TRANSPORTATION RESEARCH BOARD

NATIONAL RESEARCH COUNCIL

IDEA *Innovations Deserving Exploratory Analysis Project*

NATIONAL COOPERATIVE HIGHWAY RESEARCH PROGRAM

Report of Investigation

IDEA NCHRP

IDEA PROJECT FINAL REPORT
Contract NCHRP-40

IDEA Program
Transportation Research Board
National Research Council

July 1999

Estimating Truck Attributes from Bridge Strain Data Using Neural Networks

Prepared by:
Ian Flood
ME Rinker Sr. School of Building Construction
University of Florida
Gainesville, FL

INNOVATIONS DESERVING EXPLORATORY ANALYSIS (IDEA) PROGRAMS
MANAGED BY THE TRANSPORTATION RESEARCH BOARD (TRB)

This NCHRP-IDEA investigation was completed as part of the National Cooperative Highway Research Program (NCHRP). The NCHRP-IDEA program is one of the four IDEA programs managed by the Transportation Research Board (TRB) to foster innovations in highway and intermodal surface transportation systems. The other three IDEA program areas are Transit-IDEA, which focuses on products and results for transit practice, in support of the Transit Cooperative Research Program (TCRP), Safety-IDEA, which focuses on motor carrier safety practice, in support of the Federal Motor Carrier Safety Administration and Federal Railroad Administration, and High Speed Rail-IDEA (HSR), which focuses on products and results for high speed rail practice, in support of the Federal Railroad Administration. The four IDEA program areas are integrated to promote the development and testing of nontraditional and innovative concepts, methods, and technologies for surface transportation systems.

For information on the IDEA Program contact IDEA Program, Transportation Research Board, 500 5th Street, N.W., Washington, D.C. 20001 (phone: 202/334-1461, fax: 202/334-3471, http://www.nationalacademies.org/trb/idea)

Abstract:

The project was concerned with the development of a neural network-based method of accurately estimating truck attributes (such as axle loads) from strain response readings taken from the bridge over which the truck is traveling. The approach is designed to remove the need for intrusive devices (such as tape switches) on the deck of the bridge to obtain such data so as to provide a convenient and viable means of collecting bridge loading statistics.

The problem with tape switches is their longevity. They are rapidly destroyed by heavy traffic, particularly following precipitation. They can also warn truck drivers about the presence of instrumentation on the bridge. As a result, truck drivers may avoid the bridge or reduce speed, thus introducing bias into the collected data. The system can estimate truck attributes solely from strain readings taken from girders supporting the bridge deck, thus eliminating the need for surface instrumentation. The system is inexpensive to implement since the only bridge instrumentation required is the existing weigh-in-motion (WIM) strain transducer technology.

The system has many practical applications, which can be broadly classified under two headings: (1) to enforce legal weight limits on trucks without the need to stop those vehicles for weighing; and (2) to obtain comprehensive statistics for use in, for example, highway bridge design or fatigue rating of existing bridges. A detailed database of truck types, and loading conditions can be acquired, from which it will be possible to elicit location specific and time-wise trends in bridge loading, which in turn will be useful in determining bridge refurbishment and replacement policy.

The approach to the problem was to exploit the automatic modeling capabilities of neural networks to develop an accurate model of the function that maps from bridge strain to loading conditions. Training of the networks was based on data defining the strain response of bridges to a variety of truck loading situations. A two-level neural networking system was adopted. This modularized approach was proposed, in preference to a singular network, to facilitate the task of training. The network at the first level in the system was trained to classify a given truck loading condition and, thus, select an appropriate set of networks for estimating truck attributes from the second level of the system. A variety of alternative networking systems were considered for the first level, including a self-organizing device that determines its own classification system for truck loading situations, based on natural clusterings of the training data within the problem domain. The second level in the system comprises three networks for each truck loading class, estimating velocity, axle spacings, and axle loads. These second level networks were developed using a supervised training algorithm. A variety of supervised training algorithms were considered for this purpose and their performances compared. The input to both the first and second level networks are an array of strain readings measured at a fixed location on a girder at a prespecified sampling rate during a truck crossing event.

