Chapter 22

GENERALIZATION OF NEURAL NETWORKS FOR INTELLIGENT DATA ANALYSIS

Kok Wai Wong, Chun Che Fung, Tamás D. Gedeon, and Yew Soon Ong

Abstract

This chapter has brought statistics and neural networks together to provide a better understanding of the generalization ability of neural networks. Statistical approaches are used to analyze the function of the neural networks. It also gives a better understanding of the factors contributing to the generalization of the neural networks. Data used in training plays an important role for a successful application of neural networks in intelligent data analysis. The impact of the available data is investigated in the chapter. In most cases, data preprocessing and post processing are required to ensure that the neural networks is successful in performing intelligent data analysis. This has also contributed to the impact of the generalization ability of the neural networks.

22.1 INTRODUCTION

The study of generalization ability of the neural network is among the most important areas of current research. Among most popular neural network configuration, the Multilayer Neural Network (MLNN) with back propagation error learning [7] is the most discussed and studied. In order for MLNN to be useful in real world applications, several factors relating to the generalization ability need to be taken into account. There are the number of hidden nodes related to the number of weights in the network, amount of noise in the training set and the distribution of the training set. They will be discussed in more detail in Section 22.3 of this chapter. Most of the research works in determining the best generalization ability of the MLNN have placed their focus on estimating the complexity of the network [10] and network size [8]. There are also a few effective approaches that are used to avoid under fitting and over fitting of the MLNN. The more common approaches are the weight decay [2], early stopping [19] and utilizing of hint [20].

In the past decade, the relationship and overlapping between the field of neural networks and statistical methods have been explored [3, 4]. However, statistical methods are mainly concerned with data analysis, therefore some neural networks that are intended to model biological systems have little connection with it. Those with relationship are feed forward MLNN that is similar to the projection pursuit regression, Hebbian neural network that is similar to principal component analysis, and Kohonen net to k-means cluster analysis. Those without relationship are Kohonen Self-organizing Map (SOM) and Reinforcement learning net. Although there is some crossover between the two fields, each has their own objective in performing individual research. Neural network researchers are trying to design something that act like a human being that has the ability to adapt and learn by itself. It is treated mostly like a black box that requires no human intervention, and is used to provide data in and prediction out phenomenon. Anybody without any experience should be able to use them with confidence using automatic learning. On the other hand, statisticians usually depend on human understanding of the problem under study before designing any estimation model. They then generate hypotheses, test assumptions, and many other parameters to help them understand the built model. From their different objective, it will be fruitful if both of them can be used hand in hand. As statistician has done much study in the field of data analysis over the past few decades, the basic and analysis technique is already well established. It will be very useful to make use of the statistical analysis technique to help in designing a better neural network system. As neural network is trying to model the way a human learns and how the brain functions, the notions of learning, self-organizing, dynamics and field theory may provide statisticians with some inspiration for future studies.

The purpose of the chapter is to perform a statistical analysis on the important issues of MLNN's generalization ability, and to provide solutions to them. In Section 22.2, the statistical aspect of the MLNN will be studied and analyzed. In Section 22.3, the analysis result is then used to realize some of the potential factors that have direct effect on the generalization ability of the MLNN. In Section 22.4, investigation of the statistical frameworks used to understand the behavior of MLNN is extended to other soft computing techniques. After all these analysis, they are then used as a basis in determining approaches to find solutions to the factors that contribute to the best generalization ability. The two approaches that have been discussed in this chapter are the SOM data splitting technique and the distribution modifier technique. Their experimental simulations and results will also be reported.

22.2 STATISTICAL ANALYSIS OF NEURAL NETWORKS

In performing function approximation, the MLNN is similar and comparable to non-parametric estimators [14]. The purpose is to build a model to find the relationship between the input vector (independent vector) x and the target vector (dependent vector) y without any assumed prior parameters. Given that the input vectors X and the target vectors Y, the expression that uses to describe the relationship can be:

$$Y = g(X) \tag{1}$$

When obtaining the training set (observations), there will be some environmental factors that will affect the measurements. Therefore it is not possible to have an exact function of $g(X)$ that describes the relationship between X and Y. However, a probabilistic relationship govern by joint probability law ν can be used to describe the relative frequency of occurrence of vector pair (x, y) for n training set. The joint probability law ν can further separate into environmental probability law μ and conditional probability law γ. For notation expression, the probability law can be expressed as:

$$P(\nu) = P(\mu)P(\gamma) \tag{2}$$

For environmental probability law μ, it describes the occurrence of x. As for conditional probability law γ, it describes the occurrence of y given x. A vector pair (x, y) is considered as noise if x does not follow the environmental probability law μ, or the y given x does not follow the conditional probability law γ.

From Eq. 1, the relationship $g(X)$ based on the available training set can be assume to has direct relation with the conditional probability law γ. Therefore, it is the role of γ that the MLNN is trying to find. It can also be denoted as $E(Y|X)$ as the expectation of Y given X. Therefore:

$$g(X) = E(Y \mid X) \tag{3}$$

In MLNN, $g(X)$ is not always obtained straight away from the training set (X, Y). It has to undergo certain training (estimation) process in realizing the best $g(X)$. As MLNN, the best $g(X)$ model is directly related to the internal weights W, it can then be expressed as:

$$g(X) \approx f(X, W^*) \tag{4}$$

where W^* denotes to the best weights from the above condition, the Eq. 1 is therefore:

$$Y = f(X, W^*) + \theta \tag{5}$$

where θ denotes the error function; the output vectors (predicted value) O will be:

$$O = f(X, W) \tag{6}$$

To find the best weights W^* so as to minimize the error function θ, MLNN make use of the error back propagation learning algorithm [1] to perform the mean square errors (MSEs) minimization process, $\sum_{i=1}^{n}[Y - f(X,W)]^2$, or $\sum_{i=1}^{n}[Y - O]^2$. As the prediction performance of the MLNN is very much dependent on the weights W, the expected performance functions $\lambda(w)$ could be expressed as:

$$
\begin{aligned}
&\lambda(w)\\
&= E([Y-O]^2)\\
&= E([Y-E(Y\mid X)+E(Y\mid X)+O]^2)\\
&= E([Y-E(Y\mid X)]^2)+E([E(Y\mid X)-O]^2)+2E([Y-E(Y\mid X)][E(Y\mid X)-O])\\
&= E([Y-E(Y\mid X)]^2)+E([E(Y\mid X)-O]^2)
\end{aligned}
$$

