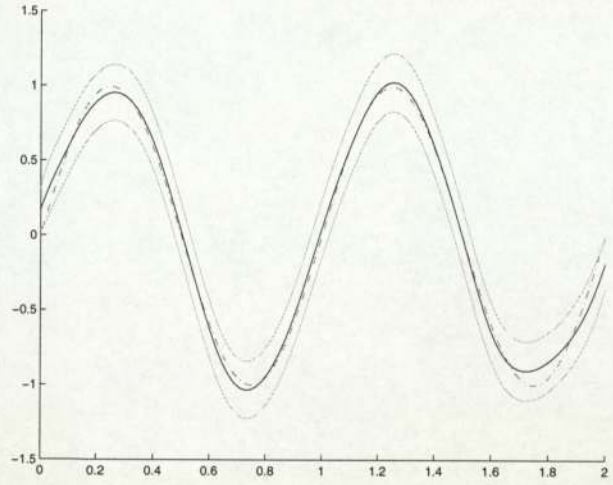
(a)



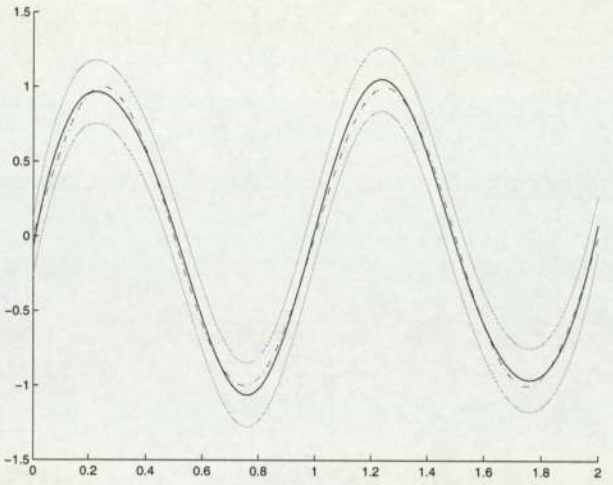
(b)

Figure 3.25: The Bayesian committee machine experimented on the toy data set. (a) Fusion of 10 MLP networks. (b) Fusion of 10 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

Data sets of different size at each node



(a)



(b)

Figure 3.26: The Bayesian committee machine experimented on the toy data set. (a) Fusion of 8 MLP networks. (b) Fusion of 8 RBF networks. Function (*dashdot line*), prediction (*solid black line*), and error bars (*solid grey line*).

quantity	regression	data set	NRMSE tr	NRMSE test
Few	MLP + MCMC	toy	0.4305	0.4467
-	-	scatterometry	0.6021	0.6244
-	MLP + Linearization	toy	0.9962	1.0176
-	-	scatterometry	1.3925	1.4245
-	RBF	toy	0.2823	0.2912
-	-	scatterometry	0.3945	0.4072
Lot	MLP + MCMC	toy	0.2585	0.2654
-	-	scatterometry	0.3723	0.3710
-	MLP + Linearization	toy	0.9950	1.0375
-	-	scatterometry	1.3400	1.4503
-	RBF	toy	0.2684	0.2790
-	-	scatterometry	0.3721	0.3902
Different	MLP + MCMC	toy	0.2509	0.2721
-	-	scatterometry	0.3607	0.3842
-	MLP + Linearization	toy	0.9965	1.1500
-	-	scatterometry	1.3933	1.6075
-	RBF	toy	0.2732	0.2812
-	-	scatterometry	0.3823	0.3938

Table 3.10: The Bayesian committee machine experimented on the toy data set and on the scatterometry data. Training normalized root mean square error (*NRMSE tr*). Test normalized root mean square error (*NRMSE test*). For multilayer perceptrons, two approximations of the prior over functions are experimented : linearization of the model (*Linearization*) and Markov chain Monte Carlo (*MCMC*).

Partial conclusion

We recall that analytical expressions exist when applying the Bayesian committee machine to combine radial basis function networks. Thus, we obtained very accurate predictions when using such models. Here, we aim a testing some approximations that we proposed in order to apply the BCM to fuse multilayer perceptrons.

First of all, we can observe that the linearization of multilayer perceptron output functions, to estimate the prior over functions, always leads to poor results. This is due to the fact that the prior over the weights is broad and so a first order Taylor expansion is not accurate. We obtained better predictions using Markov chain Monte Carlo to approximate the prior over the functions with a Gaussian distribution.