Most generally, the project demonstrated the currently most viable ways of using artificial neural networks to determine the attributes of trucks in motion, to within an acceptable degree of accuracy.

1.0 BACKGROUND

A weigh-in-motion (WIM) system capable of accurately determining the velocity, axle spacings, and axle loads of fast moving trucks from measurements of the strain response of the structure over which they are traveling, has many practical applications. In particular, it would enable enforcement of legal weight limits on trucks without the need to stop those vehicles for weighing, and enable comprehensive statistics on vehicle-bridge loading to be obtained for use in, for example, highway bridge design or fatigue rating of existing bridges Moses et al. [14, 15]. Existing WIM methods make use of tape switches laid across each lane of a bridge for determining the velocity, the number of axles, and spacings of the axles, on a passing truck. In addition, a transducer attached to a girder supporting the bridge deck is used to measure strain, at a given sampling rate, during the truck crossing. Combining these data, and by the use of simple theory of mechanics, it is possible to approximate the load on each of the truck's axles.

The main problem with the existing system is that the tape switches rapidly deteriorate in heavy traffic, particularly following precipitation. The presence of the tape switches also warns truck drivers that the bridge is instrumented. As a result, they may moderate their speed or avoid the bridge, thus introducing bias into the collected data. The solution to this problem considered here was to develop a neural network-based method of estimating truck attributes (such as, axle loads, velocity, and axle spacings) solely from the transducer strain readings. This has the advantage of removing the need for the tape switches, or other bridge deck instrumentation. In addition, the approach makes use of the existing WIM strain transducer technology, and is thus inexpensive to develop and implement.

2.0 NEURAL NETWORKS

Artificial neural networks are computing devices that emulate, to different levels of abstraction, the structure and operation of the central nervous system. They are configured from a large number of parallel operating processors, or neurons, each of which performs some primitive function, and are usually implemented through software on a general-purpose digital computer. These neurons are linked, such as illustrated in Figure 1, to form a network that performs a higher-order function. This could be anything from, for example, estimating construction productivity Chao and Skibniewski [1] to the simulation of the behavior of dynamic construction processes Flood and Worley [3].

There are many different forms of neural networks, the details of which are well documented in a number of texts such as that by Hecht-Nielsen [11]. Briefly, however, a neural network provides a function that maps from a vector of inputs (representing a problem to be solved) to a vector of outputs (representing the network's solution to the given problem). Figure 1 shows two commonly adopted example architectures (as proposed for this work), the first is a laterally connected network (with all neurons in the output layer connected to each other), and the second is a feedforward network (with the neurons divided into layers with all neurons in one layer connected to all those in the subsequent layer). In either case, information describing the problem is presented to the appropriate input neurons, from where it is transferred across the connections to the neighboring neurons. This process may be repeated a number of times (in the case of the feedforward network the process is repeated until values are available at the output neurons). The networks solution to the problem is then read as the set of values generated across the output neurons. Processing operations are performed on the values as they pass across the links. In addition, values converging at a neuron are integrated and put through a simple non-linear function. The precise nature of these operations depends on the type of neural network adopted.

Certain parameters are associated with the processing tasks performed along the links and at the neurons, the values of which define the specific mapping function implemented by the network. A major task in the development of a neural network is, therefore, concerned with determining an appropriate set of values for these parameters. Typically this involves the use of either a supervised or an unsupervised training scheme, both of which were considered for this work. Supervised training involves the use of a representative set of training patterns, each of which comprises an example of the problem to be solved and its corresponding targeted solution. The network, in effect, attempts to learn these mappings, usually by means of an iterative procedure that adjusts network parameters in a way that minimizes an error function. Error is measured as some function of the difference between the actual output from the network and the targeted output, averaged over all the training patterns. Once training has been completed, it is anticipated that the network will be able to provide accurate solutions to other examples of the problem not used in the training process. This cannot always be guaranteed however, and thus a thorough validation of the network must be undertaken.

