Tuning diversity in bagged neural network ensembles

John G. Carney and Pádraig Cunningham

Department of Computer Science, University of Dublin, Trinity College, Ireland
E-mail: John.Carney@cs.tcd.ie

Abstract

In this paper we address the issue of how to optimize the generalization performance of bagged neural network ensembles. We investigate how diversity amongst networks in bagged ensembles can significantly influence ensemble generalization performance and propose a new early-stopping technique that effectively tunes this diversity so that overall ensemble generalization performance is optimized. Experiments performed on benchmark regression data-sets demonstrate the potential of the technique.

Keywords: Bagging, diversity, ensemble, generalization, early-stopping

1 Introduction

Recently, neural network ensemble techniques have gained widespread use amongst neural network practitioners. There are many different varieties, but the most popular include some elaboration of *bagging* [2], *boosting* [11] or *stacking* [34]. The basic idea of these techniques is to generate multiple versions of a predictor. When predictions from these versions are combined (averaged for example), smoother more stable predictions are generated. When applied to neural networks, these techniques can yield dramatic improvements in generalization performance (see e.g. [5, 7]). This is because neural networks are inherently unstable [2, 4] i.e. small changes in training set and/or parameter selection can produce large changes in performance. This idea of combining predictions from multiple versions has been around for quite a while – its origins in the neural network literature can be traced back to as early as 1965 [23]. It has also been used in other fields such as econometrics where it is called "forecast combining" [13]. However, it has not gained widepread use until recently, largely because it requires significant computational resources, especially when applied to learners such as neural networks.

In this paper we focus on bagging. We study the general work of Krogh and Vedelsby [19] on neural network ensembles and show how their key insight that diversity in an ensemble can significantly influence ensemble generalization performance can be used to better understand the underlying properties of bagging. Using this, we develop an early-stopping technique that effectively tunes diversity to optimize overall ensemble generalization perfomance. Techniques that currently address this issue focus primarily on varying the re-sampling rate of the training sets used for the ensemble e.g. [30]. We adapt this idea for bagged ensembles – given an initial level of diversity (generated as part of the bootstrap re-sampling process used in bagging) we attempt to fine-tune diversity by controling the fit of the networks. Our view is that over-fitting individual networks in an ensemble will generate more diversity and under-fitting them will generate less diversity. The results of our analysis highlight an interesting feature of bagged (back-propagation) ensembles – a controled level of over-fitting can consistently improve overall ensemble generalization performance. This result is consistent with work performed on other ensembles, e.g. Sollich and Krogh who demonstrated it for simple linear ensembles [30] and Husmeier who demonstrated it for Random Vector Functional Link (*RVFL*) ensembles [18].

An outline of the paper goes as follows. In section 2 we describe in general terms how bagging works and analytically show how it can improve the generalization performance of unstable predictors such as neural networks. In section 3 we discuss how diversity amongst networks in a bagged ensemble can significantly influence overall ensemble generalization. In section 4 we discuss the merits and disadvantages of different approaches to optimizing generalization performance and argue that early-stopping is particularly suitable for bagged ensembles. In section 5 we support this argument by describing a simple but effective technique that applies early-stopping to bagged ensembles. In section 6 we introduce our new technique that optimizes generalization performance by tuning diversity and in section 7 evaluate its performance using benchmark regression data. In section 8 we dicuss how our work relates to similar work on bagged ensembles and finally in section 9 we draw conclusions.

2 Bagging

Bagging, an abbreviation for "bootstrap aggregation", is widely accepted as one of the most popular neural network ensemble techniques. It uses the bootstrap [8], a very popular statistical re-sampling technique, to generate multiple training sets and networks for an ensemble. Although competing ensemble techniques such as those from the boosting family have been shown to out-perform bagging on some data-sets (see e.g. [20]), bagging has a number of important advantages when applied to real-world tasks. One of the most important is the ease with which confidence and prediction intervals can be computed [6, 16]. Another is the robustness and stability of the technique itself

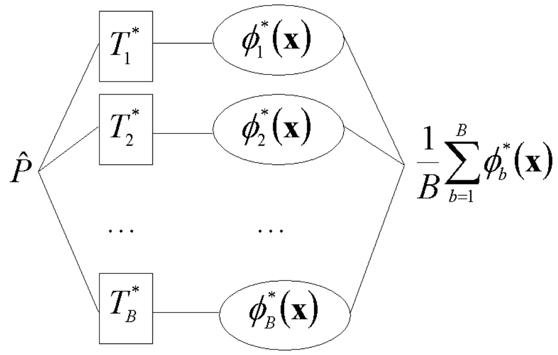

Fig. 1. A bagged neural network ensemble. Training sets are generated by sampling from the bootstrap empirical distribution $\hat{P}$. These training sets are used to train an ensemble of neural networks. The outputs of the networks in the ensemble are averaged to produce the bagged ensemble prediction.

– it can be shown that it will always perform at least as well as an individual predictor, as long as the predictor is unstable [2]. Also, in its simplest form, it is very straight-forward to implement. There are many works on bagging in the literature that study the technique in general terms i.e. without reference to any specific predictor e.g. [2, 3, 27, 32, 33]. Works that study bagging solely in the context of neural networks include [5, 17, 36].

To describe bagging (in the context of regression) in more concrete terms, let us now introduce some notation. Assume we are given a set of N (noisy regression) data pairs $T = \{(t_n, \mathbf{x}_n)\}_{n=1}^N$, described by the distribution P and generated according to

$$t = f(\mathbf{x}) + \epsilon(\mathbf{x}), \tag{1}$$

where t is the observed target value, $f(\mathbf{x})$ is the true regression and $\epsilon(\mathbf{x})$ is noise with zero mean. When we train a neural network on such data, our aim is for the network to approximate $f(\mathbf{x})$. Let us denote this neural network approximation as $\phi(\mathbf{x})$. Bagging aims to improve this individual network approximation by generating bootstrap re-samples of T and using these re-sampled training sets $\{T_b^*\}_{b=1}^B$ to generate multiple bootstrap versions. Each bootstrap re-sample T_b^* consists of N data pairs, sampled at random with replacement from T. When we sample in this way, we are said to sample from an empirical version of the distribution P [8]. Let us denote this empirical distribution as $\hat{P}$. The training sets $\{T_b^*\}_{b=1}^B$ give us a set of networks $\{\phi_b^*(\mathbf{x})\}_{b=1}^B$. Bagging aggregates these bootstrap versions by averaging to form a bagged prediction,

$$\phi_{bag}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} \phi_b^*(\mathbf{x}). \tag{2}$$

See figure 1 for a schematic outline of this.