As MSE combines the bias and variance into one measure [9, 14]. The above expression can then be separated into bias and variance term using the relationship of MSE = bias^2 + variance:

$$\text{BIAS} = E(Y\mid X)-O = E(Y\mid X)-f(X,W) \tag{7}$$

$$\text{VARIANCE} = E([Y-E(Y\mid X)]^2) \tag{8}$$

22.3 ANALYSIS OF THE GENERALIZATION OF NEURAL NETWORKS

The generalization ability of the MLNN is the most important feature in most practical application. As the realization of the best-fit model is dependent on the available training data, it is also regard as a measure on how good the MLNN can provide reasonable prediction from 'unseen' input data rather than the training data set. The MLNN using back propagation learning depends on MSE to adjust their weights *W* in minimizing the prediction error function θ, it is important to keep the MSE small. From Eqs 7 and 8, bias and variance is directly affecting the value of MSE. It is then important to keep these two components small as well. However, it is difficult to keep them small at the same time.

From Eq. 7, the bias is also dependent on the weights *W*, therefore the size of the network plays an important role in enabling the generalization ability of the MLNN. A small network with only one hidden node will most likely be biased, as the available function *f(X,W)* has limited span to adjust its weights [14]. In neural network term, it is considered under fitting. [15] and [5] have shown that a large amount of hidden nodes can make the learning fast with better training and generalization errors. [15] has also shown that with large number of hidden nodes, it is more likely to have no local minima. From these analysis, it is realized that with large hidden nodes, the bias can be reduced, thus improved the model *f(X,W*)*. Beside the weights relating to bias, the number of training vectors *(X, Y)* will also contribute to the amount of bias. In order to minimize the bias of the model, more available training vectors *(X, Y)* should be used. Usually, for most applications where training data is difficult and expensive to obtain, this has little significant in reducing the bias.

It seems that by reducing the bias, the MSE can be reduced, but this will normally increase the variance. Therefore, there is a need to keep a balance between the variance and bias. The contribution of the variance is largely dependent on the noise involved and the distribution of the training set. For the case of the noisy data, MLNN tries to reduce the MSE with small amount of bias using large number of hidden nodes. There is the danger that the variance will increase tremendously due to noisy training vectors. In effect, the final MLNN prediction model will not have good generalization ability due to the high variance. This is the phenomenon of over fitting known in neural network. In order to balance the contribution of the bias and variance in the final model, automatic smoothing technique can be applied [14]. The more common smoothing techniques used are cross-validation [13] and early stopping validation [19].

Smoothing technique is able to provide better control between bias and variance when the training set is noisy. But the distribution of the training data will generate another problem. Under the case where the 'clean' data is not evenly distributed, the probability law ν will be biased towards the majorities (large statistical frequency). As for the minorities (small statistical frequency), they will be smooth up by the automatic smoothing technique just like noise. Figure 1 shows the graphical example of a non-evenly distributed 'clean' data set, and the expected function has been smoothed up. This is valid as from Eq. 2, conditional probability law γ will affect to some extent on the final prediction model *f(X,W*)*. In effect, this will increase the bias again. Under this circumstance, a technique needs to be introduced, such that the distribution of the 'clean' data can be evened up, and the MLNN will be able to accommodate the minority characteristic function.

In this section, the generalization ability of the MLNN can be concluded as being largely dependent on the following factors:

a) number of hidden nodes or size of the weights

b) amount of noise in the training set

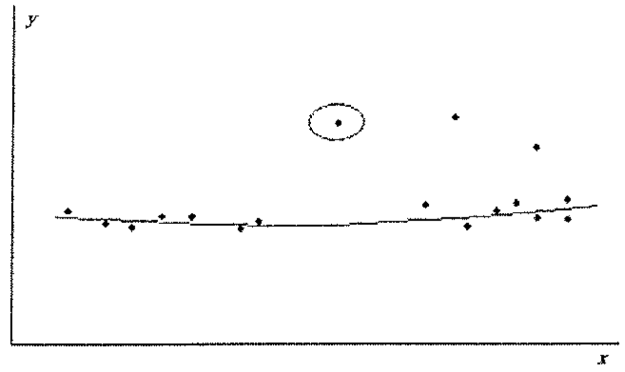

Fig. 1. Uneven Distribution of Training Data

c) distribution of the 'clean' training data in the training set.

A few points need to be noted from this analysis. First, a large number of hidden nodes or adjustable weights is favorable and can reduce bias. Secondly, the balancing of the bias and variance could be achieved by using some automatic smoothing techniques like cross-validation or early stopping validation. Thirdly, a technique should be introduced to ensure the distribution of the 'clean' data is not smooth up by the automatic smoothing technique. These will be examined in Section 22.6.

22.4 STATISTICAL FRAMEWORKS FOR OTHER SOFT COMPUTING TECHNIQUES

Although the statistical analysis presented in Section 22.2 is based on MLNNs, the basic formula can still be used for other soft computing techniques such as fuzzy inference systems and neural-fuzzy approaches.