Unsupervised training, on the other hand, makes use of training patterns that do not include targeted solutions. Typical of this class of training schemes are the clustering algorithms, such as those proposed by Kohonen [12] for problem classification. In its simplest form, training proceeds by adjusting the network parameters in a way that forces each output neuron to respond to as many training patterns as possible without overlapping with the response of other output neurons. The result is a division of the input space into regions containing distinct clusters of training patterns, each cluster embraced by a different output neuron. The class into which a problem is placed by the network is denoted by the output neuron that generates the largest value.

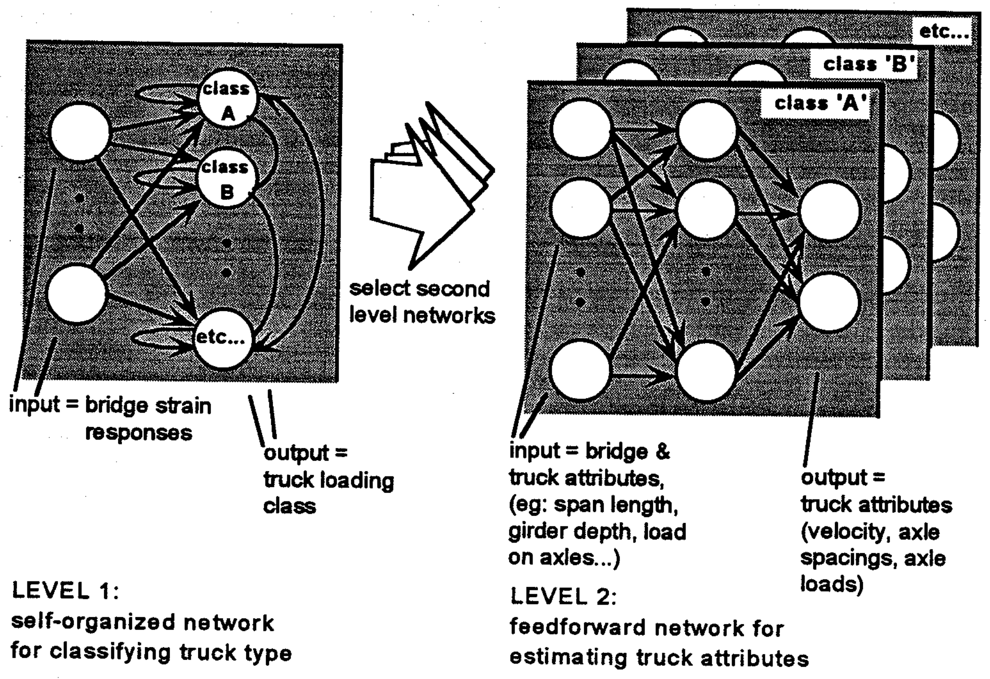
class A
class B
etc...
input = bridge strain responses
output = truck loading class
LEVEL 1:
self-organized network for classifying truck type
select second level networks
etc...
class 'B'
class 'A'
input = bridge & truck attributes, (eg: span length, girder depth, load on axles...)
output = truck attributes (velocity, axle spacings, axle loads)
LEVEL 2:
feedforward network for estimating truck attributes

FIGURE 1 Overall Structure of Neural Networking System.

3.0 APPROACH

A two-stage neural network system was considered, as illustrated in Figure 1, to estimate truck attributes from the strain response of the bridge over which the truck is traveling. The first of the two stages (that shown to the left of the figure) is designed to classify a given truck loading condition and thus select an appropriate set of networks from the second level of the system. The laterally connected architecture shown at level 1 in the figure is typical of networks used for classification purposes, though conventional feedforward networks were also experimented with for this first level network. The networks in the second level were designed to operate for a given truck loading class, providing estimates of velocity, axle spacings and axle loads.

