A Case Study on Stacked Generalization with Software Reliability Growth Modeling Data

Ping Guo
Analysis & Testing Center,
Beijing Normal University,
Beijing, 100875, P.R. China.
Email: pguo@21cn.edu.cn

Michael R. Lyu
Dept. of Comp. Sci. & Engr.,
The Chinese University of Hong Kong,
Shatin, NT, Hong Kong.
Email: lyu@cse.cuhk.edu.hk

Abstract *We study on stacked generalization performance with software reliability growth data by using a pseudoinverse learning algorithm for feedforward neural networks. The experiments show that for noisy data, using stacked generalization can not improve the network performance when overtrained networks are engaged. With properly trained networks, stacked generalization can improve the network generalization performance.*

Keywords Stacked Generalization, Pseudoinverse Learning Algorithm, Feedforward Neural Network, Software Reliability Growth Data.

1 Introduction

Multilayer feedforward neural networks have already been found to be successful for various supervised learning tasks. Both theoretical and empirical studies have shown that the networks are of powerful capabilities for pattern classification and universal approximation [1, 2, 3]. In the neural network applications, it is a common belief that "neural networks can generalize". That is, one of the important purpose to train a neural network is for generalization. When training samples set is small and deteriorates by random noise, the network is sometimes overtrained and becomes fitted to the noise, while overfitting the noisy data will degrade the prediction accuracy of the network [4]. There are several methods to avoid the overfitting and improve network generalization, such as regularization and model selection. Weight decay [5] and early stopping [6, 7] are the most popular methods of regularization. Regularization is the procedure of allowing parameters bias towards what are thought to be more plausible values, which reduces the variance of the estimates at the cost of introducing bias. In the article of Geman *et. al* [8], a more rigorous approach on the bias/variance trade-off has been discussed. Combining networks [9, 10] can be categorized into a special case of model selection, it selects all models to form ensemble networks. *Stacked generalization* [11] can be viewed as a nonlinear combination of trained networks, it engages partitioning of the data set to find an overall system with improved generalization performance. But how it works on overtrained network and under what situations it works well still needs to be further investigated.

This paper mainly focuses on investigating stacked generalization performance with real world software reliability growth modeling data. In order to efficiently investigate the performance of the stacked generalization, we adopt a learning algorithm called pseudoinverse learning algorithm (PIL) for feedforward neural networks [12] in the experiments.

2 Stacked Generalization

The method of stacked generalization provides a way of combining trained networks together, engaging partitioning of the data set to find an overall system with improved generalization performance. The idea is to train the level-0 networks first and then examine their behavior when generalizing. This provides a new training set for training the level-1 network.

The specific procedure for setting up the stacked generalization system is as follows. Let the complete set of available data be denoted by D. We first leave aside a single data point from D as a validation point, and treat the remainder of D as a training set. All level-0 networks are then trained by the training partition and their outputs are measured using the validation data point. This generates a single pattern for a new data set which will be used to train the level-1 network. The inputs of this pattern consist of the outputs of all the level-0 networks, and the target value is the corresponding target value from the original full data set. This process is repeated with a different choice for the data point which is kept aside. After cycling through the full data set of N points we have N patterns in the new data set, which is now used to train the level-1 network. Finally, all of the level-0 networks are re-trained using the full data set D. Predictions on new data can now be made by presenting new input vector to the level-0 networks and taking their outputs as the inputs to the level-1 network, whose output constitutes the predicted output.

Mathematical expression is as the following for cross-validation partition training samples (CVPS) of stacked generalization. Given a training data set $D = \{\mathbf{x}^i, \mathbf{o}^i\}_{i=1}^N$, we randomly partition the data into K almost-equal subset $Ds_1, Ds_2, \cdots, Ds_K$. Define Ds_j and $Ds_{(-j)} = D - Ds_j$ to be the validation and training sets for the jth fold of a K-fold cross-validation. These are called level-0 models. Especially, if $K = N$, the validation set only has one sample, while training set contains $N - 1$ samples. This is called leave-one-out cross-validation.

Let $\mathbf{z}_i$ denote the validation output of the model M_j on $\mathbf{x}_i$. At the end of the entire cross-validation process, the data set assembled from the outputs of the models is

$$D_{cv} = \{\mathbf{z}_i, \mathbf{o}_i\}_{i=1}^N. \tag{1}$$

This is the level-1 data set used to train level-1 model. To complete the training process, the final level-0 models are derived using all the data in D.

