# Enhancement of speed and efficiency of an Internet-based gear design optimisation

Nariman Amin and Daizhong Su

School of Engineering, The Nottingham Trent University, Burton Street, Nottingham, NG1 4BU, UK. nariman.amin@ntu.ac.uk and daizhong.su@ntu.ac.uk*

Tel: (+44) 115 8482304 and (+44) 115 848 2306

* Corresponding author

**Abstract:** An Internet-based gear design optimisation program has been developed for geographically dispersed teams to collaborate over the Internet. The optimisation program implements genetic algorithm. A novel methodology is presented that improves the speed of execution of the optimisation program by integrating artificial neural networks into the system. The paper also proposes a method that allows an improvement to the performance of the backpropagation learning algorithm. This is done by re-scaling the output data patterns to lie slightly below and above the two extreme values of the full range neural activation function. Experimental tests show the reduction of execution time by approximately 50%, as well as an improvement in the training and generalisation erros and the rate of learning of the network.

**Keywords**: Artificial Intelligent, Artificial Neural Networks, Genetic Algorithm, Gear design, Design optimization.

**Reference** to this paper should be made as follows: Amin, N. and Su, D. (2002), 'Enhancement of speed and efficiency of an Internet-based gear design optimization', Int. J. Automotive Technology and Management, Vol. X, No. X, pp. X-X.

**Biographical Note:** Ms Nariman Amin is a researcher in the School of Engineering, The Nottingham Trent University, UK. She studied BSc(Hons) Computing Systems in 1996 and MSc Engineering Multimedia in 1998 at the above university. She received PhD degree in 2002. The area of her research is intelligent optimum design with the support of Internet technique.

Daizhong Su is Professor of Design Engineering in School of Engineering, The Nottingham Trent University, UK. He chairs the Engineering Design and CAE subject group and leads the Mechanical Transmission and Concurrent Engineering research team. His research interests include artificial intelligence, evolutionary optimisation, CAD/CAM/CAE, integrated design & manufacture and Web based engineering with about 130 refereed publications.

## 1. Introduction

Gears are used in a wide range of engineering design to transmit power from one shaft to another. In most cases, the design of gears is a highly complicated task involving the satisfaction of a number of design constraints. There are various manufacturing considerations. Neglecting any of these could result in the failure of the design, therefore the design assessment requires many compromises [1]. Several approaches for gear design have been proposed. Among those, the use of optimisation techniques has received much attention, for example [1, 2, 3]. Optimisation techniques usually require the minimisation of an objective function that is usually a combination of the various parameter [3]. If there are many design parameters in the objective function, it is difficult for the designer to assess the importance of each one. This scenario exists in gear design. Slight changes in the objective function of gears would result in an entirely different design.

A gear optimisation program has been developed in the Department of Mechanical and Manufacturing Engineering at The Nottingham Trent University [4]. The software utilizes Genetic Algorithm(GA) to find the optimum design solution for spur and helical gears. The results of the genetic algorithm optimisation are displayed to the user in tabular as well as graphical format.

A methodology was developed to implement this design optimisation software package over the Internet [5,6]. This is very beneficial for geographically dispersed teams to collaborate over the Internet for the purpose of integration in design and manufacture. A combination of HTML, CGI, JavaScript and Java programming is used. The required data is obtained from the client, which are then sent to the server and the design optimisation is invoked on the server. When the execution is completed, the results are sent back to the client. The system also takes into consideration the problem of a multi-user environment. Refer to the given references, for the full description of the system.

This paper consists of two sections. In the first section, a method is proposed that improves the speed of the optimisation program. It is based on a combination of Artificial Neural Networks (ANN) and genetic algorithm. In the second section, a method is described which improves the performance of the backpropagation learning algorithm for the ANN. The method shows an improvement in the training and generalisation errors as well as the rate of learning of the network. To have a better understanding of the system, a brief description of the structure of the gear optimisation software is first given.

## 2. Characteristics of the original optimisation program

The optimisation process developed by Su et al [4] is used to optimise the design of external spur and helical gears with involute tooth profile. Up to nine gear design parameters can be optimised, including tooth facewidth, module, pressure angle, helix angle, rack tip radius, addendum coefficients, addendum modification (tooth profile shift) coefficients for pinion and wheel, and number of pinion teeth. The basic configuration of gear design is provided by the user, which includes geometry, performance and material information. The user also needs to submit the settings for genetic algorithm, such as population size, number of tests, fitness parameters, etc. Once the user has entered all the required parameters, the optimisation process can then proceed.

