Weight Training for a Multilayer Perceptron: A Comparison Study

P.M. Wong*, Y. Niu* and T.D. Gedeon**

*School of Petroleum Engineering, University of New South Wales, Sydney, Australia
pm.wong@unsw.edu.au, z2246837@student.unsw.edu.au
**School of Information Technology, Murdoch University, Murdoch, Australia
tom@dijkstra.murdoch.edu

Abstract

This paper presents a critical comparison of four optimisation algorithms for training a multilayer perceptron. The chosen algorithms are backpropagation, simulated annealing, genetic algorithms and Bayesian learning. We use a petroleum reservoir data set to compare the performance of these algorithms. The data set is randomly splitted into a training set and a test set. Error bounds are generated for all the test data. We use various statistics as performance indicators. The study shows that simulated annealing is the best algorithm for fast and efficient learning of the data set.

1. Introduction

Multilayer perceptron is a popular technique for deriving highly nonlinear relationships between data or objects. It has been successfully applied to many engineering domains, including petroleum reservoir modelling [1]. One of the crucial issues for training the popular feedforward neural network is the calculation of connection weights between neurons. Many iterative algorithms are now available for optimising such weights. This paper presents a comparison of various optimisation algorithms for training a three-layer feedforward network using a petroleum reservoir data set.

Weight training in neural networks is often posed as an error minimisation problem. The simplest error function is:

$$E = \frac{1}{n}\sum_{i=1}^{n}\left(z_i - f(\mathbf{x}_i;\mathbf{w})\right)^2$$

where E is the mean square error, n is the number of training patterns, z_i is the (one-dimensional) target output, $f(.)$ is the transfer function (feedforward neural network), $\mathbf{x}_i$ is a vector of (multivariate) inputs and $\mathbf{w}$ is the weight vector to be derived.

The most popular algorithm for deriving the weight vector is backpropagation [2], which is a type of gradient descent methods. This algorithm minimises the mean square error by setting $\partial E/\partial w_j = 0, j = 1\mathrm{K}\ m$, where m is the number of weights.

Despite the popularity of the use of backpropagation algorithm, its disadvantages are also well-known. The usual claim is its use of gradient descent which often could not provide optimal solution. It may prone to entrapment in local minima, and the calculation of partial derivatives can be difficult if the error function is multimodal and/or non-differentiable. The use of numerical approximations may further add to instability. Alternative and recent algorithms such as simulated annealing [3], genetic algorithms [4] and Bayesian learning [5] have been proposed. However, critical comparison of these algorithms in real studies is rare.

The objective of this paper is to compare the performance of the above three algorithms together with backpropagation. The basic concepts of simulated annealing, genetic algorithms and Bayesian learning are revisited. This is followed by a detailed comparison of their performance on a petroleum reservoir data set.

2. Simulated Annealing

Simulated annealing (SA) has been widely used for tackling different combinatorial optimisation problems. It is based on an analogue with the physical process of annealing (slow cooling). The elementary operation is the generation of some new candidate configuration, which is accepted if it lowers the cost

function, or accepted with a probability. The most popular update method is the Metropolis algorithm. The results obtained depend heavily on the cooling schedule used.

3. Genetic Algorithms

Genetic algorithms (GA) mimic processes in Darwinian evolution theory. They are stochastic global search methods. A genetic representation for a potential solution to a problem is encoded as a chromosome (string). A "better" solution is evolved through the processes of reproduction, crossover and mutation. In general, GA could give smaller error but it is generally more computational intensive than other optimisation algorithms.

4. Bayesian Learning

Bayesian learning (BL) in multilayer perceptron was first proposed by MacKay [5]. Unlike the conventional backpropagation neural networks and their variants, the weights are represented by a probability density function. Before the training patterns are presented to the network, the weights are described by a wide, *prior* distribution function. Once the network "sees" the training patterns, the weight distribution is updated and converted to a *posterior* one using the Bayes' theorem. With the use of a Gaussian model, the learning algorithm is able to provide a mean (y) and a variance (σ^2) for each prediction. The standard deviation (σ) can be interpreted as the error bar on the mean value, i.e. $y \pm \sigma$. Some recent geological applications can be found in Cho et al. [6] and Qu et al. [7].

5. Case Study

5.1 Data Description

The data set used came from a real reservoir with 294 wells [8]. A large-scale 2D seismic survey was carried out and the seismic velocity and amplitude data were obtained at the well locations. Average porosity (a measure of the percentage of pore volume in sedimentary rock) is available at each well. In reservoir modelling, it is important to develop a transformation from the seismic measurements to porosity so that a 2D porosity map can be generated for the whole area [9]. The issue is even more important for 3D modelling.

