An Ensemble of Degraded Neural Networks

Eduardo Vázquez-Santacruz and Debrup Chakraborty

[1] Department of Electrical Engineering and Computer Science, CINVESTAV-IPN,
Unidad Guadalajara, Av. Cientifica 1145, Colonia El Bajio,
Zapopan, Jalisco 45015, Mexico
evazquez@gdl.cinvestav.mx
[2] Department of Computer Science, CINVESTAV-IPN, Av. IPN 2508,
Col: San Pedro Zacatenco, Mexico City 07360, Mexico
debrup@cs.cinvestav.mx

Abstract. In this paper we present a new method to create neural network ensembles. In an ensemble method like bagging one needs to train multiple neural networks to create the ensemble. Here we present a scheme to generate different copies of a network from one trained network, and use those copies to create the ensemble. The copies are produced by adding controlled noise to a trained base network. We provide a preliminary theoretical justification for our method and experimentally validate the method on several standard data sets. Our method can improve the accuracy of a base network and give rise to considerable savings in training time compared to bagging.

1 Introduction

Let $\mathcal{L} = \{(\boldsymbol{x}_i, y_i) : i = 1, ..., n\}$ be a training set where $\boldsymbol{x}$ is a feature vector and y is its corresponding numerical response or a class label. There are plenty of procedures available in literature which uses this training set $\mathcal{L}$ to form a predictor function ϕ, which on input $\boldsymbol{x}$, give y as the output, i.e., the function ϕ approximates the input-output relationship between $\boldsymbol{x}$ and y. The function generally is of the form $\phi(\boldsymbol{x}, \boldsymbol{W})$, where $\boldsymbol{W}$ is a parameter vector which is decided upon using $\mathcal{L}$. A very popular procedure for obtaining the predictor ϕ is by training a neural network with $\mathcal{L}$. In this case we will call the predictor function $N(\boldsymbol{x}, \boldsymbol{W})$, where the parameter vector $\boldsymbol{W}$ contains the parameters (weights and biases) of the neural network, which are learned with the aid of the training set. The training algorithm finds that $\boldsymbol{W}$ which minimizes the error committed by the predictor on the training set (the training error). The operational performance measure of a predictor function is the error committed on future data points which are not present in the training set. The error on such points which are not in the training set is known as the generalization error (or test error) of the predictor. Practice has shown that a direct minimization of the training error does not always guarantee a small generalization error. There are plenty of methods available in the literature

J.-F. Martínez-Trinidad et al. (Eds.): MCPR 2011, LNCS 6718, pp. 278–287, 2011.

which improves the generalization ability of predictor functions learned from data. There are also particular methods in the context of neural networks, that broadly fall into the following categories: (a) early stopping [1](b) Complexity control of the network (weight pruning strategies, etc.)[16] (c) training with noise [10](d) ensemble methods [9,17]. In this paper we are interested with the last paradigm.

It have been noticed that an ensemble of predictors have better generalization abilities than a single predictor function [3,15]. In the past few years there have been numerous proposals for creating ensemble of predictors. Two of the well known proposals in this regard are Bagging [3] and Boosting [14]. Ample theoretical studies of Bagging and Boosting have been reported in the literature, and these studies clearly point out why and under which scenarios ensembles created by these methods can give better predictions [3,15]. Also, in the last few years numerous variants of bagging and boosting have been proposed [11,12,7].

Right from the early nineties neural network ensembles has also been widely studied [9,17]. The studies regarding neural network ensembles are mainly for adapting suitably the general ensemble techniques in case of neural networks [5]. The other studies have been focussed on developing heuristics to choose better candidates for an ensemble such that each candidate has good prediction power along with that the selected candidates have better diversity [4,8], which is known to effect the performance of an ensemble [11,12,13].

Bagging involves creating multiple bootstrap samples [6] from $\mathcal{L}$ and training predictors from each of the bootstrap samples. The final output is obtained by a suitable aggregation of the output of each predictor. The type of aggregation depends on the type of the output, i.e., whether it is a numerical response or a class label. Leo Brieman in [3] noted that neural network predictors are unstable, i.e., it is not necessary that for a trained neural network, small changes in the input will produce small changes in the output. In [3] it was also noted that along with neural networks other very popular methods like classification and regression trees, subset selection in linear regression are also unstable. In [3] it has been shown that for an unstable classifier bagging can improve the prediction both in terms of stability and accuracy.

