In *Neural Networks and Machine Learning*, C. M. Bishop (editor), Springer-Verlag, 1998, pp. 97-129.

Assessing Relevance Determination Methods Using DELVE

Radford M. Neal

Dept. of Statistics and Dept. of Computer Science, University of Toronto
`radford@stat.utoronto.ca`, `http://www.cs.utoronto.ca/~radford/`

Abstract. Empirically assessing the predictive performance of learning methods is an essential component of research in machine learning. The DELVE environment was developed to support such assessments. It provides a collection of datasets, a standard approach to conducting experiments with these datasets, and software for the statistical analysis of experimental results. In this paper, DELVE is used to assess the performance of neural network methods when the inputs available to the network have varying degrees of relevance. The results confirm that the Bayesian method of "Automatic Relevance Determination" (ARD) is often (but not always) helpful, and show that a variation on "early stopping" inspired by ARD is also beneficial. The experiments also reveal some other interesting characteristics of the methods tested. This example illustrates the essential role of empirical testing, and shows the strengths and weaknesses of the DELVE environment.

1 Introduction

This paper has two purposes: To show how the performance of machine learning methods can be assessed using the DELVE environment, and to investigate some methods for dealing with learning problems in which the available inputs are of varying relevance. These topics are linked by the use of the DELVE environment to assess the performance of the relevance determination methods.

Below, I discuss the need for empirical assessments of learning methods, and the deficiencies of current practice in this area. I then discuss the problem of determining the relevance of the various inputs that may be available for use in a supervised learning problem (either regression or classification). Following this introduction, I describe the DELVE environment for conducting empirical assessments in more detail, and describe two types of neural network methods, based on Bayesian learning implemented using Markov chain Monte Carlo, and on ensembles of networks trained using early stopping. Bayesian learning can be done in conjunction with the technique of "Automatic Relevance Determination" (ARD), which past experiments suggest is beneficial. The mechanics of ARD are the inspiration for a variation of early stopping that we might hope will also be beneficial when some inputs are more relevant than others.

The DELVE environment is then used to assess the performance of the Bayesian and early stopping methods, with and without the use of relevance determination techniques. The results allow conclusions to be drawn about the worth of these methods, and about the strengths and weaknesses of the current DELVE environment.

1.1 The need for good empirical assessments of learning methods

One might hope that the performance of machine learning methods could be fully analysed theoretically, eliminating any need for empirical assessments of performance on actual problems. Theoretical analysis can be difficult, however. More fundamentally, without further knowledge, a complete analysis is generally impossible in principle, since the performance of a learning method usually depends on the real situation in which it is applied.

Bayesian methods do incorporate further knowledge of the problem, in the form of a prior distribution that is meant to capture the user's beliefs about the real situation. If the prior distribution accurately reflects these beliefs, a Bayesian might regard empirical tests of the method based on this prior as irrelevant. In practice, however, one can seldom be sure that the prior distribution used for a complex model actually captures the real prior beliefs. Partly, this is because the substantial effort required to ensure this cannot be justified when it is thought likely that a cruder approximation will suffice for the problem at hand. Accurate expression of prior beliefs is also limited by the difficulty of visualizing distributions in high dimensions. Empirical tests may reveal flaws in the specification of the model or prior, and may also be needed to assess the adequacy of the computational methods used to implement a Bayesian model, which often involve approximations, at least when allowed only a limited amount of computation time.

Methods justified within other frameworks will also often require empirical testing in order to assess their practical worth. Furthermore, performance on actual problems is the only criterion by which one can compare methods that have been justified within different philosophical frameworks, or that are based on vague intuitions rather than coherent principles. Empirical testing is how we will ultimately be able to see the relative merits of the Bayesian, minimum description length, structural risk minimization, and other approaches — though of course, no single test will be decisive in this regard.

Despite their importance, good empirical assessments are rare. In the statistical literature, it is quite common for newly-proposed methods to be illustrated on a problem that is real (though often stale), with the apparent intent of demonstrating that the method can produce interesting results in actual applications. In working through such problems, it is typical for all the data to be used in training the method, with the result that none is left for checking the accuracy of the results. This practice seems to result from a confusion between solving a real problem and doing methodological research. When the goal is to solve the problem, one should of course use all the data available, in one way or another. But when assessing the worth of a newly-proposed method, we need an independent check of its accuracy, which is obtainable only by testing on data not used during training.

Tests on data separate from that used in training are more common in the neural network and machine learning literatures. One approach is to divide a dataset into a "training" set and a "test" set. Several methods are then applied to the training set, on the basis of which each makes predictions for cases in the test set. The predictive performances of the methods are then compared. This methodology suffers from two serious problems. First, the datasets used are often quite small (less than 1000 cases), the test sets are even smaller, and consequently only quite large differences in performance can be reliably discerned. Second, this methodology ignores the possibility that the relative performance of the methods may vary with the random choice of training set.

In an attempt to overcome these problems, n-fold cross-validation assessments are sometimes performed, in which the data is divided into n portions, and n training runs are done, each on all but one portion, with the held-out portion being used as a test set. Unfortunately, the overlap of the training sets in this scheme introduces unknowable dependencies between runs, making formal statistical tests impossible. Investigations by Dietterich (1998) show that applying a t-test despite these dependencies can result in invalid results, though the situation is not as bad as for the widely-used resampled t-test procedure, based on multiple random splits into training and test sets, which is guaranteed to produce an apparently "significant" result if enough random splits are used. Dietterich introduces a new "5x2cv" test intended to replace these flawed tests, but by his own evidence, this test is itself deeply flawed — it is based on numerous incorrect independence assumptions, and in one example has a Type I error probability that is off by a factor of two.

The root cause of this sad state of affairs is an insistence on trying to answer the following question: "Given two learning algorithms A and B and a small data set S, which algorithm will produce more accurate classifiers when trained on data sets of the same size as S?" (Dietterich 1998). This question is fundamentally unanswerable, given that only a single instance of a dataset of size S is available. Small datasets are simply not suitable for tests of learning methods. One should use large datasets instead, from which one can select several non-overlapping training sets of whatever size is of interest, along with a separate test set that is reasonably large.

Invalid results can be obtained with this approach too, however, as illustrated by Finnoff, Hergert, and Zimmerman (1990). In their experiments, they use six separate training sets, each accompanied by a test set with 1000 cases. Good comparisons should be possible with such a setup, but unfortunately they destroy the validity of the results by setting parameters of the methods based on information from all six training sets. Furthermore, they derive a statistical test based on the dubious assumption that the relative performance of two methods is the same for all training sets. Not wishing to go to the effort of computing this test, they use an approximation which discards the pairing information in the experiment, which usually greatly reduces the power of a test to detect differences. They then use extremely generous significance levels of 0.10 and 0.25 (or possibly 0.20 and 0.50, depending on context). Their results would therefore appear to be essentially meaningless. Other researchers have conducted equally flawed experiments.

The DELVE project (Rasmussen, *et al* 1996) aims to improve the quality of empirical assessments by building a collection of large datasets suitable for empirical assessments, along with software that supports a valid experimental methodology and valid analysis of the results. Collecting, producing, or adapting suitable datasets is not a simple task, which goes some ways toward explaining why unsuitable datasets are often used. We do not yet have the variety of datasets that we would like to have in DELVE. We do have a first version of the software, and a modest collection of results.

DELVE is described more fully in Section 2. First, however, I will discuss a particular research problem that will be used in this paper to illustrate how empirical assessments using DELVE can help answer interesting questions.

1.2 The problem of dealing with varying input relevance

For learning to be successful, we must always discard much information that is known *a priori* to be irrelevant, as otherwise we will be fooled by the chance associations that we are likely to encounter if we search for relationships with a huge number of input variables. But what should we do when we think that some inputs *might* be important, but might also be of little relevance? Can we use the data to determine the degree to which each input is relevant, and thereby avoid both the cost of not using inputs that are useful, and the cost of being misled by chance associations with inputs that have little relevance?

One approach is to select a subset of input variables using some criterion that balances data fit and model complexity (eg, AIC). When number of inputs is large, considering all possible subsets is infeasible, but one can try adding variables one at a time (forward selection), or removing them one at a time (backward elimination). This approach has long been used in the context of linear regression. Bonnlander (1996) reviews variable selection methods for neural networks, and develops one based on mutual information.

Variable selection seems to me to be a dubious procedure, since if we think an input *might* be relevant, we will usually also think that it is probably at least a *little bit* relevant. Situations where we think all variables are either highly relevant or not relevant at all seem uncommon (though not, of course, impossible). Accordingly, we should usually not seek to eliminate some variables, but rather to adjust the degree to which each variable is considered relevant. We may hope that a method of this sort for determining the relevance of inputs will be able to avoid paying undue attention to the less relevant inputs, while still making use of the small amount of extra information that they provide.

This is the philosophy behind the Bayesian method of "Automatic Relevance Determination" (ARD) (MacKay 1994; Neal 1996) for multilayer perceptron networks. In this method, a *hyperparameter* is associated with each input, which controls the size of the weights associated with connections out of that input. If the hyperparameter for an input is small, weights for that input will likely be small, and the input will therefore have only a small effect on the network's predictions. These relevance hyperparameters are set in a Bayesian fashion, according to the resulting probability of the observed data.

The ARD procedure has a high-level justification, as the correct thing to do given prior knowledge that some inputs may be less relevant than others. One can also look at the low-level mechanics of how ARD operates, however, which one can try to adapt for use in a non-Bayesian context. In this paper, I will investigate a variation of training with early stopping that uses an ARD-like method for dealing with inputs of varying relevance. The aim is to determine whether such a method is actually an improvement over a plain early stopping method, and how its performance compares to the Bayesian ARD method. Some other aspects of these methods will be examined along the way.