The description of bagging given above does not quantify the extent to which bagging can improve the generalization performance of neural networks or under what circumstances. To study bagging in more depth and clearly illustrate its properties and limitations, we will now show how the general work of Krogh and Vedelsby [19] on neural network ensembles applies to bagged ensembles.

Using the notation introduced above and the terminology introduced in [19], let us define the *ambiguity* of a single member of a bagged ensemble on a prediction for t as

$$a_b(\mathbf{x}) = (\phi_b^*(\mathbf{x}) - \phi_{bag}(\mathbf{x}))^2, \tag{3}$$

and the *ensemble ambiguity* to be

$$a_{bag}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}(\phi_b^*(\mathbf{x}) - \phi_{bag}(\mathbf{x}))^2. \tag{4}$$

The ensemble ambiguity is a variance measure – it quantifies the disagreement or *diversity* amongst the networks in an ensemble on a prediction for t. Define the generalization error of an individual network on a single prediction for t as

$$e_b(\mathbf{x}) = (\phi_b^*(\mathbf{x}) - t)^2, \tag{5}$$

and on an ensemble prediction as

$$e_{bag}(\mathbf{x}) = (\phi_{bag}(\mathbf{x}) - t)^2. \tag{6}$$

Note that here we assume the inputs are independent of the training set i.e. they are test set inputs. We also define the average of the generalization errors of the individual networks across the ensemble as

$$\overline{e}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}(\phi_b^*(\mathbf{x}) - t)^2. \tag{7}$$

This important measure will be compared to the ensemble generalization error to provide valuable insights. It can be rewritten as

$$\overline{e}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}(\phi_b^*(\mathbf{x}) - \phi_{bag}(\mathbf{x}) + \phi_{bag}(\mathbf{x}) - t)^2 \tag{8}$$

and after a little manipulation as

$$\overline{e}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}(\phi_b^*(\mathbf{x}) - \phi_{bag}(\mathbf{x}))^2 + (\phi_{bag}(\mathbf{x}) - t)^2. \tag{9}$$

Using equations (4) and (6) above this gives us

$$\overline{e}(\mathbf{x}) = a_{bag}(\mathbf{x}) + e_{bag}(\mathbf{x}), \tag{10}$$

or

$$e_{bag}(\mathbf{x}) = \overline{e}(\mathbf{x}) - a_{bag}(\mathbf{x}). \tag{11}$$

For statistical rigour we now average this over the input distribution P. We begin by denoting the average generalization error over P of an individual network in an ensemble to be

$$E_b = \int d\mathbf{x} P(\mathbf{x}) e_b(\mathbf{x}). \tag{12}$$

Using this and equation (7), we express the average generalization over P across all the networks in an ensemble as $\overline{E}$. Similarly we denote the average ambiguity over P for an individual network in an ensemble as

$$A_b = \int d\mathbf{x} P(\mathbf{x}) a_b(\mathbf{x}), \tag{13}$$

and across all the networks in an ensemble as $\overline{A}$. Finally, using equation (6) we express the average bagged ensemble generalization error over P as

$$E = \int d\mathbf{x} P(\mathbf{x}) e_{bag}(\mathbf{x}). \tag{14}$$

Using equation (11) this gives us

$$E = \overline{E} - \overline{A}. \tag{15}$$

This expression is extremely valuable. It relates diversity (ensemble ambiguity) in a bagged ensemble to the generalization error of a bagged ensemble. As long as the average of the individual network generalization errors in an ensemble remains constant, increasing diversity will improve generalization performance. It also confirms intuition – an ensemble that consists of a thousand identical networks will not perform any better than an individual network. Note that, as described in [19], this expression is a general result and can be applied to most neural network ensembles.

What specific implications does it have for bagged ensembles? The training sets in a bagged ensemble are generated by sampling with replacement from the original training set T. The probability an individual training sample from T will not be part of a bootstrap re-sampled training set is $(1-1/N)^N \approx 0.368$, where N is the number of training samples in T. This means that only approximately 63% distinct training samples from T will be included in a bootstrap training set. This, of course, directly affects the average individual network generalization error $\overline{E}$ – given fewer distinct training samples it may not remain constant but instead may fall. However, the bootstrap sampling dramatically increases the diversity amongst the training sets in the ensemble and this combined with the inherent instability in neural networks will increase the ambiguity term $\overline{A}$. This is a key point – as long as the increase in ambiguity is larger than the decrease in average individual network generalization error it is worthwhile bagging a predictor. This is easy to achieve for neural networks because they are unstable and so bagging can consistently improve generalization performance. So, in summary, we can conclude that the more unstable a predictor is the more can be gained from bagging. Breiman demonstrates this empirically by bagging stable and unstable predictors and comparing relative improvements in generalization error [2]. However, he does not quantify this analytically by including an ambiguity term in his derivations.

3 Diversity and ensemble generalization performance

Now that we have established the importance of *generating* diversity in neural network ensembles, the question that remains is: how do we *tune* diversity so that ensemble generalization performance is optimized? In this section we discuss the merits and disadvantages of two possible methods for tuning diversity. Our conclusions are used as motivation for techniques developed in later sections that optimize ensemble generalization performance.

We will begin by discussing the effectiveness of training set re-sampling as a method for generating diversity. In the previous two sections we discussed how bagged ensembles use bootstrapping to generate diversity. How can we adjust this process to tune diversity? As outlined in [30], but in the context of simple linear ensembles, one could adjust the re-sampling process. Applying this to bagged ensembles, to generate more diversity fewer distinct training examples could be sampled, pushing down the standard bootstrap distinct sample rate of $\approx 63\%$. Similarly, to generate less diversity more distinct training examples could be sampled. This approach may seem promising at first, but in practice it has some major drawbacks. For example, the optimal re-sampling rate will vary depending on a number of factors including the level of noise in the training set, the size of the training set and the size of the ensemble. Therefore a global optimal re-sampling rate does not exist. Instead, it would have to estimated for each individual training scenario. Another more serious drawback is that such "pseudo-bootstrapping" does not have a large body of theoretical work to support it as does conventional bootstrapping. It would therefore be unwise (and possibly incorrect) to use this form of bootstrapping as a method for generating other statistics of interest such as confidence and prediction intervals, which can be conveniently estimated as part of the bootstrapping process in bagged ensemble training [6, 16]. Losing such valuable side effects of standard bootstrapping is a high price to pay. These drawbacks motivate the search for an alternative method for tuning ambiguity.

