Tested Neural Network architectures

In the previous reports it was established that neural network is the most efficient machine learning method for conducting bead structure recognition as well as regression analysis in the current experimental setup. A series of networks with different architectures were trained (Table 1) on the 8 selected calibration images, of a homogeneous aquifer with 1090μm diameter. Multilayer networks 7-9, generate an acceptable amount of error, comparable to or smaller than the error generated using pixel-wise regression method (proposed by G.Robinson).

Table 1: The different neural networks trained on the selected calibration images.

Model	Method	H.Layers	Neurons/Layer	Weights & Biases	Data	MSE	RMSE
1	NeurNet	1	10	51	original	76.763	8.759
2	NeurNet	1	15	76	original	73.6159	8.58
3	NeurNet	1	20	101	original	71.8043	8.4737
4	NeurNet	1	100	501	original	60.722	7.7924
5	NeurNet	1	10	51	homogenized	17.3876	4.1698
6	NeurNet	1	100	501	homogenized	10.5341	3.2456
7	NeurNet	2	10	161	homogenized	7.7209	2.7787
8	NeurNet	3	10	271	homogenized	7.139	2.671
9	NeurNet	5	10	491	homogenized	6.736101	2.5954
10	Pixel-wise					7.2488	2.6924

The size of a feedforward or multi-layer perceptron, neural network is characterized by its weights and biases. Weights in an ANN are the most important factor in converting an input to impact the output. Weights are numerical parameters which determine how strongly each of the neurons affects the other. Bias is an additional parameter which is used to adjust the output along with the weighted sum of the inputs to the neuron. The processing done by a neuron is thus denoted as:

$$output = \sum(weights * inputs) + biases \ (1)$$

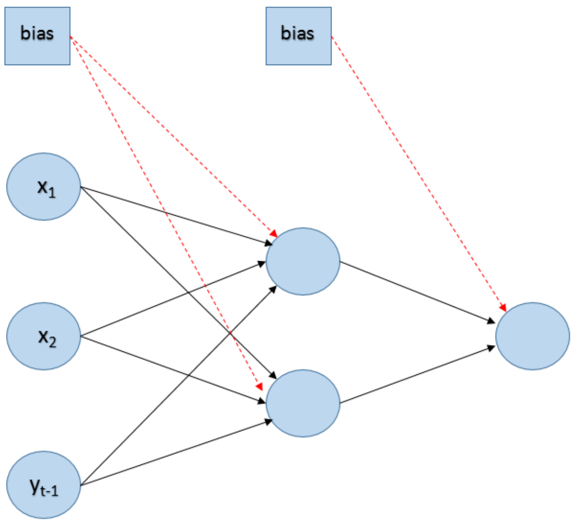

Figure 1: Representation of weights and biases in an artificial neural network.

It is safe to assume that a bigger neural network, with more weights and biases, can better approximate a complex function than a smaller one. On the other hand, the bigger the network, the higher the danger of overfitting. This minimizes network's ability for generalization. Increased neural network size greatly affects the demand in available computational power and elongates the training time needed as well. As seen in table 1, the number of hidden layers greatly affected theetwork's performance (MSE). Model 6 and model 9 have a similar number of connections between their neurons, 501 and 491 respectively, but the performance of the two networks significantly differs. Since the optimal ANN architecture depends on the available training data sets, it is of the outmost importance to determine the amount of hidden layers and neurons in each layer that best suit the regression analysis of the datasets in our disposal.

Different approaches in determining optimum ANN architecture

The most widely used method of determining an acceptable ANN architecture is through trial and error, based on the experience of each user. Since trial and error is neither the most time efficient or reliable method of optimization, a plethora of different methods have been proposed. The different approaches were categorized by Bernados and Vosniakos (2007) as follows:

(i) Empirical or statistical methods that are used to study the effect of an ANN's internal parameters and choose appropriate values for them based on the model's performance (Khaw et al., 1995; Maier and Dandy, 1998a, b; Benardos and Vosniakos, 2002). The most systematic and general of these methods utilizes the principles from Taguchi's design of experiments (Ross, 1996). The best combination of number of hidden layers, number of hidden neurons, choice of input factors, training algorithm parameters etc. can be identified with these methods even though they are mostly case-oriented.

(ii) Hybrid methods such as fuzzy inference (Leski and Czogala, 1999) where the ANN can be interpreted as an adaptive fuzzy system or it can operate on fuzzy instead of real numbers.