The software utilizes genetic algorithm to perform the search for the design configuration that will give the maximum performance for spur and helical gears. The genetic algorithm conducts an adaptive search of various configurations of gear design, derived from an initial rough design that must be supplied by the user. The numerical analysis program is invoked by the genetic algorithm to calculate the tooth strength. The results of the genetic algorithm optimisation are displayed to the user, giving both the current performance and relative performance to the initial design.

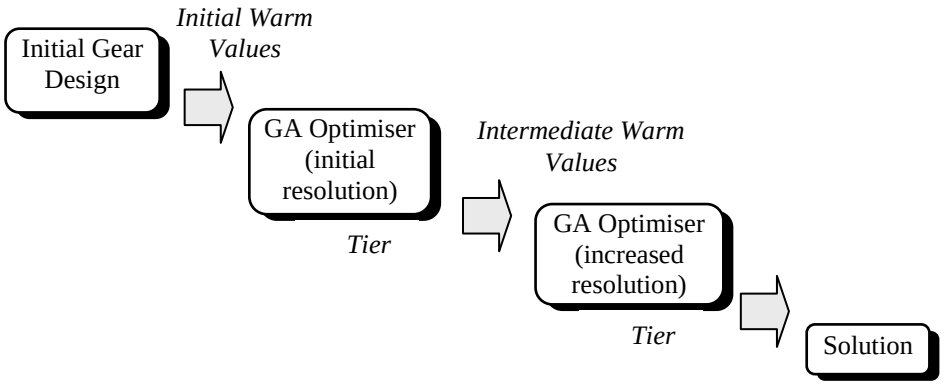


Figure 1. The cascade procedure of GA optimisation

## 2.1 The structure of the original optimisation program

The genetic algorithm optimisation process is applied in a cascade fashion. The procedure comprises of two tiers, see figure 1. The cascade procedure requires an initial starting design to base the optimisation process upon. The initial values of the design form the starting positions and limiting conditions for the parameters that are to be optimised. The first tier of the optimisation is invoked using the initial design or warm values, provided by the user, to adjust the parameters in search of a global optimum. The optimisation process continues until a limiting percentage of the genome population are identical. At this point the information encoded within the converged genome is decoded forming the solution to this tier and the intermediate warm values for the next tier. The resolution of the decoding process during pre-process in increased and the optimisation process repeated. In this case, the initial gear design that formed the base for the search is replaced by the solution obtained from the first tier. The purpose of tier 2 is to fine tune the result of the tier 1. In this way, the final result will be more accurate.

In tier 2, the genetic algorithm optimiser is initiated again with the solution from the previous optimiser, applying a narrower, more accurate band to the search. Again the search is repeated until the limiting percentage of the population are identical, at which point the converged genome is decoded to form the final solution.

## 2.2 The proposed solution

The problem with the existing optimisation program is the time consumed for the program to reach the final solution. This depends on the genetic algorithm as well as the initial design given by the user, e.g. the number of parameters need to be optimised. Genetic

Algorithm is known to be computationally expensive. One of the main factors which affects the execution time is the population. Population needs to be of adequate size to ensure that the search area is comprehensively covered. Even though the final output produces the satisfactory result, the time consumed for the program to reach the final solution could be a drawback for the user. This is particularly important in an Internet environment where speed is an essential factor. A new approach has been developed by the authors to address this problem to improve the speed of the execution drastically.

In the new method, an artificial neural network is built to replace the first tier of the cascade procedure of the genetic algorithm optimisation. The function of the ANN is to estimate the solution of genetic algorithm after the first tier. In another words, to estimate the output of the first tier. This would then be fed into the second tier as the intermediate warm values, see figure 2.

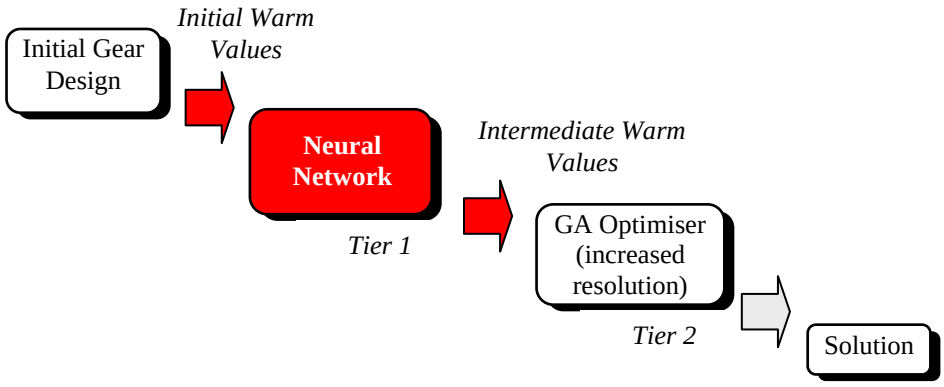


Figure 2. Implementing Neural Networks in the cascade procedure of GA optimisation

## 2.3 Implementation of artificial neural networks

A prototype software program that implements ANN, has been developed for one of the parameters, that is tooth facewidth. The training data is collected from a gear catalogue [7]. Hence the program tries to find the optimum value for the facewidth based on the condition given in the catalogue. The purpose of this proposed approach is to find a value as close as possible to the output of tier 1. It does not need to be exactly the same, since this would act as a warm input value to tier 2 and not the final output. This means that a number of initial parameters, that have slight effect on the final solution, could be excluded from the ANN training set. The idea is to reduce the data collection process by limiting the number of combinations for the ANN parameters.