Moreover, we obtained poor results when there were few training data points at

each node. Indeed, in this situation, since the posterior over the weights is *not* sharply peaked around the maximum *a posteriori*, Laplace techniques give rise to poor approximations of the posterior over functions. Otherwise, we obtained accurate predictions and our experiments suggest that Laplace approximations and Markov chain Monte Carlo can be used in the BCM framework to combine multilayer perceptrons.

3.4.9 Conclusion

In this Chapter, we described some techniques that can be used to combine models trained using initially physically distributed data. We showed that most of these methods had to be extended in order to tackle our problem. The biggest part of our work consisted in applying the Bayesian committee machine to fuse neural network predictions. Thus, we demonstrated that the posterior distribution over the functions can be approximated using the Jacobian of $F_q = (f(w, x_1), f(w, x_2), \ldots, f(w, x_q))'$ to linearize the model and Laplace techniques to estimate the posterior over the weights. Moreover, through some experiments, we found that Markov chain Monte Carlo methods give rise to very good approximations of the prior over the functions.

We observed that some algorithms to combine models, such as parametric hierarchical Bayesian modelling, weighted error averaging, and Bayesian model averaging were not stable with respect to the number of points stored on each node. Except when there was few training data, we obtained the best predictions using the Bayesian committee machine.

We chose not to specify the computational costs of the different methods we used in this Chapter since they were all negligible.

In the next Chapter, we will assume that we are given a *single* training data set. We will show that it is possible to create a distributed learning environment by distributing data on nodes. Then, some approximation methods can be used to work in the Gaussian process framework when the number N of training data points is large.

Chapter 4

Gaussian Process Regression Over Large Data Sets

In the last two decades, linear models and neural networks have been widely used for regression. They are said to be parametric methods since they use well defined output functions to approximate underlying processes. Thus, in Section 2.1, we showed that linear models are based on a set of basis functions $\{\phi_0(x), \phi_1(x), \ldots, \phi_d(x)\}$ with fixed centres whereas neural networks use hidden units which depend on parameters that can be adjusted.

In section 2.1.3 we demonstrated that, when defined in the space of functions rather than the space of parameters, linear models are instances of Gaussian processes. Moreover, as mentioned previously, Neal (1996 [11]) showed in his work that for specific choices of priors $p(w)$, the prior over functions $p(F)$ of a one hidden layer neural network converges to well known Gaussian process priors as the number of hidden neurons tends to infinity. Those two observations motivate the idea of using Gaussian processes directly instead of parametric approaches.

Unfortunately, Gaussian processes have got a poor scaling with large data sets (Choudhury et al., 2002 [2]) since they require matrix inversions of which the computational cost and the memory requirement are of order $O(N^3)$ and $O(N^2)$ respectively, where N is the number of training data points considered. To overcome such lim-

itations, some sparse approximation methods have been developed. Rasmussen and Quinonero-Candela noted (2005 [16]) that all these approximation schemes treat exactly only a subset of variables called *latent variables* whereas the remaining variables are given some approximate, but computationally cheaper treatments.

In this Chapter our approach is different. Indeed, we investigate techniques that can be described in a distributed learning environment and that aim at speeding up Gaussian process regression. Through the Sections of this Chapter we demonstrate that the training and the prediction tasks can be treated separatly. Thus, after having reviewed the Gaussian process limitations, we show in Section 4.2 how the BCM can obtain forecasts when the number of training data points is large and the model hyperparameters given. Then, in Section 4.3, we investigate the factorization of the hyperposterior and we demonstrate how it can be used during the learning process. In section 4.3.2, we describe a very recent method called *Laplace propagation* that we applied as an optimization algorithm. Finally, we present the experiments that we carried out and we draw some conclusions.

4.1 Gaussian Process Limitations

As other regression models, Gaussian processes are based on two different steps. First, given a data set $\{\chi, T\}$, an algorithm is used to optimize either the likelihood function or the hyperposterior. This operation is commonly known as the training process. Then, given the solution $\hat{\theta}$, the prediction task consists of computing forecasts for any new input point. We recall that the hyperparameter θ that defines the Gaussian process covariance function has to be positive and so we constrain $\theta = \exp(\phi)$.

In the Gaussian process framework, as we show in the two next Sections, both steps require the inversion of the covariance matrix C given by

$$C(x_i, x_j) = k(x_i, x_j) + \beta^{-1}\delta(i-j) \ \forall x_i, x_j \in \chi$$

If we assume that we are given N training data points, then C has got N^2 elements. Thus, the computational cost and the amount of memory required for the matrix

inversion are of order $O(N^3)$ and $O(N^2)$ respectively.