(iii) Constructive and/or pruning algorithms that, respectively, add and/or remove neurons from an initial architecture using a previously specified criterion to indicate how ANN performance is affected by the changes (Fahlman and Lebiere, 1990; Rathbun et al., 1997; Balkin and Ord, 2000; Islam and Murase, 2001; Jiang and Wah, 2003; Ma and Khorasani, 2003). The basic rules are that neurons are added when training is slow or when the mean squared error is larger than a specified value, and that neurons are removed when a change in a neuron's value does not correspond to a change in the network's response or when the weight values that are associated with this neuron remain constant for a large number of training epochs. Since both constructive and pruning algorithms are basically gradient descent methods, their convergence to the global minimum is not guaranteed and so they can be trapped to a local minimum close to the point of the search space from which the algorithm started.

(iv) Evolutionary strategies that search over topology space by varying the number of hidden layers and hidden neurons through application of genetic operators and evaluation of the different architectures according to an objective function (Bebis et al., 1997; Liu and Yao, 1997; Yao and Liu, 1997, 1998; Castillo et al., 2000; Arifovic and Gencay, 2001).

Following an evolutionary approach of solving this optimization problem, a genetic algorithm (GA) was created to supplement the procedure of selecting a proper architecture. Each architecture was evaluated according to its performance (MSE), generalization ability (MSE of the testing subset) and the percentage of erroneous outliers in each prediction.

Optimizing ANN architecture using genetic algorithm

A genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. The genetic algorithm was created using Matlab's Global Optimization Toolbox, version 3.4.4. A typical genetic algorithm requires: a genetic representation of the solution domain, and a fitness function to evaluate the solution domain.

- Solution Domain

In this investigation only the amount of hidden layers and hidden neurons were taken into account. The training function (Levenberg-Marquardt) is the same for all the tested ANN designs. The same applies for the activation function of each layer, that is sigmoid for each hidden layer and a linear one for the output layer. Based on the experience acquired from the networks already trained the maximum allowed number of hidden layer were set to 3, while the maximum number of neurons to 20. Each possible architecture, or chromosome in the case of Gas, consisted of 4 variables, the number of hidden layers (1-3), the neurons of first layer (1-20), neurons of second layer, neurons of third layer. In order for the genetic procedures of Crossover and Mutation to take place, the chromosomes needed to be transformed into binary form. Thus the length of chromosomes in the current GA equals to 17 (Figure2).

Hidden layers	Neurons in 1st layers	Neurons in 2nd layer	Neurons in 3rd layer
1 1	0 1 1 1 1	1 0 1 0 0	0 0 0 1 0
3	15	20	2

Figure 2: Binary and decimal representation of an ANN architecture with 3 hidden layers and 15,20 and 2 neurons in each layer.

- Objective / Fitness function

(a) Training time limit

The fitness function determines the ability of the generated neural network to successfully conduct a regression analysis of the available calibration function. The objective function minimized with this specific genetic algorithm included, the training of a multilayered feedforward neural network (hidden layers >=1). In order to make the whole optimization procedure faster an upper limit of 180 secs (3 min) were set for the training of each neural network. Another positive aspect of the applied time limit, is that it can allow for a small network (which is fast to train) to be trained for a sufficient number of epochs in order to learn the data associations and at the same time prevent a very complex network (which is slow to train) from overfitting the data.

(b) Performance

During the training of each ANN, the dataset is divided into 3 sub-sets, training, validation and testing. Training subset is used to determine the values of weights, validation to terminate the training procedure if there is evidence of overfitting, through early stopping. The testing sub-set, does not take part in the training can be used to compare different networks since it expresses the generalization ability of the ANN. The objective function calculates the performance, Mean Squared Error, generated for the whole data set $perf$, and the performance of the testing sub-set in particular $testperf$. It is possible that a neural network with a better general or validation performance have a lower performance on the testing sub-set.

(c) Solution space consistency criterion

Despite the requirement for the ANN to make accurate predictions it is just as important for these predictions to be consistent throughout the solution space. It is possible that while an ANN model may exhibit a low mean generalization error, there may be specific cases where it fails to accurately predict the desired value, i.e. it is not consistent in its predictions. The solution space consistency criterion applied here is a variation of the one proposed by Bernados and Vosniakos (2007). Its purpose is to track the percentage of outliers in the generated flow fields and apply and apply a penalty to the solutions with the higher percentages.