The original optimisation program was thoroughly investigated and every single parameter was scrutinised to filter out those data that were contributing significant effects on the output. Even though many data parameters were affecting the final output, most of these only had a slight effect. So, it is possible to omit these data from the input of the ANN. This would be very beneficial, since the presence of a large number of input data would increase the number of input patterns drastically. This might not be feasible in terms of the time required to train the ANN. The result of the detail examination carried out by Su et al [4] showed that all the input data are vital for the GA optimisation to produce an accurate result. However, ANN is not required to estimate such accurate result,

since the objective is to replace the first tier only. Any inaccuracy will be cleared in the second tier.

Consequently, after the detailed investigation for the objective of reducing tooth width, the following parameters were found to have a significant effect on the final output: two fitness parameters, which are Stress and Facewidth (these two are part of the GA settings), Number of Teeth, Module, Pressure Angle, Power, Speed, gear Ratio.

As indicated earlier, the design data was collected from a gear catalogue [7]. Four pressure angles (17.5, 20, 22.5, 24) and 12 modules (ranging from 0.5 to 6.0) are defined. To reduce the number of input data for ANN and to improve the performance, it was decided to create an ANN for each of these conditions. This means training 48 different sets of ANNs.

Further study proved that in most cases, Power, Speed and Ratio had less effect on the output. So, the input data were narrowed down to three. It was detected that the ratio of the fitness parameters, i.e. Stress and Facewidth could be used instead of using them individually. As a result, the final decision was to use two inputs, i.e. Number of Teeth and the ratio of Stress and Facewidth and one output, i.e. tooth Facewidth.

To further improve the performance, instead of using the actual output value, the difference between the output of tier 1 and the initial value would be used as the output of ANN. This proved to have a better result. Using the difference value would limit the range of data in the training set which leads to a better normalisation. This is because the difference between maximum and minimum values in the training set would be much lower compared to the case when using the actual output data.

For each training pattern, two sets of data were produced, one for training and one for testing the generalisation of the ANN. These were done for all the 48 different networks. The number of patterns used for each of these networks differ. Each network was tested intensively and additional training patterns were added if the result were not satisfactory. Due to the limited space, it is not possible to list the number of training and test patterns for all the networks. On average, approximately 160 training patterns and 100 test patterns were used.

Before feeding the patterns into the ANN, both the input data and the target need to be normalised. Data normalisation is required by the backpropagation algorithm to perform properly. The normalised data for the ANN, when sigmoid activation ($y = 1/(1+e^{-x})$) is used, lies between 0.0 and 1.0; while for the tanh activation ( y = tanh (x) = $(e^{x} - e^{-x}/e^{x} + e^{-x})$ ) function data values between -1.0 and 1.0 are used. One way to create a data that would lie within the limiting values is to use the minimum and maximum values in the dataset, that is:

$$\bar{X}_i = (X_i - min)/(max\text{-}min)$$

where $\bar{X}_i$ is the normalised value of the ith data in a dataset and *min* & *max* are the minimum and maximum values found in the dataset. The above equation normalises the dataset between 0.0 and 1.0 which is used when sigmoid function is employed. For the tanh function we have,

$$\bar{X}_i = (X_i - cent)/(max\text{-}cent)$$

Where cent is the centre distance between min and max, that is:

$$cent = ((max - min)/2) + min$$

It was found that sigmoid function suited this problem better and resulted in a more accurate output. This could be due to the fact that at certain situations, within the training patterns, there are sharp transitions of output from a high value to a low value, giving a high difference. When normalised between 0 and 1, the sudden transition would be of a lower difference (i.e. from 0 to 1) which makes ANN to behave more efficiently.

