Prediction of Reservoir Permeability Using Genetic Algorithms

Yuantu Huang[1,2], Patrick M. Wong[2],
and Tom D. Gedeon[1]
University of New South Wales
Sydney NSW 2052, Australia

Abstract. The determination of permeability is an example of many geological problems where laboratory-measured data is expensive and limited in quantity. We related permeability values to well logs. We used neural networks trained both with the popular backpropagation algorithm and with a genetic algorithm. The genetic training produced smaller errors and better generalization than backpropagation training on the same network topology. The cost includes greater average computation time as well as greater variation in computation time for the genetic training. The genetic training is robust and not sensitive to selection of the crossover and mutation parameters.

Many geological problems require data fitting and parameter estimation in multidimensional space. One example is the estimation of flow capability of sedimentary rocks, or permeability. In petroleum reservoirs, the magnitude of the permeability value directly affects hydrocarbon production and the definition of reserves. The determination of such a property is a complex problem because laboratory-measured permeability values on rock samples ("cores") are only available in limited and isolated well locations and/or intervals. Much effort is therefore required to relate permeability values to other measures, such as well logs (a series of multi-type digital measurements along the vertical depth of drilled wells), so that transformations can be developed to predict permeability in uncored wells and/or intervals where well logs are available.

Many methods have been proposed for permeability estimation in drilled wells, ranging from theoretical relations, such as the Carmen-Kozeny equation (Amaefule et al. 1993), to statistical methods, such as multiple linear regression (Jian et al. 1994), to empirical methods, such as backpropagation neural networks (Wong et al. 1995). One commonality among these methods is the existence of a (multidimensional) input vector and a (multidimensional) parameter vector, which is used in some fashion to produce a result. Mathematically, we can generalize these

[1]School of Computer Science and Engineering

[2]School of Petroleum Engineering

methods in the following form:

$$k = f(\vec{X}, \vec{\theta}) \tag{1}$$

where k denotes permeability (the dependent variable), f is the estimator, $\vec{X} = \{x_1, x_2, ..., x_n\}$ is an n-dimensional input vector (the independent variables), and $\vec{\theta} = \{\theta_1, \theta_2, ..., \theta_m\}$ is an m-dimensional parameter vector. Examples of the input and parameter vectors for the methods listed previously are shown in Table 1. All the methods aim to determine each of the elements in the parameter vector $\vec{\theta} = \{\theta_1, \theta_2, ..., \theta_m\}$.

A genetic algorithm (GA) is another example method that can be used to map well logs to permeability. It is an efficient global optimization method for solving ill-behaved, non-linear, and discontinuous problems in which exact physical relations do not exist. Other optimization methods, such as the backpropagation (gradient descent) algorithm used in supervised neural networks, are local in nature, adopting an iterative procedure using partial derivatives to improve on some initial model. These methods can lead to a dependence on the starting model and are prone to entrapment in local minima. Moreover, the calculation of derivatives can be difficult and further add to instability if numerical approximations are used.

GAs produce optimized solutions by mimicking the principle of survival of the fittest in the Darwinian theories of natural evolution on a computer (Davis 1991). This method uses a specific search strategy to look for an optimized parameter vector for a given estimator that maximizes a "fitness" function. Thus, GAs have a wide range of applications (Fang et al. 1992, Hallagher and Sambridge 1994, McKinney and Lin 1994, Moghaghegh et al. 1996, Wang and Elbuluk 1996) and can be used in conjunction with the methods listed above. For example, GAs can be employed to optimize the connection weights (which produce the parameter vector $\vec{\theta}$) within a given neural network architecture without the use of the backpropagation algorithm.

In this paper, we will first review the basics of GAs, then demonstrate how GAs can be used to optimize the parameters in a supervised neural network. We will then compare the performance of GAs and backpropagation neural networks in a case study in which well logs are used to predict reservoir permeability.

Table 1. *Comparison of various permeability (k) estimators.*

	Carmen-Kozeny	Multiple linear regression	Backpropagation neural networks
Characteristics	Theoretical; treat rocks as bundles of simple capillary tubes	Statistical; linear (can also include non-linear terms); parametric	Empirical; non-parametric; non-linear
Relation	$k = \frac{1}{F\tau^2 S^2} \frac{\phi^3}{(1-\phi)^2}$	$k = \sum_{i=1}^{n} a_i x_i$	$k = f\left(\sum_{j=0}^{n_2} b_j f\left(\sum_{i=0}^{n_1} a_{ij} x_i\right)\right)$
Input vector	1 dimension Porosity ϕ	n dimensions Well logs x_i	n_1 dimensions Well logs x_i
Parameter vector	m=3 dimensions (F, τ, S); Grain shape factor F Tortuosity τ Surface area S	m=n dimension; Weights a_i	m dimensions (see Equation 5); Weights a_{ij}, b_j
Method to obtain the parameters	Trial and error (rock-dependent)	Least-squares	Least-squares Learn by iterations

Genetic Algorithms

Background

Genetic algorithms (GAs) were first introduced in the field of artificial intelligence by Holland (1975). These methods mimic processes in the Dar-

winian theories of natural evolution in which winners survive to reproduce and pass along their "good" genes to the next generation, and ultimately, a "perfect" species is evolved. Hence the term "genetic" was adopted as the name of the mathematical algorithms.