For each input pixel, the absolute value of the relative error (2) between the target output and the ANN prediction is calculated. If the relative error is in the interval [0,15), the prediction is considered a ‘’good’’ one, if it is in the interval [15,25] it’s considered average while if the relative error is more than 25% the prediction is a “bad” one.

$$Rel.Error = \frac{|predicted.val - real.val|}{real.val} \; (2)$$

The percentage of “good”, “average” and “bad” predictions is calculated. A penalty of 33% and 100% is applied to the solution fitness value for the percentage of average and bad predictions respectively. The need for this criterion was indicated by the larger amount of bad predictions (pixels) in the sw concentration fields generated by ANN’s in contrast to the ones generated by pixel-wise regression, even though the total generated MSE was lower in the first case.

$$solspc = 1 + average\% * 0.33 + bad\% \; (3)$$

Incorporating the aforementioned criteria, the objective function, that the genetic algorithm minimizes, thus optimizing the architecture of feedforward neural networks, in our problem equals to:

$$ObjFun = perf \times testperf \times solsp \; (4)$$

Concluding the final ANN derived from this optimization procedure, should have a good performance in the whole calibration dataset, a good generalization ability and give consistent predictions, smaller networks will get a slight advantage over bigger ones.

-Genetic Procedures

The genetic procedures in the utilized GA, were kept as close to the default options of Matlab Global Optimization Toolbox as possible. After each chromosome of the initial population was evaluated according to the value of the objective function it generated, the algorithm created crossover children by combining pairs of parents in the current population. At each coordinate of the child vector, the default crossover function randomly selects an entry, or gene, at the same coordinate from one of the two parents and assigns it to the child. The algorithm usually selects individuals that have better fitness values as parents. The algorithm creates mutation children by randomly changing the genes of individual parents. The default selection option Stochastic uniform, was used. It lays out a line in which each parent corresponds to a section of the line of length proportional to its scaled value. The algorithm moves along the line in steps of equal size. At each step, the algorithm allocates a parent from the section it lands on. 80% of the new kids in the algorithm are selection kids. By default, for unconstrained problems, such as the current one, the algorithm adds a random vector from a Gaussian distribution to the parent. The best solution of the previous generation passes on to the next, being an "elite" child.

Genetic Algorithm application

The application of the genetic algorithm in order to optimize ANN architecture has the following difficulties. The objective function, includes the training of a feedforward neural network. Training of ANN's is a heuristic procedure, due to the random selection of initial weights for the network's connections. Training the exact same architecture can generate networks with different values of performance. This indicates that apart from the optimum ANN architecture, the actual neural network generating the minimum value of the objective function should be retained during each generation, and its fitting value used in subsequent genetic procedures.

Given a big enough population and sufficient generations, the genetic algorithm can find the global minimum of most functions. Its implementation in the current problem was chosen as an alternative to the time-consuming nature of the more commonly utilized method of trial and error. Nevertheless, the execution of GA's can be extremely time-consuming as well, especially with so complex objective functions such this in equation 4.

A combination between the two methods was deemed to be the appropriate solution of acquiring the optimum ANN architecture for the current calibration dataset. Data in Table 1 indicate that for 2-3 hidden layers and for about 10 neurons in each one of them, we get networks that generate MSE comparable to the one generated by pixel-wise wise regression. So using the $'InitialPopulationRange'$ option, the initial GA's population was created having between 2 to 3 hidden layers and 3 to 5 neurons in each layer. This initial range is close enough to the area of generally good solution, thus enabling the algorithm to reach it faster, but also enables the user to check the ability of the algorithm in attaining an acceptable solution. This range was applied only to the initial population, while chromosomes of subsequent generations were able to express different architectures, inside the solution domain presented in page 3 (hidden layers 1-3, neurons of first layer 1-20).

Results

The algorithm was executed between the 10th and 11th of September, and the execution lasted approximately 11 hours. In total 30×10=300 neural networks were trained. The optimal architecture acquired is that of a feedforward network with 3 layers, 9 neurons in the first, 8 neurons in the second and 6 neurons in the third layer. The total error generated by the net9N8N6N.mat equals to MSE=7.6702, RMSE=2.7695, slightly bigger than that generated by pixel-wise regression, RMSE=2.6924. 16.56% of the networks prediction are considered bad, while only 2.19% are considered average, the rest 81.25% are considered good, thus have relative error less than 15%. The generated sw concentration fields for c=0%, 50% and 100% are presented in figures 3-5. Finally, the saltwater concentration field, for the case of dH=$2mm$ in the homogenous, 1090μm, aquifer is depicted in figure 6.