Regardless of which soft computing techniques are used, the main purpose is to build a data analysis model that can be used when inferential prediction is required. The three main components in Eq. 1; inputs (independent vectors), X; the target (dependent vectors), Y; and the transfer function, g; still exist. This could also lead to Eq. 3, as most soft computing data analysis techniques try to infer output from a set of input variables. However, depending on the soft computing techniques used, Eq. 4 has to be re-written as:

$$g(X) \approx f(X, P^*) \tag{9}$$

where P^* denotes the set of parameters giving the best estimation, and f is the estimating function of the network.

In most neural network techniques, the P^* can be replaced by the interconnected weight vector W^* as shown in Eq. 4. This is the set of parameters that the learning process is trying to tune as shown in the previous section. In most fuzzy rule extraction techniques, the P^* can be replaced by the set of membership functions. The tuning of the membership functions is carried out in searching for the best estimation results. This is especially important in any case of data analysis problem as the fuzzy inference system makes use of fuzzy rules and membership functions to define the fuzzy patches in the input-output state space. As for most neural-fuzzy, neural-fuzzy-genetic, and genetic-fuzzy approaches, the main purpose is to search for the set of membership functions and defuzzification parameters to produce the best estimation results. In this case, the P^* has to be replaced by the joint parameters of the membership functions and defuzzification parameters. Of course, this basic formula shown in Eq. 9 can also be extended to other soft computing techniques that are not mentioned here.

For Eq. 9, by taking the error function into account, it can be re-written as:

$$Y = f(X, P^*) + \theta \tag{10}$$

where θ denotes the error.

The predicted output, O will then be:

$$O = f(X, P) \tag{11}$$

In most learning or tuning algorithms used in soft computing, the MSEs minimization process, $\sum_{i=1}^{n}[Y - f(X,P)]^2$, or $\sum_{i=1}^{n}[Y - O]^2$ is normally used as a basis to search for the best set of parameters, P^*. It is therefore suggested that the bias and variance phenomenon shown in Eqs 7 and 8 respectively are present in most soft computing techniques.

From these analyses, the following deductions for data analysis model using soft computing techniques can be made:-

a) the quality of the data analysis is directly proportional to the success of constructing the soft computing data analysis model;
b) the distribution of the available training data will decide on what the model will generalize eventually; and
c) the amount and quality of available training data used in building the model has direct effect on the accuracy of the data analysis model.

22.5 IMPACT OF AVAILABLE TRAINING DATA

In the previous two sections, it can be observed that the success of building a neural networks or soft computing data analysis model depends very much on the available training data. It is worthwhile to look at the problems that may exist in the data for most data analysis problems. These problems can basically be classified as too much available data, too little available data and fractured data [1].

22.5.1 Too Much Available Data

Depending on the application domains, we may find in some cases that there are too much available data. One may argue that more data is better in building the neural networks data analysis model. However, if there are noise and error among the data, they may effectively increase the error function in the data analysis model. This could also increase the search space, which directly increases the search time as well. One main reason of the increase of the error function can be due to the presence of noise and corrupted data in the available training data. Although most researchers have claimed that ANNs are good for rejecting noise, if the validation process is not handled properly, it may present the adverse outcomes. From Section 22.3 and 22.4, it is also observed that it disregards whichever soft computing techniques being used; the distribution of the noise may have direct effect on the accuracy of the model.

In some cases, the amount of input vectors (independent vectors, X) could be very large. If we are going to use all the available input variables in building the data analysis model, the transfer function could be a very complex model. Besides, any unrelated input variables may have a negative effect on the accuracy of the data analysis model. Therefore there is a need to make use of some input contribution measure to select relevant input variables for the problems under investigation [6, 17]. In cases where the number of the available data is very large, it is always safer to assume that the underlying function to be realized is very complicated. In this case, some kind of clustering before the actual analysis may help in achieving a better analysis model. As the search space has been reduced by the clustering technique, the data analysis model can be realized easier and faster.

22.5.2 Too Little Available Data

In some application domains, obtaining training data is an expensive and difficult process. Therefore, in some cases the amount of the available data may be minimum or insufficient. The data analysis model normally can still be derived from these data if the distribution of the important features in the data is obvious. However, in cases where the features are not well distributed, it may be difficult for any learning algorithm to estimate the best function. As shown in Eq. 2, most learning algorithms perform the role of estimating

$P(\gamma)$, so that the final prediction model, $Y = f(X, P^*) + \theta$, can be used to estimate the majority of the data and reject the minorities as noise or outliers.

Most smoothing or validation techniques involve the splitting of the available data into training and testing sets. If the amount of available data is small, some of the features may only appear in the validation set, and this will not give a good indication of the underlying function. However, in cases where the distribution is not evenly distributed or some minority features are important, it is difficult for the learning algorithm to incorporate them. These two issues will be examined and presented in the later sections of this chapter.

22.5.3 Fractured Data

In some cases, the preprocessing of the available data performed by any human expert or preprocessing technique is a very important factor in contributing to this class of problem. The process of preprocessing the available training data have to be handled carefully, as any data incompatibility between the data in the application domain will directly have a great effect on the final data analysis model. In order to ensure that the data are compatible in the problem domain, preprocessing using some kind of normalization is usually performed.