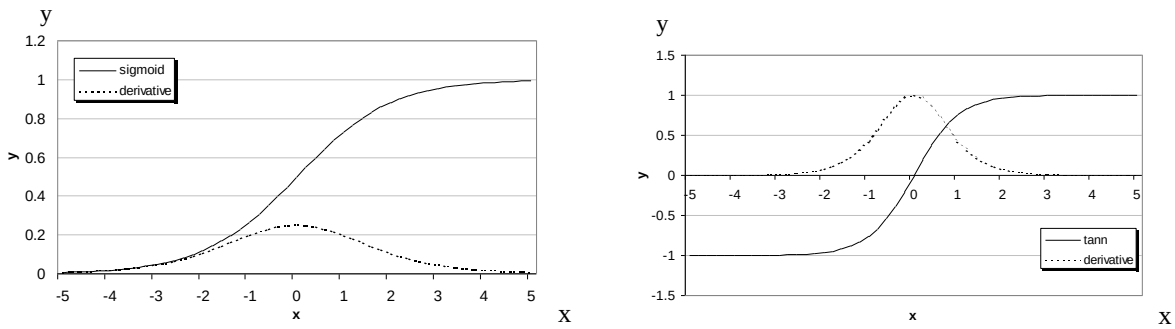


Figure 3. a) sigmoid function and b) tanh function

## 2.4 Result

The result of testing shows that ANN has managed to achieve the purpose of the proposed method. In most cases, it can estimate the output of tier 1 and the estimated value is very close to tier 1 output of the original GA program. This means that it can be used to replace the GA program in tier 1. During the training, the ANN was tested extensively with different values of learning rate, hidden nodes, momentum term, initial weight and the number of iteration, ensuring the best combination that would give the optimum result.

As an example, one of the ANNs is evaluated here. This is the condition when module is 0.7 and pressure angle is 17.5. Table 1 shows the test carried out on the number of iterations. A network with 2-3-1 structure was used, having two inputs (i.e. ratio of the fitness parameters, Stress and Facewidth, and number of teeth), three hidden nodes and one output (i.e. Facewidth). The learning rate, momentum term and initial weight were set to be 0.05, 0.05 and 0.1 respectively. The network was then tested with different number of iterations, ranging form 5000 to 60000, refer to table 1.

As shown in the table, there is only a slight change in the output errors. Increasing the number of iteration, causes a reduction in training error, $\varepsilon_1$, and an increase in testing error, $\varepsilon_g$. This means that the generalisation would be less accurate. So, the iteration number of 40000 was chosen, which shows an acceptable learning and generalisation capability.

| ANN | Learning Rate | Momentum Term | Init Weight | No. of Iteration | $\varepsilon_1$ | $\varepsilon_g$ |
|---|---|---|---|---|---|---|
| 2-3-1 | 0.05 | 0.05 | 0.1 | 5000 | 0.055 | 0.076 |
| " | " | " | " | 10000 | 0.045 | 0.076 |
| " | " | " | " | 20000 | 0.047 | 0.076 |
| " | " | " | " | 30000 | 0.045 | 0.076 |
| " | " | " | " | 40000 | 0.044 | 0.076 |
| " | " | " | " | 50000 | 0.044 | 0.077 |
| " | " | " | " | 60000 | 0.044 | 0.077 |

Table 1. Testing for the number of iterations

Similar test was carried out for the learning rate, momentum term, hidden nodes and initial weight, as well as the combination of these parameters. Their optimum values were found to be 0.045, 0.05 and 0.05 respectively. The number of iterations and the hidden nodes were found to have a strong influence in the result of the ANN. On the other hand, learning rate, momentum term and initial weights had a less effect on the output. These tests were carried out for all the 48 networks to find their best structure.

In order to analyse the efficiency of the proposed method, the output of tier 1 was tested for both the original program that uses GA only and the new one that implements ANN. A number of designs were fed into the programs and the output of the first tiers were monitored. Many tests were applied to investigate the result and the representative of all is provided in figure 4. As shown in the figure, the result has been satisfactory. In most cases, ANN has estimated a value very close to the output of GA. Only in two cases, i.e. tests number 4 and 6, the output estimated by ANN is slightly different. This does not cause any problem, since the estimated value is not totally off the target and it is still close to the GA output. Moreover, this is not the final value. This value would then be fed into Tier 2, where GA would start performing the search to find the final solution.

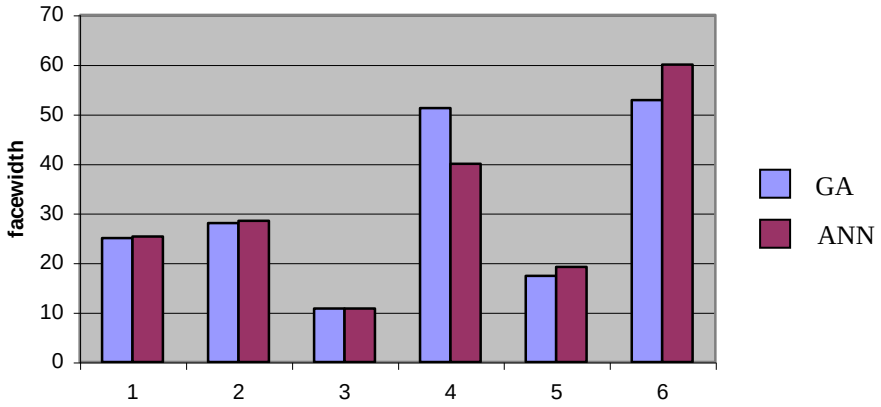


Figure 4. Comparison of the output of Tier 1 for GA and ANN

To further analyse this methodology, a number of tests were carried out on the speed and the iteration number, as shown in table 2.