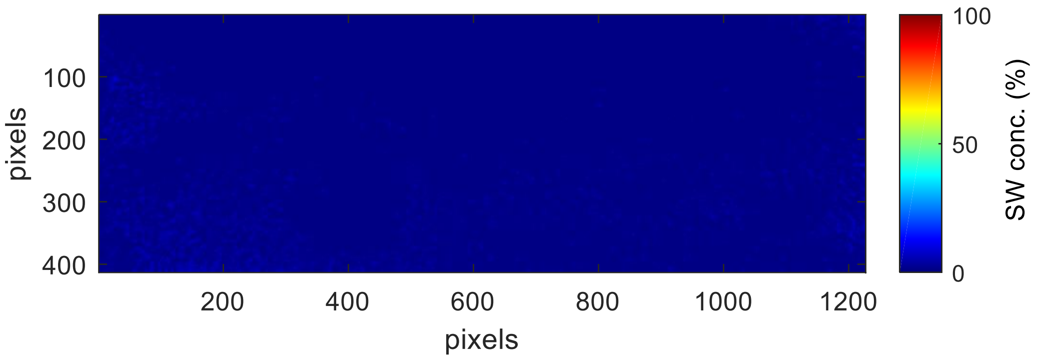

Figure 3: Concentration color maps of calibration images at 0% using optimum Neural Network trained with homogenized data.

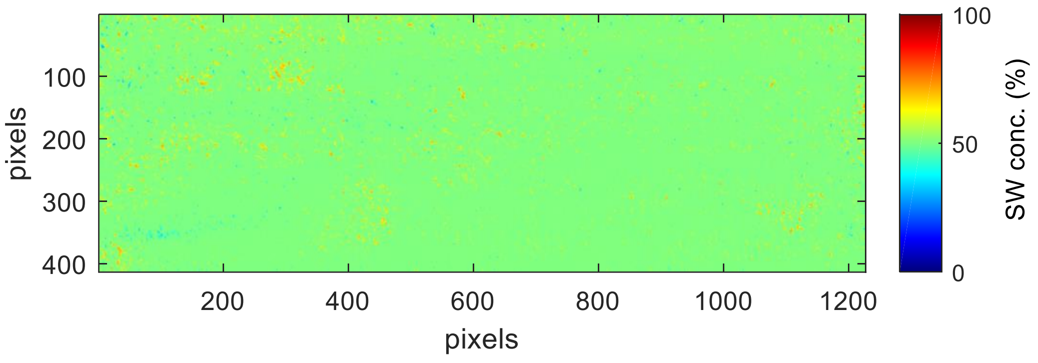

Figure 4: Concentration color maps of calibration images at 50% using optimum Neural Network trained with homogenized data.

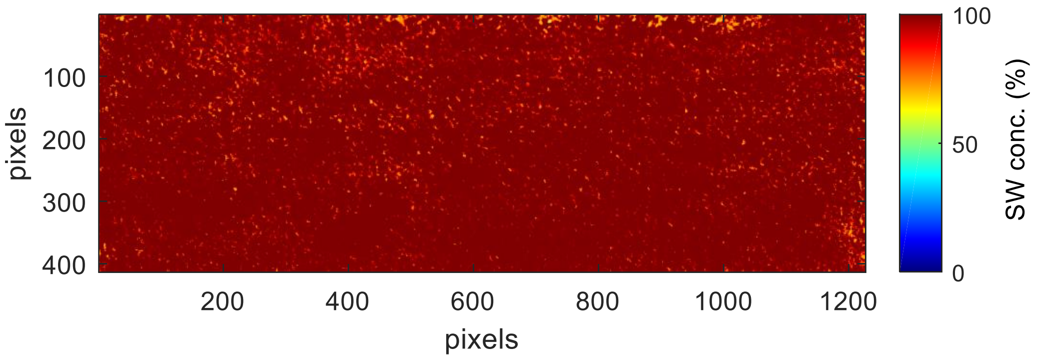

Figure 5: Concentration color maps of calibration images at 100% using optimum Neural Network trained with homogenized data.