This modularized approach was adopted, in preference to a singular network, to facilitate the task of training the neural networks [8, 9]. In addition, it enables individual modules to be retrained, and new modules added, as new training data become available, without the burdensome task of having to retrain the complete network system. Each of these subsystems is described in turn in the following.

3.1 FIRST LEVEL NETWORK: TRUCK CLASSIFICATION

A variety of different types of neural network were considered for the first level in the network. Of these, the most promising have been found to be:

(a) SORG: a self-organizing network that develops its own classification system (in this case of truck types) by identifying clusters in the training patterns in the input domain. Specifically, a system that places radial-Gaussian functions over each cluster was adopted. The mode of operation and development of this type of network is described in detail by Moody and Darken [13].

Input to the SORG networks was an array of strain readings measured at a fixed location on a girder of the bridge during the passage of a truck, as indicated in Figure 2. As a truck crosses a bridge it induces strain in the girders. This is measured for the entire period of the truck crossing by a transducer, providing a strain-time curve such as that indicated in the figure. Each truck generates a characteristic strain-time curve, the form of which is a function of the velocity of the truck, the number of axles, the spacing between axles, and the load on each axle. The objective is to recognize the type of truck from its strain-time curve. The resultant strain curve for a truck crossing event was divided into an array of 32 real-valued strain readings distributed evenly over the duration of the truck crossing event (see, for example, Figure 3a).

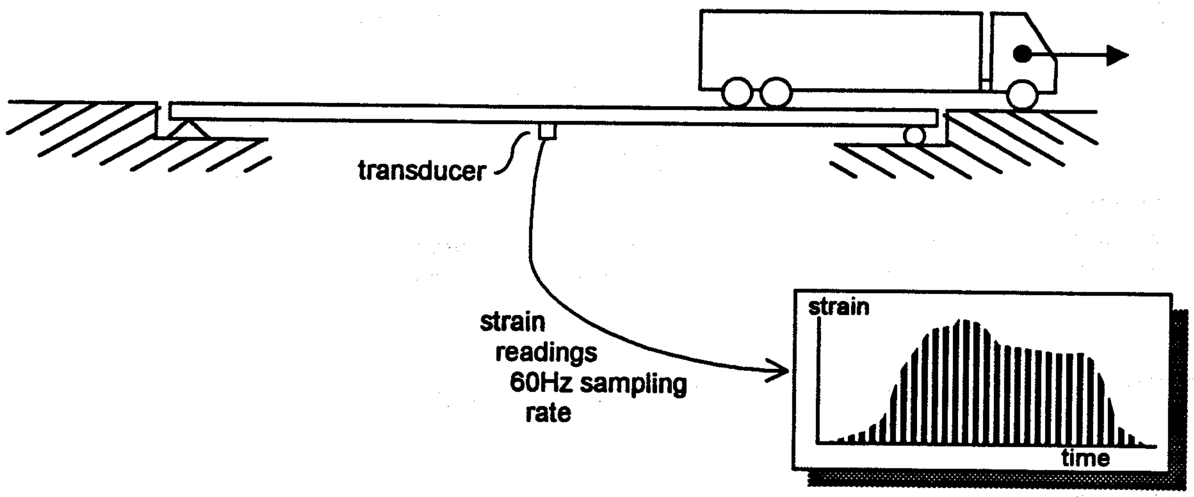

FIGURE 2 Sampling of Strain Data Induced by a Truck Crossing Event.

Each output from the SORG network represents a different truck loading class. The class to which a given truck loading situation belongs is indicated by the output neuron that generates the value closest to 1.0 (all other outputs should generate a value close to 0.0).