2 The DELVE environment

The DELVE environment, developed at the University of Toronto, aims to support valid empirical assessments of learning methods for regression and classification applications. We are especially interested in methods for finding non-linear relationships in high-dimensional data. Neural networks, classification trees, and nearest neighbor methods are among the many approaches that have been taken to such problems. Each such approach has many variations. We hope that DELVE will help in determining which variations are actually useful, and will allow new variations to be easily assessed.

DELVE consists of an archive of datasets for use in assessing learning methods, a set of conventions and software that support experiments with this data, and an archive of the results of such assessments. I will here give a brief and somewhat simplified sketch of the main aspects of DELVE. For complete details, see the DELVE manual (Rasmussen, *et al* 1996) and the PhD thesis by Rasmussen (1996). The DELVE archive is located at `http://www.cs.utoronto.ca/~delve/`.

2.1 From datasets to task instances

A DELVE dataset consists of a set of *cases*. For each case, the values of certain *attributes* are recorded. At present, the cases in a DELVE dataset are considered to be independent, though in future we may support experiments with time series data or in which other sorts of dependencies are present.

DELVE datasets can have various origins. Natural datasets from real-world learning problems are useful because they are representative of the problems we would actually like to solve. Artificial datasets based on simple formulas are easy to construct, and can allow tests that are narrowly focused on particular issues. At present, most DELVE datasets are of an intermediate type, produced by realistic simulation programs that mimic a situation where one might well wish to use a learning method. This allows us to generate an unlimited number of cases, and also lets us produce families of related datasets that can provide information about which factors influence the performance of the learning methods. We hope that the simulations used are realistic enough that they are representative of real applications.

The first step in using a dataset to assess learning methods is to define what we call a *prototask*, in which one attribute is identified as the *target* to be predicted, and some other attributes are identified as *inputs*, which are available for use in predicting the target. When the target is a real number, we have a regression prototask; when the target is from a finite set, we have a classification prototask. In this paper, only regression prototasks are used, but DELVE supports classification as well.

For each prototask, we can define one or more *tasks*, in which the number of *training cases* is specified. It is meaningful to ask how well a particular learning method does on a task, as judged by the expected loss when it is used to predict the target in a test case using the information from a training set of the specified size. Several loss functions are supported by DELVE, but in this paper only squared error loss will be used.

The expected loss of a method on a task is the average loss with respect to the random selection of a training set of the size specified for the task and the random selection of a test case. Both these random selections are from the real distribution from which the cases were drawn, *not* from the particular set of cases that we have available for testing. The expected loss can therefore only be estimated, with some uncertainty, on the basis of performance on several *task instances*, each of which consists of a training set and a set of test cases. The training sets for different task instances are always disjoint, because there is no way of accounting for the dependencies that would occur if training sets overlapped. In this paper, the test sets used are also non-overlapping. This simplifies the statistical analysis, and is more efficient when there is no limit on the amount of simulated data that can be generated.

Most DELVE prototasks are associated with a set of five tasks with training set sizes of 64, 128, 256, 512, and 1024. Eight task instances are usually used for each of these tasks, except that often only four instances are used for the task with 1024 training cases. It is important that such conventions be established for each prototask, so that the results of experiments by different researchers can be compared. However, different conventions might be used for some new prototask if this was thought desirable.

2.2 Defining and running a learning method

Before a learning method can be assessed, it must first be carefully defined. A fully automatic method can be defined by a program that implements it, perhaps supplemented by some explicit instructions on what the human operator should do in certain situations (eg, reduce the gradient descent learning rate for a neural network by a factor of two if instability is observed). Methods that require substantial human judgement could in theory be defined by a description of the method that could be understood by any data analyst with a suitable background. Properly assessing such a manual method would be quite costly, however, as many analysts would need to learn how to use the method, and then analyse instances of various tasks. All the results currently archived in DELVE are for automatic methods.

A program implementing a method reads the inputs and targets for the training cases, plus the inputs for the test cases, and produces guesses at the targets in the test cases. The guess for one test case is meant to be made without reference to the inputs for other test cases. Some methods will make use of random numbers — for example, to randomly initialize the weights in a neural network, or as part of a Monte Carlo method for evaluating integrals. Such methods should use a different random number seed for each task instance. Otherwise, the results might depend on the particular seed chosen, and if so would be very fragile, subject to change whenever a minor modification to the program alters the sequence of calls to the random number generator, or when the task is slightly altered (eg, the inputs are re-ordered).

It is up to the writer of a learning method how the training data and test inputs are used to make predictions for the test targets. However, DELVE provides support for some common encoding and normalization procedures, which many researchers may choose to use if they are not specifically investigating this aspect of learning. All the methods discussed in this paper use the default DELVE normalization for inputs and targets, which is to shift them so that they have median zero over the training set, and to scale them so that over the training set their average absolute deviation from the median is one. The DELVE software creates data files for each task instance using this normalizing transformation (which is usually different for each instance). The program implementing the method can then be run, after which DELVE will transform the method's guesses for the normalized test targets back to the original form. For each test case, the squared error loss with respect to the actual target value is then computed.

2.3 Analysing the results

The result of running a learning method on a task is a set of files that record the guesses the method made for each test case in each task instance. These guesses are retained for later analysis, which might potentially include a detailed comparison of the guesses made by different methods. At present, however, the only analyses supported by DELVE are estimation of the expected loss (eg, squared error) on a task for a single method, and the comparison of the expected losses on a task for two methods.

These analyses are fairly simple when both the training sets and the test sets in different task instances are disjoint. Suppose that we have I task instances, for each of which we have both a training set and a set of J test cases. Suppose that when method A is trained on the training set for instance i, its loss when predicting the target for test case j of instance i is y_{ij}. We can compute the average loss for test cases in instance i, written $\bar{y}_i$, and the average loss over all instances, written $\bar{y}$, as follows:

$$\bar{y}_i \;=\; \frac{1}{J}\sum_{j=1}^{J} y_{ij}, \qquad \bar{y} \;=\; \frac{1}{I}\sum_{i=1}^{I} \bar{y}_i \tag{1}$$

Note that $\bar{y}$ is an unbiased estimate of the true expected loss for this method with respect to the random selection of a training set and a test case. Because the task instances are independent, it is possible to estimate the standard error of $\bar{y}$ (see Rasmussen, *et al* (1996) for details). A confidence interval for the true expected loss could be obtained if one were willing to assume that the distribution of $\bar{y}$ is close to Gaussian, which is reasonable if I is large, though perhaps not when $I = 8$. Such confidence intervals are not computed by DELVE at present, however.

Our main interest is usually in comparing two methods, A and B, with respect to their expected loss on a task. If these two methods have been run in the standard way within the DELVE environment, we will have the results for both methods on the *same* training sets and test cases. We can therefore do a paired comparison of these results, which is more powerful than a comparison using results on different training and test sets. Let x_{ij} be the *difference* in the loss for method A and the loss for method B when each is trained on the training set for instance i and then used to predict the target in test case j of instance i. We can compute averages for these differences in the same way as we did for the losses of a single method:

$$\bar{x}_i \;=\; \frac{1}{J}\sum_{j=1}^{J} x_{ij}, \qquad \bar{x} \;=\; \frac{1}{I}\sum_{i=1}^{I} \bar{x}_i \tag{2}$$

Here, $\bar{x}$ is an unbiased estimate for how much greater the expected loss of method A is than the expected loss of method B. If $\bar{x}$ is positive, then we have evidence that method B is better than A, and conversely, if $\bar{x}$ is negative, we have evidence that A is better.

However, even if there is no true difference in expected performance between A and B, we will generally obtain some non-zero value for $\bar{x}$, and hence some indication that one or the other method is better than the other. Indeed, even if B is better than A, there is some chance that $\bar{x}$ will be negative, indicating the reverse. We need some way of telling whether the sign of $\bar{x}$ is a reliable guide to which method is better. Because the $\bar{x}_i$ are independent, we can use a t-test for this, based on the following t-statistic:

$$t = \bar{x} \left[\frac{1}{I(I-1)} \sum_{i=1}^{I} (\bar{x}_i - \bar{x})^2 \right]^{-1/2} \tag{3}$$

If the $\bar{x}_i$ have a Gaussian distribution, this t-statistic will have a t-distribution with $I-1$ degrees of freedom. On this basis, we can find the p-value for the comparison, which is the probability of obtaining a t-statistic at least as large (in absolute value) as that observed, if in fact the two methods have the same expected loss. If the p-value is small (traditionally, less than 0.05), we have good evidence that the apparently better method is actually better.

There is no strong reason to believe that the distribution of the $\bar{x}_i$ is actually Gaussian, so use of this t-test is not beyond question. Fortunately, however, the t-test is fairly robust in this respect, so the results are usable, if treated with appropriate caution. The impact of the $\bar{x}_i$ being non-Gaussian would be less if I were larger, but we have nevertheless chosen $I = 8$ for most DELVE tasks because many interesting methods take quite a long time to run.

DELVE uses a more complex analysis of variance procedure when the same test set is used for each task instance, as may be necessary for a natural dataset with a fairly small number of cases. DELVE does not use overlapping training sets, since there is no way of obtaining valid standard errors or p-values in this context, at least not without detailed prior knowledge of the behaviour of the methods, which is presumably not available for methods whose behaviour is being investigated empirically.

3 The methods to be assessed

The methods assessed in this paper are built around multilayer perceptron networks. Methods in the `mlp-mc` family use Bayesian learning implemented using Markov chain Monte Carlo techniques. Methods in the `mlp-bgd` family find an ensemble of networks trained using batch gradient descent with early stopping.

I will start by describing some characteristics common to all these methods, after which I describe the two basic methods, which do not try to adapt to varying relevance of the inputs. I then describe the Bayesian Automatic Relevance Determination (ARD) method, and the variation on early stopping inspired by it. A simpler variation on early stopping which might also help when inputs vary in relevance is also described.