| | | $\tau_1 + \tau_2$ | Total | $\tau_{NN} + \tau_2$ | Total | %Improvement |
|---|---|---|---|---|---|---|
| Test 1 | Execution Time (s) | 25.15 + 25.49 | 50.64 | 0.01 + 25.82 | 25.83 | 48.99 % |
| | No. of Iteration | 82 + 76 | 158 | 1 + 80 | 81 | 48.73 % |
| | | | | | | |
| Test 2 | Execution Time (s) | 23.84 + 21.84 | 45.68 | 0.01 + 25.49 | 25.50 | 44.18 % |
| | No. of Iteration | 62 + 85 | 144 | 1 + 70 | 71 | 50.69 % |
| | | | | | | |
| Test 3 | Execution Time (s) | 44.87 + 26.64 | 66.747 | 0.01 + 25.18 | 25.19 | 62.26 % |
| | No. of Iteration | 85 + 102 | 187 | 1 + 72 | 73 | 60.96 % |

Table 2. Comparing the speed of the execution of the original optimisation program and the new one that implements ANN

Table 2 shows the execution time and the number of iteration for the original optimisation program that uses GA for both tier 1 and 2, and also for the new program which implements ANN for tier 1. Here $\tau_1$ and $\tau_2$ indicate the time taken for the GA tier 1 and tier 2 respectively, while $\tau_{NN}$ indicates the execution time for the artificial neural network computation. For a better understanding, the execution time and the iteration number have been recorded for both tiers.

In the first test, both the execution time and the iteration number of the two tiers of the new program, i.e. $\tau_{NN} + \tau_2$ have been reduced almost by a half, i.e. execution time has been dropped from 50.64s to 25.83s and the iteration number from 158 to 81. Similar trend appears in the next two tests shown in the table, with the third test having percentage improvement of even higher than 60. The tests shown in table 2 are only representative of many tests carried out. The improvements were between 44% and 62.26%.

## 3. Improving the performance of backpropagation learning algorithm

This section describes a method that allows an improvement in the performance of the Backpropagation learning algorithm. This is done by normalising the output values between 0.1 and 0.9 instead of 0 and 1 for the sigmoid function and between –0.9 and 0.9 instead of –1 and 1 for the tanh function. This is related to the effect of the output derivative near the saturation levels. At these levels the derivatives tend to zero and hence the weight changes would be near zero. By re-scaling the output data, the minimum value of the derivative will be slightly above zero which allows a slight weight change at the saturation level. This leads to a faster and more efficient learning process for the artificial neural network

The proposed method is tested on two applications: 1) Function approximation and 2) Gear optimisation. Both produce satisfactory results and show an improvement in the training and generalisation errors.

### 3.1 Backpropagation algorithm

In this algorithm, the goal is to train a multi-layer perceptron network to approximate an unknown function, based on some training data consisting of pairs ($x$,$d$). The vectors $x$ and $d$ represent a pattern of input to the network and desired output (target) respectively. The

backpropagation calculation can be divided into three segments, feed-forward, weight-adaptation, and feed-back (or error-backpropagation)[8].

*Feed-Forward*:

In feed-forward operation all inputs to each neuron are multiplied by their associated weights, which are then summed up and sent to a limiting function to find the output values,

$$y_j = \sum_{i=1}^{n} x_i * w_{ij} \qquad (1a)$$

$$y_j = f(net_j) \qquad (1b)$$

where $w_{ij}$ is the weight value between input i and output j and *f(.)* is a nonlinear activation function which is commonly represented by Sigmoidal approximation function [9] with the advantage of having a simple derivative, that is,

$$f(x) = 1/(1+e^{-x}) \qquad (2a)$$

$$f'(x) = f(x)*(1-f(x)) \qquad (2b)$$

Sigmoid function bounds values between 0 and 1. However for certain applications, where a broader range of input and output values might be required, the tanh function is used instead. This function limits the activation values between –1 and 1.

$$f(x) = \tanh(x) \qquad (3a)$$

$$f'(x) = 1-(f(x)*f(x)) \qquad (3b)$$

The derivatives are used during the learning process where it behaves somewhat like a filter. It allows a greater change in the weight values when the neuron output is near zero while it permits almost no changes in the weight values when a neuron has reached the saturation (0 or 1 for the *sigmoid* and –1 or 1 for the *tanh* function).