22.6 IMPROVING GENERALISATION ABILITY OF MLNN

In ensuring the generalization of MLNN, cross-validation techniques and early stopping validation techniques are commonly used as the smoothing techniques. Although cross-validation and early-splitting validation are considered two different types of automatic smoothing techniques, they do have their similarities. Both of them divide the whole sample of the available data set into training and validation sets. The difference is the way they perform smoothing in training of the MLNN. In cross-validation technique, the available data set is usually divided into *k* subsets of equal size. A *k* number of MLNN is set up, each time leaving out one of the subsets from the training. The validation error is then calculated only based on the omitted subset. This is sometime known as 'leave-one-out' cross-validation. However, the main disadvantage of this automatic smoothing technique is the training time needed to train *k* networks. As for early stopping validation, it works on the basis of split-sample methods. This only requires one network to be trained. The advantages include that only one major decision to be made by the user and it is fast as compare to the 'leave-one-out' cross-validation technique.

When applying early stopping validation, the available data set is first split into training and validation sets. A very large number of hidden nodes are used to set up the MLNN. This is favorable to our discussion in the previous section by reducing bias. By using a small learning rate, the validation error (which is also MSE) is calculated periodically. The training process is stopped when the validation error starts to rise. In this case, the validation is just like a teacher guiding a student. It is therefore plays a very important part in obtaining the best generalization ability of the MLNN.

Just like the error function used in training the MLNN, the validation process also contributed by the bias/variance dilemma. Beside this, the splitting of the training and validation set is the major contribution of the final generalization ability of the MLNN. [11] has realized that the validation error will oscillate up and down several times during the process of training. [11] has then proposed that training should let the MLNN converge, and then observed where the smallest validation error is.

Until now, all the factors that directly affect the generalization ability of the MLNN have been identified. The automatic smoothing techniques have also been examined. It can be concluded that to reduce the bias of the MSE term, large number hidden nodes is more practical (to prevent under fitting). An automatic smoothing technique is needed to ensure that the variance is kept small as well (to prevent over fitting). Early stopping validation is more preferable as it is fast and used in the situation where the number of hidden nodes is large. To find the lowest validation error, the training process is allow to converge, and the weights at the lowest validation error is used as the best weights W^*. However, there are still problems that need to be solved:

a) How to split the training and validation set?
b) How to modify the distribution of the 'clean' data?

The following two sub-sections will provide answers to these two problems.

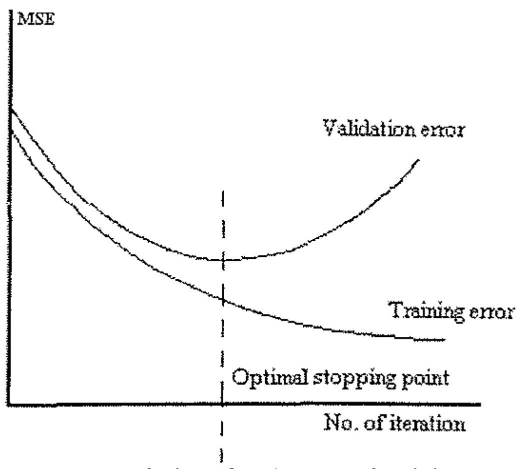

Fig. 2. Typical plot of training and validation error

22.6.1 Self-organizing Map Data Splitting Approach

When using early-stopping validation, split-sample validation is commonly used for estimating the generalization capability of a MLNN [16]. In this approach, a set of validation data that is not used in the training process is used to calculate the validation error. The validation error is found in the same way as the average training system error of the MLNN:

$$V_e = \frac{\Sigma(T_p - O_p)^2}{2P} \tag{12}$$

where V_e= average validation error; P= no. of patterns; T_p = target patterns; and O_p= output patterns.

The stopping point in this method is suggested to be the point when the validation error starts to rise. This point also indicates that the generalization ability starts to degrade. Figure 2 shows a typical graphical plot of the training system error and the validation error. When training starts, the errors for both data sets will normally reduce. After many training iterations, the validation error normally starts to rise although the training error may continue to fall. In most early stopping validation technique, the MLNN training process can be stopped at this point, as further training will result in over fitting. As mentioned in the previous section, [11] has pointed up that the validation error may oscillate up and down several times during the process of training. Therefore, training should let the MLNN converge, and observe where the smallest validation error is. Although this appears to oppose the characteristics of the early-stopping validation technique, this feature is desirable. Therefore, in this case the best generalization point is where the validation error is the smallest.

Using the above approach, the generalization ability of the MLNN is highly dependent on the set of validation data. Hence, the splitting method used is important. However, there are no rules to suggest the best splitting methods. Nevertheless, the validation data set should demonstrate two characteristics:

a) the validation set should be statistically close to the training set, and
b) the validation error should indicate the generalization ability of the final MLNN.

When using the above validation technique, for applications with sufficient data, they can be divided into three sets; training, validation and testing. The training and validation sets are used in the training process and the testing set is then used to determine the generalization error of the trained network. The main problem in this case is how to perform the data splitting.

If U is the universal sample space of all the cases of data to be processed by the network, then the training set TR should be statistically equal to U:-

$$s(TR) = s(U) \tag{13}$$

where $s()$ indicates the statistical characteristics of a data set.

If $s(TR)$ covers the complete sample space, validation set, *VA* and testing set, *TE* should be statistically similar to the training set, that is,

$$s(VA) \subseteq s(U) \tag{14}$$
$$s(TE) \subseteq s(U) \tag{15}$$

and with the condition: $VA \cap TE = \varnothing$.

However, if the conventional random approach of data splitting is used, this may result in the worst-case situation as illustrated by the following equations.

$$s(TR) \subseteq s(U) \tag{16}$$
$$s(VA) \subseteq s(U) \tag{17}$$
$$s(TE) \subseteq s(U) \tag{18}$$

with conditions: $TR \cap VA \cap TE = \varnothing$, and $s(TR) \neq s(VA) \neq s(TE) = \varnothing$

In this case, the statistical characteristics of the three data sets are all mutually exclusive. The training set does not cover all the sample space, and the validation and testing sets will not be able to give a fair indication of the generalization ability of the network.