One possible approach highlights a fascinating feature of neural network ensembles. If we over-fit the networks in an ensemble we generate more diversity.

If we under-fit the networks we generate less diversity. Therefore, if we use standard bootstrapping to generate a basic level of diversity we can fine-tune the diversity by controling the fit of each network in the ensemble. For some ensembles, a controled level of over-fitting can significantly improve generalization performance. We will demonstrate this for non-linear bagged backpropagation ensembles in this paper. A key advantage of this approach is that ensemble diversity tuning and individual network parameter tuning can be unified and performed simultaneously using the same algorithm. In sections 6 and 7 we develop and evaluate such an algorithm.

4 Optimizing generalization performance using early-stopping

To optimize a neural network's generalization performance is to optimize its performance on test or out-of-sample data i.e. data not used during training. Much of modern neural network research is focused on developing techniques that optimize the performance of neural networks in this way. A large number of these techniques exist. They can however be loosely divided into two main categories – pruning techniques and regularization techniques.

Pruning techniques work by explicitly choosing an optimal number of hidden units and weights for a network. Examples include optimal brain damage pruning [35] and optimal brain surgeon pruning [14]. Regularization techniques work by constraining or penalizing the training of a network so that smoother, more general models are built. Examples of this approach include weight decay [15] and early-stopping, which cannot be fairly attributed to any single author.

It is generally accepted that, in practice, early-stopping based techniques are amongst the most popular for optimizing generalization performance. This is especially true for ensemble learning – see for example [5, 17]. We attribute this to a number of factors. Firstly, they are much more general than pruning techniques and can be easily adapted to a large variety of architectures and algorithms. Secondly, they do not require any *a priori* assumptions about the model (network) or training data – all pruning algorithms require prior assumptions, some of which are difficult to justify – see for example [28]. Thirdly, they are easy to understand and implement. Finally, and most importantly for this paper, they work very well with ensembles. For example, as we will show in this paper they allow diversity to be easily tuned, which as discussed in section 3, is a very important consideration for bagged ensemble training. Given these factors, we pursue the development and evaluation of an early-stopping based technique in this paper.

The basic idea of early-stopping is to terminate neural network training as soon as some estimate of generalization error begins to increase. This estimate of generalistion error is usually generated using cross-validation [31] or some related statistical re-sampling technique. The simplest and possibly most popular early-stopping technique for neural networks uses “hold-out validation” (often just called cross-validation) to estimate generalization error.

Hold-out validation consists of using $N - N_v$ of the available N training examples for training the network i.e. estimating the network weights $\hat{\mathbf{w}}_{N-N_v}$, and the remaining N_v examples for computing the generalization error estimate. This generalization error estimate can be expressed in terms of the neural network cost function as

$$G_{hov} = \frac{1}{N_v} \sum_{n=N-N_v+1}^{N} (t_n - \phi(\mathbf{x}_n; \hat{\mathbf{w}}_{N-N_v}))^2, \tag{16}$$

The estimate G_{hov} is measured at regular intervals during training (usually every epoch or training iteration). When it begins to increase, training is stopped.

This is early-stopping in its simplest form. However, it is widely accepted that the estimate G_{hov} can fluctuate during training if there is noise and/or non-linearity in the training data. Therefore, to simply terminate training as soon as it begins to increase can result in choosing a sub-optimal set of weights. One way to overcome this is to train a network to convergence and then choose the set of weights that correspond to the minimum value of G_{hov} as the optimal set. Other more elaborate techniques also exist. For example, in [25] a patience threshold, set by the user, is used to determine for how long training should continue after G_{hov} begins to increase during training. Although such techniques can improve overall training speed by circumventing the requirement that a network should be trained to convergence, the patience threshold is not easy to estimate given that it can vary significantly depending on the training data and network architecture used. The simpler option of training a network to convergence and then choosing the optimal set of weights will always be more robust and reliable. Given this, combined with the general view today that training speed is not as vital an issue as it used to be given modern computer processing power, motivates the use of the simpler approach in this paper.

To describe in more concrete terms the variety of early-stopping adopted in this paper, let us extend equation (16) above so that the generalization error estimates and the network weights are “indexed” by the number of epochs they correspond to,

$$G_{hov}(e) = \frac{1}{N_v} \sum_{n=N-N_v+1}^{N} (t_n - \phi(\mathbf{x}_n; \hat{\mathbf{w}}_{N-N_v}(e)))^2. \tag{17}$$

Here, $G_{hov}(e)$ is the hold-out validation estimate of generalization error for a network trained to epoch e. Values for $G_{hov}(e)$ are found for $e = 1, .., E$, where E is the "maximum" number of epochs and chosen to ensure convergence. A simple patience threshold idea, similar to that used above could be used here e.g. if training error does not change significantly for 100 epochs, terminate training. Once the network has been trained through the range of epochs specified, the user chooses the value for e that provides the best estimated generalization performance. The set of weights saved on disk corresponding to this best e is chosen as the optimal set.

An obvious drawback of early-stopping techniques that use hold-out validation is that only $N - N_v$ of the available training data can be used for estimating network weights. This damages the generalization performance of a network. If a small value for N_v is chosen to minimize this effect, then the estimate of generalization error will be compromised. One way to overcome this dilemma is to use a different variety of cross-validation. For example, "leave-one-out" cross-validation [31] does not require any training data to be sacrificed by the user. There are a number of different ways this technique (and generalizations of it such as k-fold cross-validation) can be applied to estimate generalization error for neural networks – see [12] for a general discussion.

However, in the context of this paper, a key point regarding the leave-one-out cross-validation family of techniques is that when applied to bagged ensembles they fail. This is because the bootstrapped training sets in a bagged ensemble are generated by sampling *with* replacement. This means that a single training example may occur several times in a training set. If a training example is removed from a training set for leave-one-out cross-validation, a replicate of it may still remain. This will produce an estimate of generalization error that is biased downwards, as the training example removed for cross-validation is not truly out-of-sample. However, sampling with replacement also has some important advantages in the context of bagged ensembles. As we will describe in the next section, it can be used as the basis for a simple but effective method for applying early-stopping to bagged ensembles.