*Weight Adaptation*:

The output weights are updated by an error value that is obtained by,

$$\delta_j^L = f'(y_j^L)*(d_j - y_j^L) \qquad (4)$$

where $d_j$ is a target value and $\delta_j^L$ is error value for the jth neuron in the output layer L, and $f'(y_j^L)$ is the derivative of the activation value of the output layer neuron.

This error value is then used to update weights of the jth node,

$$\Delta w_{ij}^L = \eta * \delta_j^L y_i^h \qquad (5a)$$

$$w_{ij}^{L}(t+1) = w_{ij}^{L}(t) + \Delta w_{ij}^{L} \qquad (5b)$$

where $\eta$ is learning rate coefficient, $y_i^h$ is the output value of neuron i in hidden layer h, $w_{ij}$ is weight value between hidden neuron i and output neuron j, and t is a time step (i.e. iteration).

*Feed-back*:

Since there is no explicit target for the internal representation of the network (or hidden nodes), the back-propagated error signal from the output layer is used to obtain the error values for the hidden neurons. This is a reverse calculation to the feed-forward going from output to hidden layer. Here for each hidden node, the error values of all output neurons are multiplied by their associated weight values and the summation of all multiplication will represent the error value,

$$\delta_i^h = f'(y_i^h) * \sum_{j=1}^{n} \delta_j^L * w_{ij} \qquad (6)$$

A similar weight adaptation is performed for the output weight.

### 3.2 Effect of the output derivative

In most applications, the values of the outputs used for training data are not in the ranges of the artificial neural networks activation function (i.e. sigmoid or tanh). This means that the output data must be pre-processed by scaling. Scaling data before training the ANN is done by mapping the desired range of the outputs to the full working range of the ANN outputs.

One problem with scaling data to full ranges of activation functions is that the values near the two ends (0 and 1 for sigmoid and –1 and 1 for tanh) reach the saturation level. This causes the weight adaptation of such neurons to seize or change insignificantly. The reason lies in the equation of the backpropagation weight adaptation (see Eq. 4 & 5).

In weight adaptation equation the derivative of the neurons have significant effect on the weight values. As can be seen in figure 5a, the value of the derivative reaches a peak value (0.25) at the centre of the sigmoid function (i.e. when y=0.5 for x=0) and progresses towards zero very quickly at the two ends of the activation function (when y=0 or y=1). Therefore the derivative of the output neurons whose target values are close to 1 or 0, become almost zero. Hence any weight adaptation for such output neurons would be zero (or significantly low), since derivative affects the weight change formula (i.e. if derivative=0 ➔ weight change=0).

As an example if an output value is 0.999, its derivative value would be 0.000999. Therefore the weight change, as indicated in Equations 4 & 5, will be very small for the neurons that have reached saturation. The same scenario exists for the tanh function (see figure 5b). The value of the derivative reaches the peak value at the centre of the activation function, (i.e. when y=0 for x=0) and inclines towards zero at the two ends (i.e. when y = -

1 and y = 1). So for tanh activation function, those output neurons that have target values close to 1 and –1, their derivatives becomes zero causing no change in the value of the weights.

One solution to the problem of the insignificant weight adaptation of the neurons with saturated target values is to allow the derivatives of such neurons to have small effect even though they have reached their maximum or minimum values (1 or 0). This can be done by lowering the range of the output data to lie between slightly above zero and slightly below one for the sigmoid function (e.g. between 0.1 and 0.9). In this case the lowest derivative of the neurons would be 0.09 (i.e. 0.1(1-0.1) or 0.9(1-0.9) refer to Eq.2b) as compared to 0.0 (i.e. 0(1-0) or 1(1-1) ) for the 0 to 1 data scaling. Therefore the weight values will still be changing even though the neurons might have reached their saturation levels.

So the output data are normalised between 0.1 & 0.9, and –0.9 & 0.9 for the sigmoid and tanh function respectively.

### 3.3 Experimental Results

As was mentioned before, the theory was tested on two applications, i.e. function approximation and a gear optimisation problem.

*Function Approximation:*

The function [10] that was used for this test is defined as:

$$f(x) = \sin(2\Pi x) * \exp(-x) + Noise(0,0.1)$$

where x is in the range [-1, 1] and the Noise is a random value between 0.0 and 0.1. It is a sinus-line function with noise added to it. It is a non-linear function, which is difficult for AI to estimate the output. The purpose for the addition of the noise is to make it even harder for the ANN to predict the output. The network size of 1:10:1 was used for the test and output values were scaled in the range [0, 1] for sigmoid activation function and [-1, 1] for tanh activation function.

