Neural Networks Fusion for Regression Problems

Ali Shamsoddini and John C. Trinder

***Abstract*—A common solution to improving the generalization problem and increasing the efficiency of different ANNs is to use ANN ensembles. These methods focus on the possibility of generating different neural nets for a dataset and combining the results for acquiring a more accurate regression. In this paper, a new ensemble method called machine learner fusion-regression (MLF-R) is proposed to increase the accuracy of the results through focusing on difficult samples. The architecture of MLF-R includes two different parts: the first is a training phase from which final nets are selected after a filtering process; the second part is a weighted decision maker including a backpropagation structure which fuses the different nets derived from the first step to predict the outputs. The results demonstrate MLF-R is more efficient than bagging, different boosting methods and the implementation of single ANN methods with 18% to 51% higher accuracy. Moreover, MLF-R offers more stable results compared to the other methods which have been tested in this paper.**

***Index Terms*—Adaptive threshold, ensemble, fusion, neural networks.**

I. Introduction

There is a major impetus for using backpropagation neural networks for prediction in different applications because they can lead to an acceptable level of accuracy derived from a desired size of training dataset [1]. However, there are some regression problems which cannot be adequately predicted when based on a single ANN because of the complexity of the problem and large volume of data [2]. Therefore the motivation for fusion of different ANNs is the potential for obtaining more accurate predictions compared with those which would be obtainable using single ANN [3]. Moreover, ANNs are known as unstable learners due to the inherent data and the random process which is used in the training. This instability is further motivation for applying ANNs in an ensemble mode [4]. According to [5], one or more of the following strategies have been applied in the currently developed ANN ensembles:

- Training a set of ANN on a constant dataset, but varying the initial random weights during training.
- Using different ANN methods on a constant training dataset.
- Altering ANN architecture, e.g. changing the number of hidden units, during the training process of constant datasets.
- Varying training datasets through different procedures such as sampling data, boosting, using different data sources, and disjoint training sets. This strategy is applied more frequently than the others.

It is also possible to create a neural network ensemble using a combination of the above strategies [5]. In general, there are two designs of ANN ensemble methods [6]:

- An ensemble of ANNs which are error-independent, without removing some of them such as those in [1], [7].
- Select the most error-independent ANNs among a large number of nets which have been produced in earlier stages [8], [9].

In this paper, a new ensemble method called machine learner fusion-regression (MLF-R) is proposed which is based on using a combination of variable initial random weights for training datasets and different ANN methods. The design of this method is similar to the second group above, since it produces different nets, but some of them are selected for more training as discussed later. In the next section, details will be given on some ensemble methods, especially bagging and boosting which will be used in this paper. In section III, the methodology of the new algorithm will be presented. The results will be shown in section IV and section V will conclude this paper.

II. Ensemble Methods

There have been many approaches for aggregating different neural networks following the mentioned strategies in the above dot points, such as Bayesian voting and its derivatives [10], manipulation of training and output sets [7], [11], [12], and injection of randomness [13]. Bagging and boosting are well-known among the methods belonging to the group of methods which varying training datasets through different procedures, and are more popular examples for addressing regression problems.

Bagging provides different versions of the training samples in a random process with replacement. Replacement means that the chance of selecting a sample is the same as for the samples which were selected in the last sampling. For an arbitrary number of epochs, the training datasets are divided into two parts including in-bag (almost 63% of training sample size) and the remainder as out-bag in each training step, and then the learner will be fed by in-bag data. At the end of the iterations, the final output is determined by simple averaging over all generated nets [11]. This technique is appropriate especially for unstable learning machines such as Tree-based algorithms and neural networks [14].

Boosting applies the hypothesis to the training data set manipulation and was developed originally by Schapire [15]

Manuscript received May 23, 2012; revised July 27, 2012.

Ali Shamsoddini is with School of Surveying and Spatial Information Systems, University of New South Wales, Sydney, Australia (e-mail: a.shamsoddini@ student.unsw.edu.au).

John C. Trinder is Visiting Emeritus Professor at School of Surveying and Spatial Information Systems, University of New South Wales, Sydney, Australia (e-mail: j.trinder@unsw.edu.au).

but it demanded a large number of training sets. Different versions of this technique have been proposed; the demands for large training data led to the development of other boosting versions including: Adaboost (adaptive boosting); Adaboost M1 and Adaboost M2 for classification; and Adaboost.R for regression problems [7]. Adaboost.R2 which will be used in this paper was developed to overcome some deficiencies of Adaboost.R [16].