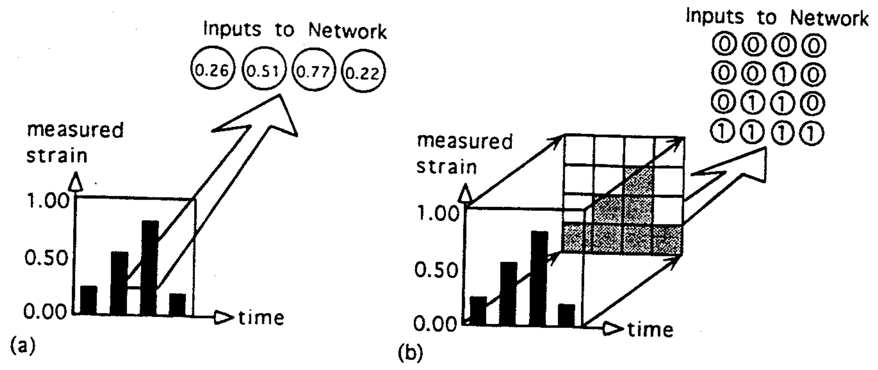

FIGURE 3 Formatting Strain-Time Curves for Input to a Neural Network: (a) vector of real-values; (b) matrix of binary values.

(b) EHAM: a binary networking system that is an extension of the simple Hamming network. This system uses a supervised training algorithm, and so a classification system for trucks must be prescribed. The FHWA system of truck classification was therefore adopted. The extended Hamming network and its training algorithm were developed by the principal investigator and are not yet published – therefore, a description of the system and its training algorithm are provided in Appendix II.

Input to the EHAM networks was a matrix of binary values representing a projection of the strain readings measured at a fixed location on a girder of the bridge during the passage of a truck. The binary map was a 32 by 32 matrix, with one dimension representing sample strains taken at different points in time during a truck crossing event, while the other dimension indicating the magnitude of the strain readings (see, for example, Figure 3b).

Each output from the EHAM network represents a different truck loading class. The class to which a given truck loading situation belongs is indicated by the output neuron that generates the binary value 1 (all other outputs should generate a binary value of 0).

(c) RGIN: a radial-Gaussian feedforward networking system that uses an incremental training algorithm. As for EHAM, training in RGIN is supervised and so the FHWA system of truck classification was adopted a priori. The RGIN network and its training algorithm were developed by the principal investigator and are not widely published – therefore, a description of the system and its training algorithm are provided in Appendix I.

Input to the RGIN networks was an array of strain readings measured at a fixed location on a girder of the bridge during the passage of a truck. Each output from the RGIN network represents a different truck loading class. The class to which a given truck loading situation belongs is indicated by the output neuron that generates the value closest to 1.0 (all other outputs should generate a value close to 0.0). In terms of the input-output data format, the RGIN network operates in the same manner as the SORG networks, as outlined in (a) above.

The RGIN system was adopted since it had been found to perform well in a similar problem considered by Gagarin et al. [9]. The SORG and EHAM network systems were selected to see if their radically different and unconventional approaches to the problem would provide a better solution. SORG, in particular was selected to overcome a limitation of RGIN when applied to truck classification. RGIN experienced some difficulty in distinguishing between certain of the FHWA truck classes (some classes were very similar, whereas others embraced trucks that could differ in characteristics significantly). It was thought that the self-organized approach of SORG might enable the network to divide the problem domain into well-defined regions (with less ambiguity than the partially arbitrary FHWA classification scheme) and thus reduce the likelihood of confusion when making a classification.

3.2 SECOND LEVEL NETWORK: TRUCK ATTRIBUTE ESTIMATION

The networks in the second level of the system are designed to operate, not just for a truck loading class as determined by the first level network, but also for a specific class of bridges. To keep the research effort within a reasonable bound, the work focused on developing a system for simply supported multispan steel bridges with negligible skew. Extensions to other set-ups, such as skewed and prestressed concrete bridges, will be the subject of future work.

Inputs to the second level network were an array of strain readings measured at a fixed location on a girder of the bridge during the passage of a truck (see, for example, Figure 3a). Supervised training was adopted, using training patterns that define both input to the network and corresponding targeted output. Two neural network architectures and supervised training schemes were evaluated. These were:

(a) RGIN: this is the same system outlined in (c) above used for truck classification in the first level of the networking system (see Appendix II)

(b) GDR: the most widely used neural networking system, comprising a feedforward network architecture and using the Generalized Delta Rule for training, a detailed description of which is provided by Rumelhart et al. [16].