4.1.1 Training of the hyperparameters

As described in Section 2.1.3, the log-likelihood function is given by

$$\ln p(T|\theta) = -\frac{1}{2}\ln\det C - \frac{1}{2}T'C^{-1}T - \frac{N}{2}\ln(2\pi). \tag{4.1}$$

In order to use an optimization algorithm, such as scale conjugate gradient, the gradient of (4.1) has to be computed. Since $\theta = \exp(\phi)$, we have

$$\frac{\partial C}{\partial \phi} = \frac{\partial C}{\partial \theta}\frac{d\theta}{d\phi} = \frac{\partial C}{\partial \theta}\exp(\phi).$$

Using standard formulae for matrix differentiation, we obtain

$$\frac{\partial}{\partial \phi}\ln p(T|\theta) = -\frac{\exp(\phi)}{2}\left\{\mathrm{tr}\left(C^{-1}\frac{\partial C}{\partial \theta}\right) - T'C^{-1}\frac{\partial C}{\partial \theta}C^{-1}T\right\}.$$

If we now consider the hyperposterior, we have

$$\ln p(\phi|T,\sigma^2_{prior}) = -\frac{1}{2}\ln\det C - \frac{1}{2}T'C^{-1}T - \frac{\phi'\phi}{2\sigma^2_{prior}} - \frac{N}{2}\ln(2\pi) - \frac{d}{2}\ln(2\pi) - \frac{d}{2}\ln(\sigma^2_{prior}) - \ln(Z).$$

and its corresponding gradient is given by

$$\frac{\partial}{\partial \phi}\ln p(\phi|T,\sigma^2_{prior}) = -\frac{\exp(\phi)}{2}\left\{\mathrm{tr}\left(C^{-1}\frac{\partial C}{\partial \theta}\right) - T'C^{-1}\frac{\partial C}{\partial \theta}C^{-1}T\right\} - \frac{\phi}{\sigma^2_{prior}}.$$

4.1.2 Predictions

Given a new input x_{N+1}, we recall that the predictive distribution is given by

$$\hat{t}(x) \sim p(t_{N+1}|T) = N\left(t; k'C^{-1}T, c - k'C^{-1}k\right).$$

4.2 The Bayesian Committee Machine for Gaussian Process Predictions

In Chapter 3, we saw that the Bayesian committee machine can be used to combine any kind of Bayesian regression models. For our experiments, we applied it to fuse

radial basis function networks and multilayer perceptrons trained using the evidence procedure. We now see the BCM as a technique to obtain approximations of Gaussian process predictions when the number of training data points is large.

Assume that we are given a *single* training data set $\{\chi, T\}$ of size N. In order to use the BCM, we split the data into several data sets that are then stored on nodes. Following Tresp, better approximations can be achieved by first using some clustering techniques and assigning the data of each cluster to a separate node. The predictive distribution is then estimated using

$$p(T_q|\{\chi_j, T_j\}_{j=1}^m) \propto \frac{1}{p(T_q)^{m-1}} \prod_{j=1}^m p(T_q|\chi_j, T_j), \tag{4.2}$$

where, following Section 3.3.2, the prior and the different posteriors are given by

$$p(T_q) = N(T_q; 0, \Sigma),$$

and

$$p(T_q|\chi_j, T_j) = N(T_q; E[T_q]_j, \mathrm{cov}(T_q)_j).$$

Here, we used $E[T_q]_j = A' C_j^{-1} T_j$ and $\mathrm{cov}(T_q)_j = B - A' C_j^{-1} A$, where C_j defines the Gaussian process covariance matrix of node j. Moreover, to simplify the notations we ignored the dependency of A on the training data set $\{\chi_j, T_j\}$. Since all these distributions are Gaussians, following (3.5), the mean and the covariance matrix of the predictive distribution have got a simple form

$$E[T_q] = \mathrm{cov}(T_q) \sum_{j=1}^m (\mathrm{cov}(T_q)_j)^{-1} E[T_q]_j, \tag{4.3}$$

with

$$\mathrm{cov}(T_q)^{-1} = -(m-1)\Sigma^{-1} + \sum_{j=1}^m (\mathrm{cov}(T_q)_j)^{-1}. \tag{4.4}$$

Thus, we observe that instead of having to invert a N-dimensional matrix as in the Gaussian process framework, the BCM approximation requires inversions of α-dimensional and q-dimensional matrices, where α is the number of points stored on each node and q the number of query points. The computational cost to calculate each term in the sums (4.3) and (4.4) is of order $O(\alpha^3)$ if $\alpha \gg q$. Conversely, if $q \gg \alpha$, the

computational cost becomes $O(q^3)$. Tresp suggests to use $\alpha = q$ since then all matrices that have to be inverted have the same size. Thus, the BCM overall computational cost is linear with respect to N

$$O(m\alpha^3) = O(\frac{N}{\alpha}\alpha^3) = O(N\alpha^2) = O(N),$$

if $\alpha \ll N$.