3.1 Common characteristics of the methods

All the methods defined in this paper are based on multilayer perceptron ("backprop") networks with one layer of 20 hidden units, with tanh activation

		Training set size				
		64	128	256	512	1024
`mlp-bgd-*`	8 inputs	1.6	3.3	7	14	27
	32 inputs	3.2	6.2	13	26	52
`mlp-mc-*`	8 inputs	2.4	3.5	9	16	31
	32 inputs	6.2	9.4	16	31	64

Fig. 1. Computation time in minutes, on a 194 MHz MIPS R10000 processor.

function. This number of hidden units was chosen in the hopes that it is large enough for most tasks with up to 1024 training cases — ie, that little would be gained by using more hidden units with that amount of data. Both the Bayesian and the early stopping methods are supposed to avoid overfitting, so the number of hidden units is not reduced for smaller training sets. The hidden units have connections to all inputs, and also have a bias input. The single output (whose activation function is the identity) is connected to all the hidden units, and again has a bias input. One of the Bayesian Monte Carlo methods (`mlp-mc-2`) had direct connections from the inputs to the output as well.

Bayesian Monte Carlo and early stopping ensemble methods have previously been used in DELVE assessments by Rasmussen (1996). Both these approaches can be automated relatively simply. In contrast, many commonly-used approaches to network training require that human judgements be made regarding how to set weight decay parameters, exactly how many hidden units to use, or when to stop one training run and start another. Often, it is not clear on what basis these decisions are to be made. Assessments of such methods will only be possible if and when they have been properly defined.

Following Rasmussen (1996), the methods used in this paper are defined such that they take a specified amount of computation time. However, I have defined time in terms of the number of network forward and backward passes, rather than actual computation time on a particular computer. In particular, all the methods (except those in Section 5.3) were allowed the equivalent of 60,000 passes over the complete training set. With this definition, two methods designed to take the same amount of time may actually differ somewhat in time taken, due to differing overheads, and this relationship may vary somewhat depending on the computer used. The advantage of this approach is that the results of the methods will be the same on any computer, aside from possible differences due to random number generators and floating-point roundoff error.

The actual times taken by the methods are shown in Figure 1, for different sizes of training set, and different numbers of inputs. Time varied little for different methods within the same family. Rasmussen's `mlp-ese-1` procedure used somewhat less time, and his `mlp-mc-1` procedure used considerably more time. (Due to differences in machines used and the time scaling employed, these time differences cannot be summarized by a simple factor.)

All the methods in this paper are implemented using the software for flexible Bayesian modeling available from my web page (specifically, the release of 1997-07-22). The exact descriptions of the methods are in the DELVE archive, along with the results of the experiments.

3.2 Bayesian learning using Markov chain Monte Carlo

In a Monte Carlo implementation of Bayesian neural network learning (Neal 1996), we do not try to find a single "optimal" set of network weights, but instead try to sample from the posterior distribution for network weights, based on a prior distribution and on the likelihood given the training cases. We then make predictions for test cases by averaging the outputs produced by the networks in this sample from the posterior distribution.

We can try to obtain a sample of networks from the posterior distribution by simulating a Markov chain that has the posterior as its equilibrium distribution. For the methods in the `mlp-mc` family, this Markov chain is built using two types of updates. One type changes the weights and biases in the network using a dynamical technique that resembles gradient descent with "momentum". The other type of update changes the hyperparameters that control the distribution of the weights and biases, along with the standard deviation for the "noise" — the difference between the network output and the actual target. For more details on these updates, see (Neal 1996) and the software documentation.

The Markov chain converges to the desired distribution asymptotically, but will not sample from the exactly correct distribution in a run of finite length. In an application without strict time constraints, one should examine the progress of the chain, in order to decide how long to let it run so as to produce a nearly-correct result. The initial part of the run, before equilibrium appears to have been (nearly) reached, should be discarded. Predictions can then be made by averaging the outputs of networks from the remaining part of the run.

In a time-critical application, one might not have the option of running the chain for longer. In this case one can do nothing except use the latter part of the chain for predictions, and hope for the best. There is not much to be gained by judging exactly how much to discard of the beginning of such a fix-length run — just discarding the first third and using the last two-thirds for predictions seems to be generally suitable. This approach is convenient for automated assessments, since it involves no human judgement.

The `mlp-mc-1` method defined by Rasmussen (1996) uses this approach of running the Markov chain for a fixed amount of time, and discarding a fixed fraction. For the assessments done in this paper, I defined three related methods, `mlp-mc-2`, `mlp-mc-3`, and `mlp-mc-4`, which also use this approach, but which use somewhat different Markov chain updates, involving "partial gradients" (see Neal 1996, Section 3.5.1). These methods are also allowed much less time than `mlp-mc-1`.

The three new `mlp-mc` methods differ in their network architecture and in whether they use Automatic Relevance Determination. The `mlp-mc-4` method has no direct input-output connections, and does not use ARD. It does use a hierarchical prior that accounts for the differing roles of connections of different types (see MacKay 1992). The input-to-hidden weights are placed in one group, whose standard deviation is a hyperparameter, the hidden unit biases form another group, controlled by another hyperparameter, and the hidden-to-output weights form a third group, controlled by a third hyperparameter. The noise standard deviation is also a variable hyperparameter. The output bias has a fixed standard deviation of one.

The hope is that the posterior distribution of these hyperparameters, which determine characteristics such as the smoothness of the relationship, will be concentrated on values that are suitable for the actual problem. If this works as intended, "overfitting" will not occur.

3.3 Ensembles of networks trained with early stopping

The technique of *early stopping* (Morgan and Bourland 1990) is a simple and computationally cheap way of trying to avoid overfitting when training a neural network. In the simplest form of this technique, the available training data is divided into two portions. One portion is often called the "training set", but to avoid confusion with the full training set, I will here call it the *estimation set*. The network is trained by some optimization method so as to minimize squared error (or some other error function) on the cases in this estimation set, starting from an initial state in which the network weights are small. The other portion of the training data is called the *validation set*. The error on cases in the validation set is used to select a network from among those found in the course of minimizing the error on the estimation set. Due to overfitting, the best network as judged by error on the validation set is often not the one with smallest error on the estimation set (which will be the last one found, if the optimization method is stable). The network with the smallest validation error is used to make predictions for test cases.

This procedure seems rather *ad hoc*. The results will clearly depend on the exact optimization procedure employed. Only part of the data is used for estimating the weights, which seems wasteful. It's hard to see how the method can be justified within any general framework for learning. On the other hand, the procedure requires only a single training run, and hence may be feasible when other methods are not. Early stopping is also relatively easy to automate.

Rasmussen (1996) has defined an early stopping method (`mlp-ese-1`) that uses an *ensemble* of several networks. The ensemble is found by training networks with early stopping using several random splits of the training data into estimation and validation sets. Predictions for test cases are then found by averaging the outputs of all the networks in the ensemble. Training an ensemble obviously requires more time than training a single network, but it has the advantage that all the data is likely to be used for estimating the weights, in at least some of the networks. Predictions using an ensemble are in fact guaranteed to be better than predictions from a randomly selected network in the ensemble (Perrone 1994), though there is no guarantee that using all the available time to continue training a single network would not have been better.

I have defined a similar but somewhat simpler method, `mlp-bgd-1`. This method randomly partitions the training data into four equal portions and then trains an ensemble of four networks by early stopping. Each training run uses three of the four portions as the estimation set and the remaining portion as the validation set. In contrast, `mlp-ese-1` uses an ensemble of variable size, with the data being split randomly for each member of the ensemble. The `mlp-bgd-1` method minimizes the error on the estimation set using 20,000 epochs of simple batch gradient descent. It is therefore not an "early stopping" method in the literal sense, since it continues training for this number of iterations regardless of what is happening to the error on the validation set. In contrast, `mlp-ese-1` uses a variable number of iterations of the conjugate gradient method, though it too generally continues training for a while past the point where validation error starts increasing, in case it should go down again later.

In detail, each batch gradient descent iteration of `mlp-bgd-1` changes the weights in following way (with biases being handled similarly):

$$w'_{ij} \;=\; w_{ij} \;-\; \frac{\epsilon}{N+1}\sum_{c=1}^{N}\frac{\partial E_c}{\partial w_{ij}} \tag{4}$$

Here, w_{ij} is the weight on the connection from unit i to unit j, N is the number of cases in the estimation set, and E_c is half the squared error on case c. The stepsize, ϵ, was set to 0.1, which was adequate for all the tests done, but if instability is observed, it should be reduced as necessary. (Note that the inputs and targets are normalized, so there is no general need to change ϵ to match changes in the scaling of the inputs or targets.)

Networks are saved after 63 of the 20,000 batch gradient descent iterations. These 63 iterations are spaced approximately logarithmically. Once training is finished, the best of the 63 saved networks is selected, as judged by squared error on the validation set.

Unlike `mlp-ese-1`, the networks trained by `mlp-bgd-1` have no direct input-to-output connections. In tests on toy problems, including such connections changed the behaviour of early stopping substantially, and apparently for the worse.

3.4 Bayesian Automatic Relevance Determination (ARD)

The Automatic Relevance Determination (ARD) method (MacKay 1994; Neal 1996) uses an elaboration of the hierarchical prior used in `mlp-mc-4` (see Section 3.2). Rather than placing the weights on all input-to-hidden connections in one group, controlled by a single hyperparameter, input-to-hidden weights are grouped according to which input they are associated with, and a separate hyperparameter is used for each such group. The hyperparameter for an input determines the standard deviation of the weights on connections from that input, and represents the relevance of that input to the task of predicting the target. We hope that the relevance hyperparameters for the less relevant inputs will take on small values, preventing these inputs from having an undue effect on the predictions.