In SOM [18] data-splitting, the available data are first classified into different clusters using unsupervised learning. If U is classified into C_1 to C_n clusters, then U can be written as:

$$U = \{C_1, C_2, C_3, \ldots, C_n\} \tag{19}$$

If the training data set is selected from each one of the n clusters and the rest are left for testing and validation, then the conditions on Eqs 14 and 15 are satisfied. In this case, the training set will cover all the desired underlying cases. The validation set and testing set are subsets from the clusters from which the training set is selected. The case studies and results related to this proposed method is presented in Section 22.7.1.

22.6.2 *Distribution Modifier of MLNN*

As has been shown in Section 22.2, in order to balance between bias and variance in estimating the data analysis model, it is important to make use of some validation or smoothing techniques. At the same time, the previous sub-section also suggested that the splitting of the available data is a very important process in ensuring that the data analysis model can cover most of the features in the underlying function. This is not normally possible, especially when all the features of the underlying function are not distributed evenly. One possible solution to this problem is to go back to the field or experiment and extract more training data that could present that specific feature in the underlying function. However, this is not feasible as the cost and effort involved is normally huge and difficult for most problems. The other solution is to modify the distribution of the available training data such that most features in the underlying function will be obvious for any learning algorithm to identify them. The objective here is therefore, with the use of any validation techniques, that it is important while performing the balancing of bias and variance, most features in the underlying function should be included in the data analysis model.

Before this can be done, an analysis into the normal error minimization process is required. Effectively, the learning algorithm is searching for the function that best describes most of the available data. For those data that are significantly small in the overall distribution, they will be treated as noise in the smoothing process. This characteristic is good to reject noise. But at the same time, any important features that are presented in this small amount of data as compared to the overall distribution will also be excluded from the validation process.

The problem now is with the splitting of the available data. It is hard to tell where the small number of important features will appear. In the previous sub-section, we have proposed a systematic splitting technique using SOM. In this method, most minorities will only be present in the training set, as the idea is to make the validation set explicitly a subset of the training set. We make use of this SOM splitting method to help us in splitting the available data into training and validation sets for use to train the MLNN. After the data has been split into the two sets, the distribution modifier using MLNNs are as follows:

1. Train a MLNN with the split sample validation technique using the training and validation set based on SOM data splitting.
2. At the end of the training, generate a distance measure between the target output and the actual output for each training point:
   $$dm = |y - o|$$
   where y is the target output and o is the actual output from the MLNN.
3. Generate a report with all the input and output vectors with high distance measures.
4. The human analyst will need to examine the report. If any of the training patterns presented in the report seem to be important, those will be the points that need to be reinforced in the distribution modification stage.
5. After the points have been identified that need to be reinforced, distance measure observation points are set to monitor them when the MLNN is undergoing training.
6. Start to reinforce those points by duplicating them to double the original number in the training set.
7. As we know that the validation set is important in providing generalization indications, the points that are reinforced in the training set should also appear in the same number in the validation set.
8. Start training the MLNN with the new training and validation sets with split-sample validation.
9. Perform distance measure *dm* on those reinforced points.
10. Perform total error measure based on the sum of square error on the training set error and the validation set error:
    $$TotE = \sqrt{TE^2 + VE^2}$$
    where TE = training set error and VE = validation set error
11. Normally, the distance measure on those points and the total error, *TotE* should both present better results before those points have been reinforced. If the total error is worse than before, the number of the duplication points in steps 6 and 7 needs to be reduced.
12. The process from steps 6 to step 11 can be repeated until the distance measures are below a certain threshold set by the human analyst.

The case studies and results related to this proposed method is presented in Section 22.7.

22.7 CASE STUDIES AND DISCUSSIONS

The case results for the SOM data splitting validation technique presented in Section 22.6.1, and the distribution modifier presented in Section 22.6.2 are investigated and discussed in the following two subsections.

22.7.1 Case Studies for SOM Data Splitting

One of the problems for the determination of petrophysical properties from well-log data is used to illustrate the application of SOM data-splitting approach used to estimate the generalization ability of a MLNN. The performance of networks using different numbers of hidden nodes is also investigated. In petroleum exploration, the prediction of petrophysical properties [12] from well log data is usually a complex problem. Measurements from several log instruments are first obtained from the site. Samples from various depths are extracted for examination; followed by extensive laboratory analysis to determine the petrophysical properties of the corresponding depth. Data obtained from these measurements and their corresponding analyses are known as *core data*. Based on information from a number of wells, decisions on important issues such as viability of production can be addressed. Traditionally, statistical and graphical methods are used to establish models to relate the well-log data to petrophysical properties of the well. With vast amount of data from increasing number of logs, alternative methods are sought to solve the problem. The MLNN has been used widely in this problem. The MLNN-based data analysis models are trained using core data. They are then used to predict the petrophysical properties of other depths or other wells within the region.

In this study, a set of typical well-log data is used and the number of available core data is 303. This set of data consists of 9 input logs (PEF, RHOB, NPHI, CALI, RT, RXO, GR, DT and SP). Typical petrophysical properties to be determined are porosity, permeability, volume of clay (VCL) and a number of other parameters. In this study, results of VCL are examined.

The available 303 core data are first classified using the unsupervised SOM method. Testing and validation data sets are then selected from each cluster. If a cluster contains only one data point, it is selected as the training data. This is to ensure that the training of MLNN covers all possible features. Table 1 shows the number of data in each set from the SOM data-splitting approach.