Table 3 summarises the original ANN architecture used for both problems, i.e. function approximation and one of the tests for gear optimisation, and lists the training error and generalisation error for each network. For these tests, the output data were normalised between 0 and 1.

| **Problem** | **ANN** | $\varepsilon_t$ | $\varepsilon_g$ |
|---|---|---|---|
| Gear Optimisation | 2-3-1 | 0.044 | 0.076 |
| Function Approximation | 1-2-1 | 0.022 | 0.024 |

Table 3 – Test result for the original network

Table 4 summarises the new ANN architecture where the output data were normalised between 0.1 and 0.9 and table 5 lists the percentage improvement between the new method and the original one.

| **Problem** | **ANN** | $\varepsilon_t$ | $\varepsilon_g$ |
|---|---|---|---|
| Gear Optimisation | 2-3-1 | 0.040 | 0.061 |
| Function Approximation | 1-2-1 | 0.011 | 0.014 |

Table 4 – Test result for the new network

| **Problem** | **% Improvement for $\varepsilon_t$** | **% Improvement for $\varepsilon_g$** |
|---|---|---|
| Gear Optimisation | 9.1 % | 19.74 % |
| Function Approximation | 50 % | 41.67 % |

Table 5 – Percentage improvement for the new method

As shown in tables 4 and 5, in both cases the new network resulted in a better training and generalisation errors. For the gear optimisation, the training error has improved by 9.1%, i.e. from 0.044 to 0.04, but the percentage of improvement for the generalisation error is much higher, i.e. 19.74%. The generalisation error has been reduced from 0.076 to 0.061. These figures were relatively the same for different tests carried out on gear optimisation. For function approximation problem, the improvement is even higher, where there is a 50% improvement in the training error and 41.67% improvement in the generalisation error.

Figure 5 and 6 shows this effect graphically for the function approximation problem. The average error is calculated using the equation: $\sqrt{\sum (Y-T)^2 / N}$ , where Y is the calculated value, T is the target value and N is the number of training patterns. The graphs of the ANN training results were plotted for the two-dimensional function approximation problem. Figure 5 displays the graph for the original network and figure 6 displays the graph for the new network with re-scaled output data. There is an obvious improvement, specially where the outputs become close to maximum. Scaling the output to 0.9, brings the target closer to the output. Similar trend exists where the outputs get closer to minimum.

Another important point which was observed during testing of the gear optimisation problem was that the new network was learning much faster than the original one. For the gear optimisation, it required 40000 iterations for the original network to reach the solution, whereas for the new network, only 5000 iterations were needed. To further analyse this, a fixed value was chosen for the final error to see how many iterations both the network require in order to reach that fixed value. It was tested for the final value of 0.049 for the gear optimisation problem and the result was outstanding. The original network needed 19,440 number of iterations, whereas the new network reached the final value only in 340 iterations. Similar test was carried out for the function approximation problem and it followed the same pattern.

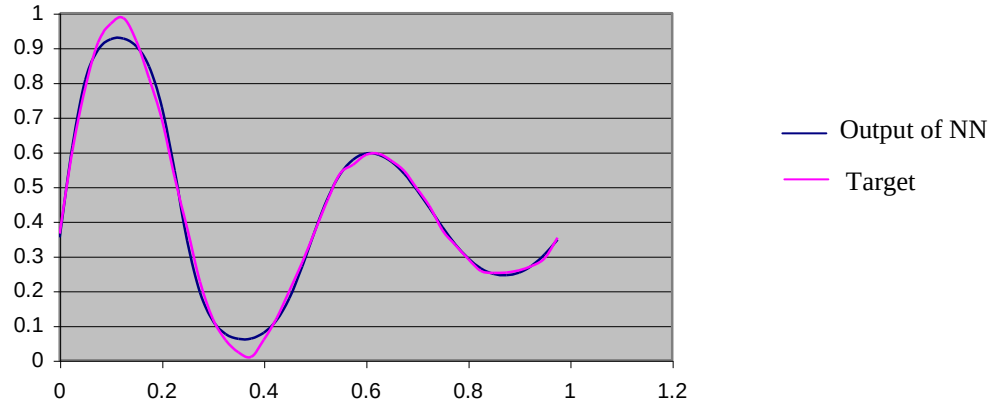


Figure 5 – ANN training result for the original network (average error = 0.022)