The above problem can be treated as a regression-type problem. A feedforward neural network was used to derive the transformation. We used the x-y coordinates of the wells, the seismic velocity and amplitude as inputs (4) and the output was porosity. All the data were normalised into the range (0,1).

5.2 Network Setup

A three-layer neural network (one hidden layer) was used. By trial and error, four hidden units were found to be appropriate for this problem. The total number of weights was 25 including the bias weights at both the hidden and output layers. Sigmoid transfer function was used at all processing units. The maximum number of epochs was set to 1,000. The BP, SA and GA work were performed using a C++ program developed by Huang [10]. The BP, SA and GA model parameters were set at the default values.

In order to obtain a better understanding of the model performance, all the networks (except Bayesian) were run for 10 times using different random seeds (initial weights). The test patterns were presented to the network after each run. This provided 10 predictions for each input data so that some indicators of prediction uncertainty (e.g. mean and variance) can be obtained. For the Bayesian analysis, we used Netlab [11]. Only one run was done because this technique is able to produce a mean and a variance from a parametric model.

The whole data set was randomly splitted into two sets: the training set and the test set. In order to utilise all the available patterns, no validation set was generated (e.g. for early-stopping). The number of training and test patterns was 250 and 44 respectively.

5.3 Statistical Indicators

This study uses a number of statistics as performance indicators. The notations and meanings are listed below:

n = Number of test data (=44)

p = Number of predictions for each input (=10)

t_j = The j^{th} actual data, $j = 1 \mathrm{K}\ n$.

y_{ij} = The i^{th} neural network prediction for the j^{th} input data, $i = 1 \mathrm{K}\ p; j = 1 \mathrm{K}\ n$.

$y_{j\min}$ = Minimum of $(y_{1j} \mathrm{K}\ y_{pj})$

$y_{j\max}$ = Maximum of $(y_{1j} \mathrm{K}\ y_{pj})$

$\bar{y}_j$ = Average of $(y_{1j} \mathrm{K}\ y_{pj})$

σ_j = Standard deviation of $(y_{1j} \mathrm{K}\ y_{pj})$

$$= \sqrt{\frac{1}{p}\sum_{i=1}^{p}\left(y_{ij} - \frac{1}{p}\sum_{i=1}^{p} y_{ij}\right)^2}$$

Min = Minimum $y_{ij} = \min(y_{11} \mathrm{K}\ y_{pn})$

Q1 = First quartile of $(\bar{y}_1 \mathrm{K}\ \bar{y}_n)$

Q2 = Second quartile of $(\bar{y}_1 \mathrm{K}\ \bar{y}_n)$

Q3 = Third quartile of $(\bar{y}_1 \mathrm{K}\ \bar{y}_n)$

Max = Maximum $y_{ij} = \max(y_{11} \mathrm{K}\ y_{pn})$

IQR = Interquartile range = Q3 – Q1

STD = Standard deviation of $(\bar{y}_1 \mathrm{K}\ \bar{y}_n)$

$$= \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(\bar{y}_j - \frac{1}{n}\sum_{j=1}^{n}\bar{y}_j\right)^2}$$

E_j = Error $= t_j - \bar{y}_j$

AE = Average error $= \frac{1}{n}\sum_{j=1}^{n} E_j$

RMSE = Root mean square error $= \sqrt{\frac{1}{n}\sum_{j=1}^{n} E_j^{\,2}}$

R2 = R-squared, the square of the Pearson correlation coefficient between t_j and $\bar{y}_j$.

I_j = Indicator of $y_{j\min} \le t_j \le y_{j\max}$

$$= \begin{cases} 1 & if\ y_{j\min} \le t_j \le y_{j\max} \\ 0 & otherwise \end{cases}$$

PI = Proportion of $y_{j\min} \le t_j \le y_{j\max}$

$$= \frac{100\%}{n}\sum_{j=1}^{n} I_j$$

STD* = Average σ_j for $y_{j\min} \le t_j \le y_{j\max}$

$$= \frac{1}{\sum_{j=1}^{n} I_j}\sum_{j=1}^{n}(I_j \cdot \sigma_j)$$

RMSE* = RMSE for $y_{j\min} \le t_j \le y_{j\max}$

$$= \sqrt{\frac{1}{\sum_{j=1}^{n} I_j}\sum_{j=1}^{n}(I_j \cdot E_j)^2}$$

Note that the two performance indicators (STD* and RMSE*) aim to evaluate the average spread of the predictions and error for the actual data falling only within the min-max bounds of the prediction. This allows closer examination of the uncertainty of predictions. It is easy to see that, the model is good if the actual data falls within the bounds. However there exists a counter-example. When a model gives extremely wide bounds (indicates low reliability and inconsistency), there is a large probability that the actual data would also fall within the bounds. To avoid such incorrect conclusion, we use STD* and RMSE* to evaluate the uncertainty the predictions. Thus, the model is good only if the actual data falls within narrow bounds (i.e. *precise* and *consistent*) and close to the actual value (i.e. *accurate*). In simple words, we look for models with small STD* and small RMSE*.