5 Applying early-stopping to bagged ensembles

As described in section 2, training sets in a bagged ensemble are generated by sampling with replacement from the original training set T. The probability a training example from T will not be part of a bootstrap re-sampled training set is $(1 - 1/N)^N \approx 0.368$, where N is the number of training examples in T. This means that approximately 37% of the original training examples in T will not be used for training i.e. they will be out-of-sample. These can be used to estimate generalization error.

Let us describe this in more concrete terms by outlining a technique that uses these out-of-sample training examples for estimating generalization error in an early-stopping context. We present the technique in algorithmic form for clarity.

Step 1: *Set-up bootstrap training sets*
Generate B bootstrap re-samples of $T : T_1^*, T_2^*, ..., T_B^*$ where $T = \{(t_n, \mathbf{x}_n)\}_{n=1}^N$.

Step 2: *Compute generalization error estimates for each network in the ensemble*

for $b = 1$ *to* B

for $e = 1$ *to* E

Compute:

$$G_b(e) = \sum_{n=1}^{N} \gamma_n^b (t_n - \phi(\mathbf{x}_n; \hat{\mathbf{w}}_{T_b^*}(e)))^2 \qquad (18)$$

where $\gamma_n^b = 1$ is an indicator variable that indicates whether training sample n is out-of-sample for training set T_b^*; $\gamma_n^B = 1$ if it is and $\gamma_n^B = 0$ if it is not. A value for E that guarantees convergence is chosen as discussed in section 4 above. In section 7, we discuss what value for B should be chosen.

Step 3: *Find the best value for e for each network in the ensemble*

for $b = 1$ *to* B

Compute:

$$OPT_b(e) = \overset{argmin}{e} (G_b(e)) \qquad (19)$$

Here, for each network in the ensemble, the user finds the value for e that minimizes generalization error. The corresponding networks are chosen as the optimal set for the ensemble.

Although there is no record of this technique in the neural network literature, it is generally accepted that, in practice, it enjoys widespread use. It is fast, easy to implement and easy to understand. More importantly, it does not require data to be sacrificed for estimating generalization error. However, as we will discuss in much detail in the next section, it only provides "local" estimates of generalization error i.e. it does not consider how diversity amongst networks in an ensemble can significantly influence overall ensemble generalization error. This point was highlighted in section 3 – a controlled level of over-fitting can in fact improve overall ensemble generalization performance by generating more diversity. We support these assertions by experimentation in section 7 of this paper.

6 The NeuralBAG algorithm: Tuning diversity to optimize generalization performance

In this section we propose a new early-stopping algorithm we call NeuralBAG that optimizes generalization performance in bagged neural network ensembles. It operates in a similar fashion to the technique presented in section 5 above and is also presented in algorithmic form for clarity. However, it tunes diversity i.e. it will over-fit/under-fit networks in an ensemble if it estimates that this will improve overall ensemble generalization performance. An earlier version of NeuralBAG was first introduced in [5]. However this presentation was only an outline of the technique – no concrete justification or evaluation was included.

So how does NeuralBAG tune diversity? The key to solving this problem is to compute estimates of *ensemble generalization error* rather than *individual network generalization error*. This issue has not been directly addressed for neural networks – most techniques proposed have concentrated on optimizing the networks in an ensemble locally e.g. in [36] weight-decay is used to regularize the individual networks. Although some techniques do use estimates of ensemble generalization error e.g. [17] they are used for a different purpose and not with the explicit aim of tuning diversity. The details of the NeuralBAG algorithm go as follows.

Step 1: *Set-up bootstrap training sets*
Generate B bootstrap re-samples of $T : T_1^*, T_2^*, ..., T_B^*$ where $T = \{(t_n, \mathbf{x}_n)\}_{n=1}^N$.

Step 2: *Compute ensemble generalization error estimates for each training example*

for $n = 1$ *to* N

for $e = 1$ *to* E

Compute:

$$G_n(e) = \left(t_n - \frac{\sum_{b=1}^{B} \gamma_n^b (\phi(\mathbf{x}_n; \hat{\mathbf{w}}_{T_b^*}(e)))}{\sum_{b=1}^{B} \gamma_n^b} \right)^2 \tag{20}$$

Here we find the networks in the ensemble that were trained with bootstrap re-sampled training sets that do not contain training example n. We propagate training example n through these networks, average (bag) their outputs and calculate the squared error. This error is an estimate of the *ensemble generalisation performance* for a single test example. We repeat this procedure for each training example in T throughout the specified range of epochs. At the end of this step the user will have ensemble generalization error estimates for each of the N training examples in T for $e = 1, .., E$.

Step 3: *Aggregate the ensemble generalization error estimates specific to each network in the ensemble*

for $b = 1$ *to* B

for $e = 1$ to E

Compute:

$$G_b(e) = \frac{1}{N_b} \sum_{n=1}^{N} \gamma_n^b G_n(e) \tag{21}$$

where $N_b = \sum_{n=1}^{N} \gamma_n^b$ denotes the number of training examples for training set T_b^*. At the end of this step, for each network in the ensemble, the user will have aggregated ensemble generalization error estimates throughout the range $e = (1, .., E)$.

Step 4: *Find the best value for e for each network in the ensemble*

for $b = 1$ *to* B

Compute:

$$OPT_b(e) = \overset{argmin}{e} \; (G_b(e)) \tag{22}$$

Here, for each network in the ensemble, the user finds the value for e that minimizes generalization error. The corresponding networks are chosen as the optimal set for the ensemble.

As already mentioned, the very important difference of this algorithm is that diversity is tuned. This is achieved by computing estimates of ensemble generalization error as opposed to estimates of individual network generalization error. From these we can approximate the optimal number of epochs individual networks should be trained for so that overall ensemble generalization performance is optimized.

Up to now we have not mentioned the number of hidden units each network in a bagged ensemble should have. In [5], a different version of NeuralBAG is presented that includes another loop to estimate experimentally a (possibly) different number of hidden units for each network in the ensemble. Over a small range of hidden units this provides modest improvements in performance. However, computationally, a very high price is paid. We suggest that for small problems i.e. problems represented by small training sets and relatively few dimensions/inputs, such an approach should be adopted. However, for larger problems we suggest that the version of NeuralBAG presented in this paper should be used. The number of hidden units can be estimated prior to training using some simpler method (e.g. the technique described in [1]) and remain fixed for each network in the ensemble.