Adaboost.R2 focuses on the difficult cases whose predictions are subject to larger errors than others, to improve the performance of machine learner. This algorithm gives initial weights to the training examples and updates these weights according to the predicted accuracy of the training examples in such a way that the probability of the selection of difficult examples is increased for the next training. Consequently, the next learning iteration is expected to be based on more difficult examples since this selection is random with replacement. In order to evaluate the prediction quality of each example, the technique uses a loss equation, which is used to calculate the difference between real and predicted values, written in either linear, square law or exponential forms [16]. The learning process will continue until the desired number of epochs is finished or terminated earlier if the value of the mean loss function is higher than 0.5. At the end of a preset number of training episodes, the final result is derived using the weighted median. Adaboost.RT was developed to apply Adaboost M1 specifically for regression problems [17]. The method in this paper uses a different loss function compared to the previous versions of Adaboost. Moreover, it uses a threshold to determine the probability of difficulty for the examples in a different way compared to Adaboost.R. Moreover, a weighted mean is used to calculate the final output of this method. There are some drawbacks with this method which will be addressed in the next section.

III. Methodology

Fig. 1 show the structure of MLF-R, which is in two parts. For the first part, machines are provided by different subsets derived randomly with replacement in the same way as in bagging and boosting. These subsets are used along with some reserved datasets to train learners in a number of iterations. The pseudo code of the step one is shown in Fig. 2. After generating different versions of the nets, the more reliable nets are selected in a filtering step as demonstrated by the pseudo code in Fig. 3. Then, the selected learners are used to form the inputs for the decision learner. In step two, the decision learner is trained for a number of iterations using these inputs and the reserved inputs which have been predicted using the best learners derived from the previous step. Finally, the weighting function will determine the final output of the ensemble method. Fig. 4 shows the pseudo code of this step.

A. Adaptive Threshold to Determine Difficult Examples

One of the most important issues related to the cases which focus on difficult examples, is the criterion which is applied to differentiate between the two different groups of examples, including easy and difficult ones. The difficulty of this task will increase when the ensemble method is dealing with a regression problem. In classification problems there are usually few classes, and a predefined margin for each class is used to correctly identify easy examples from incorrect predictions which are called difficult examples [9].

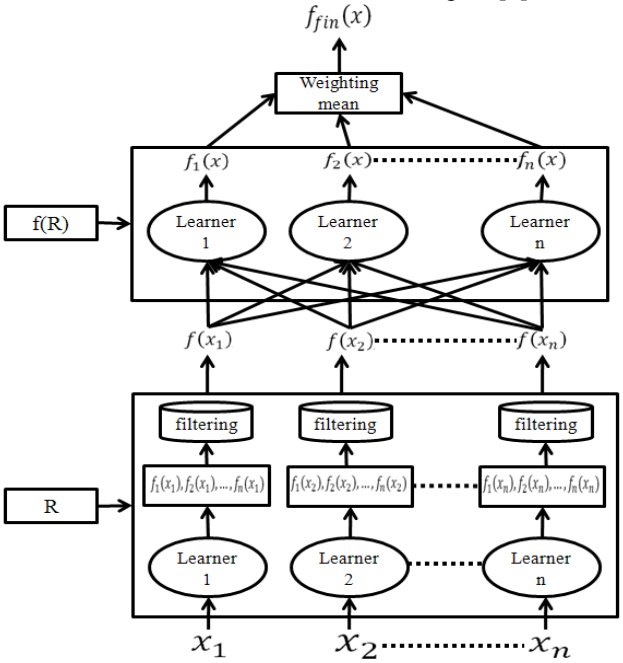

Fig. 1. The architecture of MLF-R; $X_1,\ldots,X_n$ are different training subsets which are collected randomly with replacement. R is reserved dataset. f(R) and $f(X_1),\ldots,f(X_n)$ are reserved data and different training subsets which are predicted by selected regression models to be used in the second step after filtering.

For the regression problem it is more difficult to determine a reliable boundary between easy and difficult examples due to an inevitable discrepancy between predicted and actual values. In [17], a constant value is preliminarily selected according to the prediction error statistics of a single machine. The problem is that the range of this value is infinite; however, the authors have suggested executing a calibration process before starting the ensemble method to find the optimum value of the threshold to separate easy and difficult examples. This calibration will add more time to the training.