The applications of GAs in geosciences were hindered in the early days by the unavailability of fast computers. The methodology became popular during the beginning of 1990s. Most applications have been limited to seismology. Gallagher and Sambridge (1994) presented an excellent review on this subject. Only a few papers in the literature apply the methodology in drilled wells. Fang et al. (1992) demonstrated the use of GAs to predict porosity-permeability from compositional and textural information, and the Archie parameters in petrophysics. The same authors later used the same method to map geochemical data into a rock's mineral composition (Fang et al. 1996). Mohaghegh et al. (1996) applied GAs to a hydraulic fracture treatment design in a gas storage field.

GAs can be used to solve both unconstrained and constrained large-scale optimization problems. They are stochastic search algorithms that perform particularly well in cases where the global minimum is hidden among many local minima. Unlike other optimization methods, the implementation of GAs is independent of the nature of both the forward problem and the form of objective function in that we avoid the need to calculate partial derivatives or perform matrix inversion.

Biological Representation

The basic idea of using GAs as an optimization method is to represent a population of possible or "parent" solutions in a chromosome-type encoding and manipulate these encoded solutions through simulated reproduction, crossover, and mutation. Reproduction involves a stochastic selection of two parents for mating and producing the next generation. Crossover allows genetic information exchange on each parent solution. Mutation allows a random introduction of new characteristics, which are unrelated to the parent solutions, to the offspring. This process is repeated for many generations. During the evolutionary process, the offspring with characteristics similar to the optimum or "perfect" offspring, as measured by some fitness function, would tend to survive, whereas the offspring less similar to the optimum offspring would die off, in a manner analogous to the Darwinian theories of the survival of the fittest in nature.

Computer Implementation

The following steps outline the general structure of genetic algorithms implemented in a computer:

Population Initialization

Generate a population of s parent solutions. Each parent solution contains a parameter vector $\bar{\theta}_j = \{\theta_1, \theta_2, ..., \theta_m\}_j$ that is initialized by random numbers. Each of the elements θ_i is commonly encoded with a certain length of binary numbers (or bit-string), or $\theta_i = \{b_1^i, b_2^i, ..., b_\ell^i\}$, $b_j^i \in \{0,1\}$, where ℓ is the length of the binary string (or bits). The binary numbers represent the "genetic" information of the solution. For example, a bit-string of (0,1,0,1), with ℓ equals 4 bits, represents a decimal value of 5. When putting all the elements in the parameter vector $\bar{\theta}_j$ one after another, each of the parent solutions will have a total bit-string of (m x ℓ) bits, or

$$\bar{\theta}_j = \{\{b_1^1, b_2^1, ..., b_\ell^1\}, \{b_1^2, b_2^2, ..., b_\ell^2\}, ..., \{b_1^m, b_2^m, ..., b_\ell^m\}\}_j.$$

Performance Evaluation

A "fitness" function is often used to evaluate the performance of each of the parent solutions. A fitness function to assess model fit may be defined as follows:

$$F(\bar{\theta}_j) = \frac{10}{1 + E(\bar{\theta}_j)} \tag{2}$$

and

$$E(\bar{\theta}_j) = \sum_i^{n_p} (k_i - k_{ij}^*)^2 \tag{3}$$

where $F(\bar{\theta}_j)$ is the fitness value for the parent solution j, j=1,...,s where s is the number of parent solutions in the population. $E(\bar{\theta}_j)$ is the sum of squared differences (i.e., errors) between the observed data k_i (n_p of them) and the model predictions k_{ij}^* de-

fined in Equation (1). The higher the $F(\bar{\theta}_j)$ value, or the lower the $E(\bar{\theta}_j)$, the better the solution.

Reproduction

Selecting a pair of parents for reproduction is a very important aspect of GAs. There are many methods of selection. A parent solution can be selected more than once. The most popular selection method is to choose the parent solutions with high fitness function values relative to the average fitness value, $\overline{F}$ (Wang and Elbuluk 1996):

$$\overline{F} = \frac{1}{s}\sum_{j}^{s} F(\bar{\theta}_j) \tag{4}$$

If any of the $F(\bar{\theta}_j)$ values is less than the average fitness value $\overline{F}$, it will be removed from the population. This is analogous to the extinction process of non-competitive species in the Darwinian theories of evolution. If the number of removed solutions is r, then the number of remaining solutions will be $(s\text{-}r)$. In order to keep the number of solutions in the original population constant, there is a need to reproduce r solutions. It is usually done by duplicating the remaining parents (Wang and Elbuluk 1996). In this paper, we duplicate the remaining solutions based on their $F(\bar{\theta}_j)$ values using a simple Monte Carlo sampling. The basic idea is to duplicate the "good" parent solution with a high probability. This is analogous to the survival of the fittest, but it still has a chance of being a non-dominant species.