We found that (4.3) and (4.4) can be computed with an iterative algorithm, such that the memory which is used at one iteration is reused at the next iterations. Indeed, agent j calculates $\mathrm{cov}(T_q)_j)^{-1}$, $\mathrm{cov}(T_q)_j)^{-1}E[T_q]_j$, and repectively adds those terms to

$$-(m-1)\Sigma^{-1} + \sum_{i=1}^{j-1}(\mathrm{cov}(T_q)_i)^{-1},$$

and

$$\sum_{i=1}^{j-1}(\mathrm{cov}(T_q)_i)^{-1}E[T_q]_i.$$

This process is repeated until $j = m$. As before, if we fix $\alpha = q$, the overall memory required is of order $O(\alpha^2)$.

To conclude, for a fixed α, if we assume that the number m of agents is *not* bounded, then the Bayesian committee machine can be used to obtain approximations of Gaussian process predictions when the number N of training data points is large.

4.3 Factorization of The Hyperposterior

In the previous Section, we showed that the Bayesian committee machine can be used to give to Gaussian process predictions some approximate but computationally cheaper treatments. However, we did not specify how to train the hyperparameter θ. In his work, Tresp suggests to choose randomly some points of the original training data set $\{\chi, T\}$. Then, the corresponding Gaussian process likelihood function or hyperposterior can be optimized. However, because of the problems we saw previously, if N is large, then only a very small portion of $\{\chi, T\}$ can be used for training.

We found that it was possible to approximate the learning process by keeping the original data set as a whole. Indeed, using equations inspired by the BCM (4.2), the

hyperposterior can be factorized

$$p(\phi|\{\chi_j, T_j\}_{j=1}^m, \sigma^2_{prior}) \propto \quad p(\phi|\sigma^2_{prior}) \prod_{j=1}^m p(T_j|\chi_j, \phi), \tag{4.5}$$

$$\propto \quad \frac{1}{p(\phi|\sigma^2_{prior})^{m-1}} \prod_{j=1}^m p(\phi|\chi_j, T_j, \sigma^2_{prior}), \tag{4.6}$$

where $p(\phi|\sigma^2_{prior})$ is a chosen distribution that reflects our prior belief and to go from (4.5) to (4.6), we used Bayes' rule on each term in the product.

As before, this expression is exact if the different data sets $\{\chi_j, T_j\}$ are unconditionally independent. A good approximation might be achieved by clustering the data and then assigning the data of each cluster to a separate node.

In this part, our work consisted of experimenting two methods to optimize the factorized hyperposterior.

4.3.1 Shared hyperparameters

This approach is inpired by Schwaighofer's work (2005, [17]) and aims at using directly an algorithm, such as scale conjugate gradient, to optimize the factorized hyperposterior. It requires evaluations of the logarithm of (4.6)

$$\ln p(\phi|\{\chi_j, T_j\}_{j=1}^m, \sigma^2_{prior}) = -(m-1) \ln p(\phi|\sigma^2_{prior}) + \sum_{j=1}^m \ln p(\phi|\chi_j, T_j, \sigma^2_{prior}). \tag{4.7}$$

and of the corresponding gradient

$$\nabla \ln p(\phi|\{\chi_j, T_j\}_{j=1}^m, \sigma^2_{prior}) = -(m-1) \nabla \ln p(\phi|\sigma^2_{prior}) + \sum_{j=1}^m \nabla \ln p(\phi|\chi_j, T_j, \sigma^2_{prior}). \tag{4.8}$$

After some iterations, we obtain the solution $\hat{\theta} = \exp(\hat{\phi})$.

If α represents the number of points stored on each node, then we know that the terms in the sums (4.7) and (4.8) have got a computational cost and a memory requirement of order $O(\alpha^3)$ and $O(\alpha^2)$ respectively. Thus, if we apply an iterative process, such that the memory used at one step is reused at the next step, the overall computational cost and memory requirement to evaluate either the logarithm of the factorized hyperposterior or its gradient are given by

$$O(m\alpha^3) = O(\frac{N}{\alpha}\alpha^3) = O(N\alpha^2) = O(N),$$

and $O(\alpha^2)$.