IT specifying the number of iterations;
Nt number of samples $(x_1, y_1), \ldots, (x_{Nt}, y_{Nt})$ are used for training (TR);
Reserved dataset (R=37% of Nt);
C and β is set;
Step1. Initialize:
Distribution of D=1/Nt;
Bootstrap size (B) = 63% of Nt
For t = 1: IT
For t_1 = 1: IT
Call machine learner and providing it with distribution D_{t1}
Derive regression model of $f_{t1}(x_i)$ to predict Y
Calculate the loss of $f_{t1}(x)$ for all in- and out-bag training samples:
$|f_{t1}(x_i) - Y_i|$
NL is formed to remove $\{x_{t1}(i) | l_{t1}(i) < \Phi\}$
Calculate RMSE for in- and out-bag training samples;
Calculate δ and Save it into a cache;
Replace the removed training examples from TR with R
If R is {ϕ} or Nt_t < B, Reset T and R same as the beginning
Set $t_1 = t_1+1$
End
Select $\{f_{t1}(x_i) | \delta_{t1} = \min(\delta)\}$
Reset TR and R same as the beginning
Set t= t+1
End

Fig. 2. The pseudo code of the step 1 of MLF-R algorithm

For calculating the loss of the predicted training examples the authors in [17] also proposed a relative error function which does not work when there are some examples with zero values in the training dataset. Finally, the threshold which is utilized in the proposed methods is not set automatically and consequently requires calibration.

Step1 finished.
Filtering of generated nets: $f_{t1}(x), f_{t2}(x), \ldots, f_{tT}(x)$
A=A+1;
$\alpha = \max\{\delta\}$;
If A=1
  If $\alpha < 0.5$
    $\theta = \alpha$;
  else
    $\theta = 0.5$
  End
Set F all $\{f_t(x) | \delta_t < \theta\}$
  F $\neq \{\phi\}$; finish filtering process, otherwise return to step1
elseif A=2
  If $\alpha < 0.5$
    $\theta = \alpha$;
  else
    $\theta = 0.5$
  End
Set F all $\{f_t(x) | \delta_t < \theta\}$
F $= \{\phi\}$; set F as $\{f_t(x) | \delta_t < \min(\delta)\}$ and finish.
End

Fig. 3. The pseudo code of the filtering process of the nets

In order to overcome the above deficiencies in the MLF-R algorithm, the actual errors in the predicted examples are assumed to be normally distributed variables. Therefore, the losses of the predicted examples are calculated using the following equation that is the same as the loss equation in Adaboost.R2:

$$l_t(i) = |f_t(x_i) - y_i| \quad (1)$$

where, $f_t(x_i)$ is the predicted value of the example i in iteration t and y_i is the actual value of the example. Then the normalized losses (NL) of each training epoch are calculated using the following equation:

$$NL = \frac{l_t(i) - \mu}{\sigma} \quad (2)$$

where, $l_t(i)$ is loss for the example i in the iteration t. μ represents the mean value of the losses while σ is the standard deviation of these values. In this normal distribution, the values which are less than threshold, Φ, are considered as easy samples in the current training iteration. The relationship between Φ and σ_N which is the standard deviation of normalized loss values (almost equal to 1) is determined as follows:

$$\Phi = -C \times \sigma_N \quad (3)$$

where, C is in [0, 2]. If C is 0, it means the examples whose normalized loss values are less than the mean are considered as easy examples and consequently the number of easy examples will increase, whereas setting C to 2 can decrease the number of easy examples. In order to avoid these extreme situations, it is better to select a value for C between these extremes, such as 1; however as the analyses of the effect of different C values on the results (in the section IV) shows, the value of C does not significantly influence the final outcome. There are four advantages for this type of threshold:

- It is more logical and is based on statistical definition.
- It is set automatically.
- Its variation is not infinite as in previous methods.
- It is adaptive and focuses on relative errors. This type of threshold prevents a learner from treating all examples which are different in terms of difficulty, in the same way.