3 Experiments

3.1 Learning Algorithm

Stacked generalization requires to train a lot of networks to get level-1 training samples, which is very computation-time consuming when using back propagation algorithm to perform the required task. In order to reduce training time and investigate the stacked generalization properties, this paper use the pseudoinverse learning algorithm, which is a feedforward-only algorithm. Learning errors are transferred forward and the network architecture is established. The previously trained weights in the network are not changed. Hence, the learning errors are minimized separately on each layer instead of globally for the network as a whole. The learning accuracy is determined by the number of layer. By adding layers to eliminate errors, all examples of a training set can be perfectly learned. From a mathematical computational point of view, the algorithm is based on generalized linear algebraic method and employs matrix inner products and pseudoinverse operations. Unlike gradient descent algorithms, the PIL is a feed-forward only, fully automated algorithm, including no critical user-dependent parameters such as learning rate or momentum constant. For most problems, we only need one step to reach the exact learning. More detail discussion about the PIL algorithm is in reference [12].

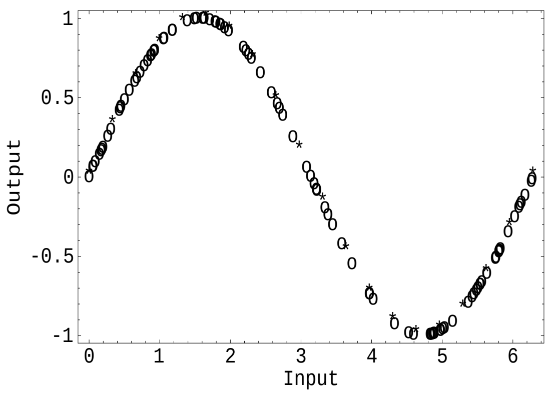

Figure 1: The trained network output for $y = sin(x)$ function mapping problem. "$*$" stands for training data output, while "o" stands for test data.

3.2 Experiments

In the experiments, multilayer neural network structure is adopted as individual network. The stacked generalization is tested with the function mapping examples first. Figure 1 shows $sin(x)$ function mapping results, which is reasonably good.

The experiments show that with smooth function or piecewise smooth function, the trained network generalization performance is

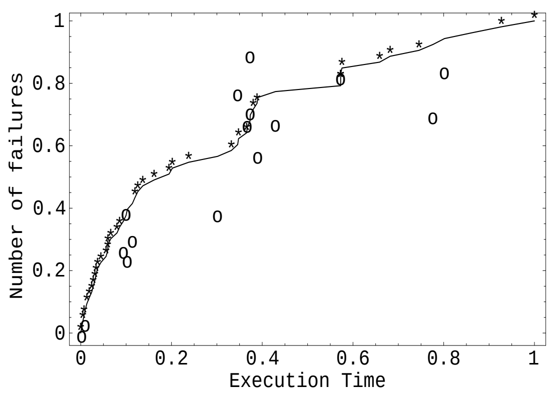

Figure 2: The neural network model trained with software reliability Sys1 data set (normalized). Solid line is the original data, "$*$" stands for training data samples, while "o" stands for test data samples. Because of overfitting the training samples, the network generalization is poor.

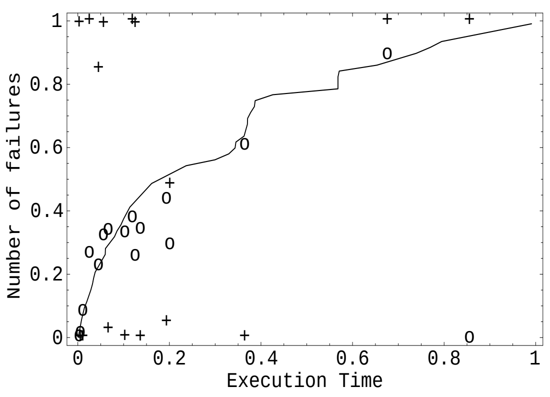

Figure 3: The stacked generalization output for Sys1 data set (normalized). Solid line is the original data, "o" stands for test data output of level-0 neural network, while "+" stands for test data output of level-1 network output. The results are also poor.

good with stacked generalization. The examples also illustrate that generalization can be expected when the underlying function is sufficiently smooth. In fact, for smoothing function, without stacked generalization, trained network performance on unseen data also good in the experiments.