To conclude, as for the Bayesian committee machine and given a fix α, if the number m of nodes is *not* bounded, then the whole data set $\{\chi, T\}$ can be used for the training even if the number N of data points is large. An important aspect of this approach is that the *exact* expression of the factorized hyperposterior is used.

4.3.2 Individual hyperparameters

In the previous Section, we described a *single* optimization procedure where the log-hyperposterior and its gradient are decomposed into sums of simpler terms thanks to the factorization (4.6). In particular, we showed that this method gives rise to a *single* hyperparameter $\hat{\theta}$. We now present a different approach. Indeed, we assume that local agents use the distributed data sets $\{\chi_j, T_j\}$ to train. In other words, they optimize independently the different distributions $p(\phi|\chi_j, T_j, \sigma^2_{prior})$ to give rise to m solutions $\hat{\theta}_j = \hat{\phi}_j$.

Local Laplace approximations

In order to obtain a single estimate $\hat{\theta}$, we experimented local Laplace approximations. Indeed, each agent j estimates its own hyperposterior with a Gaussian distribution $N\left(\phi; \phi_j, H_j^{-1}\right)$, where H_j is the hessian matrix of $-\ln p(\phi|\chi_j, T_j, \sigma^2_{prior})$. Then, using (4.6) and basic properties of the Gaussian distribution (see Section 3.2.4), we calculate

$$\frac{1}{p(\phi|\sigma^2_{prior})^{m-1}} \prod_{j=1}^{m} p(\phi|\chi_j, T_j, \sigma^2_{prior}) \approx N\left(\phi; \phi_{fus}, H_{fus}^{-1}\right),$$

where $\phi_{fus} = H_{fus} \sum_{j=1}^{m} H_j \phi_j$ and $H_{fus} = \sum_{j=1}^{m} H_j$. Thus, we use the hyperparameter $\hat{\theta} = \exp(\phi_{fus})$ for the prediction task.

In Section 4.3.2, we show that the calculation of the hessian matrix at node j requires the inversion of the Gaussian process covariance matrix C_j. Moreover, we know that the computational cost and the memory requirement for each agent to optimize its own hyperposterior are of order $O(\alpha^3)$ and $O(\alpha^2)$ respectively. Thus, if we apply an iterative process, such that the memory used at one step is reused at the

next step, the overall computational cost and memory requirement for the optimization procedures and the local Laplace approximations are given by

$$O(m\alpha^3) = O(\frac{N}{\alpha}\alpha^3) = O(N\alpha^2) = O(N),$$

and $O(\alpha^2)$.

We recall that we obtained the same characteristics in the method described in the previous Section. However, we now use local approximations and *not* the exact factorized hyperposterior. Thus, we expect to get less accurate results.

The negative log-hyperposterior hessian matrix

The expression of the negative log-likelihood hessian matrix was first derived in 2005 (Zhang and Leithead, [26]). It is given by

$$L_{ab} = -\frac{1}{2}\operatorname{tr}\left(C^{-1}\frac{\partial^2 C}{\partial\phi_a\partial\phi_b}\right) + \frac{1}{2}\operatorname{tr}\left(C^{-1}\frac{\partial C}{\partial\phi_b}C^{-1}\frac{\partial C}{\partial\phi_a}\right) - T'C^{-1}\frac{\partial C}{\partial\phi_b} \tag{4.9}$$

$$\times C^{-1}\frac{\partial C}{\partial\phi_a}C^{-1}T + \frac{1}{2}T'C^{-1}\frac{\partial^2 C}{\partial\phi_a\partial\phi_b}C^{-1}T, \quad \forall a, b \in \{1, 2, \ldots, d\}, \tag{4.10}$$

where d is the dimensionality of $\theta = \exp(\phi)$.

Thus, since $p(\phi|T, \sigma^2_{prior}) \propto p(T|\phi)p(\phi|\sigma^2_{prior})$, the negative log-hyperposterior hessian matrix is defined as

$$H_{ab} = L_{Ab} + \frac{I}{\sigma_{prior}}, \quad \forall a, b \in \{1, 2, \ldots, d\},$$

if we use an isotropic Gaussian distribution $p(\phi|\sigma^2_{prior}) = N\left(\phi; 0, \sigma^2_{prior}I\right)$.

4.4 Laplace Propagation

In the previous Section with demonstrated that after having split the data set $\{\chi, T\}$ into multiple data sets $\{\chi_j, T_j\}$ stored on nodes, an equation inspired by the Bayesian committee machine (4.2) can be used to factorize the hyperposterior $p(\phi|\{\chi_j, T_j\}_{j=1}^m, \sigma^2_{prior})$. Then, we showed that two methods are applicable for the optimization procedure even if the number N of training data points is large.