$f(TR) = \{f_{t1}(TR_1), y1), \ldots, f_{tn}(TR_{Nt}), y_{Nt}\}$;
$f(R) = \{(f_{t1}(R_1), y_1), \ldots, f_{tn}(R_{Nt}), y_{Nt})\}$
β is set;
Initialize:
Distribution of D=1/Nt;
Bootstrap size (B) = 63% of Nt
  Start: For T = 1: IT
    Call machine learner and providing it with distribution D_{t1}
    Derive regression model of $f_{T1}(f_t(x_i))$ to predict Y
    Calculate the loss of $f_{T1}(f_t(x))$ for all in- and out-bag training samples:
      $|f_{T1}(f_t(x_i)) - Y_i|$
    NL is formed to remove $\{f_{T1}(f_t(x_i)) | l_{t1}(i) < \Phi\}$
    Calculate RMSE for in- and out-bag training samples;
    Calculate δ and Save it into a cache;
    Replace the removed training examples from $f(TR)$ with $f(R)$
    If $f(R)$ is $\{\phi\}$, Reset $f(TR)$ and $f(R)$ same as the beginning;
    If $Nt_t < B$, Reset $f(TR)$ and $f(R)$;
    Set T= T+1
  End
$F_{final}(x_i) = \frac{\sum_T^{IT} \exp(\frac{1}{\delta}) \times f_T(f_t(x_i))}{\sum_T^{IT} \exp(\frac{1}{\delta})}$

Fig. 4. The pseudo code of the step 2 of MLF-R algorithm

B. Training Subset Selection

Using replacement in the sampling process generates some overlapped subsets. Although, the probability of using easy samples decreases during the iterations, they may still be used in the next training subset. This consequently leads to less emphasis on the more difficult examples. In order to overcome this deficiency and increase attention on the difficult samples, which is the aim of this new ensemble method, *hard partitioning* [9] is used instead of the probability updating. In this manner, the examples which are predicted to an acceptable level of accuracy (NL less than Φ) are removed from the training subset. Hence, there is no need to update the probability of examples during the training process.

The problem of using *hard partitioning* is over fitting the networks as the end of the training process is approached, since early-termination is not applied in this new algorithm. In order to solve this problem, some of the examples are used as reserved data (R) to be injected into the training samples whenever required. The size of this reserved data can be a proportion of the training data. For example in this study, 37% of training data size is used to form the reserve dataset. The number of injected examples from R should be the same as the number of easy training examples which are removed. This process should continue until all reserved examples are used in training or the number of training examples is less than 63%, which is the standard bootstrap distinct sample rate,

of the training dataset size at the beginning of the training process. Under each of these conditions, there are two different scenarios for the next iteration in each step of the algorithm:

- The first step of this algorithm will start with the same training and reserved datasets, because the aim of this step is to look for the best machine learners and they have to have the same conditions in terms of training and reserved dataset at the beginning of each epoch.
- For the second step, if the number of training examples is less than the standard bootstrap distinct sample rate, the training and reserved datasets will be selected among those examples which formed the previous training and reserved examples, as we are looking for the best final results which can be derived from the training and reserved datasets.

C. Filtering Less Accurate Nets

It has been shown that the performance of an ensemble method to generalize the results of the training for the rest of datasets is a function of diversity and the accuracy of the individual machines applied in the ensemble [18], [19]. The diversity of the nets is provided through the strategy which was presented in the previous subsection. The filtering process which is developed in this study is applied to select more accurate nets among all nets which are formed in step 1 of MLF-R. After generating the regression model of an individual machine, the accuracy of this model is checked through testing the model on the in-bag examples which are used to form the regression model, and out-bag examples which are not selected in the subset selection process. RMSE are calculated for them. Afterwards, the weight of the current learner is calculated as follows:

$$\delta = 0.5^{\tau} \quad (4)$$

where, τ is derived through the following equation:

$$\tau = exp^{(\beta - RMSE)} \quad (5)$$

In (5), β is the RMSE of an individual machine, which is derived before starting the algorithm. In this algorithm, δ is utilized to judge the performance of the machine learners as is shown in Fig. 2. In order to filter the generated nets, the pseudo code in Fig. 3 is used.