I have defined two ARD methods, `mlp-mc-2` and `mlp-mc-3`. The `mlp-mc-1` method defined by Rasmussen (1996) also uses ARD. The only difference between `mlp-mc-2` and `mlp-mc-3` is that `mlp-mc-2` has direct input-output connections (as does `mlp-mc-1`), whereas `mlp-mc-3` does not (like `mlp-mc-4`). When present, direct input-output connections are associated with relevance hyperparameters that are separate from those associated with input-to-hidden connections. I found in preliminary tests that the presence of direct input-output connections sometimes slowed the convergence of the Markov chain. Which model is better with unlimited computation time should depend on whether inputs that are relevant to the linear component of the relationship of inputs to target are generally also relevant to the non-linear component of this relationship.

ARD can be understood and justified in high-level terms, as the correct Bayesian procedure when one has prior information that some inputs are likely to be more relevant than others. However, one can also look at the mechanics of how a Bayesian procedure using ARD operates, as a way of gaining insights that might be applied in other contexts. ARD can be implemented either by Monte Carlo methods (Neal 1996), or by using Gaussian approximations (MacKay 1992, 1994). Although these implementations differ substantially, they can both be viewed as operating in roughly the following fashion, if we assume that both the weights and the relevance hyperparameters for all inputs start out small:

1) The data forces some input-output weights to take on larger values, in order to better predict the targets, overcoming the tendency of the small relevance hyperparameters to keep these weights small.

2) The relevance hyperparameters associated with inputs for which some input-output weights are now larger also become larger, since they must reflect the distribution of the weights they control.

3) Other weights from the inputs with larger relevance hyperparameters are now allowed to get bigger.

This is a positive feedback mechanism, in which some large weights out of an input encourage all weights from that input to be larger, including weights that were not initially forced to be large by the data. This positive feedback is the mechanistic expression of a prior belief that if an input is relevant in one way (as captured by one weight from it), it is likely to be relevant in other ways as well (as captured by other weights from it). We can try to adapt ARD to other contexts by building similar positive feedback mechanisms.

3.5 An early stopping method inspired by ARD

The `mlp-bgd-2` method is an attempt to build an ARD-like feedback mechanism into the batch gradient descent procedure with early stopping. It operates in the same way as `mlp-bgd-1` (see Section 3.3), except that the stepsizes for the weights on input-output connections vary according to the total gradient on connections from that input, mimicking to some extent the mechanics of ARD.

In detail, a batch gradient descent iteration for `mlp-bgd-2` changes the weights as follows:

$$w'_{ij} \;=\; w_{ij} \;-\; \frac{\epsilon\, r_i}{N+1} \sum_{c=1}^{N} \frac{\partial E_c}{\partial w_{ij}} \tag{5}$$

This is the same as equation (4) except that the stepsize for weight w_{ij} is potentially altered by a factor r_i. For the biases and the hidden-to-output weights, r_i is always one. Stepsizes for input-to-output weights are adjusted by factors computed as follows:

$$r_i \;=\; G_i^4 \,/\, \max_{i'} G_{i'}^4 \tag{6}$$

Here, G_i is the magnitude of the error gradient with respect to weights out of input i:

$$G_i^2 \;=\; \sum_j \left(\frac{\partial E_c}{\partial w_{ij}} \right)^2 \tag{7}$$

The effect of this modification is that the weights out of the input that is most important (at the present stage of learning) are updated with the same stepsize as in `mlp-bgd-1`, but weights out of inputs that are less important are changed by smaller amounts. We hope that this will allow the relationship of the target to the more relevant inputs to be discovered before much overfitting to the less relevant inputs has occurred. The details of this method, such as the use of the fourth power of the gradient magnitude, were arrived at empirically, based on tests on toy datasets, and a small amount of testing on datasets in the `kin` family, which DELVE designates as "development" datasets, on which such preliminary tests are allowed.

3.6 A simpler variation on early stopping

The `mlp-bgd-2` method defined above continually changes the stepsizes for weights out of different inputs during the course of learning. One could instead try to find a simpler static scheme, in which the stepsizes are fixed at the beginning, based on simple statistics that can be computed from the data. One would certainly want to know if such a scheme does as well as or better than the more elaborate scheme inspired by ARD.

The `mlp-bgd-3` method uses such a static scheme. It is the same as `mlp-bgd-1` except that the stepsize for updates of weights out of input i is adjusted by the following factor:

$$r_i \;=\; \max\left[10^{-4},\; C_i^4 \,/\, \max_{i'} C_{i'}^4\right] \tag{8}$$

C_i is computed from the inputs, $x_i^{(c)}$, and targets, $t^{(c)}$, in the cases in the estimation set, as follows:

$$C_i \;=\; \sum_{c=1}^{N} x_i^{(c)} t^{(c)} \tag{9}$$

Since the inputs and targets are normalized to have medians of zero and average absolute deviations from the median of one, C_i will be approximately N times the correlation of input i with the target. When the weights are all zero, C_i is also the error gradient with respect to unit i of a linear regression model for the target. At the beginning of training, when all the weights are small, it will thus be analogous to G_i of Section 3.5.

The actual implementation of this method uses the same stepsize for all weights, but scales input i (after the usual normalization) by the factor $\sqrt{r_i}$. This has the same effect as using a stepsize proportional to r_i, since the weights out of input i must now be larger by a factor of $1/\sqrt{r_i}$ to achieve the same effect as without scaling, while the gradient is smaller by the same factor.

4 The main experiments and their results

The previous section defined Bayesian Monte Carlo and early stopping ensemble methods, with and without techniques for adapting to varying relevance of inputs. This gives four basic combinations, but with further variations, a total of six new methods were defined. These are summarized in Figure 2, along with two related methods assessed by Rasmussen (1996). In this section, I describe how the performance of these methods was assessed in a systematic manner using DELVE.

	Early stopping ensemble methods
`mlp-ese-1`	Rasmussen's method: direct I-O connections, no ARD
`mlp-bgd-1`	New method: no direct I-O connections, no ARD
`mlp-bgd-2`	New method: no direct I-O connections, ARD
`mlp-bgd-3`	New method: no direct I-O connections, static ARD scheme
	Bayesian Monte Carlo methods
`mlp-mc-1`	Rasmussen's method: direct I-O connections, ARD, long time
`mlp-mc-2`	New method: direct I-O connections, ARD
`mlp-mc-3`	New method: no direct I-O connections, ARD
`mlp-mc-4`	New method: no direct I-O connections, no ARD

Fig. 2. Summary of the six new methods, along with Rasmussen's two methods.

4.1 Questions to be addressed

The primary motivation for the assessments described here is to answer the following questions:

1) Does ARD actually improve the performance of the Bayesian Monte Carlo method? If so, by how much?
2) Does the variation on early stopping inspired by ARD improve performance? If so, by how much?
3) Is the performance of early stopping with an ARD-like procedure better or worse than the Bayesian ARD method?

Experiments relevant to question (1) have previously been done by Vivarelli and Williams (1997). Their results suggest that ARD was beneficial, but the differences observed lacked statistical significance, and were for a single task.

While investigating the performance of the relevance determination methods, we will also inevitably obtain further information about whether the Bayesian Monte Carlo method is better than early stopping ensembles, a question previously addressed by Rasmussen (1996). One should note that the relative performance of these techniques may well depend on the amount of computation time they are allowed. The six new methods are defined to all take roughly the same amount of time, which is much less than that allowed for the Bayesian Monte Carlo method defined by Rasmussen (`mlp-mc-1`). Rasmussen found that `mlp-mc-1` was better overall than his early stopping ensemble method (`mlp-ese-1`), even when it was restricted to using the same amount of time. However, since the early stopping methods defined in this paper differ in important ways from `mlp-ese-1`, it is possible that the comparison with the Bayesian Monte Carlo methods might turn out differently.

4.2 Tasks on which the methods were tried

DELVE contains several *families* of datasets intended for use in systematic experiments. Data in each family is generated using a realistic simulation program. In the present families, a single prototask is defined for each dataset, in which a target attribute is to be predicted from the other input attributes. The target attribute was always real-valued for these experiments, producing a regression task. Datasets within a family are distinguished by the number of inputs, by the amount of noise in the relationship of the target to the inputs, and by the degree to which this relationship is non-linear. For each prototask, five tasks are defined, in which the number of cases in the training is 64, 128, 256, 512, or 1024.

These families allow assessments of how the performance of a method varies with the characteristics of the dataset. Running methods on all members of a family can take a long time, however. For the assessments of the relevance determination methods, I used only the datasets for which the relationship was non-linear, with a moderate amount of noise. However, I did run the methods on both the datasets with 8 inputs and those with 32 inputs, since this dimension of variation is clearly important in assessing methods for determining input relevance. These two members of a family are identified by the suffixes "`8nm`" and "`32nm`".

Three families of datasets were used in the experiments reported here:

`kin` This data comes from a simulation of the kinematics of a robot arm. The target attribute is the distance of the end of the arm from a target; the inputs are various joint positions, angles, etc. This family was created by Z. Ghahramani.

`pumadyn` This data comes from a simulation of the dynamics of the Puma 560 robot arm. The target attribute is the angular acceleration of one of the links; the inputs are various angular positions, velocities, torques, etc. This family was created by Z. Ghahramani.

`bank` This data comes from a simulation of customers being served by banks. The target attribute is the proportion of customers who go away because all the queues in the bank are full; the inputs are various parameters of the simulation, such as the maximum lengths of the queues, and the sizes of the population in different areas. This family was created by C. E. Rasmussen.

The `kin` and `pumadyn` families were also used by Rasmussen (1996) to assess `mlp-mc-1` and `mlp-ese-1`. The results of these assessments are in the DELVE archive, allowing comparison of these methods with the new methods.