There are theoretical guarantees about good prediction accuracy in case bagging is applied to neural networks, but using bagging to learn multiple neural predictors seems to be suboptimal, as neural network training is computationally expensive. Creating multiple neural networks from the bootstrap samples is costly. In this paper we propose a new method to create neural network ensembles. The method involves adding controlled noise to a base network and thus create numerous clones of the network and use an ensemble of the degraded networks for prediction. Our experiments show that such an ensemble can improve the performance of a base network, but the performance of such ensembles on the average is poorer than conventional bagging. What we gain by our method is a drastic reduction of training time, as in the average training using our method requires almost half training time compared to bagging. Under some assumptions we also provide a theoretical justification of why such an ensemble works.

2 The Strategy

In the discussion that follows by a neural network we shall mean a multilayered perceptron (MLP). As training multiple neural networks from bootstrap samples of the training data is time consuming, here we propose a method for generating multiple copies of neural networks from a single trained network. Let $N(\boldsymbol{W})$ be a neural network trained using the training set $\mathcal{L}$. We call this network as the *base network*. Here $\boldsymbol{W}$ is a vector containing all the learnable parameters of the network, thus, if the network contains s weights and r biases, then $\boldsymbol{W}$ will have $p = s + r$ components. Let $\boldsymbol{W} = (w_1, w_2, ..., w_p)$. Now a little perturbation of the weight vector will generate a different network, whose performance would be comparable with the base network. We create an ensemble of these degraded network, which acts as the final predictor network. In the following paragraphs we discuss the steps of our method in details.

Let $N(\boldsymbol{W})$ be a base network. By a base network we mean an MLP trained with the given training set $\mathcal{L}$. Any standard technique like error back-propagation or some of its variants can be used to train the base network $N(\boldsymbol{W})$. The size of the parameter vector $\boldsymbol{W}$ would depend on the architecture of the base network. We assume that the architecture of $N(\boldsymbol{W})$ is adequate to learn the problem represented by the training set $\mathcal{L}$. Once we have the base network, then we create a degraded version of $N(\boldsymbol{W})$ by adding a zero mean Gaussian noise to each of its components (weights and bias). Thus if $\boldsymbol{W} = (w_1, w_2, ..., w_p)$ be the parameter vector of the base network $N(\boldsymbol{W})$ and $\boldsymbol{W}^d = (w_1^d, w_2^d, ..., w_p^d)$ be the parameter vector of a degraded version of $N(\boldsymbol{W})$, then

$$w_i^d = w_i + e_i, \forall i = 1, 2, .., p \tag{1}$$

where $e_i \sim \mathcal{N}(0, \sigma_i)$, i.e. e_i is a random number drawn from a normal distribution of zero mean and variance σ, and to generate each component of the parameter of the degraded version, e_i is drawn independently of its previous values. Thus we see that the amount of degradation that a component receives is dependent on the value of σ. Further we shall call this σ as the *degradation parameter*. A large σ means a more degraded network, in average. Thus by controlling the parameter σ one can control the degree of degradation of a network. Let ϵ be the training error of the base network on the training set $\mathcal{L}$ and let ϵ_d be the error committed by the degraded network $N(\boldsymbol{W}^d)$ on $\mathcal{L}$. We call $N(\boldsymbol{W}^d)$ as a *valid candidate* if $\epsilon_d \leq t\epsilon$. Where t is a user defined threshold, which we call as the *selection threshold*. For our simulations we assume $t = 1.05$, i.e., we accept a degraded copy to be a valid candidate if the error committed by it on the training set is within 5% of the error that the base network commits on the training set [1]. Thus by repeated degradation we obtain the desired number of valid candidates and these are used to form the ensemble, with a suitable aggregation function. The overall strategy is summarized in the algorithm shown in Table 1.

[1] Note that, if the error committed by the base network is zero, then this multiplicative threshold does not work, in fact then there is no point in creating an ensemble.