D. Calculation of Final Output

After training the machine learners using different subsets and selecting more accurate machine learners through the filtering process, the training and reserved datasets are predicted using the selected ANNs. Then these predicted values are used together as training and reserved datasets in the second step. β should be calculated using an individual machine learner, which is supposed to be applied in step two, provided by the predicted values. In this step, the machine learner is trained as explained in subsection B. After finishing a preset number of epochs, the results of m iterations are combined together using following equation:

$$F_{final}(x_i) = \frac{\sum_{T=1}^{m} \exp(\frac{1}{\delta}) \times f_T(f_t(x_i))}{\sum_{T=1}^{m} \exp(\frac{1}{\delta})} \quad (6)$$

where, $f_T(f_t(x_i)$ represents the predictions of the results being derived in step 1 for the iteration t. In [17], a similar weighted mean function has been proposed using the natural logarithm; however our experience with the exponential form shows this function is more efficient than a logarithm function.

IV. Results

In this study two backpropagation neural networks with Gaussian and Sigmoid functions were used as machine learners in steps 1 and 2, respectively. Also, in order to demonstrate the new algorithm performance, it was compared with six different methods including bagging, Adaboost.R2 with three different loss functions, individual backpropagation neural networks separately embedded by Sigmoid and Gaussian activation functions. In order to ensure conditions were equal for all these methods, the same parameters were applied for all of these methods. In the first step of this method seven nodes were used in the hidden layer as in the other tested methods while the number of nodes in the second step was three. Three different standard benchmarks were used to test the methods. Housing and Computer Activity datasets were downloaded through the following website http://www.gatsby.ucl.ac.uk/~chuwei/regression.html, while the Friedman#1 dataset was synthesized according to [20]. Table I gives more details about these benchmarks.

TABLE I: Information Summary of Datasets

Data	Training data	Reserved data	Test data	Attributes	
				continuous	*discrete*
Housing	239	141	126	13	1
Computer Activity	945	555	500	21	-
Friedman#1	756	444	400	10	-

A. Comparison of Different Ensemble Methods

In order to compare MLF-R with other methods, training and reserved datasets were randomly selected 20 times, and all methods were tested using them. The number of epochs was set to 10 and C set to 1. Table II shows the mean RMSE and standard deviation (SD) values derived from these 20 trials for each method.

TABLE II: The Results of Different Methods

Method/ Data	Housing		Computer activity		Friedman#1	
	RMSE	SD	RMSE	SD	RMSE	SD
ANN(Gaussian)	4.69	1.1	3.83	0.9	1.60	0.47
ANN(Sigmoid)	4.89	**0.5**	3.25	1.2	2.87	0.27
Adaboost.R2 (Linear)	4.23	0.6	3.08	0.3	1.27	0.06
Adaboost.R2 (Square law)	4.22	**0.5**	3.14	0.4	1.28	0.06
Adaboost.R2 (Exponential)	4.04	**0.5**	3.04	0.4	1.30	0.07
Bagging	4.02	**0.5**	3.23	0.6	1.54	0.09
MLF-R	**3.89**	**0.5**	**2.61**	**0.1**	**1.17**	**0.05**

The superior results have been shown in bold. As the results show, MLF-R will produce more promising results than the other methods, as the RMSEs of its predictions are

less than for the others. Also, the low standard deviations of MLF-R show this method has slightly more stable performance than other methods.

In order to compare the relative performance of the methods, a scoring matrix in percent is used [17], calculated from (7). In addition to the relative performance of the methods, this matrix shows the overall performance of the methods for all datasets as shown in Table III:

$$SM_{i,j} = \frac{1}{N} \times \sum_{k=1}^{N} \frac{RMSE_{k,j} - RMSE_{k,i}}{\max(RMSE_{k,j}, RMSE_{k,i})} \quad (7)$$

where, $SM_{i,j}$ is the scoring matrix element of ith method (the header row of Table III) over jth method (header column of Table III). This table demonstrates that the relative performance of MLF-R is better than the other methods ranging from 18% improvement on Adaboost R.2 method with exponential loss function, to 51% improvement on single neural networks with sigmoid function. Also, the overall performance of MLF-R is significantly better than other methods for the tested datasets.