7 Evaluation

7.1 Data-sets

We use 4 data-sets in total to evaluate NeuralBAG. Each have numerical responses i.e. are regression data-sets. There are 3 real-world data-sets; *Boston Housing*, *Ozone* and *Hearta* and 1 synthetic data-set; *Freidman#1*. The following is an outline of and background to each:

Boston Housing. This data-set consists of 506 training vectors. There are 11 variables in the input component of each vector and 1 target output. The input variables consist mainly of socio-economic information and the output component is the corresponding median housing price. Each input-output training vector represents a census tract in the greater Boston area. The data-set has been used in many neural network and bagging studies e.g. [2, 3, 17]. It can be downloaded from the UCI Machine Learning Repository [22]. A more detailed description of the data can also be found there.

Ozone. This data-set consists of 366 training vectors. There are 9 input variables and 1 target output in each vector. The inputs consist of meteorlogical information such as humidity and temperature and the targets are the maximum daily ozone at a location in the Los Angeles basis. Following the lead of Breiman [2], we remove the 36 training vectors and 1 input predictor variable that have missing values. This leaves us with a data-set that consists of 330 training vectors and 8 variables. The data-set can be downloaded by ftp from the Department of Statistics, University of California at Berkeley web-site (ftp://ftp.stat.berkeley.edu/pub/users/breiman).

Hearta. This data-set consists of 900 training vectors. There are 35 input variables and 1 target output in each vector. The aim is to predict the number of heart vessels (1-4) that are reduced in size due to heart disease. The input variables consist of medical information such as blood pressure and electrocardiogram results, and personal information such as sex, age, smoking habits and pain descriptions. The data-set is a combination of four other data-sets and can be downloaded from the Proben1 benchmark data repository [26]. A version of it can also be found at the UCI Machine Learning Repository [22]. A full description of the data and the data-sets from which it is derived can be found in [25].

Freidman#1. This synthetic data-set was first introduced in [9]. Each training vector consists of 5 input variables and 1 target output generated according to

$$t = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \epsilon, \tag{23}$$

where ϵ is Gaussian noise, $N(0,1)$ and each of the $x_1, ..., x_5$ are uniformly distributed over the interval $[0,1]$. A data-set of 1400 sample vectors is used for the experiments. The inclusion of this data-set is potentially valuable for experimental purposes since the exact magnitude of the noise in the data is known.

7.2 Experimental set-up

Experiment 1: In this experiment we compare the generalization performance of ensembles trained using NeuralBAG to the generalization performance of ensembles trained using a benchmark technique that we adapt from [3]. We also compare NeuralBAG to the simpler early-stopping technique we proposed for training bagged ensembles which is described in section 5.

Let us first describe how the benchmark technique works. Breiman showed in [3] that a good measure of the quality of any generalization error estimate based on the training set can be found by comparing its accuracy to the estimate one would get using an independent validation set which is the same size as the training set. Adapting this idea to our problem, we train ensembles that have each individual network optimized *locally* using estimates of generalization error generated from independent validation sets that are the same size as the training sets.

Our reasoning behind this idea goes as follows. If we optimize the individual networks of an ensemble as described above, then we can expect the performance of such an ensemble to be close to the best achievable for an ensemble where only local estimates of generalization error based on the training set are used, and the importance of tuning diversity is not considered. Therefore, if we compare the generalization performance of such ensembles to the generalization performance of ensembles trained using NeuralBAG any improvement in performance yielded by NeuralBAG can be attributed (mostly) to tuning diversity.

Table 1
Experiment 1 results. Generalization (test set) mean-squared error performance of ensembles trained using NeuralBAG (NBAG) compared to the benchmark technique adapted from [3] (Bench) and the simpler technique described in section 5 (Simple). Note that Hearta and Freidman#1 results have been adjusted for clarity and ease of comparison. For example, all of the Hearta results have been multiplied by 10^3.

Dataset	NBAG	stdev	Bench	stdev	Simple	stdev
Boston	12.23	5.22	12.40	5.05	13.48	4.23
Ozone	18.37	3.59	18.58	3.40	19.14	3.21
Hearta ($*10^{-3}$)	50.58	6.20	51.96	5.72	56.43	4.75
Friedman#1 ($*10^{-1}$)	45.02	2.68	53.72	6.46	49.48	5.89

Table 2
Experiment 2 results. Average optimal number of epochs estimated for each ensemble and data-set in experiment 1.

Dataset	NBAG	Bench	Simple
Boston	7865	2944	1956
Ozone	4293	1213	1086
Hearta	113	55	18
Friedman#1	5116	1502	1338

The experiments were set up as follows. We generated 20 randomly sampled training, validation and test sets for each data-set and averaged results across 20 experiments. For the Boston Housing data the training, validation and test sets were of size 200, 200 and 106 respectively, for the Ozone data 125, 125 and 80 respectively, for the Hearta data 400, 400 and 100 respectively and for the Freidman#1 data 200, 200 and 1000 respectively. Each combination of training, validation and test set for each data-set are independent of each other. Note that the same training and test set combinations are used for ensembles trained using NeuralBAG, the benchmark technique and the simple early-stopping technique so as to get a direct comparison of performance. All ensembles trained had 30 networks (in experiment 4 below we justify this), a learning rate of 0.1 and momentum rate of 0.2. Ensembles trained with the Boston and Ozone data-sets had 5 hidden units, with the Hearta data-set 20 hidden units and with Freidman#1 6 hidden units. The results are presented in table 1 and discussed in section 7.3.

Experiment 2: In this experiment we compare the average number of epochs or training iterations that each of the 3 techniques (NeuralBAG, Benchmark and Simple) estimate as optimal. The aim is to determine whether NeuralBAG over-fits or under-fits the training sets (relative to the other two techniques) when it tunes diversity. Using estimates recorded from experiment 1, we average the optimal number of epochs estimated for each network in each ensemble. This gives us a single average number of epochs for each ensemble. We then average these figures across the 20 experiments performed for each data-set. Note that we do not include standard deviations with the results. This is because in the context of this experiment they are not meaningful or useful. The results are presented in table 2 and discussed in section 7.3.

Experiment 3: In this experiment we attempt to illustrate the importance and value of tuning diversity by plotting the ensemble test set (generalization) performance (an estimate for E) against the corresponding ensemble ambiguity (an estimate for $\overline{A}$) and the average individual network test set performance (an estimate for $\overline{E}$) as training evolves. We do this for a single training and test set pair of the Hearta data.