Crossover

Crossover produces offspring by exchanging the genetic information between the selected parent solutions. The selection criteria are based on a user-defined probability for crossover, P_c (generally between 0.5 to 0.8). This probability defines the number of candidates for crossover. For example, if P_c is 50%, it means 50% of the s parents in the population will be selected randomly and mated in pairs.

Two-point crossover is a common method to implement the crossover mechanism (Lucasius and Kateman 1993, 1994). It works by randomly selecting two points along the $(m \times \ell)$ long bit-string. The binary numbers between the two selected points of the paired solutions are switched and this results in two new solutions.

Mutation

The reproduction and crossover would only exploit the known regions in the solution space, which could lead to premature convergence for the fitness function with the consequence of missing the global optimum by exploiting some local optimum. Mutation is a genetic process to avoid such a problem. This process allows the introduction of new characteristics—unrelated to the parent solutions—to the offspring. It first requires a user-defined probability for mutation P_m (generally between 0.001 to 0.01). A uniform random number (between 0 to 1) is then simulated on each of the $(m \times \ell)$ bits of each solution. If the simulated number for a bit location is less than P_m, mutation will take place in that location and its binary value will be flipped in parity (i.e., 0 becomes 1, and vice versa). Otherwise, the bit is unchanged.

Note that if P_m is 1, then all the bits will undergo mutation and the algorithm will perform similarly to the standard Monte Carlo. Moreover, if P_m is $1/(m \times \ell)$, then, on average, one bit per bit-string will undergo mutation.

Termination Criteria

This paper uses GAs to optimize the parameters in neural networks. The termination or stopping criteria for the iterative process is similar those commonly used in neural learning. In this work, we evaluate model performance based on the minimum error on validation patterns after finishing a preset maximum number of iterations. More details are given in the field example.

Field Example

Objective

The objective of this example is to compare the generalization capability of backpropagation and genetic algorithms in predicting permeability from data for oil wells located offshore in western Australia. We use both algorithms to train a neural network. Hereafter, we will use "BPNN" to denote the neural network trained by the backpropagation algorithm, and "GANN" to denote the neural network trained by the genetic algorithms.

Data Descriptions

Two oil wells, namely #1 and #2, in a lithologically complex reservoir were used to provide well logs and core permeability values. Well logs are digital measurements obtained every 150 mm or so of reservoir depth by lowering various equipment in the drilled wells. The well logs used for the analyses were gamma ray (GR), deep resistivity (LLD), sonic travel time (DT), bulk density (RHOB), and neutron porosity (NPHI). The groupings of the rock were also incorporated in the input data set as a discrete variable. There is a total of six independent variables (n_1=6). The values of the dependent variable, the core permeability measurements (k), were available at selected well depths.

In this study, Well #1 and Well #2 contain 152 and 156 points, respectively. All the input data were normalized in the range of (0,1). This is normally done in neural computation as the network will then give comparable magnitudes of weight values. All the permeability values were normalized in the range of (0.1,0.9). In order to keep consistency, the same normalized input and output values were used for both BPNN and GANN.

Neural Network

In this study, we used a simple three-layer feed-forward neural network. There were six input units and one output unit. The optimum number of hidden units was determined by cross-validation. All connections are from units in one layer to units in the next layer, with no lateral, backward, or recursive connections. Each unit is connected to each unit in the preceding layer by a simple weighted link with the basic sigmoid logistic activation function. Bias units are also included. The network was trained using a training set of input patterns with desired outputs. The network was tested using a validation set of patterns that were never seen by the network during training and thus provided a good measure of the generalization capabilities of the network.

The number of connection weights (or the dimensions of the parameter vector) depend on the number of hidden units used. Their relation is:

$$m = (n_1 + 1) \times n_2 + (n_2 + 1) \times n_3 \qquad (5)$$

where m is the dimension of the parameter vector, n_1, n_2, and n_3 are the number of input, hidden and output units, respectively. In this study, n_1 is 6 and n_3 is 1. Hence, $m = 41n_2 + 1$. The number of unknowns (weights) increases with the number of hidden neurons.

In this study, we first used BP to determine the optimum number of hidden neurons. We then applied GAs to the same neural network architecture and determined whether GAs performed better than BP.

Simulation Setup

In order to evaluate the generalization performance of the methods, the data from Well #1 were first used for training and data from Well #2 for validation. This was referred to as "Case 1." In order to reduce the simulation bias associated with the predictions, we then swapped the data for training and validation. This was referred to as "Case 2." The maximum number of iterations was set at 5,000.