TABLE III: THE SCORING MATRIX FOR DIFFERENT METHODS IN %

Method	ANN (Gaussian)	ANN (Sigmoid)	Adaboost.R2 (Linear)	Adaboost.R2 (Square law)	Adaboost.R2 (Exponential)	Bagging	MLF-R	total
ANN (Gaussian)	0	10	-32	-31	-36	-28	-51	-168
ANN (Sigmoid)	-10	0	-28	-28	-31	-26	-39	-161
Adaboost.R2 (Linear)	32	28	0	1.6	-5.1	4.6	-24	37
Adaboost.R2 (Square law)	31	28	-1.6	0	-6.7	3.3	-25	28
Adaboost.R2 (Exponential)	36	31	5.1	6.7	0	8.9	-18	69
Bagging	28	26	-4.6	-3.3	-8.9	0	-24	13
MLF-R	51	39	24	25	18	24	0	**181**

B. The Effect of C on the Results

As mentioned earlier, *C* is used to find the examples which are predicted to an acceptable level of accuracy. In order to show the effect of this factor on the results of MLF-R, *C* was altered incrementally by 0.1in each trial and repeated 10 times on Friedman#1 data as synthesized data and Housing datasets as a real case. The epoch number of MLF-R was set to 10. Fig. 5 shows the results which demonstrates that *C* in [0, 2] has an insignificant effect on the performance of MLF-R and consequently, there is no need to calibrate *C* values; however using values near to zero causes RMSE to increase.

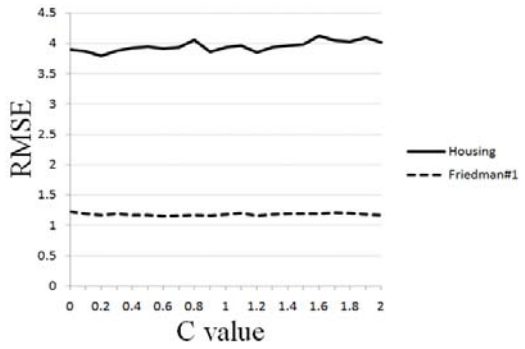

Fig. 5. The effect of varying C value on MLF-R performance

C. The Effect of Number of Epochs

Similar tests were run on Friedman#1 datasets, but the variable parameter was the epoch number and *C* was set to 1. The results are shown in Fig. 6, which demonstrates that the performance of MLF-R improves using more epochs; however, using more than 20 epochs cannot improve its performance significantly.

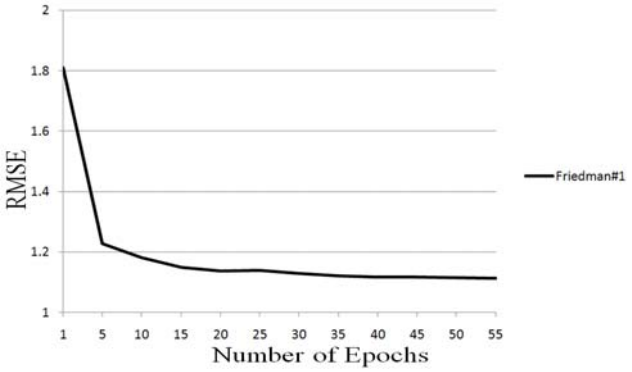

Fig. 6. The effect of varying number of epochs on MLF-R performance

V. CONCLUSION

A new ensemble method was introduced in this paper based on merging two different ANNs. The results show this method can perform more effectively than the other conventional methods such as bagging and Adaboost.R2 and single ANNs. The proposed method uses an adaptive threshold to separate difficult examples from easy examples in a *hard partitioning* manner. This adaptive threshold is set automatically, and also it was shown that this threshold does not need a calibration process before it is applied. Moreover, in the second step, a weighted ANN decision maker was introduced to fuse the results of the different machine learners. As was demonstrated, the larger number of epochs can increase the accuracy of predictions; however, the improvement rate will only slightly increase for the epoch numbers more than 20. The most important issue is that the conventional methods do not have the capability of fusing the performances of different machine learners such as ANN and support vector regression, which may provide errors in different parts of datasets. It is believed that the proposed method potentially is suitable as a hyper fusion method, but this suggestion requires more investigation.