Table 1. The overall strategy to create ensembles from degraded networks

Algorithm Make_Ensemble($N(\boldsymbol{W}), \sigma, m, t, \epsilon, \mathcal{L}$)
1. Let $\boldsymbol{W} = (w_1, w_2, \ldots, w_p)$;
2. $V \leftarrow \emptyset$; $d \leftarrow 1$;
3. **while** $|V| < n$,
4. **for** $j = 1$ to p,
5. $e \sim \mathcal{N}(0, \sigma)$
6. $w_j^d \leftarrow w_j + e$;
7. **end for**
8. $\boldsymbol{W}^d \leftarrow (w_1^d, w_2^d, \ldots, w_p^d)$;
9. Let ϵ_d be the error committed by $N(\boldsymbol{W}^d)$ on $\mathcal{L}$;
10. **if** $\epsilon_d \leq t\epsilon$, **then**
11. $V \leftarrow V \cup \{N(\boldsymbol{W}^d)\}$; $d \leftarrow d + 1$;
12. **end if**
13. **end while**
14. Create ensemble of the networks in $\{N(\boldsymbol{W})\} \cup V$;

In the algorithm described in Table 1 the inputs are a trained base network $N(\boldsymbol{W})$, the degradation parameter σ, a positive integer m where $m + 1$ is the number of candidates to be present in the ensemble, the selection threshold t, and ϵ, the error committed by $N(\boldsymbol{W})$ on $\mathcal{L}$ and the training set $\mathcal{L}$. The algorithm collects the valid candidates in the set V. It creates degraded copies by drawing a random number e from a normal distribution with zero mean and variance σ and adding this noise to the parameters of the base network. Then, it checks whether the degraded copy is a valid candidate and continues creating degraded copies until it gets m copies. At the end an ensemble of $m + 1$ candidate networks is created using the base network and the m valid candidates generated from the base network. By creation of an ensemble we mean using all networks $N(\boldsymbol{W}), N(\boldsymbol{W}^1), \ldots, N(\boldsymbol{W}^m)$ together for prediction. For a test point $\boldsymbol{x}$ we present the point to all networks and obtain $\xi = N(\boldsymbol{x}, \boldsymbol{W}), \xi^1 = N(\boldsymbol{x}, \boldsymbol{W}^1), \ldots, \xi^m = N(\boldsymbol{x}, \boldsymbol{W}^m)$. These outputs are aggregated together to get the final output. The most preferred technique of aggregation is to use the simple average or a majority vote.

Note that, m, t and σ are user defined parameters. The choice of m and t is not crucial. The number of candidates in an ensemble can be suitably selected and a guideline for selection of t has already been given A proper choice of σ is most crucial for the proper functioning of the algorithm. It is possible that the "optimum" value of σ is data dependent. We suggest to use a "small" value for sigma, the following example will illustrate some effects on the choice of σ also will serve as a motivation that our method works.

3 An Example

Here we provide a convincing example showing that the gross methodology works well. We consider the problem of learning a noisy sine curve as shown in the equation below.

$$f(x_i) = 0.4 \sin x_i + 0.5, x \in [-\pi, \pi]. \tag{2}$$

We generate 150 input-output pairs (x_i, y_i), $i = 1, 2, \ldots, 150$, with x_is generated uniformly on $[-\pi, \pi]$ and $y_i = f(x_i) + r_i$, where $r_i \sim \mathcal{N}(0, 0.001)$ accounts for a zero mean Gaussian noise with a small variance. We use these 150 input-output pairs as a training set. Additionally, we generate 50 more pairs using eq. 2 which we use as the test set. Now we train a base network using the training data and create degraded copies of the base network using different values of σ. Figure 1 shows the variation of the sum of square errors measured on the test data for different values of σ. In this example the base network had 10 hidden nodes in a single hidden layer, and was trained using the conventional back-propagation algorithm. For each run we generated 14 valid candidates from the network and thus created an ensemble of 15 networks according to the algorithm Make_Ensemble. In Fig. 1 we show two representative scenarios of variation of the sum of square error (SSE) on the test points of the ensemble against the choice of various values of σ.