In order to investigate the properties of the stacked generalization technique in noisy data case, we adopt real world data sets in further experiments. The data sets are Sys1 and Sys3 software failure data applied for software reliability growth modeling in [13].

Sys1 data set contains 54 data pairs. In the experiment, we partition the data into two parts: training set and test set. The training set consists of 37 samples which are randomly drawn from the original data set. The remaining 17 samples consist the test set. The data set are normalized to the range of [0,1]. Normalizing is a standard procedure for data preprocessing. In this problem, the network input is normalized successive failure occurrence times, and the network output is the accumulated failure number. During training, each input sample x_t at time t is associated with the corresponding output value o_t at the same time t. This kind of training is called generalization training [14].

Figure 2 shows the experimental result for software reliability growth modeling trained by using data set Sys1, which is one of the level-0 network output. Figure 3 shows the stacked generalization output for Sys1 data set. Because of overfitting the training samples, the level-0 output strays away. These samples are not in level-1 training data either, and the level-1 network outputs are further away from the desired values. The generalization ability is not improved by stacked generalization because of overfitting to the noise. Here we can see that when overfitting to the noise occurs, stacked generalization is not a suitable technique for improving network generalization performance. Poor generalization ability is not what we expected, so we should seek for the methods that can avoid overfitting in noisy data cases.

As we have mentioned early in this paper, PIL algorithm can eliminate learning errors layer by layer. For the generalization problem, we do not expect to realize the perfect learning. Therefore, we may adopt the strategy like early stopping to employ a three-layer neural network structure.

Figure 4 shows the experimental result by a three-layer structure trained with data set Sys1, which is one of the level-0 network output. To avoid overfitting, the training error is not small, and the network outputs for training samples are not completely fitting the target values. Compared with perfect leaning error which is 2.3×10^{-9}, the training error is now 0.0034. This introduces the bias to the training samples, and the output tend to be a smooth curve.

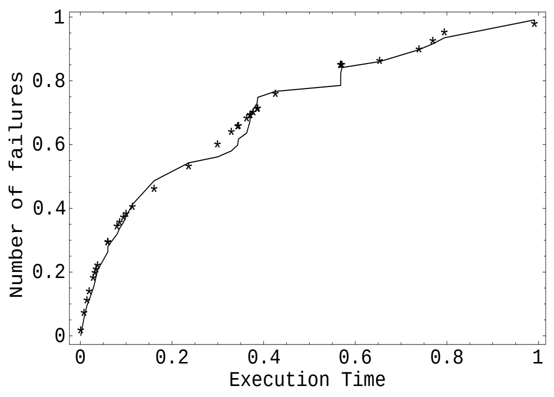

Figure 4: The three-layer network trained with software reliability Sys1 data set (normalized). Solid line is the original data, and "∗" stands for training data samples. Training accuracy is not very high and overfitting is avoided.

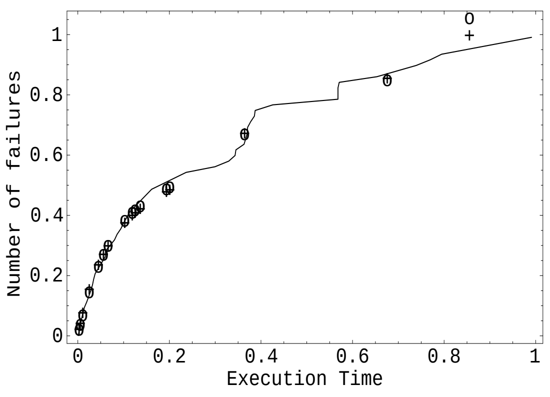

Figure 5: The stacked generalization output for Sys1 data set (normalized). Solid line is the original data, "o" stands for test data output of level-0 neural network, while "+" stands for test data output of level-1 network output. Generalization is improved at the cost of introducing training bias.

Figure 5 shows the stacked generalization output for Sys1 data set. In this case, with stacked generalization, the total sum-of-square test error is 0.0152. While without stacked generalization, the total sum-of-square test error is 0.0434. Therefore, generalization ability is improved by stacked generalization.

Another data set is Sys3. In this data set, altogether there are 278 data pairs. In the experiment, we partition the data into a training set and a test set. The number of training data is about 2/3 of the total data number, consisting of randomly drawn 186 samples from the original data set. The remaining 92 samples form the test set.