4.0 RESULTS

This training of the networks used data based on the simply supported 17.1m bridge span representing the westbound lane crossing the Bull Run on Interstate 66 near Washington DC. Training patterns were established for the nine classes of truck most common in traffic (see Figure 4), with the range in axle spacings and axle loads listed in Table I. Three values (the minimum, maximum, and central value) were considered for each axle spacing and axle load, for each truck class, giving the number of alternative truck configurations listed in the end column. Each truck configuration was run at a speed of 80.1 km/h, 96 km/h and 112 km/h, across a single girder model of the bridge, and strain readings were generated based on a 60 Hz sampling rate, providing a total of 4,131 strain-time curves. Each curve covered the duration of the truck-crossing event, and was divided into 32 equal groups of strain readings - the values within each group were then averaged. Each resultant strain-time curve was used to establish a training pattern.

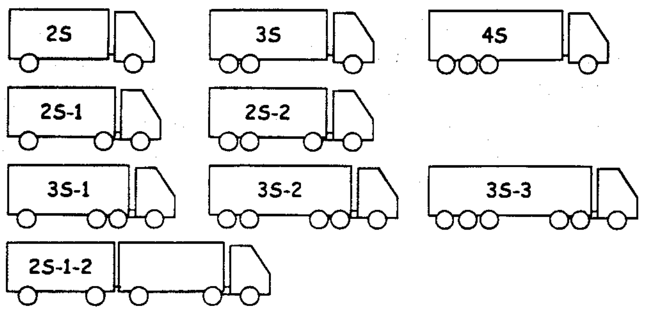

FIGURE 4 Axle Configurations of FHWA Alternative Classes of Truck

A total of 1,000 testing patterns were set-up for each type of truck using axle spacings and loads selected at random from within the ranges given in Table I, and with velocities between 80.1 km/h and 112 km/h.

These data were used for training and testing both the level 1 and level 2 networks illustrated in Figure 1 above.

4.1 TRUCK-TYPE CLASSIFICATION

The first series of experiments were concerned with the development of the truck classifier (see Level 1 in Figure 1). The EHAM networking system required a maximum of 917 hidden neurons to learn all the training patterns at an output, while the RGIN network was found to improve little beyond 1,000 hidden neurons for any output. The performance of the networks, once trained, was measured using the example truck crossing events not used for training. In each experiment, the percentage of correct and incorrect classifications by the network were registered. Table II shows the performance of both the EHAM and RGIN networks for the 9,000 test patterns (1,000 patterns per truck type). It is clear from these results that there is little to distinguish between the two types of network in terms of performance. However, the EHAM network has the advantage of being simpler in form and two to three orders of magnitude faster to train (see Section 4.3 below).