Table 1. Number of training, validation and testing data

Set	No. of data
Training	117
Validation	77
Testing	109

Table 2. Number of hidden units in each experiment

Experiment	No. of hidden units
1	3
2	5
3	18
4	52
5	82

Table 3. Training and Validation Error

Experiment	Training Error	Validation error
1	0.00269	0.00404
2	0.00198	0.00557
3	0.00200	0.00548
4	0.00213	0.00340
5	0.00191	0.00362

Table 4. Comparison of Errors for different MLNNs

Experiment	No. of Hidden units	Error from Data Set, *P*	Error from Data Set, *T*
1	3	0.00345	0.00591
2	5	0.00362	0.00579
3	18	0.00359	0.00571
4	52	0.00275	0.00431
5	82	0.00262	0.00392

A comparison study on the performance of the MLNN due to different number of hidden units has also been carried out. It is intended to verify whether reduction of training and generalization errors can be accomplished by using large amount of hidden units as mentioned by Lawrence et al [15]. The number of hidden units varied from very small to 8 times the number of training cases. Table 2 shows several experiments and the corresponding number of hidden units being used.

The MLNN is trained using the training data set, and the validation error is calculated for every cycle of the training process. As the training and validation errors may go up and down a few times during training, the process is run until the network converges and the validation error rises steadily indicating the network is over fitted. At such point, training is stopped and the network configuration with the lowest validation error is used. After the training and validation processes have been completed, the testing data set is then used to indicate an unbiased estimation on the MLNN generalization ability.

The MLNNs from Experiment 1 to Experiment 5 in Table 2 have been trained and stopped at the lowest validation error point. The formula in Eq. 22 is used to calculate the errors presented in this section. Table 3 shows the training and validation errors of five networks.

In order to assess the generalization ability, the training and the validation data set are combined to form Data Set, *P*, which was used previously during the training process. The Testing Data Set, *T*, is one that has not been applied to the network. Results from these two data sets are used to compare the performance of the networks. The results are tabulated in Table 4.

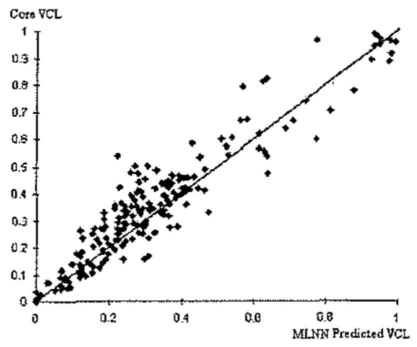

Fig. 3. Cross-plot of core data set *P* vs MLNN predicted output from Experiment 1

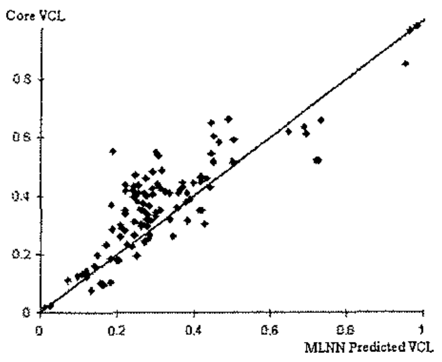

Fig. 4. Cross-plot of core data set *T* vs MLPNN predicted output from Experiment 1

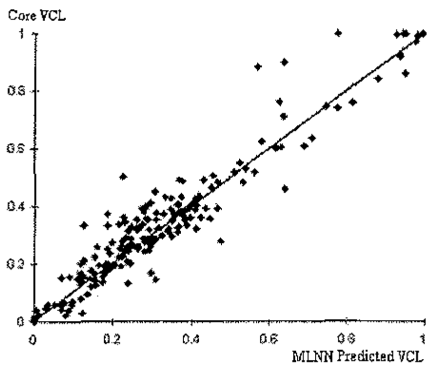

Fig. 5. Cross-plot of core data set *P* vs MLNN predicted output from Experiment 5

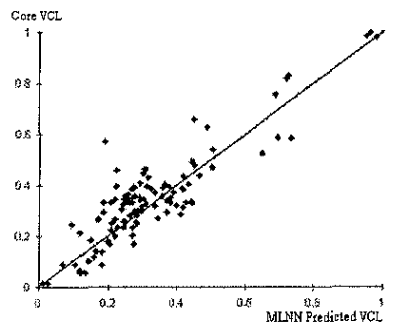

Fig. 6. Cross-plot of core data set *T* vs MLNN predicted output from Experiment 5

From these results, errors due to Data Set *P* in Table 4 are dependent on both training and validation errors in Table 3. The value is approximately the average of the two previous errors. In Experiment 2, although the training error was the lowest, but it has high validation error. The overall error became the highest as compared to other experiment cases. This shows that the validation error has large effect in determining the overall error. It also suggests that by using the training or validation error alone is not a good estimation of the generalization ability of the MLNN. The use of an unbiased testing set such as Data Set *T*, which is not used previously in the training process, should be used to estimate the generalization ability. From Table 4, it is observed that the overall error decreases as the number of hidden units increases. Experiment 5 with the largest number of hidden units gives the lowest errors from both data sets *P* and *T*. Results from other experiments using much higher number of hidden units reduce the errors insignificantly, but they require excessive long training time. This suggests that when the number of hidden units increases, it can fit the underlying function better and avoiding under fitting. However, one may argue that an oversized network may result in over fitting. This problem is now solved by using the split-sample validation approach to locate the best generalization point, and similarity of the data sets is ensured by using the SOM data-splitting approach.

Figures 3 and 4 are cross-plots of the predicted outputs generated by the MLNN in Experiment 1 as compared to core data. Fig. 3 is compared to core data set *P* while Fig. 4 is compared to core data set *T*. Figures 5 and 6 show similar cross-plots from the outputs generated from the MLNN in Experiment 5.