5.4 Results and Discussions

After running for 10 times, ten (10) predictions were generated for each input data. The median of the predictions was taken as the final prediction for subsequent error analyses. For BL, only one run was done due to its ability to generate a mean and a variance for each prediction. Table 1 shows the statistics of the median predictions compared to the test data. All the models seemed to give reasonable match to the test data statistics. This was due to the fact that the statistics of the training and test data sets were similar.

The cross-plot of the predictions with the actual data is shown in Figure 1 together with the R2 values. Table 2 tabulates various performance indicators. The results from the conventional multiple linear regression (MLR) are also displayed. Comparing the five models, all the neural network models gave higher R2 and lower RMSE than those obtained from MLR.

	Test	BP	SA	GA	BL
Min	0.206	0.205	0.232	0.196	0.186
Q1	0.360	0.325	0.363	0.338	0.343
Q2	0.465	0.415	0.442	0.444	0.457
Q3	0.658	0.562	0.622	0.598	0.612
Max	0.998	0.907	0.901	0.894	0.908
IQR	0.298	0.238	0.259	0.260	0.269
STD	0.188	0.187	0.183	0.186	0.183

Table 1. Statistics of the predictions and the test data.

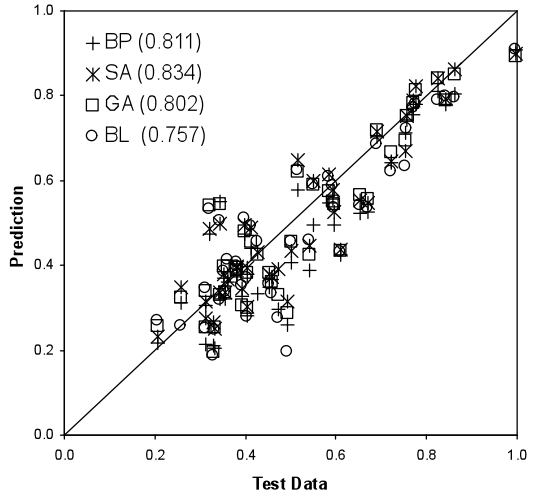


Figure 1. Cross-plot of the predictions versus test data with the R2 values.

	MLR	BP	SA	GA	BL
R2	0.722	0.811	0.834	0.802	0.757
RMSE	0.105	0.095	0.079	0.088	0.100
AE	0.026	0.046	0.017	0.021	0.031

Table 2. Performance indicators for different models.

Figure 2 shows the histograms of the prediction error (E_j) for the four neural network models. The distributions were fairly symmetrical with mean values (AE) slightly higher than zero. This means that all the models slightly underestimated the actual values on average, but no significant bias was present. Note that the AE of the BP model was about 3 times higher than the SA's value. For BP, there exists a trade-off between speed and accuracy. For GA however, it took a long time to run but the performance was less good compared to SA. BL performance was similar to BP. Based on the statistics in Table 2, SA gave the best overall performance for this data set.

Table 3 shows the uncertainty analysis results of the four models. From this table, we can see that 50% of the actual data fell between the min-max bounds predicted by the SA and GA models. In the BP model, the value was slightly smaller with only 41%. BL has the largest value due to the resulting wide bounds. According to the STD* and RMSE* values, the GA model gave the most precise (small STD*) and also the most accurate predictions (small RMSE*). The BL model gave the least precise predictions (very large STD*) with relatively large RMSE*.

Figure 3 shows the corresponding bounds for the test data set. The x-axis represents the test data ID ranked by the actual porosity values. As shown, the BL bounds were relatively wide compared to the others. The locations of the peaks and troughs of the bounds were similar for all the models due to the use of the same training set and network topology.

For a practical solution, it is important to select the algorithm with high R2, low RMSE, low AE, high PI, low STD*, low RMSE* as well as fast. Thus, we recommend the use of SA for this data set.

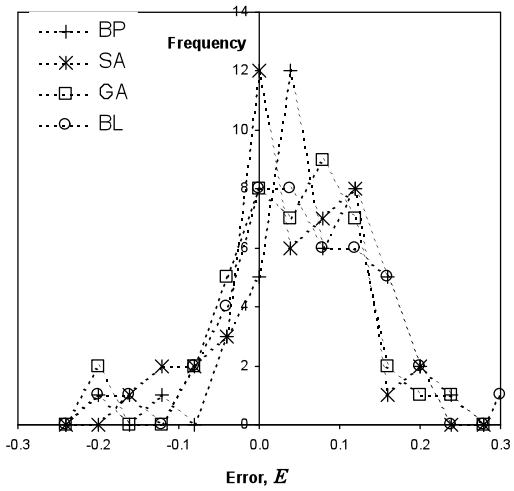


Figure 2. Histograms of the prediction error.

