# The NeuralBAG Algorithm: Optimizing Generalization Performance in Bagged Neural Networks

**John G. Carney and Pádraig Cunningham**
Department of Computer Science
Trinity College
University of Dublin
Ireland
*John.Carney,Padraig.Cunningham@cs.tcd.ie*

## Abstract

In this paper, we address the issue of how to optimize the generalization performance of bagged neural networks. We propose an algorithm we call "NeuralBAG" that estimates the set of weights and number of hidden units each network in a bagged ensemble should have so that the generalization performance of each is optimized. Experiments performed on noisy non-linear foreign exchange rate time-series data demonstrate that, on average, bagged networks trained using NeuralBAG out-perform individual networks trained using "cheating" and *k*-fold cross-validation by 31% and 40% respectively. We also show that, on average, bagged networks trained using NeuralBAG out-perform bagged networks trained using *k*-fold cross-validation by 17%.

## 1 Introduction

One fundamental weakness of neural networks is that they are unstable or exhibit high variance [3] (i.e.) small changes in training set and/or parameter selection can cause large changes in performance. This instability is magnified when real-world systems such as foreign exchange markets are modelled because, typically, only a limited amount of useful training data is available. One way to overcome this instability is to use bagging, a statistical re-sample and combine technique, first proposed by Breiman [1] and applied to neural networks by Heskes [6]. The basic idea of bagging is to use bootstrap re-sampling to generate multiple versions of a predictor which, when combined, should perform better than a single predictor built to solve the same problem. For bagging to work it is very important that the predictor is unstable – bagging can actually degrade the performance of stable predictors [1]. Breiman showed [3] that techniques such as neural networks, classification trees and regression trees are unstable, while techniques such as *k*-nearest neighbour methods are stable.

To symbolically formalise how bagging works, let us first assume we have a predictor (in our case a neural network) constructed using a training set $T$ that consists of $N$ training samples. Parameters (the weights) $\hat{\omega}$ are estimated using a learning algorithm (e.g. back-propagation) and the network generates an output $\hat{y}$ when presented with an input $x$. Let us denote this as:

$$\hat{y} = \phi(x; T; \hat{\omega}) \tag{1}$$

Now let us imagine an ideal, hypothetical situation where we have a large number of training sets $\{T_k\}_{k=1}^{K}$ (where $K \to \infty$), each consisting of $N$ independent observations from the same distribution as $T$ (let us call this distribution $P$). Given this ideal situation we could build an ensemble of networks trained using the $\{T_k\}_{k=1}^{K}$ and combine the estimates for $\hat{y}$ produced by each network to improve the quality of predictions. In reality however, we only have one training set $T$. Bagging proposes to remedy this situation by generating bootstrap re-samples of $T$ and using these re-sampled training sets $\{T_b\}_{b=1}^{B}$ (where $B$ is the number of bootstrap training sets) to mimic the training sets $\{T_k\}_{k=1}^{K}$. Each bootstrap re-sample $T_b$ consists of $N$ input-output training vectors, sampled at random from $T$ with replacement (see [4] for more on bootstrapping). The bootstrap distribution $\hat{P}$ is an empirical approximation to $P$. Using the training sets $\{T_b\}_{b=1}^{B}$ we now have a set of networks $\{\phi(x; T_b; \hat{\omega})\}_{b=1}^{B}$. Bagging proposes to aggregate these bootstrap versions by averaging to form a bagged ensemble of networks:

$$\hat{y}_{BAG} = \frac{1}{B} \sum_{b=1}^{B} \phi(x; T_b; \hat{\omega}) \tag{2}$$

In practice, however, bagging neural networks is not so straight-forward. There are a number of related questions that first need to be answered including; how do we choose the $\hat{\omega}$'s for each network in the ensemble so that the generalization performance of each network is optimized? How many hidden units should each network in the ensemble have?

In this paper we attempt to answer these questions. The NeuralBAG algorithm estimates the set of weights and number of hidden units each network in an ensemble should have so that generalization performance of each is optimized. It does not require any *a priori* information, does not require the modeller to sarcrifice any of his training samples and is statistically robust enough to work even when highly non-linear, noisy training data is used.

## 2 The NeuralBAG Algorithm

In this section we describe our proposed NeuralBAG algorithm. It estimates optimal values for $e$, the number of epochs (training iterations) and $h$ the number of hidden units for each network in a bagged ensemble. Although for clarity the serial version of the algorithm is presented here it is easily converted to parallel form[1].

The probability a training sample from (a large) training set $T$ 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 $T$. Following the lead of [2], [6] and [7] we use such "out-of-bag" samples for generalization error estimation.