4.3 Running the methods on the task instances

In the main series of experiments, the six methods were each run on instances of 30 tasks: 3 families, 2 datasets per family (with 8 and 32 inputs), and

5 tasks per prototask (with varying training set sizes). Performance on each task was assessed using 8 instances, each of which consisted of a training set and a set of test cases, except that only 4 instances were used for the tasks with 1024 training cases. The total computation time used in running one of the methods on all these task instances was approximately 50 hours.

Before a method is run, the DELVE software is used to produce data files for each instance, in accordance with the encoding and normalization options specified for the method. When the targets are normalized (as they were here), DELVE also translates the guesses the method makes back to the original format. DELVE conventions for directory hierarchies and file names make it easier to keep straight what is happening during this process.

These runs produce quite a bit of data. At a minimum, a guess for the target in each test case of each instance must be saved. Many methods produce more than one guess, intended for use when different loss functions are used (this is the case for the `mlp-mc` family, though not for `mlp-bgd` family). The losses that result from using these guesses are usually saved as well, so that they will not have to be recomputed whenever the method is compared with another. Some methods also produce files recording what happened during learning for each instance — eg, the `mlp-bgd` methods record which iteration was selected for use based on validation error. These final results can be submitted to the DELVE archive.

4.4 Presentation and interpretation of the results

Once a method has been run on the instances of a task, the DELVE software can be used to estimate the expected squared-error loss of the method on that task, as described in Section 2.3. An estimate of the standard error of this estimate is obtained as well.

The estimated expected loss is printed both in "raw" form, corresponding to the original scaling of the targets, and in "standardized" form. For squared-error loss, losses are standardized by dividing by the variance of the targets in the complete set of test cases used for all instances of the task. A method that just guesses the mean of the targets in the training set would therefore be expected to have a standardized expected loss of close to one. Note that standardization of losses is purely a convention for presenting results in a more meaningful form. It has nothing to do with the normalization of attributes that may (or may not) be part of the definition of a method.

Figures 3, 4, and 5 show the standardized squared-error losses that the methods achieved on tasks from the three families. The name of the dataset, followed by the name of the (only) prototask, are shown at the top of each plot. Each plot shows results for the six new methods, along with `mlp-ese-1` and `mlp-ese-1` for the `kin` and `pumadyn` families. For the `bank` family, these older methods have not been run, so the results of a simple linear regression method, `lin-1`, are included for comparison instead.

These figures have a format due to Rasmussen (1996). The results for one task are contained in a vertical rectangle labeled at the top with the size of the training set for that task. Within this rectangle, each method has a column, with the left-to-right order of the methods being the same as the top-to-bottom order of the list of methods lower in the figure. The horizontal line in the column for a method marks the estimated expected loss of that method on the task. The vertical line extends up and down a distance corresponding to plus or minus the estimated standard error.

Below the rectangle for each task is an array of p-values for pairwise comparisons of the performance of two methods on this task, using the paired t-test described in Section 2.3. If the estimated expected loss of method A is significantly less than that for method B, a digit giving the significance level is placed in the column for method A and the row for method B. This digit is 100 times the p-value for this comparison, rounded up (eg, the digit 5 indicates a p-value between 0.04 and 0.05). A dot is shown in a row and column position if the observed difference between the methods is not significant (p-value greater than 0.09), or if the better method is the one associated with the row, rather than the column.

When examining these figures, the plots of the estimated expected losses for the various methods can be viewed first, in order to get a general impression of the results. One can then check whether those observed differences that seem of interest are statistically significant by looking at the p-value for the relevant comparison. If this is small, one can be reasonably confident that the apparent difference in performance of the two methods reflects at least the sign of the actual difference. On the other hand, if the p-value for a comparison is not small, the apparently better method might actually be worse.

Another way of viewing the results is to look across the row of p-values for a method. A digit will appear in this row if some other method was observed to be better, with the difference being significant at the indicated level. Similarly, looking down the column of p-values for a method will reveal which methods it has been shown to be better than.

5 Preliminary conclusions and further experiments

The results of the main experiments can be interpreted to give some preliminary answers to the questions asked in Section 4.1. As is typical, however, really understanding the results requires further investigation, including some further experiments.

5.1 Summary of results of the main experiments

I will start by summarizing what one can conclude from the results of the main experiments on the three families of tasks.

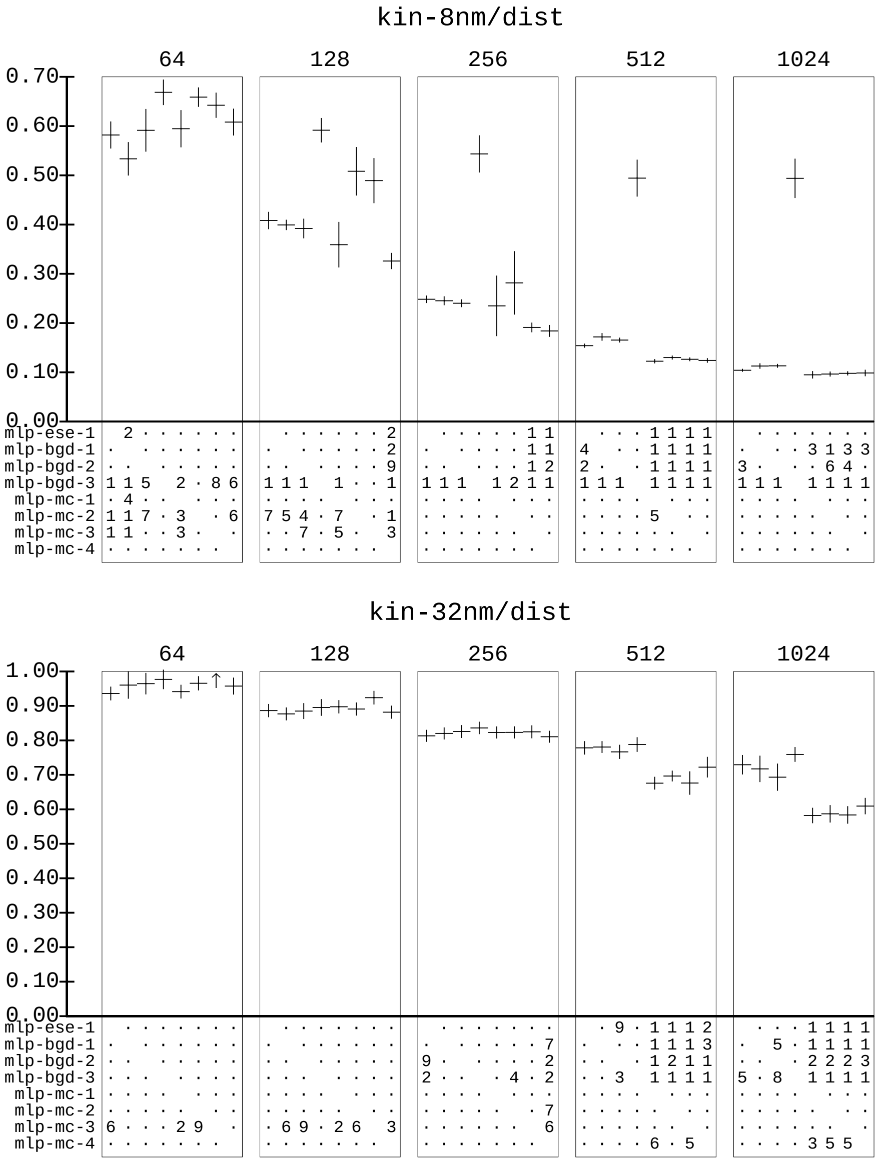

Fig. 3. Results on the `kin` family of tasks.

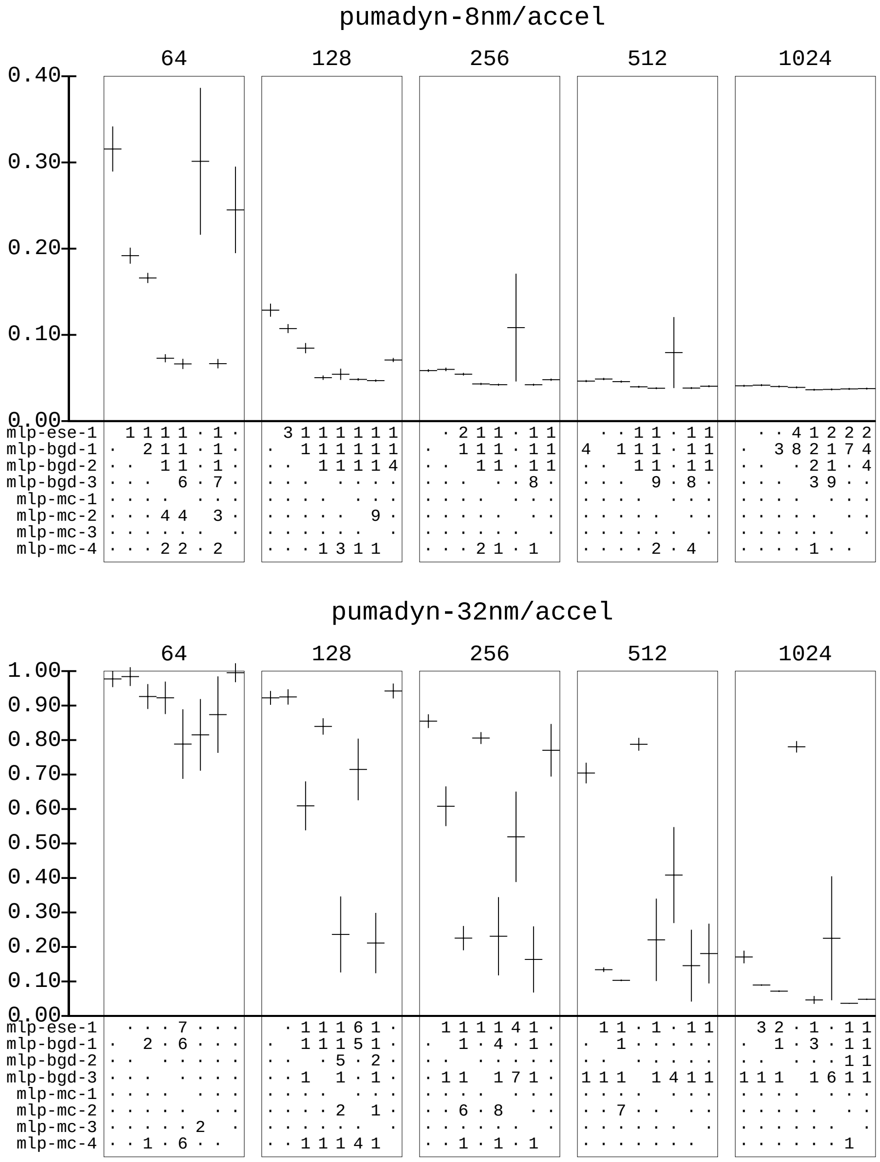

Fig. 4. Results on the pumadyn family of tasks.