Note that we could not directly use NeuralBAG to generate such a plot. This is because it estimates a (possibly) different optimal number of epochs for each network in the ensemble. This makes a single plot that compares these measures at each epoch impossible. To overcome this and to generate a meaningful plot, we recorded the responses generated by each network in the ensemble at each epoch for the test set. Following training we computed the ensemble test set performance, the ensemble ambiguity and the average individual network test set performance at each epoch. The resulting plot is illustrated in figure 2 and discussed in section 7.3.

Experiment 4: In this experiment we attempt to investigate the effect the size of the ensemble has on NeuralBAG and how it tunes diversity. Using the 20 Hearta training and test set pairs, we plot the average test set performance across 20 experiments against the number of networks in the ensemble. We also plot the average number of epochs across 20 experiments against the number of networks in the ensemble. The plots are illustrated in figures 3 and 4 and discussed in section 7.3.

7.3 Discussion

In this section we discuss the results of the experiments described above. The results of experiment 1 demonstrate the potential of NeuralBAG as a technique for tuning diversity. On every data-set, it out-performs the benchmark technique. It does this despite having fewer validation samples ($\approx 63\%$ less than the benchmark technique per network in each ensemble). As expected it also out-performs the simple technique. The simple technique has approxi-

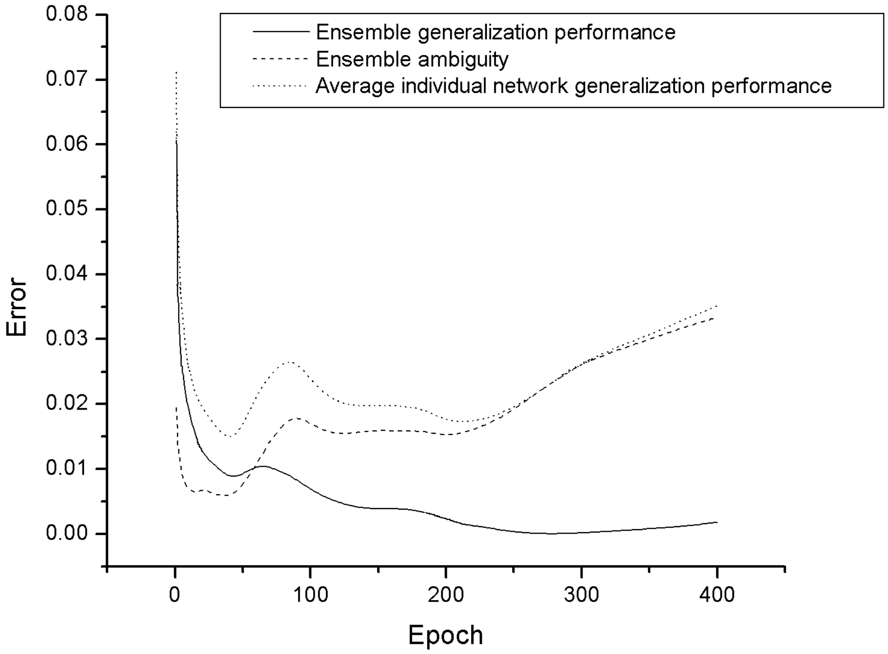

Fig. 2. Experiment 3 results. Here we compare the ensemble generalization performance to the ensemble ambiguity and the average individual network generalization performance as training evolves for a single training and test set pair of the Hearta data.

mately the same number of validation samples per network in each ensemble but does not tune diversity. There is only one anomolous result in this experiment – for the Freidman#1 data, the simple technique out-performs the benchmark technique. It is not obvious why this occured. However, a small number of the validation sets used for the benchmark experiments provided very poor estimates of generalization error which in turn led to ensembles with poor generalization performance. The relatively high standard deviation of the results for the benchmark technique reflect this. However, the fact that this occured only for the Freidman#1 data is puzzling. Further study that examines the cause and significance (if any) of this anomoly is required.

One of the key aims of these experiments is to verify that NeuralBAG improves performance by tuning diversity and not by any other means. The results of experiments 2 and 3 help to confirm this. Taking experiment 2 first, the average number of epochs estimated as optimal for NeuralBAG is consistently larger than that for the benchmark technique and the simple technique. This suggests that networks are over-fitting their training sets (relative to the other techniques) in order to tune (which translates to generating more) diversity. It is important to note however that the ensemble generalization error begins to increase after the optimal number of epochs – it does not converge, instead

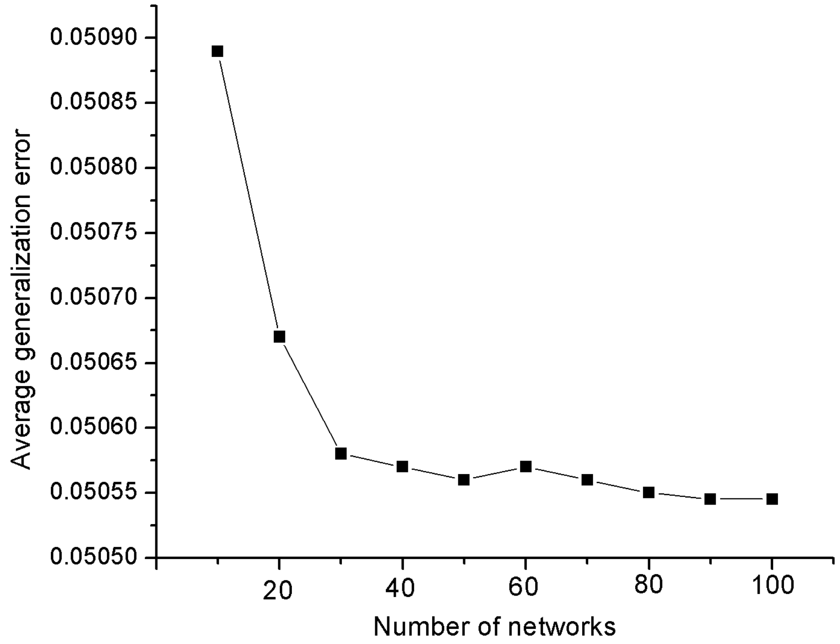

Fig. 3. Experiment 4 results, part 1. Here we plot the average generalization error across 20 experiments on the Hearta data as a function of the number of networks in ensembles trained with NeuralBAG.

the ensemble itself begins to over-fit the training set. This is important as it implies that it is not good enough to just train every network in a bagged ensemble to convergence – a *controled* level of over-fitting is required. The results of experiment 3 illustrate this effect very clearly. Here, the average individual network generalization error is minimized much earlier (epoch 42) than the ensemble generalization error (epoch 287). However, the ensemble generalization error begins to increase again following this minimum.

