ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 135-140

The NeuralBAG algorithm: Optimizing generalization performance in bagged neural networks

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

Abstract. In this paper 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 the ensemble is optimized. Experiments performed on noisy synthetic data demonstrate the potential of the algorithm. On average, ensembles trained using NeuralBAG out-perform bagged networks trained using cross-validation by 53% and individual networks trained using "cheating" by 32%.

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 are modeled 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]. For bagging to work it is important that the predictor is unstable – bagging can actually degrade the performance of stable predictors [1].

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).

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 135-140

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 [5] 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.

2. The NeuralBAG Algorithm

In this section we describe our proposed NeuralBAG algorithm. We view neural network (backpropagation) learning as an evolution through models. Using notation similar to the above, this can be represented as

$$\Upsilon = \{\phi(T; \hat{\omega}_e)\}_{e=1}^E \tag{3}$$

where e is the current epoch (training iteration) and E is some "maximum" number of epochs specified by the modeler. This ignores however the number of hidden units a network has. In reality, the number of hidden units is very important and so we extend equation (3) above to a set of sets

$$\Upsilon = \{\{\phi(T; \hat{\omega}_{he})\}_{e=1}^E\}_{h=1}^H \tag{4}$$

where H is some "maximum" number of hidden units specified by the modeler.

Our goal should be to find the combination of values for e and h that provide optimal generalization performance. For individual networks these values could be estimated by cross-validation for example. For bagged networks, we have $\{\Upsilon_b\}_{b=1}^B$ where B is the number of networks in the ensemble. For each Υ_b we must estimate values for e and h. In this section we describe how the NeuralBAG algorithm can be used to do this. Although for clarity the serial version of the algorithm is presented here, it is easily converted to parallel form. 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] and [6] we use such "out-of-bag" samples for generalization error estimation.

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

Generate B bootstrap re-samples of $T : T_1^*, T_2^*, ..., T_B^*$ where $T = \{(x_n, y_n)\}_{n=1}^N$, each (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

$$E_n(h,e) = \left(\frac{\sum_{b=1}^{B} \gamma_n^b \left(\phi(x_n; T_b^*; \hat{\omega}_{he})\right)}{\sum_{b=1}^{B} \gamma_n^b} - y_n \right)^2 \tag{5}$$

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] and [6] 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 (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

$$E_b(h,e) = \frac{1}{N_B} \sum_{n=1}^{N} \gamma_n^b E_n(h,e) \tag{6}$$

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

$$OPT_b(h,e) = \overset{argmin}{h,e} \; (E_b(h,e)) \tag{7}$$

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.

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 135-140

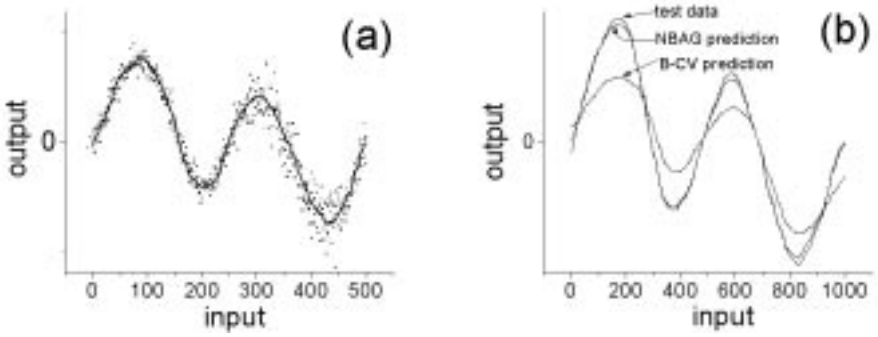

Figure 1: (a) 500 input-output noisy training vectors generated by sampling input values at random from the interval $[-1, 1]$ and plugging these values into (8). A noise free version of these values is also included. (b) 1000 test set noise free target values. Sample predictions for this test set also included. Note the local minimum produced by the bagged ensemble trained with cross-validation.