	BP	SA	GA	BL
PI	41%	50%	50%	70%
STD*	0.032	0.037	0.030	0.107
RMSE*	0.055	0.039	0.032	0.052

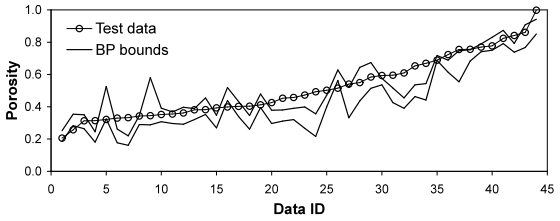
Table 3. Uncertainty indicators for different models.

6. Conclusions

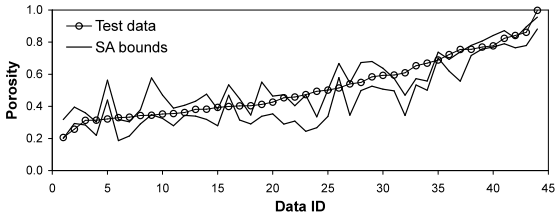
This paper shows a critical comparison of four algorithms for deriving the weights of a multilayer perceptron applied to a petroleum data set. The algorithms compared are backpropagation, simulated annealing, genetic algorithms and Bayesian learning. Based on the use of various performance indicators on precision and accuracy, we recommend the use of simulated annealing for training the neural network using the given data set.

Acknowledgments

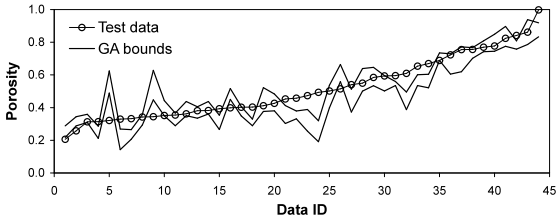
Dr PM Wong would like to thank Institut Français du Pétrole at Rueil Malmaison (France) where he was appointed as Visiting Professor from February to August 2000 when the major technical work of this paper was done. This work is supported by an Australian Research Council's Small Grant awarded in 2000.



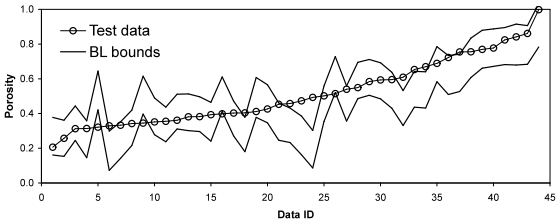
a) BP predictions.



b) SA predictions.



c) GA predictions.



d) BL predictions.

Figure 3. The min-max predictions for the test data set.

References

[1] D. Tamhane, P.M. Wong, F. Aminzadeh and M. Nikravesh, "Soft Computing for Intelligent Reservoir Characterization," SPE 59397, SPE Asia Pacific Conference on Integrated Modelling for Asset Management, Yokohama, 11 pp., 2000.

[2] C.M. Bishop, *Neural Network for Pattern Recognition*, Oxford University Press, NY, 1995.

[3] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, 220(4598), 671-680, 1983.

[4] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, MA, 1989.

[5] D. MacKay, "A Practical Bayesian Framework

for Backpropagation Networks," *Neural Computation*, 4(3), 448-472, 1992.

[6] S. Cho, S. Choi and P.M. Wong, "Data Selection based on Bayesian Error Bar," International Conference on Neural Information Processing, Perth, vol. 1, 418-422, 1999.

[7] D. Qu, A.G. Bruce, P.M. Wong, "Bayesian Neural Networks: A New Tool for Permeability Prediction," International Conference on Advances in Intelligent Systems: Theory and Applications, Canberra, 6 pp., 2000.

[8] P.M. Wong and S.A.R. Shibli, "Combining Multiple Seismic Attributes with Linguistic Reservoir Qualities for Scenario-Based Reservoir Modelling," SPE 64421, SPE Asia Pacific Oil and Gas Conference and Exhibition, Brisbane, 5 pp., 2000.

[9] Y. Niu, P.M. Wong and L. Chen, "Sequential Neural Simulation: A New Approach for Stochastic Reservoir Modelling," SPE 65123, SPE European Petroleum Conference, Paris, 6 pp., 2000.

[10] Y. Huang, *Soft Computing Models and Optimisation Algorithms for the Prediction of Petroleum Reservoir Properties*, PhD dissertation, University of New South Wales, Sydney, unpublished (1999).

[11] Neural Computing Research Group, Aston University, http://www.ncrg.aston.ac.uk/netlab/index.html.