The model parameters employed in the BPNN and GANN methods are shown in Table 2. Note that the BP is a gradient descent method which attempts to reduce the model error (here we used root mean-square-error, or RMSE) by iteration. It is generally characterised by two parameters: a learning rate and a momentum term. Their values are shown in Table 2. In GAs, we aimed to maximize the fitness function as defined in Equation (2).

Table 2. *The configurations of the algorithms.*

Backpropagation algorithm	
Learning rate	0.95
Momentum	0.50
Genetic algorithms	
Population size, s	50
Bit-string for each unknown, ℓ	32
Probability for crossover, P_c	0.6
Probability for mutation, P_m	0.003

Results

To determine the optimum number of hidden neurons, we ran a sensitivity analysis using two to nine hidden neurons. This was done on both cases: Case 1 using Well#1 for training and Well #2 for validation; Case 2 using Well #2 for training and Well #1 for validation. In both cases, the performance was evaluated at the minimum RMSE on the validation data after finishing the 5,000 iterations. This is a common method to avoid over-fitting. For each hidden size, 10 runs were performed using different sets of initial weights. The results are displayed in Figure 1. The diagrams show the minimum, average or mean, and maximum RMSE values (see the legend bar in the plot) for each hidden size. The best numbers of hidden neurons, on average, were five in Case 1 and two in Case 2. The "absolute" best results (with minimum errors) were with a hidden size of three in both cases. These errors were about 0.087 and 0.081, respectively. These two values served as a basis for comparing the GANN performance.

In GANN, we ran analyses on the optimum (3) hidden neurons. Ten runs were performed on each case using different initial populations. The results are shown in Figure 2, together with the best BPNN results. Figure 2a shows that the minimum and average errors from GANN were both less than the minimum errors from BPNN for both cases. In Case 2, even the maximum error from GANN was less than the minimum error from BPNN. Comparing their average errors, GANN is about 4.4% and 7.5% better than BPNN in cases 1 and 2, respectively. Figure 2b displays the statistics of the number of iterations required to produce a generalized solution. It shows that GANN took much more computational time to generalize on average compared to BPNN. Their respective values are tabulated in Table 3.

In this study, GAs produced a smaller error compared to BPNN. We have also examined the sensitivity of the choice of the crossover and mutation probabilities in GAs. The results showed that GAs for this problem were not sensitive to the selection of these parameters. The important model parameters in GAs are the values of the (randomly generated) initial genes in the population that affects the performance, as we have shown in this study. From these results, we can conclude that GAs are more robust than BPNN, as they avoid local optima.

It is important to note that the speed of convergence in GAs is slow compared to BPNN iterations. In this exercise, a single run of GANN was about 100 times (in terms of CPU time) slower than that of BPNN. Moreover, GANN also took more iterations to generalize (Figure 2b). It therefore illustrates a trade-off between accuracy and time for practical applications. This is relevant, especially if the simulation takes a lot of CPU time.

Table 3. *Comparison of BPNN and GANN using three hidden neurons.*

	Method	RMSE Min	RMSE Mean	RMSE Max
Case 1	BPNN	0.0871	0.0894	0.0935
	GANN	0.0823	0.0855	0.0889
Case 2	BPNN	0.0811	0.0857	0.0891
	GANN	0.0771	0.0792	0.0805
		No. of iterations at min RMSE		
Case 1	BPNN	413	596	1108
	GANN	143	273	4927
Case 2	BPNN	29	303	809
	GANN	672	2576	4642