If we assign the training error as 10^{-7}, after two hidden layers are added, the final training error reaches the order of 10^{-14}. But with this trained network, the test error (20.329) is large. Figure 6 shows the results.

Now we still use leave-one-out CVPS to train level-0 neural networks for stacked generalization. At this time, the three-layer network structure is adopted. For individual network, the training error is about 0.0442, while the test error is 0.0221. Figure 7 shows the individual network training output, while Figure 8 is for stacked generalization results.

From these real-world experimental results, we can see that it is at the cost of introducing the bias (training error) to reduce the variance (generalization error) [8]. For most generalization problems the stacked generalization can be expected to reduce the generalization error rate. For example, in the Sys1 experiment, the test error is 0.0434 without stacked generalization, while the test error reduces to 0.0152 with stacked generalization. However, for some particular data set such as Sys3, stacked generalization dose not show significant improvement (test error is reduced from 0.0221 to 0.0215), but the computation time is dramatically increased. The results are summarized in Table 1.

From experiments we have observed, for noisy data, if the network is overtrained (overfit to noise), the generalization will be poor. Using stacked generalization can not improve the network performance when overtrained networks are engaged. The reason is that the overtrained network is biased to particular training samples, therefore, forecasting the values which are not in the training set will be far away from the expected values. Stacked generalization only improves generalization performance with properly trained networks.

Stacked generalization is a computation-intensive technique, when large-scale data set is given, it is not effective at all. For a large-scale data set, one of the well-known strategy is *divide-and-conquer* method. That is, partition the data set into subsets, so as to reduce the individual network size. We can also use other methods such as ensemble networks [9] to improve the network performance and then ap-

Table 1: Training error and generalization error for software reliability growth model data set

Data Set		Sys1	Sys3
Training number		37	186
Test number		17	92
Individual net (4-layers)	Training error	3.49×10^{-4}	1.47×10^{-14}
	Test error	58.003	20.329
Individual net (3-layers)	Training error	0.0181	0.0442
	Test error	0.0434	0.0221
Stacked (level-1)	Test error	0.0152	0.0215

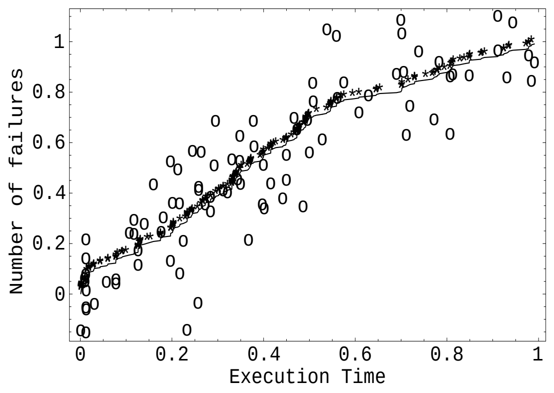

Figure 6: The network output for Sys3 data set (normalized). Solid line is the original data, "o" stands for test data output. Because of overfitting the training samples, the network generalization is poor.

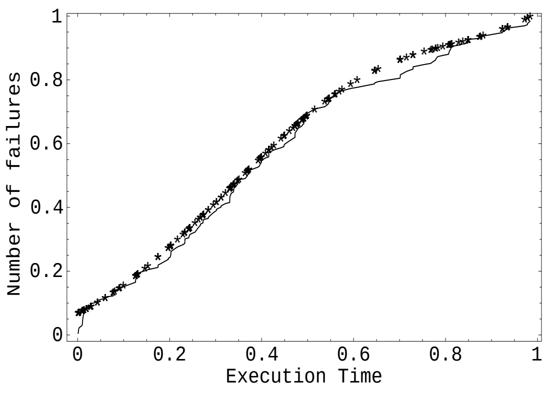

Figure 7: The three layer network model trained with software reliability Sys3 data set(normalized). Solid line is the original data, and "$*$" stands for training data samples. Training accuracy is not very high and overfitting is avoided.

ply weight parameter average to reduce the network size [15]. For example, we can employ k-fold CVPS to train neural networks. Details on ensemble neural network generalization ability and its application to software reliability growth model is beyond the scope of this paper.