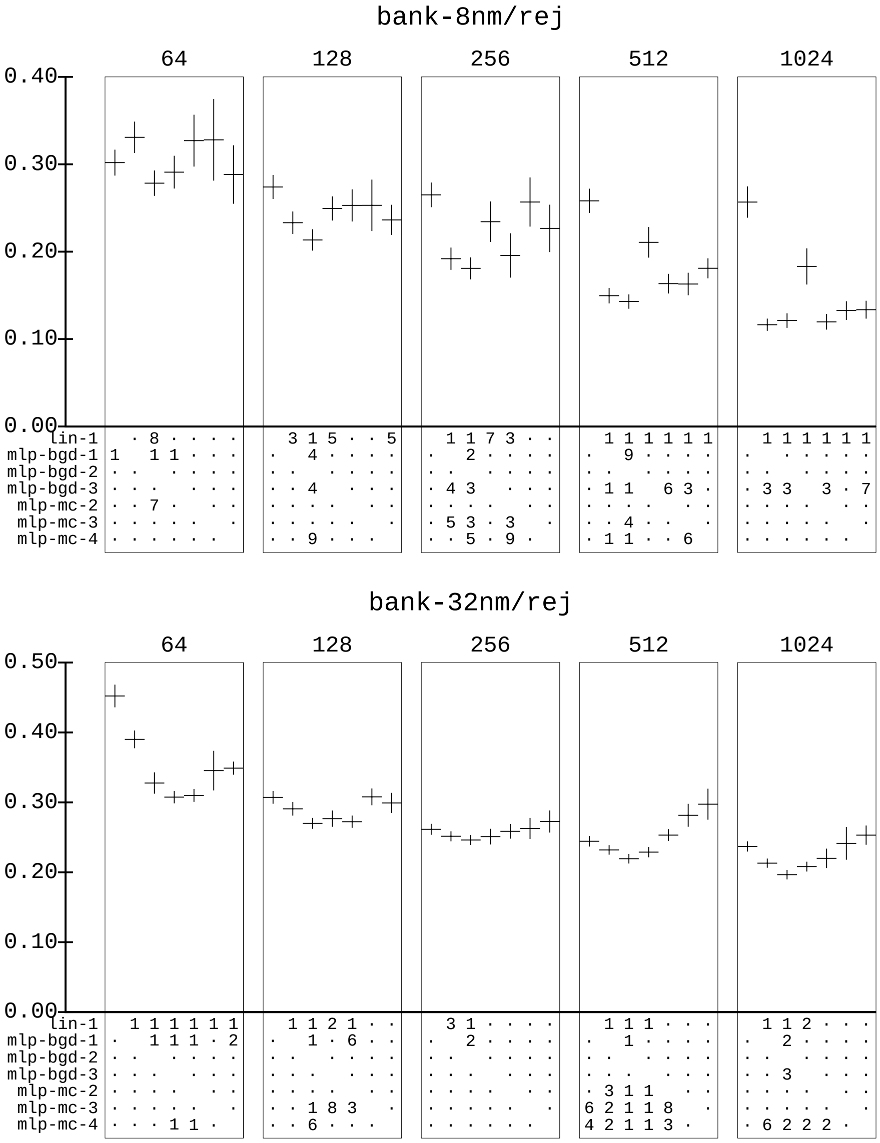

Fig. 5. Results on the bank family of tasks.

Results for the `kin` family (Figure 3) are generally favourable for the Bayesian Monte Carlo methods, particularly on the tasks with larger training sets. This is very clear for the task with 8-inputs and 512 training cases and the tasks with 32-inputs and 512 or 1024 training cases, on which all the Bayesian Monte Carlo methods are better than all the early stopping methods, with the p-values for these comparisons being 0.03 or lower. The magnitude of the advantage is not huge, but is large enough that it would often be practically important. For smaller training set sizes, the Bayesian Monte Carlo methods are not consistently better. For some tasks, it seems that the Markov chain used for sampling may not have come close to converging. This would explain the fairly large differences among the Bayesian Monte Carlo methods for the 8-input tasks with 128 or 256 training cases — in particular, since the models for `mlp-mc-1` and `mlp-mc-2` are quite similar, it is most plausible that the differences between them are due to the longer computation time allowed for `mlp-mc-1`.

Among the early stopping ensemble methods, the most striking result is that `mlp-bgd-3`, which uses the simple static approach to relevance determination, does very poorly on the 8-input tasks. It also appears worse than the other early stopping ensemble methods on the 32-input task, though the difference there is less dramatic. No large differences are seen among `mlp-ese-1`, `mlp-bgd-1`, and `mlp-bgd-2`. There is some indication that `mlp-bgd-2`, which uses the relevance determination technique, is a bit better than `mlp-bgd-1`, which does not, but is similar otherwise. However, when judged on a task-by-task basis, the observed advantage for `mlp-bgd-2` is statistically significant only for the 32-input task with 1024 training cases.

Differences among the Bayesian Monte Carlo methods are small, except for the two tasks where it seems that convergence was a problem. For the tasks with 32-inputs and 512 or 1024 training cases, a small but statistically significant disadvantage is seen for the method without ARD (`mlp-mc-4`). This non-ARD method performed better on some of the tasks with smaller training sets, but one might suspect that this is due to faster convergence of the Markov chain, rather than the ARD model being bad.

Some large differences in performance are seen on the `pumadyn` datasets (Figure 4). Looking first at the relative performance of the four early stopping ensemble methods, we see that `mlp-bgd-3` does very well on the 8-input tasks, particularly when the training set is small, but it does very badly on the 32-input tasks. The `mlp-ese-1` method also does much worse than `mlp-bgd-1` and `mlp-bgd-2` on some tasks, and it never does much better. There is also a consistent advantage of `mlp-bgd-2` (with relevance determination) over `mlp-bgd-1`. For the tasks with 32-inputs and 128 or 256 inputs, this advantage is quite large.

The results of the Bayesian Monte Carlo methods on the `pumadyn` tasks are confusing. Some large differences are seen among these methods, which are very likely the result of the Markov chain being far from convergence for some methods on some task instances. Large standard errors are also seen, which are probably the result of large variation in how well the Markov chain converged on different training sets (or with different random numbe seeds). Because of the convergence problems, which are especially pronounced for the 32-input tasks, detailed comparisons of the Bayesian Monte Carlo methods are probably not very fruitful. Useful comparisons with the early stopping ensemble methods are also difficult. When convergence seems to be a problem, the early stopping ensemble methods (especially `mlp-bgd-2`) may perform better, but the Bayesian Monte Carlo methods seem to do better when convergence is presumed to not be a problem (eg, with `mlp-mc-1`, `mlp-mc-3`, and `mlp-mc-4` on the 32-input task with 1024 training cases).

For the `bank` tasks (Figure 5), the `mlp-bgd-2` is consistently a bit better than `mlp-bgd-1`, and both are usually much better than both `mlp-ese-1` and `mlp-bgd-3`. The Bayesian Monte Carlo methods do surprisingly poorly here. None of them are ever significantly better than `mlp-bgd-2`, and they are sometimes substantially worse. On the 32-input task with 512 training cases, `mlp-mc-3` and `mlp-mc-4` are even worse than the simple linear regression model (`lin-1`).

This preliminary look at the results needs to be supplemented by a more detailed examination of the behaviour of the methods, so that the source of the differences seen can be identified.

5.2 More on the early stopping ensemble methods

To better understand the results for the early stopping methods, the first thing to look at is what iteration was selected for use on the basis of the error on the validation set.

For `mlp-bgd-1` and `mlp-bgd-2`, quite early iterations are often chosen. Indeed, for several task instances, the network after only one iteration of batch gradient descent was used! Only rarely is the network chosen the last one found, after the full 20,000 iterations. It therefore seems that these methods would not benefit much from being allowed more computation time. The fact that very early networks are sometimes chosen may explain why `mlp-ese-1` sometimes performs worse than `mlp-bgd-1`, as it uses the more sophisticated conjugate gradient optimization method, which may be "too good", moving past the optimal point in the first iteration. However, it is also possible that the presence of direct input-output connections in `mlp-ese-1` is the source of the performance difference.

The last iteration was chosen much more often by `mlp-bgd-3`. This might be expected whenever its static selection of stepsizes for the weights out of different inputs turns out to be inappropriate. Although the method might then benefit from being allowed more time, the objective of slowing down learning for the less relevant inputs will not have been achieved in any event.