We now present an algorithm, called Laplace Propagation (LP), that was introduced very recently in the machine learning community (Smola et al., 2004 [18]). Assume that we are given the factorized distribution

$$p(\phi|\{\chi_j, T_j\}_{j=1}^m, \sigma^2_{prior}) \propto p(\phi|\sigma^2_{prior}) \prod_{j=1}^m p(T_j|\chi_j, \phi).$$

It is important to note that this expression is similar to our previous factorized hyperposterior. We simply do not use Bayes' rule to invert each term $p(T_j|\chi_j, \phi)$ in the product (4.6). According to Smola, the LP strategy relies on the assumption that if we succeed in finding good approximations of those terms by $\tilde{p}(T_j|\chi_j, \phi)$, then we will obtain an approximate maximizer of $p(\phi|\{\chi_j, T_j\}_{j=1}^m, \sigma^2_{prior})$ by maximizing

$$p(\phi|\sigma^2_{prior}) \prod_{j=1}^m \tilde{p}(T_j|\chi_j, \phi).$$

This requires good approximations of each of the $p(T_j|\chi_j, \phi)$ at the maximum of $p(\phi|\{\chi_j, T_j\}_{j=1}^m, \sigma^2_{prior})$. This can be insured by maximizing

$$p(\phi|\sigma^2_{prior}) p(T_j|\chi_j, \phi) \prod_{i \neq j}^m \tilde{p}(T_i|\chi_i, \phi). \tag{4.11}$$

Laplace methods are then used to approximate $p(T_j|\chi_j, \phi)$ using

$$\ln \tilde{p}(T_j|\chi_j, \phi) \approx \ln p(T_j|\chi_j, \phi_j) + g_j'(\phi - \phi_j) + (\phi - \phi_j)' H_j (\phi - \phi_j),$$

where ϕ_j is the maximum of (4.11), $g_j = \nabla \ln p(T_j|\chi_j, \phi_j)$, and $H_j = \nabla\nabla \ln p(T_j|\chi_j, \phi_j)$.

The algorithm starts by fixing $\tilde{p}(T_j|\chi_j, \phi) = \text{cst}$, $\forall j \in \{1, 2, \ldots, m\}$. Then, we have to establish methods for updating the estimates. One approach consists of performing such updates sequentially. In other words, at iteration k, we approximate $p(T_k|\chi_k, \phi)$ by $\tilde{p}(T_k|\chi_k, \phi)$. This new estimate is then directly used for the optimization of (4.11) at iteration $k + 1$. It is also possible to update our approximations in parallel. Thus, at iteration k, each node j optimizes (4.11) and we obtain m solutions $\theta_j = \exp(\phi_j)$. Then, all the nodes update their corresponding approximations $\tilde{p}(T_j|\chi_j, \phi)$. In this approach, the algorithm stops when all the hyperparameters agree

$$\theta_j = \hat{\theta}, \ \forall j \in \{1, 2, \ldots, m\}.$$

We chose this method to update the different estimates. Thus, the LP has got a very interesting property in our framework. Indeed, at the first iteration, since we are given $\tilde{p}(T_j|\chi_j, \phi) = \text{cst}, \ \forall j \in \{1, 2, \ldots, m\}$, each node j optimizes (4.11) which is equivalent to optimize

$$p(T_j|\chi_j, \phi) \propto p(\phi|\sigma^2_{prior}) p(T_j|\chi_j, \phi).$$

In other words, all the nodes optimize indendently their own hyperposterior and we have $g_j = \nabla \ln p(T_j|\chi_j, \phi_j) = 0$ and

$$\ln \tilde{p}(T_j|\chi_j, \phi) \approx \ln p(T_j|\chi_j, \phi_j) + (\phi - \phi_j)' H_j (\phi - \phi_j). \tag{4.12}$$

Thus, after the first iteration, all the solutions $\theta_j = \exp(\phi_j)$ are exactly the ones that we obtained using the method presented in Section 4.3.2. Whereas the algorithm stopped at this point, we are now free to continue the approximation.

Since each node optimizes (4.11) and evaluates its own negative log-likelihood hessian matrix, then the overall computational cost and memory requirement are the same as with local Laplace approximations, that is $O(N)$ and $O(\alpha^2)$.

4.5 Experiments

In the previous Sections, we showed that the Bayesian committee machine can approximate Gaussian process predictions when the number N of training data points is large. Moreover, we demonstrated through three different algorithms that using factorized distributions, all the data can be used for the training procedure. We recall that we showed that all these methods have got the same order $O(N)$ of computational cost if the number m of models is *not* bounded. However, some of those techniques use optimization algorithms, such as scale conjugate gradient, and we have not discussed yet how fast they converge to solutions $\hat{\theta} = \exp(\phi)$.