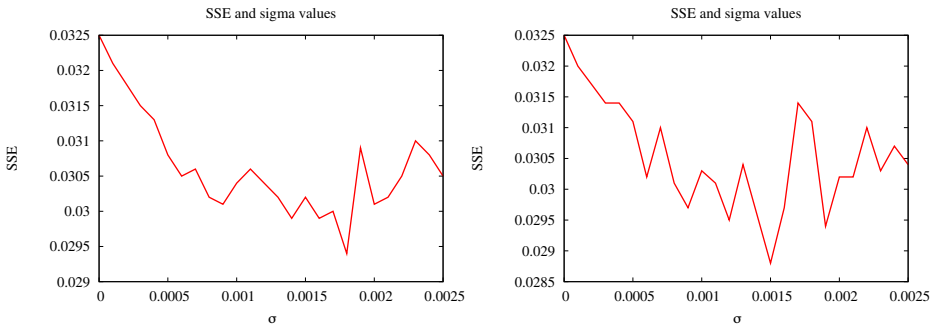

Fig. 1. Two representative runs showing the error of ensemble with variation of the degradation parameter (σ). In the figures the σ is plotted in the x axis and the SSE in the y axis.

Figure 1 clearly shows that as the value of the degradation parameter increases the SSE of the ensemble decreases, and after a certain value of the degradation parameter the SSE starts to increase. In the two scenarios shown in Fig. 1, the optimum value of σ lies in the interval [0.001,0.002]. In all the runs that we made with this data this was true. The explanation of the variation of the SSE with the degradation parameter is probably that for very small values of σ, the valid candidates generated are too similar to the base network, thus there is too little variability among the candidates of the ensemble, thus an improvement over the base network is not possible. For high values of the degradation parameter,

the structure of the base network gets altered and thus the degraded candidates created cannot really sustain the learning capabilities of the base network. Thus, it seems that there would be an optimal value of the degradation parameter which will give rise to a good ensemble. But, in this study we could not give a method to select the parameter σ, but our experience show that a small value of σ can give rise to good ensembles. Based on our experiments we suggest to select σ in the range of [0.001, 0.002], and this seems to give acceptable results across data sets.

4 A Theoretical Justification

In this section we give a theoretical justification of why our method works. We first consider an ideal scenario, where we assume an ensemble of neural networks whose parameters are sampled from a specific distribution. But this specific distribution will always be unknown and sampling from that would not be possible. Later we argue why our scenario closely resembles the ideal scenario.

When a neural network architecture and the internal activation functions are fixed and also the parameters of a learning algorithm (like the learning rate in back-propagation) are fixed then the learning algorithm becomes a deterministic algorithm. A learning algorithm then can be viewed as a function which takes as input a training set and outputs the weights and biases of the network which can be viewed as a parameter vector $\boldsymbol{W} \in \Re^p$. In other words, the learning algorithm on a fixed architecture A can be characterized by the function $\Lambda_A : \Re^q \times \mathcal{C} \rightarrow \Re^p$. Where the training set $\mathcal{L} \subset \Re^q \times \mathcal{C}$, we implicitly assume that the input feature vector is a q dimensional real vector and $\mathcal{C}$ is the set of possible class labels or the numerical responses. Also we assume that the specific architecture A has p learnable parameters. In all machine learning tasks it is assumed that the training data (also the test data) are generated from a fixed (but unknown) time invariant probability distribution. Let the the unknown distribution from which the training data $\mathcal{L}$ has been generated be $\mathcal{P}$. Let $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_r$ be r training sets generated independently from the distribution $\mathcal{P}$. Then for a fixed architecture A, the learning algorithm Λ_A will produce different parameter vectors $\boldsymbol{W}_1, \boldsymbol{W}_2, \ldots, \boldsymbol{W}_r$ corresponding to the r different training sets. The distribution $\mathcal{P}$ on the training data will induce a distribution on the parameter vectors $\boldsymbol{W}_i s$, let $\mathcal{P}_W$ denote this distribution. Now, if this distribution $\mathcal{P}_W$ is known then we can sample parameter vectors using this distribution, which can be treated as parameters of a neural network trained by the learning algorithm Λ_A using the data generated following the distribution $\mathcal{P}$. We assume that we construct an ensemble of neural networks with parameters drawn from the distribution $\mathcal{P}_W$. The ensemble of such neural networks be denoted by N_E. Let $\mathcal{W}$ be a random variable following the distribution $\mathcal{P}_W$, and $\mathcal{E}$ denote the expectation operator, then we have

$$N_E(\boldsymbol{x}) = \mathcal{E}_{\mathcal{W}}[N(\boldsymbol{x}, \mathcal{W})].$$