**Step 1:** *Set-up bootstrap training sets.*

[1]All experiments described in this paper were performed on the Trinity College Dublin/Queens University Belfast IBM SP2 48 node supercomputer.

Generate $B$ bootstrap re-samples of $T$ : $T_1^*, T_2^*, ..., T_B^*$ where $T = \{(x_n, y_n)\}_{n=1}^N$, each $(\boldsymbol{x}_n, y_n)$ pair denotes an input-output training vector and $N$ denotes the total number of training vectors in training set $T$. For notational convenience we assume a network has only one output, but can have a vector of inputs.

**Step 2:** *Calculate out-of-bag estimates for each training sample.*

*for* $n = 1$ *to* $N$

*for* $h = 1$ *to* $H$

*for* $e = 1$ *to* $E$

Calculate:

$$E_n(h, e) = \left( \frac{\sum_{b=1}^{B} \gamma_n^b \left( \phi(\boldsymbol{x}_n; T_b^*; \hat{\omega}_{he}) \right)}{\sum_{b=1}^{B} \gamma_n^b} - y_n \right)^2 \tag{3}$$

where $\gamma_n^b = 1$ is an indicator variable that indicates whether training sample $n$ is out-of-bag for training set $T_b^*$; $\gamma_n^b = 1$ if it is and $\gamma_n^b = 0$ if it is not. These out-of-bag estimates are less noisy than individual out-of-sample estimates and are therefore more useful (see [2] for more on this). The values $E$ and $H$ are chosen by the modeler and should be large enough to ensure overfitting is observed and all dynamics of the system under study can be modeled by the networks. At the end of this step, for each training sample $(\boldsymbol{x}_n, y_n)$, the modeler will have out-of-bag error estimates throughout the range of epochs $e = (1, 2, ..., E)$ for each $h = (1, 2, ..., H)$.

**Step 3:** *Aggregate the out-of-bag estimates specific to each network in the ensemble.*

*for* $b = 1$ *to* $B$

*for* $h = 1$ *to* $H$

*for* $e = 1$ *to* $E$

Calculate:

$$E_b(h, e) = \frac{1}{N_B} \sum_{n=1}^{N} \gamma_n^b E_n(h, e) \tag{4}$$

where $N_b = \sum_{n=1}^{N} \gamma_n^b$ denotes the number of out-of-bag training samples for training set $T_b^*$. At the end of this step, for each network in the ensemble, the modeler will have aggregated out-of-bag error estimates throughout the range $e = (1, 2, ..., E)$ for each $h = (1, 2, ..., H)$.

**Step 4:** *Find the best values for* e *and* h *for each network in the ensemble.*

*for* $b = 1$ *to* $B$

Calculate:

$$OPT_b(h, e) = \overset{argmin}{h, e} \; (E_b(h, e)) \tag{5}$$

Here, for each network in the ensemble, the modeler finds the values for $e$ and $h$ that minimize out-of-bag error. The corresponding networks are chosen as the optimal set for the ensemble.

| | *I-CH* | *stdev* | *I-CV* | *stdev* | *B-CV* | *stdev* |
|---|---|---|---|---|---|---|
| usd/jpy | 33% | 6.6 | 39% | 7.2 | 19% | 5.2 |
| chf/jpy | 22% | 6.0 | 34% | 6.8 | 20% | 3.6 |
| gbp/dem | 38% | 5.8 | 45% | 5.2 | 16% | 4.7 |
| usd/dem | 31% | 5.4 | 41% | 6.1 | 12% | 5.6 |
| **Average** | **31%** | **6.0** | **40%** | **6.3** | **17%** | **4.8** |

Table 1: Average percentage decreases in generalization error yielded by NeuralBAG. Each entry corresponds to an average of results across 5 experiments. The standard deviation of results across each set of 5 experiments is also included. Entries in column *I-CH* are the average percentage decreases in generalization error provided by NeuralBAG when compared to the generalization error of individual networks trained using cheating. Entries in column *I-CV* are the average decreases in generalization error when compared to individual networks trained using *k*-fold cross-validation. Entries in column *B-CV* are the average decreases in generalization error when compared to bagged networks trained using *k*-fold cross-validation.

## 3 Experiments

In this section we summarize the results of 120 experiments that demonstrate how the generalization performance of bagged ensembles of networks trained using NeuralBAG clearly out-perform the generalization performance of individual networks trained using cheating and *k*-fold cross-validation, and bagged networks trained using *k*-fold cross-validation.