In experiment 4 we attempt to investigate what effect the size of an ensemble has on NeuralBAG. The results for the first part of the experiment are not a surprise – as the size of an ensemble increases, generalization error decreases. However, little is to be gained by using more than 30 networks and improvements seem to level out at around 100 networks. This is consistent with previous work on bagged ensembles e.g. [2, 17]. The results on the second part of the experiment are a little more interesting. Here, the average optimal number of epochs increases because the variance (and therefore also the diversity) of a larger ensemble will naturally be lower. To compensate for this effect NeuralBAG will over-fit the networks a little more to generate more diversity.

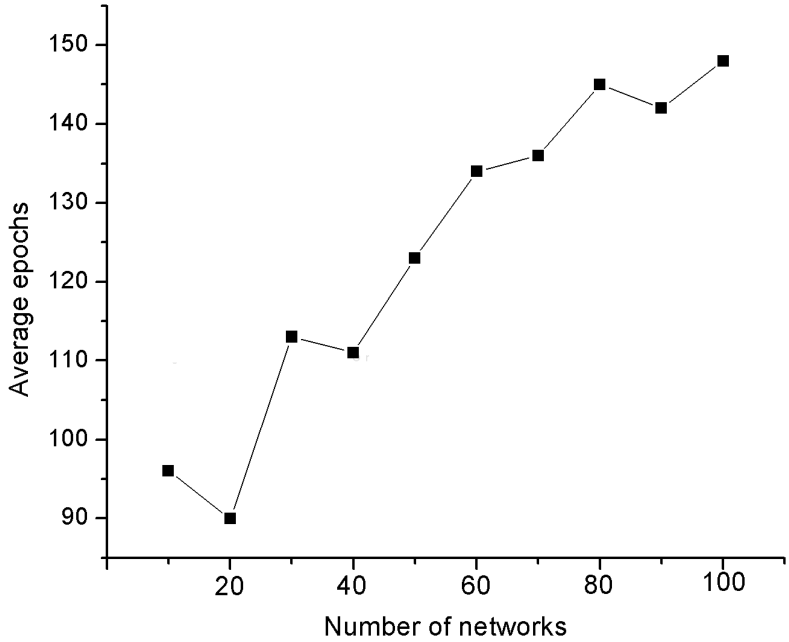

Fig. 4. Experiment 4 results, part 2. Here we plot the average estimated optimal number of epochs across 20 experiments on the Hearta data as a function of the number of networks in ensembles trained with NeuralBAG.

8 Relation to other work

In this section we discuss how the work presented in this paper relates to similar, previous studies on neural network ensembles. Much of the work presented in this paper was inspired by the the work of Sollich and Krogh [30]. They studied the characteristics and properties of simple linear ensembles and together with Krogh and Vedelsby [19] were the first to study in depth the importance and value of diversity in neural network ensembles. More specifically, in [30] they proposed that for large *linear* ensembles, under-regularized networks should be used and that a globally optimal ensemble generalization error can be reached by also varying the training set sizes of the individual networks. If only small ensembles can be trained due to computational restrictions, then they suggested that varying the re-sampling rate of the training sets is unnecessary (especially if the original training set is very noisy) and that a better approach is to use an optimized weighted average to combine the networks in the ensemble.

So, how does their work relate to ours? Although there are a number of obvious similarities in some of the conclusions, there are a number of important differences in the analysis and aims of the research. Firstly, our analysis is based on experiments performed using ensembles of *non-linear* back-propagation neural networks trained using both real and synthetic data. Secondly, we use an early-stopping technique to regularize the networks, they use weight-decay. Thirdly, we are only interested in bagged ensembles – we do not attempt or desire to vary the re-sampling rate of the bootstrapping process. We believe that this can potentially be cumbersome to optimize and may cause difficulties when confidence and prediction intervals are estimated. Finally, we don't attempt to use a weighted average – we find the robustness of a simple average very attractive. We prefer to simply increase the size of the ensemble to compensate for any problems caused by poorly performing individual networks. Given the wide availability of significant computer processing power today, we believe this approach is justified. Also, as described in [6], an added advantage of this approach is that large ensembles can produce more accurate confidence and prediction intervals.

Other related work includes Husmeier's analysis of *RVFL* ensembles [18]. He was the first to provide empirical evidence that over-fitting an ensemble of non-linear networks can be useful. However, his focus in [18] is solely on *RVFL* ensembles whose properties are significantly different to conventional back-propagation ensembles. For example, in [18] he showed how bagging does not usually improve the performance *RVFL* networks. He suggests that enough inherent instability exists in RVFL networks. Also, his focus is a little different from ours in that he doesn't expicitly attempt to tune diversity.

Finally, the work of Heskes [17] also has some similarites with ours. In [17], he develops and evaluates a technique for optimizing the performance of weighted back-propagation bagged ensembles. Although he does compute estimates of ensemble generalization error he uses them for a different purpose and does not explicitly attempt to tune diversity. However, as Sollich and Krogh [30] suggest, if small weighted ensembles are used, tuning diversity is not as valuable anyway.

9 Conclusion

In this paper we demonstrated the importance and value of tuning diversity in bagged neural network ensembles. Our approach of tuning diversity by varying the fit of the networks is a simple idea, but nevertheless works very well for bagged ensembles in practice. Also, it has one key advantage over other approaches – diversity tuning and network parameter (weights) tuning are unified – both can be performed simultaneously using the same algorithm. This would not be easy to achieve if a different method (e.g. varying the re-sampling rate of the training set) was used to tune diversity.

A very encouraging result of our analysis is the performance of our new NeuralBAG algorithm on the benchmark data-sets. Despite using small training sets, the generalization performance of ensembles trained using NeuralBAG is excellent when compared to other results on this data e.g. see [2, 17] for some results on the Boston Housing data, [2] on the Ozone and Freidman#1 data and [25] on the Hearta data.