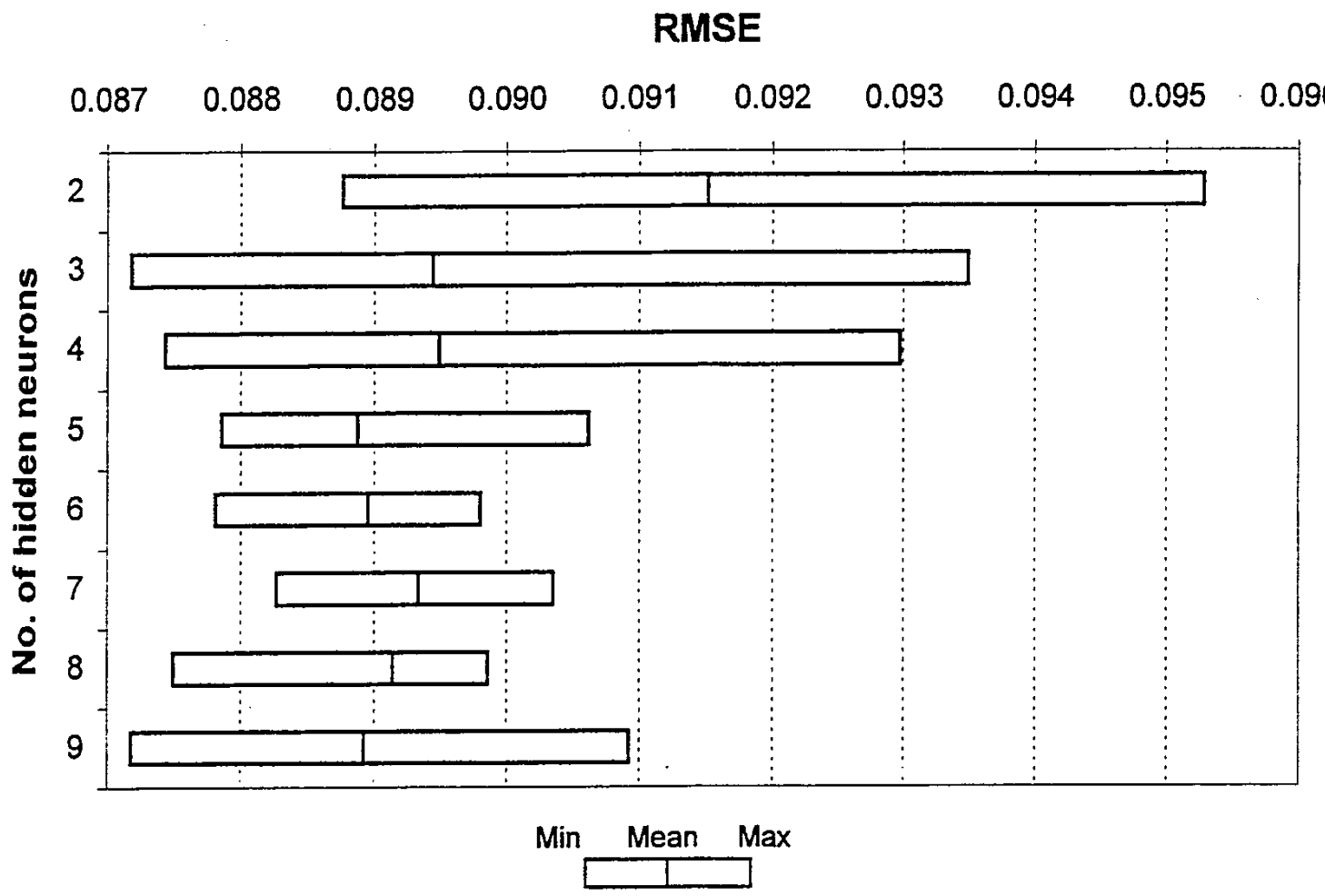
Conclusions

Genetic algorithms (GAs) are useful in the searching global minimum in a large and complex space. GAs can be used to predict permeability from well logs, providing their function relation (or estimator) is given, such as the use of neural networks. This methodology optimizes the parameters used in the estimator.

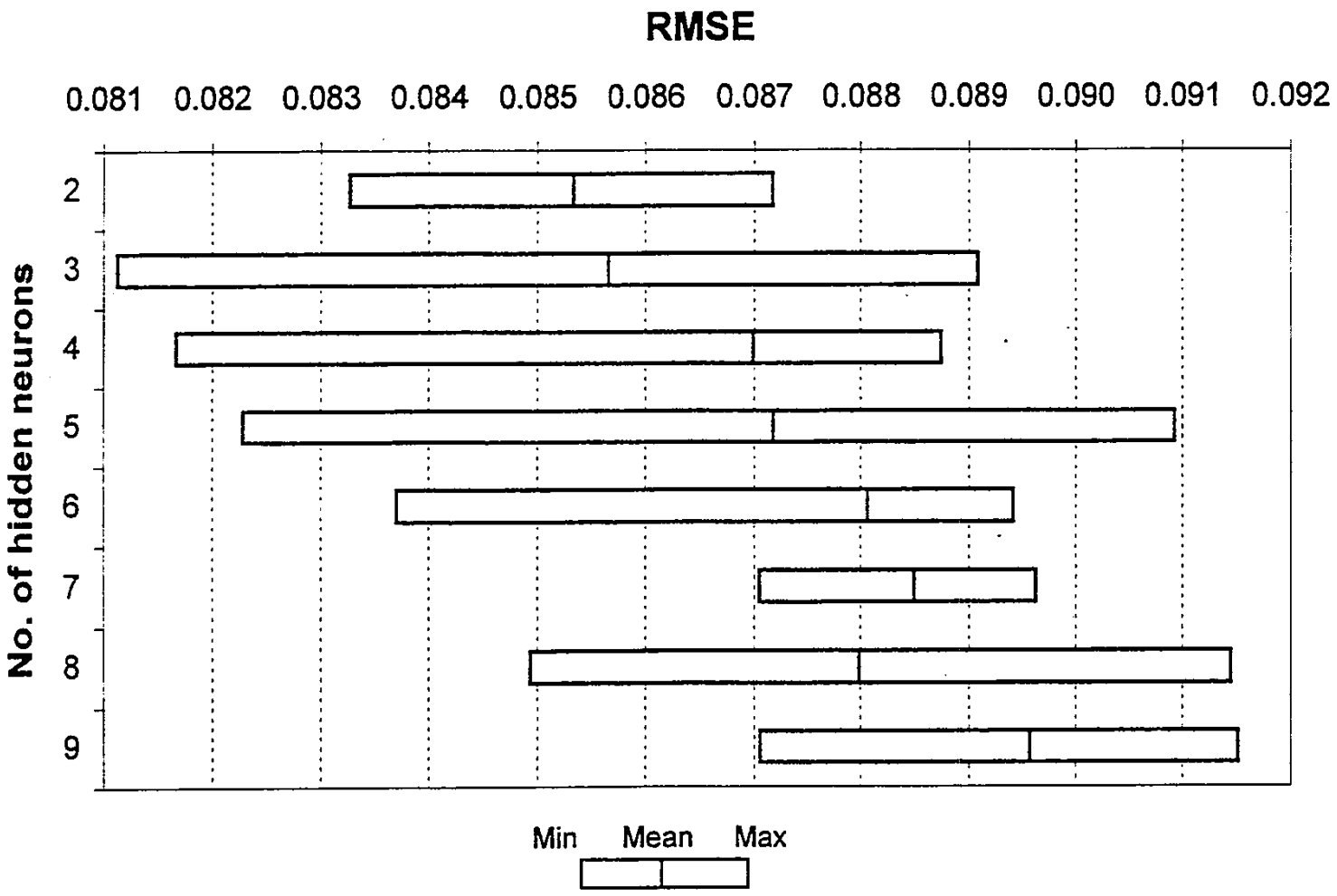
In this study, the average errors obtained from GAs were smaller than those obtained from the backpropagation algorithm. In one case, the worst performance of GAs was still better than the best result obtained from backpropagation.