In this Section, through the experiments on the scatterometry data, our goal is twofold. First, we want to analyse how fast and accurate the methods that we have seen in this Chapter are. Then, we want to compare those results with multilayer perceptrons and radial basis function networks trained using the evidence procedure.

We are given a data set with 180000 data points. We split the data into two data sets of the same size. The first one is used for the training whereas the second aims at testing the models. Because the number N of training data points is very large, Gaussian processes are *not* applicable. Indeed, we found that programs based on those regression methods stop a few seconds after they started because of *out-of-memory* problems. Thus, to estimate Gaussian process predictions, we use the Bayesian committee machine and to approximate the training procedure, we apply methods presented in Sections (4.3.1), (4.3.2), and (4.4). We also want to experiment if, as noted by Tresp, better approximations can be obtained by clustering the training data and then assigning the data of each cluster to a separate node.

4.5.1 Results

Model	RMS train	RMS test	Training	Prediction
RBF (2 hidden units)	0.8160	0.8327	27s	25s
RBF (10 hidden units)	0.5724	0.5867	2.29min	36s
RBF (50 hidden units)	0.3489	0.3454	9.28min	1.53min
MLP (2 hidden units)	0.4428	0.4534	2s	15s
MLP (10 hidden units)	0.3453	0.3479	25s	17s
MLP (50 hidden units)	0.3444	0.3482	13.04min	19s
SH + BCM + *non* clustered data	0.3635	0.3642	4.31h	5.26h
SH + BCM + clustered data	0.3489	0.3517	4.29h	5.10h
LLA + BCM + *non* clustered data	0.4032	0.4057	3.57h	5.10h
LLA + BCM + clustered data	0.3960	0.3975	3.41h	5.20h
LP + BCM + *non* clustered data	0.3721	0.3739	4.20h	5.32h
LP + BCM + clustered data	0.3491	0.3530	4.20h	5.20h

Table 4.1: Results obtained using several regression models (*Model*) on the scatterometry data. Five techniques are applied : radial basis function networks (*RBF*), multilayer perceptrons (*MLP*), shared hyperparameters (*SH*), local Laplace approximations (*LLA*), Laplace propagation (*LP*). Training root mean square error (*RMS train*). Test root mean square error (*RMS test*). Time to train a model using the training data set as a whole (*Training*). Time to obtain predictions for all the inputs in the test set (*Prediction*)

4.5.2 Conclusion

To analyse our results, we use the multilayer perceptrons and radial basis function networks as references. First of all, we can point out that using approximation methods, both for the training and prediction task, we managed to work in the Gaussian process framework using a very large training data set. Thus, on average, we obtained predictions for all the inputs in the test set in five hours using the Bayesian committee machine. Moreover, it took us between three and four hours and a half to find solutions $\hat{\theta} = \exp(\hat{\phi})$ depending on the methods considered. Regarding the accuracy of our approximations, we always obtained better results when the data was clustered before being distributed and stored on different nodes. More precisely, we got the best results using shared hyperparameters. This training method is also the slowest one. Techniques based on local Laplace approximations are faster but give rise to less accurate predictions. The best compromise was found using Laplace propagation. Indeed, we obtained almost the same forecasts as the method based on shared hyperparameters, in a bit more reasonable time.

To work in the Gaussian process framework, when the number of training data points is large, our experiments suggest to use Laplace propagation for the optimization of the hyperparameters and the Bayesian committee machine to approximate Gaussian process predictions. However, we found that a 10 hidden unit multilayer perceptron and a 50 radial basis function network can obtain similar predictions much faster. Nevertheless, it is important to note that the underlying process of the data set that we are given is known to be quite simple to characterize. Indeed, it maps only three inputs to one target and it is monotome. It would be particularly interesting to consider a more complex data set and to observe if our approximations give rise to a better model than multilayer perceptrons or radial basis function networks.

Chapter 5

Conclusion

Through this thesis we tackled two critical applications that can be described in a distributed learning environment. First, in Chapter 3, we defined some techniques that can be applied to combine multilayer perceptrons and radial basis function networks trained using physically distributed data. In particular, we showed that is is possible to work in the space of parameters or to fuse directly the different model predictions, for a given input x, in order to obtain a more accurate predictive model. We also derived some approximations of priors and posteriors over the functions so that the Bayesian committee machine can be used to combine multilayer perceptrons. Thus, we demonstrated that the posterior distribution over the functions can be approximated using the Jacobian of $F_q = (f(w, x_1), f(w, x_2), \ldots, f(w, x_q))'$ to linearize the model and Laplace techniques to estimate the posterior over the weights. Moreover, we found that Markov chain Monte Carlo methods give rise to very good approximations of the prior over the functions.