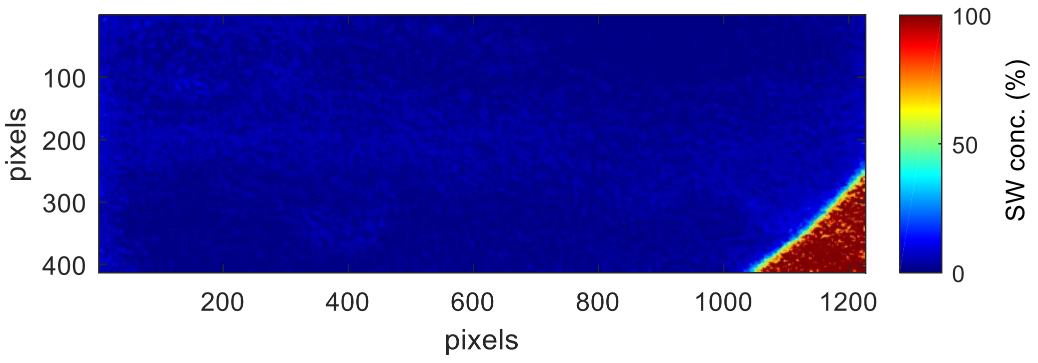

Figure 6: Saltwater concentration field for the $dH = 2mm$ in 1090μm case using optimum Neural Network trained with homogenized data.

The default diagram of average and best solutions per generation were plotted in order to check the optimization procedure. Unfortunately, the presence of an unfitted architecture among the chromosomes of the 7th generation, probably introduced through the mutation procedure, gave an extremely big value of the mean penalty value (objective value), making reading the graph impossible (Figure 7). A similar plot of previous executions of the GA, for different versions of the objective function as well as initial population and solution domain is presented in figure 8. Possible ways of minimizing the impact of such outliers on the GA training plots, would be to plot the y axis (Fitness value) in logarithmic scale, exclude the worst value from the calculation of the average fitness value or even plot the optimum values alone.

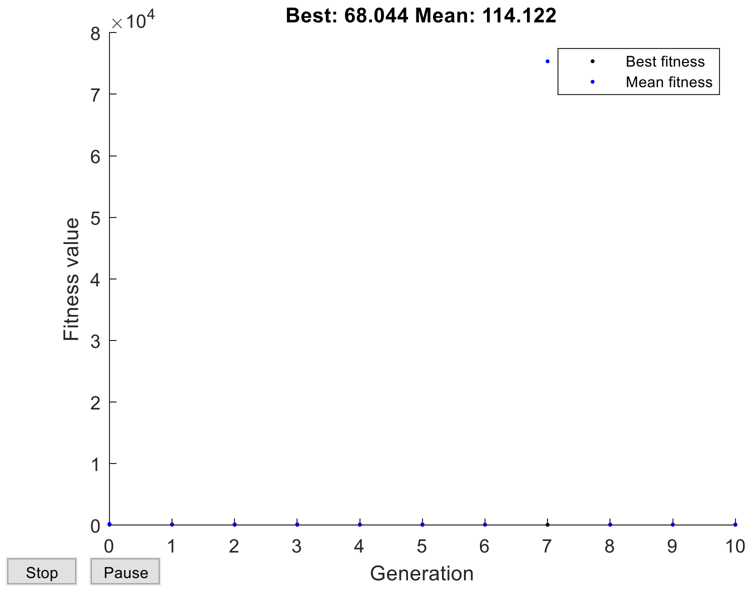

Figure 7: Average and best value of the objective function per generation.