Assuming $\boldsymbol{X}$ and Y to be random variables having a joint distribution $\mathcal{P}$ the average prediction error for a single network would thus be

$$e = \mathcal{E}_{\mathcal{W}}[\mathcal{E}_{\boldsymbol{X},Y}[\{Y - N(\boldsymbol{X}, \mathcal{W})\}^2]].$$

The error in the ensemble N_E would be

$$e_E = \mathcal{E}_{\boldsymbol{X},Y}[\{Y - N_E(\boldsymbol{X})\}^2]$$

Now we have

$$\begin{aligned} e &= \mathcal{E}_{\mathcal{W}}[\mathcal{E}_{\boldsymbol{X},Y}[\{Y - N(\boldsymbol{X}, \mathcal{W})\}^2]] \\ &= \mathcal{E}_{\mathcal{W}}[\mathcal{E}_{\boldsymbol{X},Y}[\{Y^2 - 2YN(\boldsymbol{X}, \mathcal{W}) + N^2(\boldsymbol{X}, \mathcal{W})\}]] \\ &= \mathcal{E}_{\boldsymbol{X},Y}[Y^2] - 2\mathcal{E}_{\boldsymbol{X},Y}[YN_E(\boldsymbol{X})] + \mathcal{E}_{\boldsymbol{X},Y}[\mathcal{E}_{\mathcal{W}}[N^2(\boldsymbol{X}, \mathcal{W})]] \end{aligned} \tag{3}$$

As for any random variable Z, we have $(\mathcal{E}[Z])^2 \leq \mathcal{E}[Z^2]$, hence

$$\mathcal{E}_{\mathcal{W}}[N^2(\boldsymbol{X}, \mathcal{W})] \geq [\mathcal{E}_{\mathcal{W}}[N(\boldsymbol{X}, \mathcal{W})]]^2 = [N_E(\boldsymbol{X})]^2. \tag{4}$$

Hence using eqs. (3) and (4), we have

$$e \geq \mathcal{E}_{\boldsymbol{X},Y}[(Y - N_E(\boldsymbol{X}))^2] = e_E. \tag{5}$$

Thus we see that the error committed by the ensemble is less than that of an individual predictor, if we create an ensemble of neural networks whose parameters have been sampled from $\mathcal{P}_W$. The above explanation that we gave is due to function approximation type problems, and it cannot directly applicable for general classification problems where the outputs are discrete and do not generally bear a metric relationship between them. But, while an multilayered perceptron is trained for a classification task, the class labels are suitably coded as binary vectors and the network learns a function approximation task associated with binary outputs. Due to the choice of the activation functions (typically a sigmoid function is used in case of MLPs), the outputs of the MLP are not binary, they are real numbers in the interval $[0, 1]$. And, those real output vectors are suitably interpreted to get the final solution. So the above analysis is valid for an MLP trained as a classifier also, but may not be valid for a general classifier.

The above analysis represents an idealistic scenario, where we assume that the ensemble is created using parameter vectors $\boldsymbol{W}_i$s which follow a certain distribution $\mathcal{P}_W$. $\mathcal{P}_W$ is the distribution induced by the distribution of the input output data through the learning algorithm. Needless to say that the distribution $\mathcal{P}_W$ is unknown, in-fact a knowledge of the input-output data distribution $\mathcal{P}$ does not guarantee a closed form formula for the distribution $\mathcal{P}_W$ as they are related in a highly non-linear manner through the learning algorithm. And, to us $\mathcal{P}$ is also unknown. The best algorithm which would be faithful to the above analysis would be a technique to sample parameter vectors from the distribution $\mathcal{P}_W$, and the first step towards it would be an estimate for the distribution $\mathcal{P}_W$. We claim that our algorithm samples parameter vectors from $\mathcal{P}_W$ under certain assumptions.