3. Experiments

Here we compare the performance of NeuralBAG to competing techniques using noisy, synthetic training and test data similar to that used in [7]. We use noisy data here because we want to highlight how NeuralBAG is more successful than competing techniques at preventing overfitting. Inputs x are randomly drawn from the interval $[-1, 1]$ and outputs y are generated according to:

$$y = sin(\pi x)cos(5\pi x/4) + \epsilon(x) \tag{8}$$

where $\epsilon(x)$ is Gaussian noise with zero mean and $var[\epsilon(x)] = 0.005 + 0.005(1 + sin(\pi x))^2$.

We randomly generate 10 training sets consisting of 500 input-output vectors and 10 test sets consisting of 1000 input-output vectors. Figure 1(a) above illustrates one of these training sets and figure 1(b) a corresponding test set. Using these, we compare the generalization performance of bagged ensembles of 200 networks[1] 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

[1] 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 $E = 30000$ and $H = 12$ for the maximum number of epochs and hidden units to be investigated by NeuralBAG. Using 8 nodes on an IBM SP2 supercomputer each experiment took approximately 1 hour of (wall-clock) time to compute.

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 135-140

I-CH	*stdev*	*B-CV*	*stdev*	*NBAG*	*stdev*	*NBAG-hf*	*stdev*
0.0445	0.0037	0.0643	0.0112	0.0303	0.0052	0.0312	0.0084

Table 1: Test set performance of ensembles trained using NeuralBAG versus competing techniques. Each entry in this table corresponds to an average of test set root-mean-squared-error performance across 10 experiments (10 different randomly generated training/test sets).

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 can be a useful benchmark measure – it provides a close estimate of the best performance any network could achieve on a given test set.
B-CV (Bagged ensemble, Cross-Validation): Here we optimize the generalization performance of each network in an ensemble of 200 networks using cross-validation. However, rather than removing part of the training set and using it as the validation set as is usually done, we use those training vectors that are excluded from each bootstrap training set following the bootstrap re-sampling as validation vectors. However we don't "bag" these vectors in any way as described in section 2 (see equation (5)).
NBAG-hf (Bagged ensemble, NeuralBAG, fixed number of hidden units): 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, to improve computational efficiency 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 estimated by the algorthm for each network as an estimate for a fixed value of h.

The results summarized above in table 1 demonstrate the potential of NeuralBAG. On average ensembles trained using NeuralBAG (*NBAG*) out-perform individual networks trained using cheating (*I-CH*) by 32% and bagged networks trained using cross-validation (*B-CV*) by 53%. Note the poor performance of bagged ensembles trained using cross-validation – this technique consistently provided a very poor estimate of generalization error and therefore also provided poor estimates for the optimal number of epochs and hidden units. Mostly only local minimums are found (see (e.g.) figure 1(b)). This highlights how important it is to bag the vectors before combining them to form a validation set as prescribed in equation (5). This provides less noisy, smoother estimates of generalization error. A full analytical exposition of why this works is beyond the scope of this paper but [2] and [6] provide some good general guidelines. We also compare the performance of ensembles trained using NeuralBAG to ensembles trained using a version of NeuralBAG where h the number

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 135-140

of hidden units remains fixed and is the same for each network in the ensemble. Estimating the (possibly different) number of hidden units each network in the ensemble should have is justified and provides a modest but consistent 4% improvement in performance. Note that these results are preliminary. The results of more experiments, that compare the performance of NeuralBAG to other competing techniques using real-world financial time-series data, are also available [4].

4. Conclusion

The only real drawback of NeuralBAG is its computational cost. However, since estimating different values for the number of hidden units each network in the ensemble should have is costly and provides only modest improvements in performance, for most applications a version of NeuralBAG that does not estimate the number of hidden units experimentally would be adequate. However, for real-world safety-critical or financial applications, for example, small improvements in generalization performance can make a big difference. For such applications an accurate estimate for the number of hidden units each network in an ensemble should have remains important.

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] J. Carney and P. Cunningham. Technical Report TCD-CS-1998-23, Computer Science Department, University of Dublin, Trinity College, Ireland, 1998.

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

[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] T. Heskes. Practical confidence and prediction intervals. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Processing Systems 9* pages 176-182, Cambridge, 1997. MIT Press.