Finally, we might wonder how much the use of an ensemble contributes to the performance of the early stopping methods. This is not directly rele-

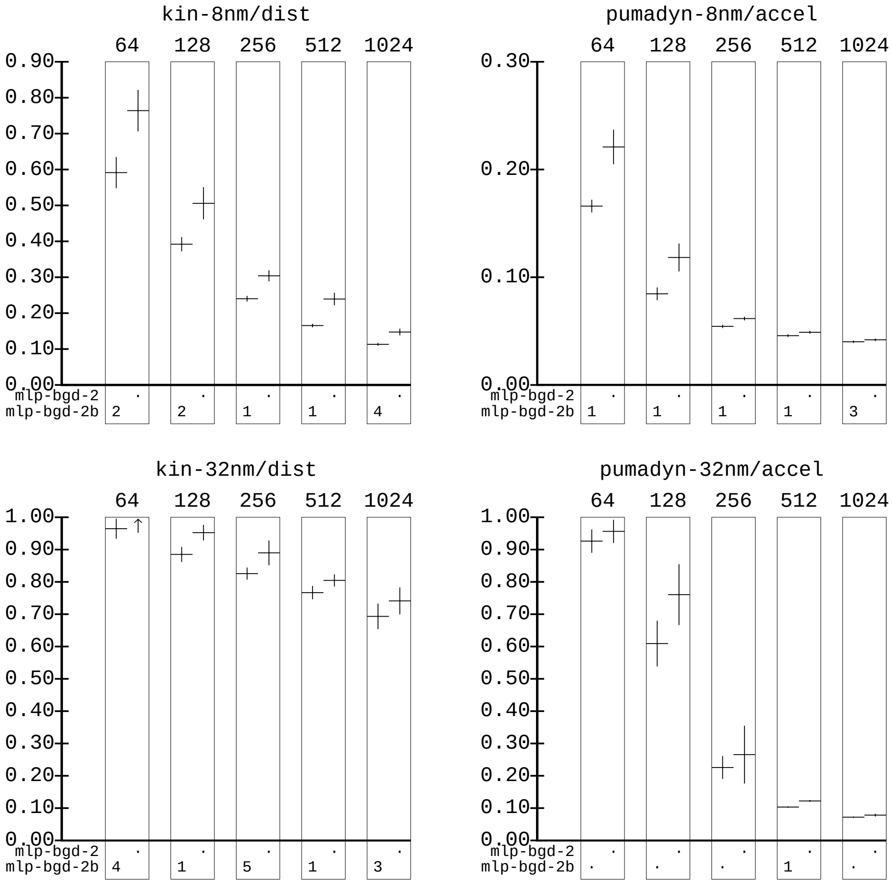

Fig. 6. Results using early stopping with and without an ensemble.

vant to the main objective of this investigation, but it is easily to determine, by just making predictions based on the first of the four networks already trained in the main experiments. Figure 6 shows the results of a comparison of the performance of `mlp-bgd-2` and a modification of it without the ensemble (`mlp-bgd-2b`). As can be seen, the ensemble often reduces the expected loss by a substantial amount, especially when the training set is small. Note that for this question, the p-values for the comparisons are irrelevant. Theory guarantees that using an ensemble is better than using just one arbitrarily selected network from the ensemble. The only question is whether the advantage of the ensemble is big enough to justify the computational cost of training four networks rather than one.

5.3 More on the Bayesian Monte Carlo methods

Two issues regarding the Bayesian Monte Carlo methods need to be resolved: Are the variable results seen on the `pumadyn` tasks in fact due to the methods not always having sufficient time to approach convergence? And what is the reason for the poor performance on the `bank` tasks? One possibility is that this poor performance also results from the methods not having enough time to converge.

To investigate these questions, I did further runs on the `pumadyn-32nm` and `bank-8nm` tasks, using variations on the Bayesian Monte Carlo methods called `mlp-mc-2b`, `mlp-mc-3b`, and `mlp-mc-4b`. These methods were identical to `mlp-mc-2`, `mlp-mc-3`, and `mlp-mc-4` except that they were allowed three times as much computation time. The results are shown in Figure 7.

More time did indeed improve the performance of the Bayesian Monte Carlo methods on some of the `pumadyn-32nm` tasks. With the extra time, `mlp-mc-3b` is very much better than the best early stopping ensemble method (`mlp-bgd-2`) for all training set sizes, except perhaps for training sets of size 64, for which the p-value of the comparison is 0.13. (See Figures 4 and 7.)

The results for `mlp-mc-2b` are still anomalous, as the model used by this method is very similar to that of `mlp-mc-1`, but its performance is sometimes much worse. On the task with 1024 training cases, its poorer performance is due to bad results on one of the four instances. When the run on this instance is allowed even more time, the Markov chain moves to a state with much different hyperparameter values, similar to those found with the other instances. It therefore seems that the performance of `mlp-mc-2b` is still limited by computation time.

With or without extra time, the use of ARD appears beneficial on the `pumadyn-32nm` tasks, as seen by comparing `mlp-mc-3` or `mlp-mc-3b` with `mlp-mc-4` or `mlp-mc-4b`. However, one might still wonder whether at least part of the very large benefit of ARD seen for the tasks with 128 and 256 training cases might be due to poorer convergence when ARD is not used, rather than to the ARD model being better. Examination of the hyperparameter values during individual runs of `mlp-mc-4b` suggests that poor convergence may still be a problem for this method.

The results of allowing the Bayesian Monte Carlo methods more time for the `bank-8nm` tasks are rather surprising — with more time, the performance is *worse*, especially on the tasks with larger training sets. Examining these runs in detail shows that the Markov chains eventually converge to states in which the noise level is quite small.

Recall that the target for the `bank` tasks is a proportion, which will be between 0 and 1. The distribution of this target for `bank-8nm` is quite skewed, with many values at or near zero, but a few that are much larger (up to about 0.5). It seems quite likely that the conditional distributions for the target given particular values for the inputs are also skewed, and furthermore have a variance that varies depending on the inputs (ie, the data is

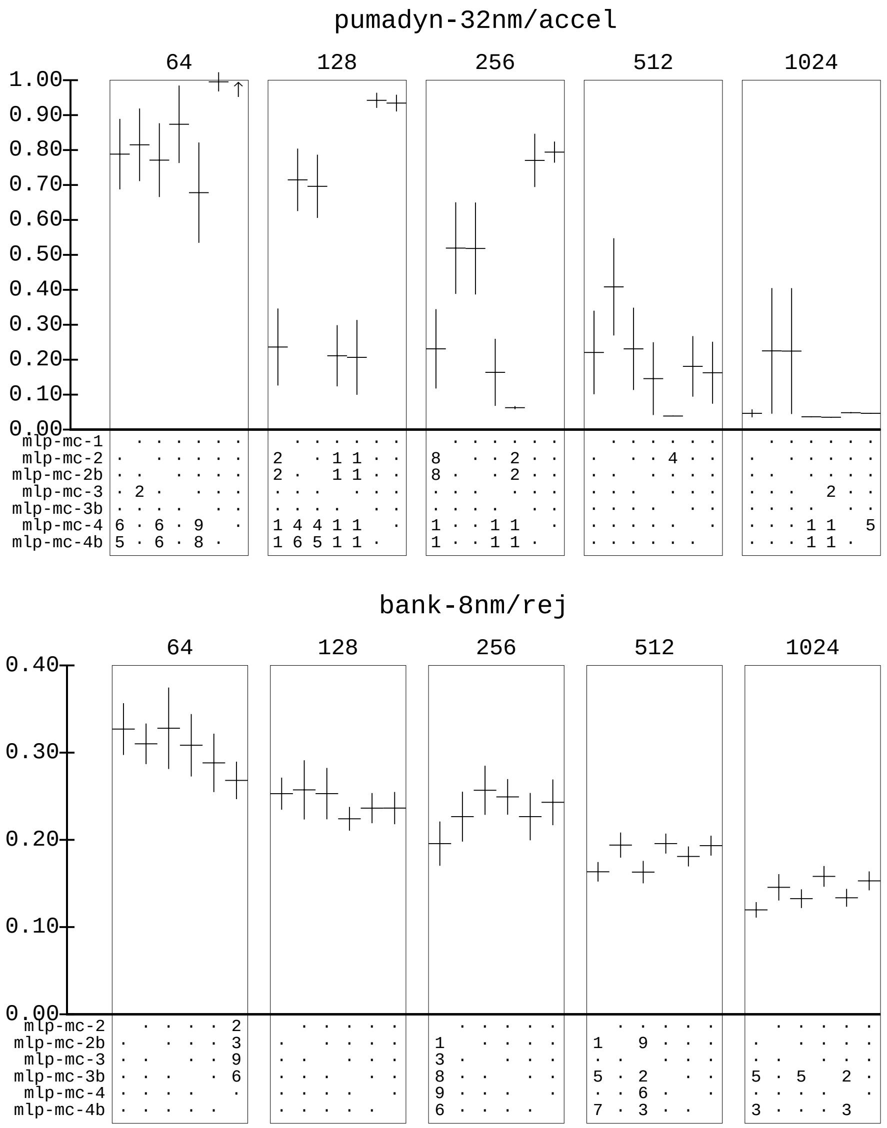

Fig. 7. Results when the Bayesian Monte Carlo methods are allowed more time.

"heteroskedastic"). In contrast, the models used by all the Bayesian Monte Carlo methods tested assume that the noise is Gaussian, and that its variance does not vary with the inputs. This mismatch between model and reality is likely the problem. The model thinks the noise variance is small because in some regions of the input space the targets can indeed be predicted quite accurately. This small noise variance then produces poor results in the parts of the input space where the real noise variance is large.

6 What was learned about the methods

In light of these experimental results, what can we say concerning the questions posed in Section 4.1?