To see this, let us observe our algorithm a bit closely. Our algorithm starts with a base network $N(\boldsymbol{W}^b)$. The parameters of the base network (i.e. $\boldsymbol{W}^b$)gives us one sample from $\mathcal{P}_W$. If we assume that $\mathcal{P}_W$ follows a multidimensional normal distribution centered around $\boldsymbol{W}^b$, then our process of degradation do generate parameter vectors following the distribution $\mathcal{P}_W$, with a small variance σ. This naive assumption is unlikely to capture the whole distribution $\mathcal{P}_W$ but probably restricts us to a small area in the whole distribution. But as the results in the following section suggest, this naive approximation also can give us encouraging results.

5 Experimental Results

We report performance of our method on six classification data sets from [2]. For training the neural networks we use the *traingdx* algorithm as implemented in the Neural-network toolbox of MATLAB. All the networks we use in the experiments have a single hidden layer with 10 nodes. As usual, this decision was rather ad-hoc and a change in the number of hidden nodes would not change the conclusions of our experiments. For each run we use the degradation parameter $\sigma = 0.0015$.

Table 2. Performance Comparison

Data set	Base Network Performance (in %)	Degraded Ensemble Performance (in %)	Conventional Bagging Performance (in %)
Iris	91.26 ± 6.11	91.33 ± 6.81	96.09 ± 5.66
Glass	67.87 ± 3.61	71.11 ± 9.76	72.96 ± 8.05
Waveform-40	60.19 ± 5.99	**71.02 ± 3.62**	85.41 ± 0.94
Waveform-21	62.75 ± 5.95	**72.64 ± 3.80**	84.11 ± 1.89
Pima-Diabetes	66.35 ± 2.10	**68.41 ±1.97**	75.11 ± 4.06
Wine	83.90 ± 7.99	85.54 ± 7.63	97.18 ± 1.89

Table 3. Training Times

Data set	Degraded Ensemble (time in secs)	Conventional Bagging (time in secs)
Iris	18.92 ± 4.37	134.01 ± 2.32
Glass	45.41 ± 3.25	66.11 ± 1.32
Waveform-40	367.12 ± 19.61	4077.50 ± 166.80
Waveform-21	294.79 ± 18.60	3669.75 ± 142.14
Pima-Diabetes	41.30 ± 10.34	797.81 ± 8.12
Wine	29.99 ± 8.26	85.54 ± 7.63

In Table 2, we show the comparative performance of our network. All the results reported are on 10 fold cross validation repeated 10 times. The second column of Table 2 gives us the average performance of a single network. For

each of the trained base networks 10 degraded copies were created and then aggregated with the principle of majority voting. The third column of the table shows the average performance of the ensemble of the degraded networks. The last column give result of conventional bagging, where also 10 candidate networks trained from bootstrap samples of the training data were trained and aggregated using the principle of majority voting.

Table 2 clearly shows that our method of creating ensembles can significantly enhance the performance of the base network. The entries in bold in Table 2 suggests that the improvement in the degraded ensemble was statistically significant [2]. But in all cases the results obtained by our method are poorer than that obtained by conventional bagging. But, what we gain in comparison to bagging is the training time. As discussed earlier, conventional bagging in neural networks amounts to training multiple networks, but in our method the candidates of the ensemble are created by perturbing the parameters of the base network. This gives rise to a huge savings of time compared to bagging. The training times for our method and bagging are depicted in Table 3. Tables 2 and 3 clearly indicates that our method can improve the performance of a single network to a large extent in very less time.