GAs are slow in convergence compared to the backpropagation learning. Optimizing its speed of convergence is currently a challenging research area.

Further work will concentrate on: 1) the integration of fuzzy logic with genetic algorithms in order to handle uncertain and vague data; and 2)

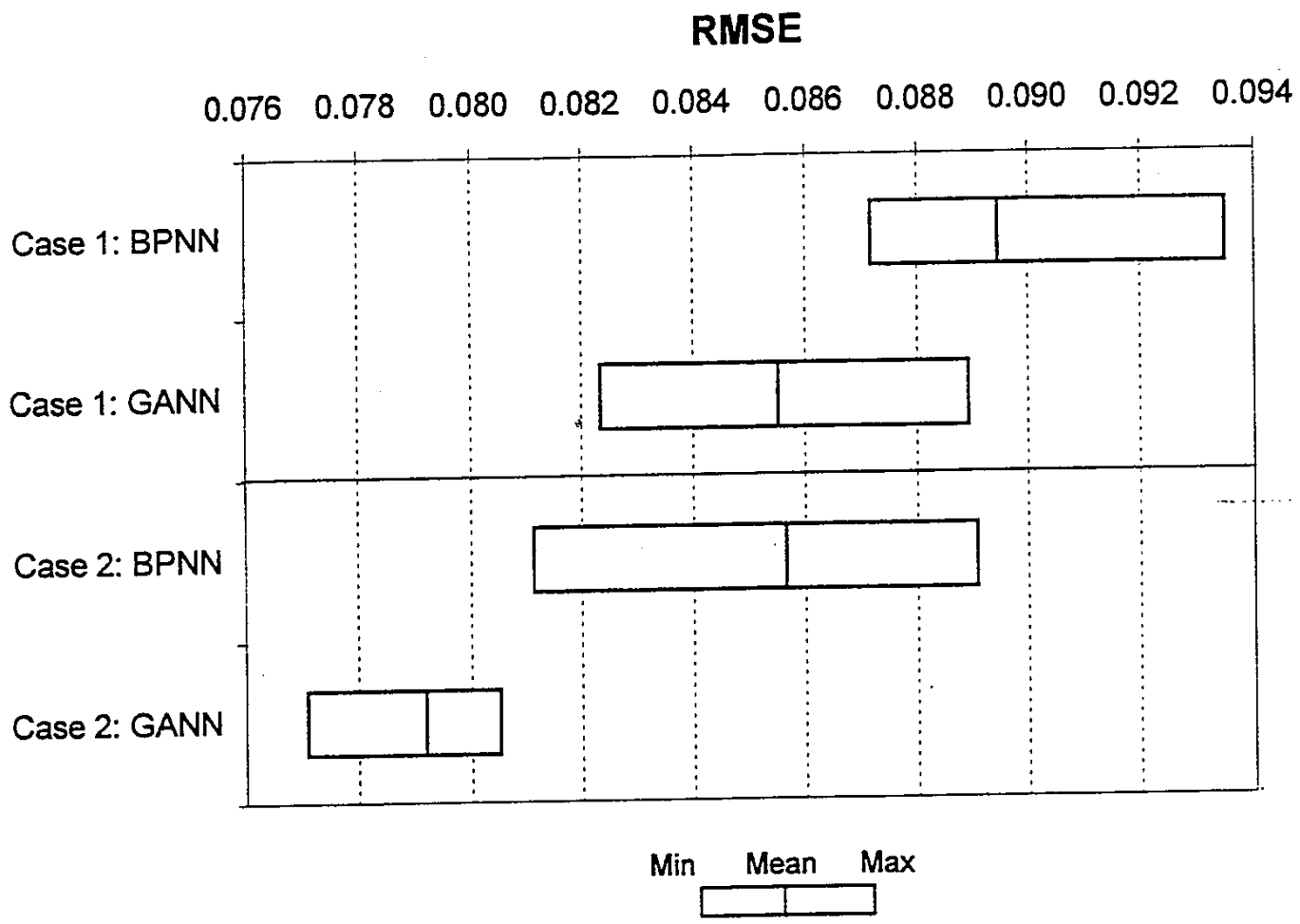


a. Case 1: Well #1 for training and Well #2 for validation.

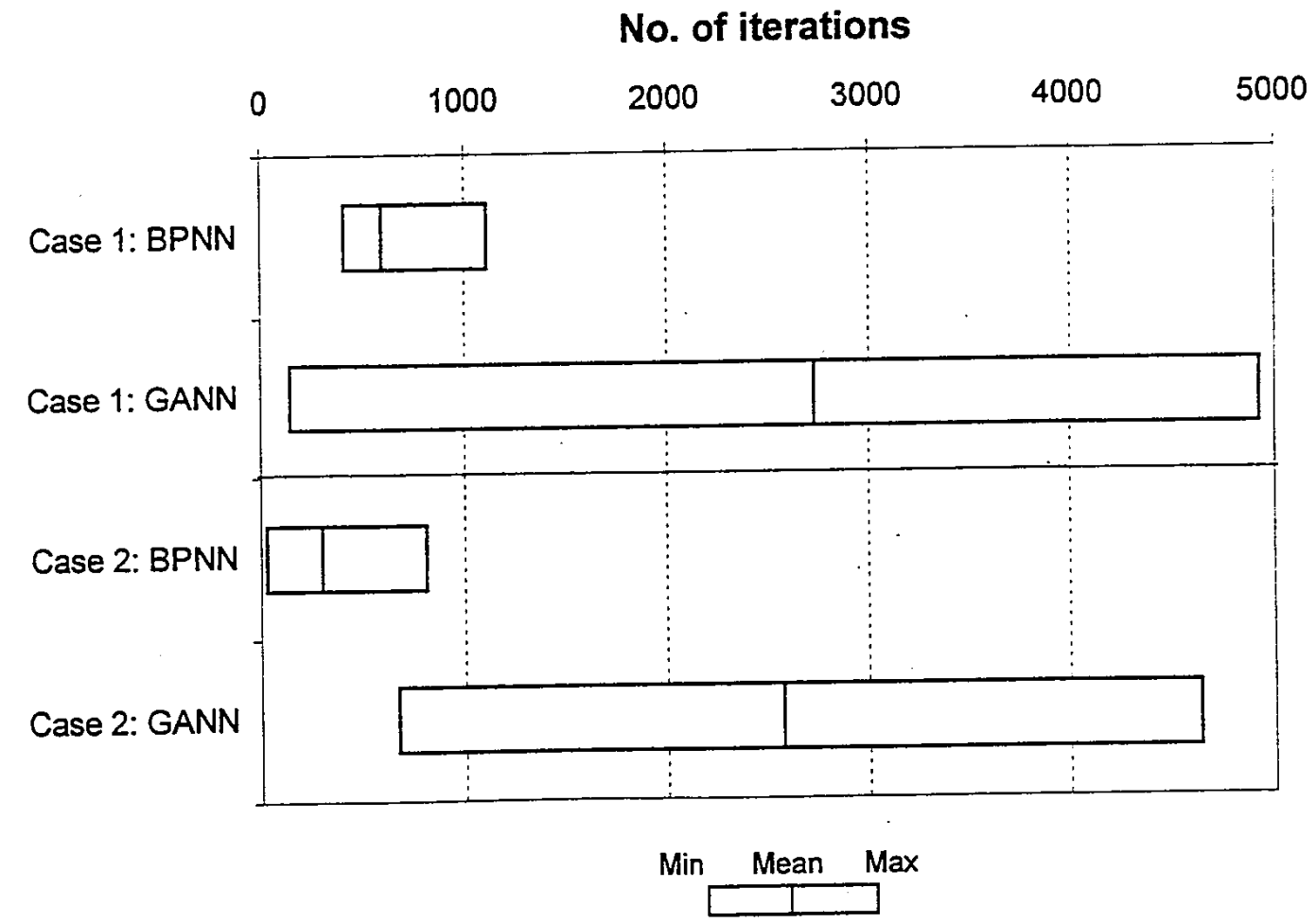


b. Case 2: Well #2 for training and Well #1 for validation.