The Bayesian Monte Carlo method using ARD (`mlp-mc-3`) generally performed a bit better than the corresponding method without ARD (`mlp-mc-4`). For a few tasks, the advantage of ARD was dramatic (`pumadyn-8nm` with 64 training cases, `pumadyn-32nm` with 128 and 256 training cases). More often, however, ARD performed only slightly better, and in some cases no significant difference was observed. A significant disadvantage for `mlp-mc-3` was seen for the `kin-8nm` and `kin-32nm` tasks with 128 training cases. Since this disadvantage was not shared with the other two ARD procedures (`mlp-mc-1` and `mlp-mc-2`), it was probably due to lack of convergence within the allowed time.

Although the benefit of ARD in terms of predictive performance is often modest, identifying the less relevant inputs may also help in interpreting the results of training. Interpretability is important in many applications, but is not addressed by DELVE assessments.

Another conclusion from these experiments is that obtaining reliable results using Bayesian Monte Carlo methods requires somewhat more computation time than was allowed for the methods assessed here. Even increasing the time allowed by a factor of three was not completely satisfactory. Nevertheless, even with the time restriction used here, the Bayesian Monte Carlo methods sometimes performed better than the early stopping methods.

The variation on early stopping inspired by ARD (`mlp-bgd-2`) appears to be very successful. It was sometimes substantially better than the corresponding procedure without relevance determination (`mlp-bgd-1`), and was never seen to be significantly worse. The simpler static method (`mlp-bgd-3`) occasionally performed well, but more often it performed very badly.

In contrast with the Bayesian Monte Carlo methods, the early stopping methods `mlp-bgd-1` and `mlp-bgd-2` appear to have reached most of their potential within the computation time allowed here. The advantage of using an ensemble of four networks in these methods was fairly substantial, and probably worthwhile in most applications, given that training time was not too large. These methods generally performed as well as or better than Rasmussen's (1996) `mlp-ese-1` method, even though it uses a more sophisticated optimization method and a more complex heuristic for building an ensemble. The conjugate gradient optimization method used by `mlp-ese-1` may actually be too good, sometimes moving past the optimum point in the first iteration.

An ugly aspect of the ARD-like modification of early stopping is that, apart from the inspiration from Bayesian ARD, it is rather arbitrary and *ad hoc*. This may be a general characteristic of early stopping methods. Certainly the success of the ARD-like modification is a further demonstration of how sensitive early stopping is to the exact method used for optimization.

The comparison of Bayesian ARD with early stopping using ARD-like relevance determination is complicated by the apparent lack of convergence of some runs of the Bayesian ARD methods and by the unusual behaviour of the Bayesian methods on the `bank` tasks. Reliable results using Bayesian Monte Carlo methods can be expected only if two conditions are satisfied:

1) There is sufficient time for the Markov chain sampler to converge to a good approximation to the posterior distribution.

2) The probabilistic model used is a fairly close approximation to reality.

Sometimes, a Bayesian Monte Carlo method may produce better results than other methods even though it has not come close to converging, but relying on this is not advisable if allowing more computation time is an option. Models seldom correspond exactly to the real situation, and we may hope that minor flaws will not have drastic effects, but there is no reason to expect good results from a Bayesian model with serious flaws in the specification of either the likelihood or the prior.

When the above two conditions are met, Bayesian Monte Carlo using ARD appears to perform better than early stopping ensembles, even with the ARD-like modification. However, the results on the `bank` tasks show that early stopping ensembles can perform better when the Bayesian model is not appropriate for the situation.

When manual intervention is possible, one should of course check for major departures from model assumptions, and modify the model if necessary. For the `bank` tasks, it might be appropriate to apply some non-linear transformation to the target in order to better match the model's assumption of Gaussian noise with constant variance. It would be interesting to compare the various methods on a variation of the `bank` tasks with such a transformed target, or in which the methods were provided with prior information that such a transformation might be useful (but still made predictions for the original target).

Finally, one should note that all the conclusions in this study are based on only three families of datasets, for all of which the input attributes represent diverse quantities. It is to be expected that some of these heterogeneous inputs will turn out to be more relevant than others. On the other hand, tasks certainly exist for which any sort of relevance determination method will be disadvantageous. For example, if the inputs are pixels of an image, and the task is known to be invariant with respect to translation of the image, all pixels must be equally relevant.

7 What was learned about DELVE

I hope the experiments reported here have served to illustrate how DELVE can be used to assess the performance of learning methods. Major assessments using DELVE are also described by Rasmussen (1996) and by Waterhouse (1997).

The goal of such assessments is not simply to obtain a table of numbers, but to gain insight into the methods assessed. In the StatLog project (Mitchie, *et al* 1994) many methods were assessed on many datasets, and an attempt was then made to obtain insight by exploratory methods such as multidimensional scaling and hierarchical clustering, without any attempt to assess statistical significance. In my opinion, it is more useful to address explicit questions, and in doing so, it is important to obtain valid indications of statistical significance, as otherwise one may be led to "explain" phenomena that were simply chance occurrences.

This approach has here produced an important result — that the ARD-like variation on early stopping is beneficial for tasks of the sort used in this assessment — as well as several other interesting results. This demonstrates that the present DELVE conventions for numbers of instances and numbers of test cases can produce statistically significant results when interesting differences are present. Experiments on smaller datasets may instead produce inconclusive results except when extreme differences exist (Rasmussen 1996).

The results reported here also illustrate the importance of running the methods on tasks with varying numbers of training cases. Many of the interesting phenomena seen might have been missed if a single size of training set had been arbitrarily chosen.

One of the strengths of the DELVE environment seen here is the ease with which comparisons can be made to the results of other methods held in the DELVE archive. Note that for paired significant tests to be possible, it is necessary for the losses on each test case to be archived for each method. When only the average loss for a method is known (eg, from a report in the literature), no significance testing is possible. When the average loss is accompanied by an estimated standard error, significance tests are possible, but these tests will be of low power, since the pairing information is ignored.

One weakness of DELVE at present is that we have only a few families of datasets, along with some isolated datasets. The number of families could certainly be increased, but how best to decide on the types of datasets to create is an open question. Interpretation of results can also be difficult if the nature of a dataset is not clear. The programs that generated the DELVE datasets are available from the DELVE archive, but some more accessible characterization of each dataset would be desirable.

A more fundamental problem is that running a method on all the tasks in many families can take a large amount of computation time. This practical difficulty is why the experiments in this paper used only the "**nm**" tasks (non-linear, medium noise). The DELVE software is itself a bit slow, due to its being written in the **tcl** language. Fixing this would be helpful, especially when the methods being assessed are fast. Assessing methods that are slower, such as neural networks, will inevitably require a substantial amount of computation time, but one may hope that the benefit obtained from this effort will increase as the results of more methods are placed in the DELVE archive.

Acknowledgements

The DELVE environment was developed at the University of Toronto by Carl Rasmussen, Geoff Hinton, Drew van Camp, Mike Revow, Zoubin Ghahramani, Rafal Kustra, Rob Tibshirani, and myself. I thank Zoubin Ghahramani, David MacKay, and Rob Tibshirani for helpful comments. This work was supported by the Institute for Robotics and Intelligence Systems and by the Natural Sciences and Engineering Research Council of Canada.

References

Bonnlander, B. V. (1996) *Nonparametric Selection of Input Variables for Connectionist Learning*, PhD thesis, University of Colorado, Department of Computer Science. `http://www.cs.colorado.edu/~brianb/`

Dietterich, T. G. (1998) "Approximate statistical tests for comparing supervised classification learning algorithms", to appear in *Neural Computation.*

Finoff, W., Hergert, F., and Zimmerman, H. G. (1993) "Improving model selection by nonconvergent methods", *Neural Networks*, vol. 6, pp. 771-783.

MacKay, D. J. C. (1992) "A practical Bayesian framework for backpropagation networks", *Neural Computation*, vol. 4, pp. 448-472.

MacKay, D. J. C. (1994) "Bayesian non-linear modeling for the energy prediction competition", *ASHRAE Transactions*, vol. 100, pt. 2, pp. 1053-1062.

Mitchie, D., Spiegelhalter, D. J., and Taylor, C.C., editors (1994) *Machine Learning, Neural and Statistical Classification*, New York: Ellis Horwood.

Morgan, N. and Borland, H. (1990) "Generalization and parameter estimation in feedforward nets: Some experiments", in D. Touretzky (editor), *Advances in Neural Information Processing Systems 2*, pp. 630-637. San Mateo, California: Morgan Kaufman.

Neal, R. M. (1996) *Bayesian Learning for Neural Networks*, Lecture Notes in Statistics No. 118, New York: Springer-Verlag.

Perrone, M. P. (1994) "General averaging results for convex optimization", in M. C. Mozer, *et al* (editors), *Proceedings of the 1993 Connectionist Models Summer School*, pp. 364-371. Hillsdale, New Jersey: Lawrence Erlbaum.

Rasmussen, C. E. (1996) *Evaluation of Gaussian Processes and other Methods for Non-Linear Regression*, PhD Thesis, University of Toronto, Department of Computer Science. `http://www.cs.utoronto.ca/~carl/`

Rasmussen, C. E., Neal, R. M., Hinton, G. E., van Camp, D., Revow, M., Ghahramani, Z., Kustra, R., and Tibshirani, R. (1996) *The DELVE Manual*, Version 1.1. `http://www.cs.utoronto.ca/~delve/`

Waterhouse, S. R. (1997) *Classification and Regression using Mixtures of Experts*, PhD Thesis, University of Cambridge, Department of Engineering. `http://www.ultimode.com/stevew/`

Vivarelli, F. and Williams, C. K. I. (1997) "Using Bayesian neural networks to classify segmented images", to appear in *Proceedings of the Fifth IEE International Conference on Artificial Neural Networks*.