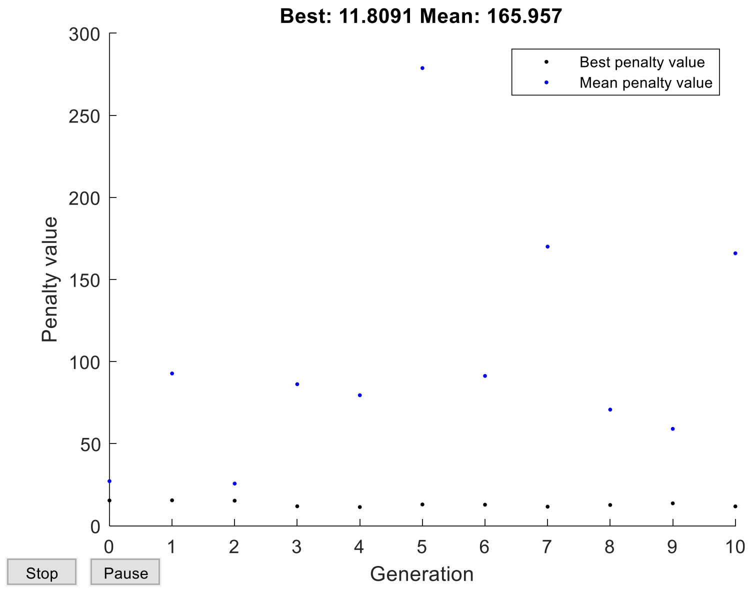

Figure 8: Average and best value of the objective function per generation. Generations=10, initial population = 10. Objective function= performance of ANN.

We are unable to present the 4 plots representing the training procedure of optimum network, performance, training state, regression plot and error histogram, since their data cannot be retrieved after the completion of the ANN training. The correct commands have since been incorporated into the objective function script providing the user with the aforementioned plots.

Conclusions

-The genetic algorithm works correctly. Even with a small population (20) and a very limited number of generations, just 10, it was able to provide a network that accurately generates the sw concentration fields of the calibration image and successfully depicts the saltwater wedge in the same aquifer.

-The use of the genetic algorithm in a predetermined space of relatively good ANN designs is essential in order to achieve a good design in a time-efficient way.

-The Objective function can and should be modified according to the user's need for each specific case. Possible changes could include:

1) Changing the maximum training time. 3 minutes might not be enough time for neural networks with more than 100 connections to achieve an acceptable value. Changes in the number of chromosomes and available generations could also be beneficial.

2) Modifying the Solscc criterion. Both the limits defining bad and average predications can be modified as well as the penalty applied for each category.

3) The final value of the objective function itself could take different forms such as those in equations 5-7.

$$ObjFun = (perf + testperf) \times solsp (5)$$

$$ObjFun = (perf \times testperf) \times e^{solsp} (6)$$

$$ObjFun = (perf + testperf) \times e^{solsp} (7)$$

4) Selecting different approaches for the genetic procedures such as tournament or roulette wheel for the selection, or antimetathesis instead of mutation.

- Using a metaheuristic procedure, GA, to optimise a heuristic procedure, such as the training of ANN's would definitely demand multiple executions in order to achieve an optimum or really good feedforward neural network.

-Even though the procedure might be time consuming it does not rely on the user's previous knowledge of NN, architecture.

Future Investigations (Until October 1st 2018)

See report 5.

References

Bebis, G., Georgiopoulos, M., Kasparis, T., 1997. Coupling weight elimination with genetic algorithms to reduce network size and preserve generalization. Neurocomputing 17, 167–194.

Benardos, P., & Vosniakos, G. (2007). Optimizing feedforward artificial neural network architecture. *Engineering Applications of Artificial Intelligence, 20*(3), 365-382.

Benardos, P.G., Vosniakos, G.-C., 2002. Prediction of surface roughness in CNC face milling using neural networks and Taguchi's design of experiments. Robotics and Computer Integrated Manufacturing 18, 343–354.

Fahlman, S.E., Lebiere, C., 1990. The Cascade-Correlation Learning Architecture. Advances in Neural Information Systems 2. Morgan-Kaufmann, Los Altos, CA.

Khaw, J.F.C., Lim, B.S., Lim, L.E.N., 1995. Optimal design of neural networks using the Taguchi method. Neurocomputing 7, 225–245.

Leski, J., Czogala, E., 1999. A new artificial network based fuzzy interference system with moving consequents in if-then rules and selected applications. Fuzzy Sets and Systems 108, 289–297.

Maier, H.R., Dandy, G.C., 1998a. The effect of internal parameters and geometry on the performance of back-propagation neural networks: an empirical study. Environmental Modelling & Software 13, 193–209.

Maier, H.R., Dandy, G.C., 1998b. Understanding the behavior and optimising the performance of back-propagation neural networks: an empirical study. Environmental Modelling & Software 13, 179–191.

Ross, J.P., 1996. Taguchi Techniques for Quality Engineering. McGraw-Hill, New York.