4 Summary

We have made the following observations and analysis in our case study on stacked generalization with software reliability growth data. (i) For noisy data, if the network is overtrained, the generalization will be poor. Using stacked generalization can not improve the network performance when overtrained networks are engaged. (ii) With properly trained networks, stacked generalization can improve generalization performance. (iii) With smooth function or piecewise smooth function, the trained network generalization performance is good with stacked generalization. (iv) The stacked generalization is not necessarily as effective as individual network when data set is large or the underlying function is sufficiently smooth.

References

[1] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 1995.

[2] S. Haykin and C. Deng, "Classification of Radar Clutter Using Neural Networks," *IEEE Transactions on Neural Networks*, vol. 2, pp. 589–600, 1991.

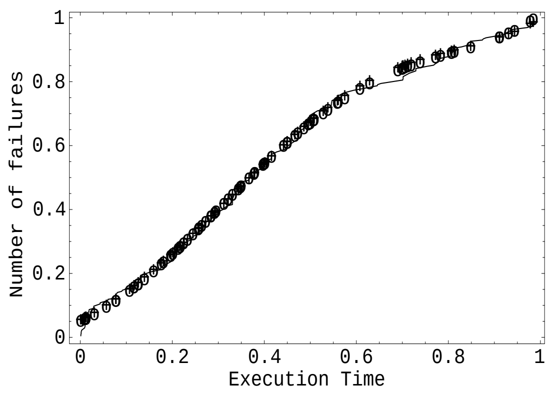

Figure 8: The stacked generalization output for Sys3 data set (normalized). Solid line is the original data, "o" stands for test data output of level-0 neural network, while "+" stands for test data output of level-1 network output. Generalization is improved at the cost of introducing training bias.

[3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard and W. Hubbard, "Handwritten Digit Recognition with a Back-propagation Network," in *Advanced in Neural Information Processing Systems*, D. S. Touretsky, Ed., San Mateo, CA, pp. 396–404, 1990.

[4] M. Smith, *Neural Networks for Statistical Modeling*, International Thomson Computer Press, Boston, MA, 1996.

[5] J. E. Moody, "The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems," in *Advanced in Neural Information Processing Systems*, J.E. Moody, S.J. Hanson, and R.P. Lippmann, Ed., MIT Press, Cambridge, MA, vol. 4, pp. 847–854, 1992.

[6] A. Weigend, "On Overfitting and the Effective Number of Hidden Units," in *Proceedings of the 1993 Connectionist Models Summer School*, M. C. Mozer, Ed., LEA/Lawrence Earlbaum Association, Boulder, CO, pp. 335–342, 1994.

[7] Warren S. Sarle, "Stopped Training and Other Remedies for Overfitting," in *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, Convention Center and Vista Hotel, Pittsburgh, PA, vol. 27, pp. 352–360, 1995.

[8] S. Geman, E. Bienenstock, and R. Doursat, "Neural Networks and the Bias/Variance Dilemma," *Neural Computation*, vol. 4, pp. 1–58, 1992.

[9] M. P. Perrone and L. N. Cooper, "When Networks Disagree: Ensemble Methods for Hybrid Neural Networks," in *Artificial Neural Networks for Speech and Vision*, R. J. Mammone, Ed., Chapman & Hall, London, pp. 126–142, 1993.

[10] Amanda J.C. Sharkey, Ed., *Combining Artificial Neural Nets: Ensemble and Modular Multi-net Systems* , Springer, London; New York, 1999.

[11] D. H. Wolpert, "Stacked Generalization," *Neural Networks*, vol. 5, pp. 241–259, 1992.

[12] Ping Guo and Michael R. Lyu, "Pseudoinverse Learning Algorithm for Feedforward Neural Networks," in *Advances in Neural Networks and Applications*, N. E. Mastorakis, Ed., WSES Press, Puerto De La Cruz, Spain, pp. 321–326, 2001.

[13] Michael R. Lyu, *Handbook of Software Reliability Engineering*, IEEE Computer Society Press, McGraw Hill, 1996.

[14] N. Karunanithi, D. Whitley and Y. K. Malaiya, "Prediction of Software Reliability Uisng Connectionist Models," *IEEE Transaction on Software Engineering*, vol. 18, pp. 563–574, 1992.

[15] Ping Guo, "Averaging Ensemble Neural Networks in Parameter Space," in *Proceedings of fifth International Conference on Neural Information Processing*, IOS Press, Kitakyushu, Japan, pp. 486–489, 1998.