Also, an important finding of our work that must be stressed is that although some over-fitting amongst networks in a bagged ensemble is usually required, it must be controled. If it is not, even large ensembles will over-fit and ensemble generalization performance will be compromised. Nevertheless however, it is fascinating to observe over-fitting improving the generalization performance of neural networks for a change.

References

[1] E. Baum and D. Haussler, What size net gives valid generalization? *Neural Computation* 1 (1988) 151-160.

[2] L. Breiman, Bagging predictors, *Machine Learning* 24 (1996) 123–140.

[3] L. Breiman, Out-of-bag estimation, Technical Report, Statistics Department, University of California at Berkeley, California, 1996.

[4] L. Breiman, Heuristics of instability in model selection, Technical Report, Statistics Department, University of California at Berkeley, California, 1994.

[5] J.G. Carney and P. Cunningham, The NeuralBAG algorithm: Optimizing generalization performance in bagged neural networks, in: M. Verleysen, ed., *Proceedings of the 7th European Symposium on Artificial Neural Networks* (D-Facto, Brussels, 1999) 35–40.

[6] J.G. Carney and P. Cunningham, Confidence and prediction intervals for neural network ensembles, in: *Proceedings of the International Joint Conference on Neural Networks 1999* (in press).

[7] H. Drucker, R. Schairpe and P. Simard, Improving performance in neural networks using a boosting algorithm, in: S.J. Hanson, J.D. Cowen and C.L. Giles, eds., *Advances in Neural Information Processing Systems 5* (Morgan Kaufman, 1993) 42–49.

[8] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap* (Chapman and Hall, London, 1993).

[9] J. Freidman, Multivariate adaptive regression splines (with discussion), *Annals of Statistics* 19 (1991) 1–141.

[10] Y. Freund and R. Schapire, Experiments with a new boosting algorithm, in: *Proceedings of the Thirteenth International Conference on Machine Learning* (Morgan Kaufman, 1996) 148–156.

[11] Y. Freund and R. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, in: *Proceedings of the Second European Conference on Computational Learning Theory* (Springer Verlag, 1995) 23–37.

[12] S. Geman, E. Bienenstock and Rene Doursat, Neural networks and the bias/variance dilemma, *Neural Computation* 4 (1992) 1–58.

[13] C.W. Granger, Combining forecasts – twenty years later, *Journal of Forecasting* 8 (1989) 167–173.

[14] B. Hassibi and D. Stork, Second order derivatives for network pruning: Optimal Brain Surgeon, in: S.J. Hanson, J.D. Cowen and C.L. Giles, eds., *Advances in Neural Information Processing Systems 5* (Morgan Kaufman, 1993) 164–171.

[15] J. Hertz, A. Krogh and R. Palmer, *Introduction to the Theory of Neural Computation* (Addison-Wesley, Redwood City, CA, 1991).

[16] T. Heskes, Practical confidence and prediction intervals, in: M. Mozer, M. Jordan and T. Petsche, eds., *Advances in Neural Information Processing Systems 9* (MIT Press, 1997) 176–182.

[17] T. Heskes, Balancing between bumping and bagging, in: M. Mozer, M. Jordan and T. Petsche, eds., *Advances in Neural Information Processing Systems 9* (MIT Press, 1997) 466–472.

[18] D. Husmeier, *Neural Networks for Conditional Probability Estimation* (Spinger-Verlag, London, 1999).

[19] A. Krogh and J. Vedelsby, Neural network ensembles, cross-validation and active learning, in: G. Tesauro, D. Touretzky and T. Lean, eds., *Advances in Neural Information Processing Systems 7* (MIT Press, 1995) 231–238.

[20] R. Maclin and D. Opitz, An empirical evaluation of bagging and boosting, in: *Proceedings of the Fourteenth National Conference on Artificial Intelligence* (Providence, Rhode Island, 1997).

[21] R. Maclin and J. Shavlik, Using competitive learning to initialize neural networks, in: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (1995) 524–530.

[22] P. Murphy and D. Aha, librarians, The UCI Repository of Machine Learning Databases and Domain Theories, *http://www.ics.uci.edu/ mlearn/MLRepository.html* (1995).

[23] N.J. Nilsson, *Learning Machines: Foundations of Trainable Pattern-Classifying Systems* (McGraw Hill, NY, 1965).

[24] D. Opitz and J. Shavlik, Generating accurate and diverse members of a neural network ensemble, in: D. Touretzky, M. Mozer and M. Hasselmo, eds., *Advances in Neural Information Processing Systems 8* (MIT Press, 1996) 535–541.

[25] L. Prechelt, Proben1: A set of benchmarks and benchmarking rules for neural network training algorithms, Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, Germany, 1994.

[26] L. Prechelt, librarian, Proben1: A set of benchmarks and benchmarking rules for neural network training algorithms, *ftp://ftp.ira.uka.de/pub/neuron* (1994).

[27] J. Rao and R. Tibshirani, The out-of-bootstrap method for model averaging and selection, Technical Report, Statistics Department, Stanford University, 1997.

[28] R. Reed, Pruning algorithms – a survey, *IEEE Transactions on Neural Networks* 4 (1993) 740-747.

[29] A.J. Sharkey, editor, *Connection Science: Special Issues on Combining Artificial Neural Networks* 8 (1996).

[30] P. Sollich and A. Krogh, Learning with ensembles: How over-fitting can be useful, in: D. Touretzky, M. Mozer and M. Hasselmo, eds., *Advances in Neural Information Processing Systems 8* (MIT Press, 1996) 190–196.

[31] M. Stone, Cross-validatory choice and assesment of statistical prediction, *Journal of the Royal Statistical Society* 36 (1974) 111–147.

[32] D. Wolpert and W. Macready, Combining stacking with bagging to improve a learning algorithm, Technical Report, Santa Fe Institute, Santa Fe, 1996.

[33] D. Wolpert and W. Macready, An efficient method to estimate bagging's generalization error, Technical Report, Santa Fe Institute, Santa Fe, 1996.

[34] D. H. Wolpert, Stacked generalization, *Neural Networks* 8 (1996) 1341–1390.

[35] Y. Le Cun, J. Denker and S. Solla, Optimal brain damage, in: D. Touretzky, ed., *Advances in Neural Information Processing Systems 2* (Morgan Kaufman, 1990) 598–605.

[36] J. Zhang, Developing robust non-linear models through bootstrap aggregated neural networks, *Neurocomputing* 25 (1999) 93-113.