Then, in Chapter 4, using approximation methods, both for the training and prediction task, we managed to work in the Gaussian process framework using a very large training data set. We used the Bayesian committee machine to approximate Gaussian process predictions. Moreover, through three different algorithms, we demonstrated that using factorized hyperposteriors, all the data can be used for the training procedure.

Bibliography

[1] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer Science+Business Media, LLC, first edition, 2006.

[2] A. Choudhury, P. B. Nair, and A. J. Keane. A data parallel approach for large-scale gaussian process modelling. In *the Second SIAM International Conference on Data Mining*, 2002.

[3] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistics Society*, 39:1–38, 1977.

[4] R. L. Grossman, S. Bailey, S. Kasif, D. Mon, A. Ramu, B. Malhi, and A. Turinsky. The preliminary design of papyrus : A system for high performance, distributed data mining over clusters, meta-clusters and super-clusters. *Advances in Distributed and Parallel Knowledge Discovery*, pages 259–275, 1999.

[5] J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky. Bayesian model averaging : A tutorial. *Statistical Science*, 14:382–417, 1999.

[6] D. Lowe. The relevance and application of information fusion to financial analysis. Technical report, NCRG group, Aston University, 2001.

[7] D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4:415–447, 1992.

[8] D. J. C. Mackay. Introduction to gaussian processes. *Neural Networks and Machine Learning*, 1998.

[9] T. P. Minka. Bayesian model averaging is not model combination. July 2000. URL `http://www.stat.cmu.edu/minka/papers/bma.html`.

[10] I. T. Nabney. *Netlab, Algorithm for Pattern Recognition.* Springer, second edition, 2003.

[11] R. M. Neal. *Bayesian Learning for Neural Networks.* Springer, 1996.

[12] B-H. Park and H. Kargupta. Distributed data mining: Algorithms, systems, and applications. Technical report, Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 2002.

[13] D. Pena and D. Redondas. Bayesian curve estimation by model averaging. *Science Direct*, Computational Statistics & Data Analysis 50 (2006):688–709, September 2004.

[14] A. E. Raftery, F. Balabdaoui, T. Gneiting, and M. Polakowski. Using bayesian model averaging to calibrate forecast ensembles. Technical report, Deparment of Statistics, University of Washington, 2003.

[15] C. E. Rasmussen and C. K. I. Williams. *Gaussian Process for Machine Learning.* The MIT Press, 2006.

[16] R. C. E. Rasmussen and J. Quinonero-Candela. A unifying view of sparse approximate gaussian process rgression. *Journal of Machine Learning Research*, 6: 1939–1959, 2005.

[17] A. Schwaighofer. Gaussian process regression with bayesian committe machine. 2005. URL `http://ida.first.fraunhofer.de/~anton/software.html`.

[18] A. Smola, S. V. N. Vishwanathan, and E. Eskin. Laplace propagation. *Advances in neural information processing systems. MIT Press*, 16, 2004.

[19] A. Stoffelen. Scatterometry. *PhD Thesis, Utrecht University*, 1998.

[20] E. B. Sudderth, A. T. Ihler, W. T. Freeman, and A. S. Willsky. Nonparametric belief propagation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[21] V. Tresp. A bayesian committee machine. *Neural Computation*, 12:2719–2741, 2000.

[22] V. Tresp and K. Yu. An introduction to nonparametric hierarchical bayesian modelling with a focus on multi-agent learning. In *Proceedings of the Hamilton Summer School on Switching and Learning in Feedback Systems. Lecture notes in computer science*, 2004.

[23] C. K. I. Williams and C. E. Rasmussen. Gaussian processes for regression. In *Advances in Neural Information Processing Systems*. The MIT Press, 1996.

[24] R. Wright and Z. Yang. Privacy-preserving bayesian network structure computation on distributed heterogeneous data. In *Conference on Knowledge discovery and data mining*, 2004.

[25] K. Yamanishi. Distributed cooperative bayesian learning strategies. *Information and Computation*, 250:22–56, 1999.

[26] Y. Zhang and W. E. Leithead. Exploiting hessian matrix and trust-region algorithm in hyperparameters estimation of gaussian process. *Applied Mathematics and Computation*, 171:1264–1281, 2005.