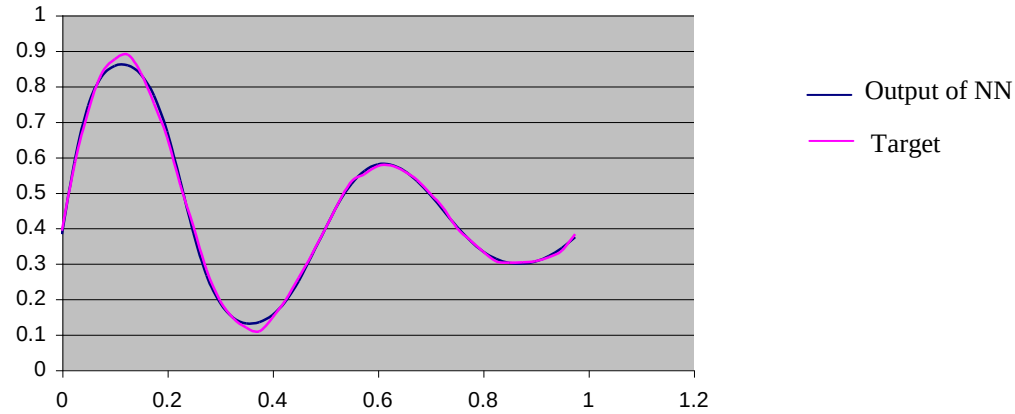


Figure 6 – ANN training result for the new network (average error = 0.022)

## 4. Discussion and conclusion

This paper presented a method to improve the speed and performance of a gear design optimisation program. The first section described the integration of ANN into the optimisation program which resulted in a dramatic reduction in speed. Even though ANN requires a lot of time to go through the learning process, once the training is completed, it is a matter of a simple calculation to produce the result (i.e. feed-forward computation). Therefore, it does not need to go through any iteration. Whereas in GA, the program needs to go through a number of iterations to search for the best possible solution. As a result, by replacing one tier that performs GA with ANN, it acts as if that tier has been eliminated, causing in an impressive reduction in the execution time. The experimental results show 50% improvement in the execution time. Reducing the execution time improves the efficiency of the program. The method is of a particular benefit to the Internet-based version of the optimisation program where speed is a crucial factor.

ANN has the capability of replacing both tiers, in such a case the result would be outstanding. Since ANN would be replacing the whole GA in the application, resulting in an immediate output of the result. At the moment, the training and testing errors are not low enough to replace tier 2. However, with further research in this area, this might be accomplishable.

The second section dealt with the problem of weight adaptation for the situation when the output data reaches the saturation level (0 and 1 for sigmoid and –1 and 1 for tanh). The nature of the weight adaptation equation causes insignificant change in the value of the weights in such conditions. This problem was addressed by lowering the range of the output data to lie between, but not including, the full range of the activation function. The range of 0.1 to 0.9 was tested for the sigmoid function. Function approximation and gear optimisation were used for testing the proposed method. Experimental results illustrates that the proposed method is efficient, resulting in an improvement in the training and generalisation errors. It also speeded up the rate of learning of the artificial neural network.

**References**


[1] Wang, H. and Wang, H.P., 1994, "Optimal engineering design of spur gear sets", International Journal of Mechanism and Machine Theory, Vol. 29, pp. 1071-1080.

[2] Savage, M., Coy, J.J. and Townsend, D.P., 1982, "Optimal tooth numbers for compact standard spur gear sets", Journal of Mechanical Design, Vol. 104, pp. 749-757.

[3] Houser, D.R., Harianto, J., Chandrasekaran, B., Josephson, J. and Iyer, N., 2000, "A multi-variable approach to determine the best gear design", Proceedings of DETC'2000 - 2000 ASME Power Transmission and Gearing Conference, Baltimore, Maryland, USA, on CD-ROM, no. DETC/PTG-14362.

[4] Su, D., Wakelam, M. and Jambunathan, K., 1999, "Evolutionary optimization within an intelligent hybrid system for design integration", Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Vol. 13, pp. 351-363.

[5] Su, D. and Amin, N., 2001, "A CGI-based approach for remotely executing a large program for integration of design and manufacture over the Internet", International Journal of Integrated Manufacturing, Vol. 14, pp. 55-65.

[6] Amin, N. and Su, D., 2000, "Utilisation of Java Applets for gear optimisation", Proceedings of the International Conference on Gearing, Transmissions, and Mechanical Systems, Nottingham, pp. 425-434.

[7] Davall Stock Gears Catalogue, 2000, UK.

[8] Gurney, K., 1997, "Introduction to Neural Networks", U.C.L. Press, London.

[9] Lippmann, R.P., 1987, "Introduction to computing with neural nets", IEEE Acoustics, Speech and Signal Processing Magazine, vol. 4, No. 2, pp. 4-22.

[10] Engelbrecht, A.P., Fletcher, L. and Cloete, I., 1999, "Variance Analysis of Sensitivity Information for Pruning Multilayer Feedforward Neural Networks", IEEE IJCNN, Washington DC, paper 379.