REFERENCES

[1] B. E. Rosen, "Ensemble Learning Using Decorrelated Neural Networks," *Connection Science*, vol.8**,** pp.373 – 384, Dec. 1996.
[2] M. M. Islam, Y. Xin and K. Murase, "A constructive algorithm for training cooperative neural network ensembles," *IEEE Transactions on Neural Networks*, vol.14**,** pp. 820-834, July 2003.
[3] C.J. Merz and M.J. Pazzani, "Combining neural network regression estimates with regularized linear weights," in *Advances in Neural Information Processing Systems 9*, MIT Press, 1997, pp. 564–570.
[4] H. Navone, P. Granitto, P. Verdes and H. Ceccatto, "A Learning Algorithm for Neural Network Ensembles", *Revista Iberoamericana de Inteligencia Artificial*, vol. 3, pp. 70-74, 2001.
[5] A. J. C. Sharkey, "On Combining Artificial Neural Nets," Connection *Science*, vol.8**,** pp. 299 – 314, Dec. 1996.
[6] G. Giacinto and F. Roli, "Design of effective neural network ensembles for image classification purposes," *Image and Vision Computing*, vol.19, pp. 699-707, Aug. 2001.
[7] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of

On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, vol.55, pp. 119-139, Aug. 1997.
[8] A. J. C. Sharkey and N. E. Sharkey, "Combining diverse neural nets," *Knowl. Eng. Rev.*, vol.12, pp. 231-247, Sep. 1997.
[9] F. Zuo and P. H. N. D. With, "Cascaded face detection using neural network ensembles," *EURASIP J. Adv. Signal Processing*, vol.2008, pp. 1-13, Jan. 2008.
[10] A., Tsymbal, S. Puuronen and D. W. Patterson, "Ensemble feature selection with the simple Bayesian classification," *Information Fusion*, vol.4, pp. 87-100, June 2003.
[11] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol.24, pp. 123-140, Aug. 1996.
[12] Y. Freund, "Boosting a Weak Learning Algorithm by Majority," *Information and Computation*, vol.121**,** pp. 256-285, Sep. 1995.
[13] G. P. Zhang, "A neural network ensemble method with jittered training data for time series forecasting," *Information Sciences*, vol.177, pp. 5329-5346, Dec. 2007.
[14] T. Dietterich, 2000. "Ensemble Methods in Machine Learning," *Multiple Classifier Systems*, vol.1857, pp. 1-15, 2000.
[15] R. E. Schapire, "The Strength of Weak Learnability," *Mach. Learn.*, vol. 5**,** pp. 197-227, June 1990.
[16] H. Drucker, "Improving Regressors using Boosting Techniques," in *Proc. the Fourteenth International Conference on Machine Learning*, Morgan Kaufmann Publishers Inc., 1997, pp. 107-115.
[17] D. P. Solomatine and D. L. Shrestha, "AdaBoost.RT: a boosting algorithm for regression problems," in *Proc. IEEE International Joint Conference on Neural Networks*, vol.2, July 2004, pp. 1163-1168.
[18] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.12, pp. 993-1001, Oct. 1990.
[19] S. Hashem, "Optimal Linear Combinations of Neural Networks," *Neural Networks*, vol.10**,** pp. 599-614, June 1997.
[20] J. H. Friedman, "Multivariate Adaptive Regression Splines," *The Annals of Statistics*, vol.19**,** pp. 1-67, 1991.

Ali Shamsoddini is a PhD student at School of Surveying and Spatial Information Systems, University of New South Wales, in Sydney Australia, since 2009. He is working on the quantification of pine plantation using different remotely sensed datasets including lidar, radar and optical data and their fusion. Also, he is interested to use different machine learning techniques for forest structure mapping. His other interests are synthetic aperture radar filtering, image fusion, and remote sensing of vegetation. He has a bachelor degree in natural resource management from Agricultural Sciences and Natural Resource Management University of Gorgan, Iran and graduated a master of remote sensing and GIS from Tarbiat Modares University, Tehran, Iran.

John Trinder graduated a BSurv from the University of New South Wales (UNSW) in Sydney Australia in 1963, and MSc from ITC in The Netherlands in 1965 and PhD from UNSW in 1971. He was employed at the University of NSW since 1965, progressing to the position of Professor in 1991. He was Head of the School of Geomatic Engineering (now School of Surveying and SIS) at UNSW from 1990-1999, and currently holds the position of Visiting Emeritus Professor. John has undertaken teaching and research at UNSW for more than 40 years, specializing in Photogrammetry and Remote Sensing, and has published more than 170 scientific papers in journals and conference proceedings. He was President of the International Society for Photogrammetry and Remote Sensing (ISPRS) for the period 2000-2004 and is currently an Honorary Member of ISPRS.