22.7.2 Case Studies for Distribution Modifier of MLNN

The problem of determining petrophysical properties from well-log data is again used to illustrate the

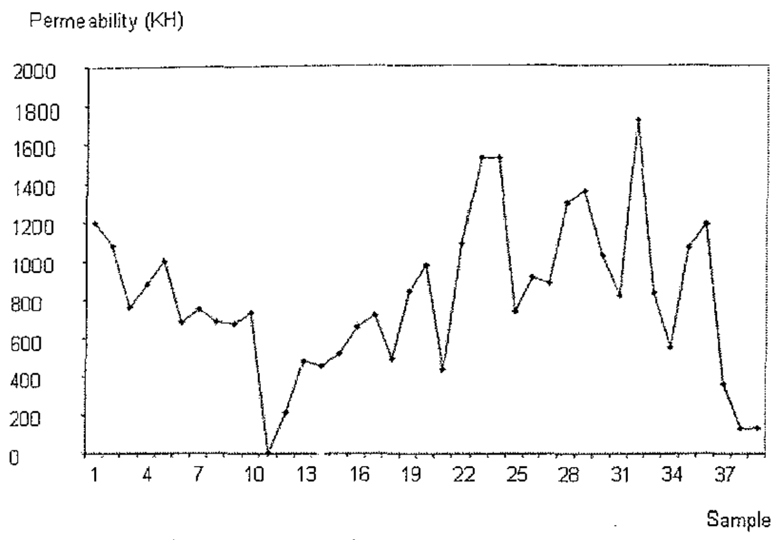

Fig. 7. Plots of the core permeability used in this case illustration

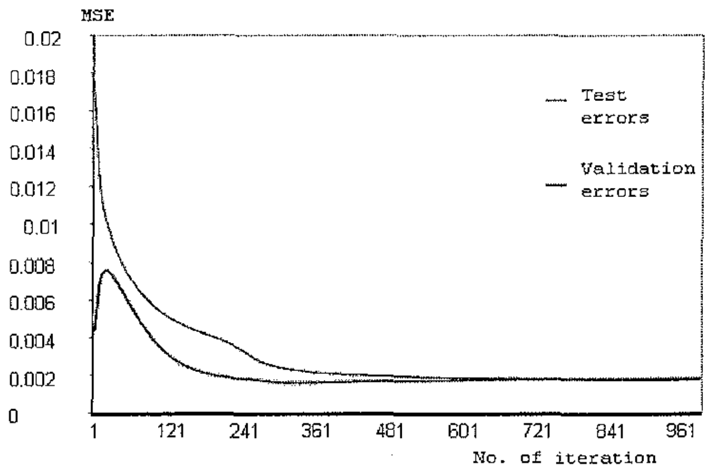

Fig. 8. Plots of the training error and the validation error in every training cycle

distribution modifier using MLNN. In this case, it has 40 core permeability data. The input logs used are neutron (NPHI), sonic travel time (DT), bulk density (RHOB), and gamma ray (GR). Figure 7 shows the plot of the core permeability. After the SOM splitting, the available core data are divided into 23 training data and 16 validation data. A MLNN was trained and the distance measures were calculated. It was found that the point near 11 at the y-axis has the largest distance measure of 0.03015. The effect of validation can be viewed in Fig. 8 that shows the training error, *TE*, and the validation error, *VE*. Figure 9 shows the distance measure for the point at 11. From Fig. 7, validation errors started to increase at around 350 epochs, even though the training errors continue to fall. It can also be observed in Fig. 9 that the distance measure for point 11 is also decreasing. This suggests that beyond this point the MLNN is trying to fit the minority data points. In the normal split-sample technique, with the purpose of rejecting noise and allowing balancing between bias and variance, the MLNN will stop at 350 epochs. This should be the point with best generalization capability. After this initial distance measure stage, the human analyst must decide whether this point at 11 should be allowed to be included in the final permeability determination model.

Two experiments are carried out to examine the proposed distribution modifier. They are summarized in Table 5 with their corresponding results. The experiments are stopped at the second experiment because the distance measure, *DM*, presents reasonable accuracy.

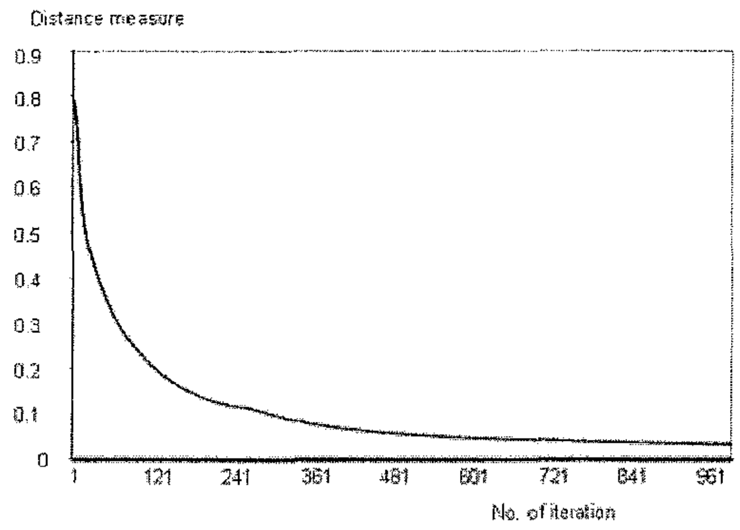

Fig. 9. Plots of the distance measure in every training cycle

Table 5. Experiments summary and results

Experiment	Number of duplications in Training set	Number of duplications in Validation set	Total Error, *TotE*	Distance Measure, *DM*, for point 11
Point Identifications	0	0	0.00276	0.03015
1	1	1	0.00232	0.01573
2	2	2	0.00219	0.01113

From the case illustration presented above, it can be seen that the proposed distribution modifier using MLNN has already affected the distribution of the available logs and core permeability. With the distribution modifier technique, one can be sure that most of the important features in the underlying function should be able to be seen by any neural networks learning algorithm regardless of their original distribution in the data set. In this way, noise could be rejected and at the same time, the significant small data features could also be incorporated in the final permeability data analysis model.