Figure 1. *Results from backpropagation neural networks for various numbers of hidden neurons.*



a. Distribution of root-mean-equare errors (RMSE).



b. Distribution of the number of iterations at minimum RMSE.

Figure 2. *Comparison of BPNN and GANN using three hidden neurons.*

the use of trained weights from backpropagation networks to initialize GAs.

References

Amaefule, J.O., M. Altunbay, D. Tiab, D.G. Kersey, and D.K. Keelan. 1993. Enhanced Reservoir Description: Using Core and Log Data to Identify Hydraulic (Flow) Units and Predict Permeability in Uncored Intervals/Wells. SPE 26436. SPE Annual Technical Conference and Exhibition, Houston, 15 pages.

Davis, L. 1991. Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York.

Fang, J.H., C.L. Karr, and D.A. Stanley. 1992. Genetic algorithm and application to petrophysics. SPE 26208. Society of Petroleum Engineers, unsolicited paper.

Fang, J.H., C.L. Karr, and D.A. Stanley. 1996. Transformation of geochemical log data to mineralogy using genetic algorithms. The Log Analyst 37(2): 26-31.

Gallagher, K., and M. Sambridge. 1994. Genetic algorithms: A powerful tool for large-scale nonlinear optimisation. Computers & Geosciences 20(7/8): 1229-1236.

Holland, J. 1975. Adaptation in Natural and Artificial Systems. University of Michican Press, Ann Arbor.

Jian, F.X., C.Y. Chork, I.J. Taggart, D.M. McKay, and R.M. Barlett. 1994. A genetic approach to the prediction of petrophysical properties. Journal of Petroleum Geology 17: 71-88.

Lucasius, C.B., and G. Kateman. 1993. Understanding and Using Genetic Algorithms: Part 1. Concepts, Properties and Context. Chemometrics and Intelligent Laboratory System 19: 1-31.

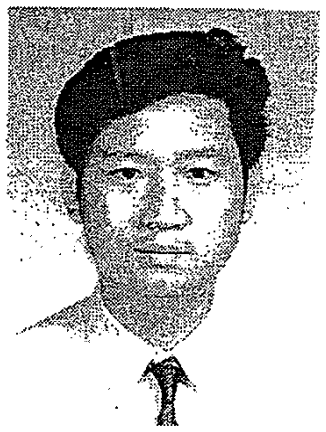
Lucasius, C.B., and G. Kateman. 1994. Understanding and Using Genetic Algorithms: Part 2. Representation, Configuration and Hybridisation. Chemometrics and Intelligent Laboratory System 19: 99-145.

McKinney, D.L., and M.S. Lin. 1994. Genetic algorithm solution of groundwater management models. Water Resources Research 30: 1897-1906.

Mohaghegh, S., B. Balan, S. Ameri, and D.S. McVey. 1996. A Hybrid, Neuro-Genetic Approach to Hydraulic Treatment Design and Optimization. SPE 36602, SPE Annual Technical Conference and Exhibition, Denver, 6 pages.

Wang, X., and M. Elbuluk. 1996. Neural network control of induction machines using genetic algorithm training. 31st IAS Annual Meeting, IEEE, New York Volume 3: 1733-1740.

Wong, P.M., I.J. Taggart, and T.D. Gedeon. 1995. Use of neural network methods to predict porosity and permeability of a petroleum reservoir. AI Applications 9(2): 27-38.



Yuantu Huang received a B.Sc. in mathematics in 1982 from the Zhejiang Normal University in China. Late, in 1989, he received an M.Sc. in mathematical geology and computer applications in geology from the Research Institute of Petroleum Exploration and Development in China. He is currently a Ph.D. student at the School of Computer Science and Engineering and the School of Petroleum Engineering at the University of New South Wales. His research interests are in the integration of neural networks, fuzzy logic, and genetic algorithms in petroleum engineering.

Patrick M. Wong received a B.E. (Hons.) and a Ph.D. in petroleum engineering in 1993 and 1996, respectively, from the University of New South Wales. He is currently a Lecturer in Petroleum Engineering at the same university. His research interests are in the area of petrophysics, geostatistics, and AI applications in reservoir characterization. He was formerly with Petroconsultants Australasia Pty. Ltd. in Sydney.

Tom D. Gedeon received a B.Sc. (Hons.) and a Ph.D. in computer science in 1981 and 1989, respectively, from the University of Western Australia, He is currently Head of Department of Information Engineering in the School of Computer Science and Engineering at the University of New South Wales and a Visiting Professor in the Department of Telecommunications and Telematics, Technical University of Budapest, Hungary. His research interests are in information retrieval and index generation, extracting knowledge (data mining) from trained neural networks, and AI applications.