6 Discussions and Conclusion

The results in section 5 show that the method of creating ensembles from degraded networks is able to improve over the base network. The accuracy of these kind of ensembles are not better than bagging, the reason behind this is probably the lack of diversity among the candidate classifiers. The training time is also significantly low. An important feature of our methodology is that the ensemble can be created without access to the training data. It may be possible that a user has a trained network to perform a specific task but does not have access to the training data. In such a scenario, improving the accuracy through other available ensemble methods is not possible, as in all of the reported methods access to the training data is necessary to create ensembles, but our method does not require access to the training data. Additional clones can be generated from a trained network to create ensembles of the clones. This feature may find application in certain scenarios.

Some future work of immediate interest are as follows:

1. The most crucial part of the proposed algorithm is the selection of the degradation parameter σ. Unfortunately we were unable to provide a procedure to obtain an optimal value for σ in this work. But,our experience shows that small values of σ do work well. We are investigating ways to find out an optimal value of σ for a given data. We believe that this will have an immediate impact on the performance of our algorithm.
2. We noted down in Section 4 that there is a theoretical guarantee that an ensemble of networks created from parameters sampled from the distribution $\mathcal{P}_W$ would give rise to less prediction errors. With certain assumptions

[2] These results are based on a studentized t-test with 95% confidence.

we viewed our scheme for degradation as sampling vectors from the distribution $\mathcal{P}_W$. A possible technique to overcome some assumptions may be to start with multiple base networks and thus try to estimate the distribution $\mathcal{P}_W$ by a better technique (say a kernel density estimate). This would have implication on the training time, but may give rise to better accuracy.

References

1. Amari, S., Murata, N., Muller, K.-R., Finke, M., Yang, H.H.: Asymtotic statistical theory of overtraining and cross-validation. IEEE Transactions on Neural Networks 8(5), 985–996 (1997)
2. Asuncion, A., Newman, D.J.: UCI machine learning repository (2007)
3. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
4. Chen, R., Yu, J.: An improved bagging neural network ensemble algorithm and its application. In: Third International Conference on Natural Computation, vol. 5, pp. 730–734 (2007)
5. Drucker, H., Schapire, R.E., Simard, P.: Improving performance in neural networks using a boosting algorithm. In: Hanson, S.J., Cowan, J.D., Lee Giles, C. (eds.) NIPS, pp. 42–49. Morgan Kaufmann, San Francisco (1992)
6. Efron, B., Tibshirani, R.: An Introduction to the Bootstrap. CRC Press, Boca Raton (1993)
7. Gao, H., Huang, D., Liu, W., Yang, Y.: Double rule learning in boosting. International Journal of Innovative Computing, Information and Control 4(6), 1411–1420 (2008)
8. Georgiou, V.L., Alevizos, P.D., Vrahatis, M.N.: Novel approaches to probabilistic neural networks through bagging and evolutionary estimating of prior probabilities. Neural Processing Letters 27(2), 153–162 (2008)
9. Hansen, L.K., Salamon, P.: Neural network ensembles. IEEE Trans. Pattern Anal. Mach. Intell. 12(10), 993–1001 (1990)
10. Holmstrom, L., Koistinen, P.: Using additive noise in backpropagation training. IEEE Transactions on Neural Networks 3, 24–38 (1992)
11. Kuncheva, L.I.: Diversity in multiple classifier systems. Information Fusion 6(1), 3–4 (2005)
12. Kuncheva, L.I., Rodríguez, J.J.: Classifier ensembles with a random linear oracle. IEEE Trans. Knowl. Data Eng. 19(4), 500–508 (2007)
13. Kuncheva, L.I., Whitaker, C.J.: Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. Machine Learning 51(2), 181–207 (2003)
14. Schapire, R.E.: A brief introduction to boosting. In: Dean, T. (ed.) IJCAI, pp. 1401–1406. Morgan Kaufmann, San Francisco (1999)
15. Schapire, R.E.: Theoretical views of boosting. In: Fischer, P., Simon, H.U. (eds.) EuroCOLT 1999. LNCS (LNAI), vol. 1572, pp. 1–10. Springer, Heidelberg (1999)
16. Setiono, R.: A penalty function approach for pruning feed forward neural networks. Neural Computation 9, 185–204 (1997)
17. Zhou, Z.-H., Wu, J., Tang, W.: Ensembling neural networks: Many could be better than all. Artificial Intelligence 137(1-2), 239–263 (2002)