22.8 CONCLUSIONS

As statistics is a powerful tool in performing data analysis, it is used in this chapter to establish an analysis framework for neural networks to determine the factors that contribute to the best generalization ability. Statistics provides a more understanding and meaningful explanation of the factors thus indicating the direction of searching the solution. The factors that have been identified through the statistical analysis are size of the weights or number of hidden nodes, amount of noise in the training set, and the distribution of the 'clean' data. From the statistical analysis, solutions are provided to ensure that the best generalization function of the MLNN can be obtained. To avoid under fitting, a large number of hidden nodes is used to set up the MLNN. This will also enable that the vector of the weights can learn any complexity of the problem. In order to avoid MLNN from over fitting, the early stopping validation is used to perform automatic smoothing. The SOM data splitting approach is proposed to enable the confidence in splitting the training and validation set. In cases where the distribution of the 'clean' data is not even, the distribution modifier technique will enable the MLNN to accommodate them while rejecting noise at the same time. After the statistical analysis of the generalization problem and the techniques are proposed, the confidence level in the generalization ability of the MLNN could be increased. This will enhance the use of neural networks for any data analysis problems.

22.9 REFERENCES

[1] A. Famili, W. Shen, R. Weber, and E. Simoudis, "Data Preprocessing and Intelligent Data Analysis," *Intelligent Data Analysis*, 1(1), 1997, pp. 1-18.

[2] A. Weigend, D. Rumelhart, and B. Huberman, "Generalization by Weight Elimination with Application to Forecasting," *Advances in Neural Information Processing Systems* 3, 1991, pp. 875-882.

[3] B. Cheng and D.M. Titterington, "Neural Networks: A Review from a Statistical Perspective," *Statistical Science*, 9, 1994, pp. 2-54.

[4] B.D. Ripley, "Statistical Aspects of Neural Networks," In O.E. Barndorff-Nielsen, J.L. Jensen and W.S. Kendall (Eds) *Network and Chaos: Statistical and Probabilistic Aspects,* 1993, pp. 409-456.

[5] C.C.Fung, K.W. Wong, and H. Eren, "Determination of a Generalised BPNN using SOM Data-splitting and Early Stopping Validation Approach," *Proceedings of ACNN'97 Eighth Australia Conference on Neural Networks*, 1997, pp. 129-133.

[6] C.C. Fung, K.W. Wong, and H. Crocker, "Determining Input Contributions for a Neural Network Based Porosity Prediction Model," in *Proceedings of Eighth Australian Conference on Neural Network (ACNN'97)*, 1997, pp. 35 – 39.

[7] D.E. Rumelhart, G.E. Hinton, G.E., and Williams, R.J., "Learning Internal Representations by Error Propagation," in D.E. Rumelhart and J.L. McClelland (Eds) *Parallel Distributed Processing: Explorations in the Mirostructures of Cognition – Vol. 1: Foundations,* MIT Press, 1986, pp. 318-362.

[8] E.B. Baum and D. Haussler, "What Size Net Gives Valid Generalization?" *Neural Computation,* 1, 1989, pp. 151-160.

[9] H.M. Wadsworth, *Handbook of Statistical Methods for Engineers and Scientists,* 1990, pp. 12.21-12.25, McGraw-Hill Inc.

[10] J. Moody, "The Effective Number of Parameters: an Analysis of Generalization and Regularization in Nonlinear Learning Systems," *Advances in Neural Information Processing Systems*, 1992, pp. 847-854.

[11] L. Prechelt, "PROBEN1 – A Set of Neural Network Benchmark Problems and Benchmarking Rules," *Technical Report 21/94,* Universitat Karlsruhe, 1994.

[12] M. Rider, *The Geological Interpretation of Well Logs*, Second Edition, Whittles Publishing, 1996.

[13] M. Plutowski, S. Sakata, and H. White, "Cross-validation Estimates IMSE," *Advances in Neural Information Processing Systems,* 6, 1994, pp. 391-398.

[14] S. German, E. Beinenstock, and R. Doursat, "Neural Networks and the Bias/Variance dilemma," *Neural Computation*, 4, 1992, pp. 1-58.

[15] S. Lawrence, C.L. Giles, and A.C. Tsoi, "What Size Neural Network Gives Optimal Generalization? Convergence Properties of Back propagation," *Technical Report UMIACS-TR-96-22 & CS-TR-3617,* Institute for Advanced Computer Studies, University of Maryland, 1996.

[16] S.M. Weiss and C.A. Kulikowski, *Computer Systems That Learn*, Morgan Kaufmann, 1991.

[17] T.D. Gedeon, "Measuring the Significance and Contributions of Inputs in Back propagation Neural Networks for Rules Extraction and Data Mining," in Amari, S., and Kasabov, N. (eds.) *Brain-like Computing and Intelligent Information Systems*, 1997, pp. 417-434.

[18] T. Kohonen, *Self-Organizing Map*, Springer-Verlag, 1995, pp. 78-141.

[19] W.S. Sarle, "Stopped Training and Other Remedies for Over fitting," *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, 1995, pp. 352-360.

[20] Y. Abu-Mostafa, "Learning from Hints in Neural Networks," *Journal of Complexity*, 6, 1990, pp. 192-198.