The 4 data-sets chosen for the experiments described in this section are typical noisy foreign exchange rate data-sets. Each contains 1200 daily cross-rates from 17/6/92 to 12/3/97 and is combined with daily volatility and interest rate market information to create training vectors that encode a simple 5-day lag-space. These are arranged into pools of 1200 training samples for each data-set. Each of these training samples contains a vector of 7 inputs (5 foreign exchange cross-rates, a volatility and an interest rate differential input) and 1 output (cross-rate 5 days ahead).

For each data-set and for each technique compared, experiments were performed on 5 different random re-samples of test set and training set (i.e.) each entry in table 1 overleaf corresponds to an average of results across 5 experiments. The training and test sets were formed by first sampling at random *without* replacement 200 samples from each data-set pool, using these as test samples, and then using the remaining 1000 as training samples. This random re-sampling of test sets is not normally done in time series prediction experiments - one usually trains on the past and tests on the future. However, given a limited amount of useful time-series data this technique suffices as a method to increase the statistical significance of results.

The experiments compare the generalization or test set performance of bagged ensembles of 200 networks[3] trained using NeuralBAG to the generalization performance of the following:

*I-CH (Individual network CHeating):* Networks built using cheating are networks that are built with the number of hidden units and epochs that will provide optimal performance on a specific test set. This is done by using the test set as a validation set during training and choosing the network configuration that performs best with this data. This is called cheating because, in reality, the network would not have access to the test set. Cheating is

[3]As described in [6] little improvement in generalization performance is achieved when values for $B$ larger than 100 are chosen. As a conservative safety measure we chose $B = 200$. We chose $E = 3000$ and $H = 8$ for the maximum number of epochs and hidden units to be investigated by NeuralBAG. Using 8 nodes on the TCD/QUB supercomputer each experiment took approximately 4 hours of (wall-clock) time to complete.

| | *NeuralBAG* | *stdev* |
|---|---|---|
| usd/jpy | 7% | 5.0 |
| chf/jpy | 5% | 4.2 |
| gbp/dem | 6% | 5.2 |
| usd/dem | 3% | 4.1 |
| **Average** | 5% | 4.6 |

Table 2: Each entry in this table corresponds to an average of results across 5 experiments and is a measure of the average percentage improvement in performance yielded by NeuralBAG when compared to a version of NeuralBAG where $h$, the number of hidden units remains fixed. As is evidenced by these results, estimating the number of hidden units each network in a bagged ensemble should have is justified. It provides a modest but consistent improvement in generalization performance.

a very useful benchmark measure – it provides a close estimate of the best performance an individual network could achieve on a given test set.

*I-CV (Individual network k-fold Cross-Validation):* Here we estimate the optimal number of hidden units and epochs an individual network should have using *k*-fold cross-validation, a well established technique for parameter and model selection.

*B-CV (Bagged network k-fold Cross-Validation):* Here we optimize the generalization performance of each network in an ensemble of 200 networks using *k*-fold cross-validation as described for *I-CV*.

Results are summarized in table 1.

An interesting experiment is to determine how much estimating $h$, the number of hidden units improves generalization performance. It is interesting because one can argue that each network in a bagged ensemble essentially approximates the same function. If this is the case, a simpler mechanism for estimating the number of hidden units could be used (perhaps even prior to the training process) and each network could be given an equal number of hidden units. We performed a number of experiments to test this. We repeated the NeuralBAG experiments but this time used an average of the number of hidden units estimted by the algorthm for each network as an estimte for a fixed value of $h$. Results are summarized in table 2.

## 4 Conclusion

As is evidenced by the results, the NeuralBAG algorithm can yield significant improvement in the generalization performance of neural networks.

Its only real drawback is its computational cost. However, since estimting values for $h$, the number of units is costly and only provides a modest improvement in performance, for most applications a version of NeuralBAG that does not estimate $h$ would be adequate. However, for applications such as ours where a small improvement in generalization performance can translate into large sums of money being lost or earned, estimating the number of hidden units is important. The same applies for safety-critical applications.

## References

[1] L. Breiman. Bagging predictors. *Machine Learning*, 24:123-140, 1996.

[2] L. Breman. Out-of-bag estimation. Technical Report, Statistics Department, University of California at Berekely, California, 1996.

[3] L. Breiman. Heuristics of instability in model selection. Technical Report TR416, Statistics Department, University of California at Berkeley, California, 1994.

[4] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, London, 1993.

[5] J. Hertz, A. Krogh and R. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, California, 1991.

[6] T. Heskes. Balancing between bagging and bumping. In M. Mozer, M. Jordan, and T. Peskes, editors, *Advances in Neural Information Processing Systems 9*, pages 466-472, Cambridge, 1997. MIT Press.

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