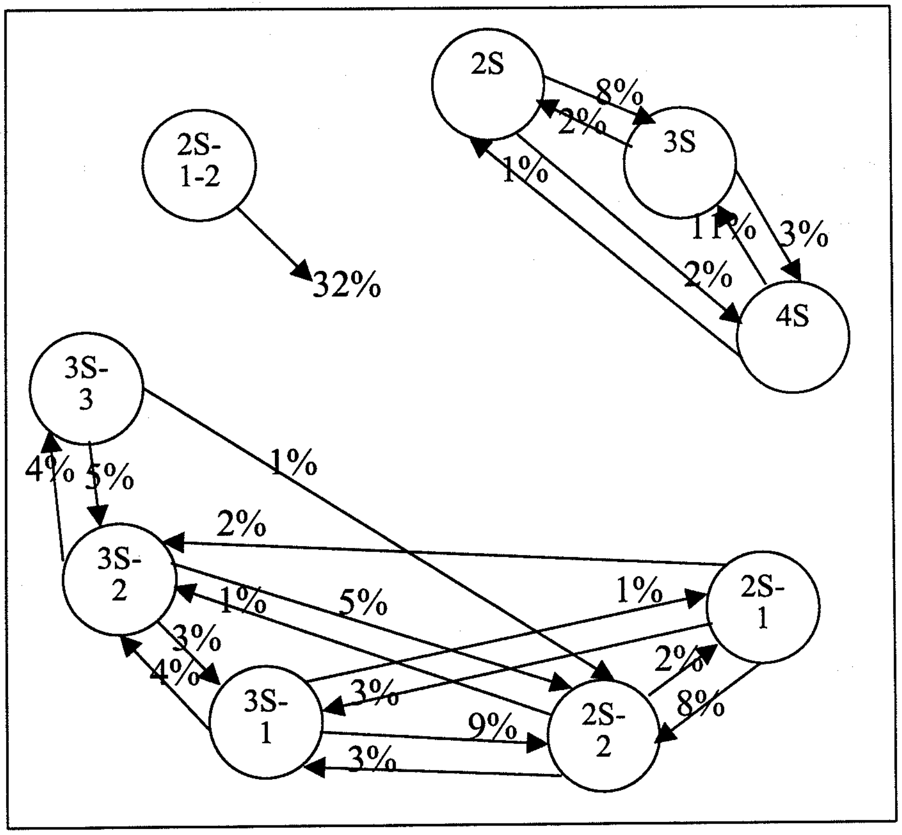

FIGURE 5 Average Percentage Misclassification of Trucks by EHAM Network Classifiers, for each FHWA truck type.

Referring to Table II, it can be seen that the neural networks had an average of around 89% success rate at truck classification, ignoring the truck type 2S-1-2. The poor performance of the classifier for the truck type 2S-1-2 can be attributed to the significantly different axle configuration of this type of vehicle compared to the others in the FHWA classification system. This type of truck is relatively rare and thus should not significantly affect the accuracy of a truck loading database for a bridge. Moreover, ways of improving performance for the 2S-1-2 truck class are being considered, including the use of more training patterns for this category.

An important observation made in this series of experiments was that the truck misclassifications tended to be between trucks with the same number of axle clusters. This is illustrated in Figure 5, which represents each FHWA truck type by a circle, and the percentage of misclassifications as a number adjacent to an arrow[i]. For example, truck class 3S-3 experienced 1% misclassifications as class 2S-2 and 5% misclassifications as class 3S-2. Referring to the figure, it can be seen that the truck types 2S, 3S and 4S all have two clusters of axles and the misclassifications of these trucks were confined within this group. Similarly, trucks of type 2S-1, 2S-2, 3S-1, 3S-2 and 3S-3 all have three clusters of axles and their misclassifications were all confined within the three-axle cluster group. This suggests that an improvement could be achieved classifying trucks in two tiers of neural network, the first of which would identify the number of axle clusters on a truck, and the second to refine this classification to a specific FHWA category. The ability of such an approach to decrease the percentage of misclassifications is currently under investigation.

Finally, SORG, the self-organizing network approach to classification, was able to converge on a classification system rapidly. A range of between 4 and 15 radial-Gaussian neurons were considered in these experiments, allowing the SORG to develop classification systems comprising from 4 and 15 distinct classes. However, an inspection of the resultant classification systems developed by the network was inconclusive. That is, the SORG divided the example training patterns into classes of truck loading situations that were indistinct – for example, one class may contain truck loading examples generated by a range of different axle configurations, and loading configurations. There was no consistent partitioning of the problem space into groups of truck loading situations that could be characterized by tangible parameters such as load distribution, or numbers of axles. Further work is recommended, however, using alternative types of self-organizing network, and using concepts such as normalization of input values to make sure each input has the same significance on the problem as the others.

4.2 TRUCK ATTRIBUTE ESTIMATION

At the second level in the system (see the right hand side of Figure 1) are a set of neural networks designed to estimate truck attributes. For each class of truck, there is a separate neural network to determine axle spacings, axle loads, and velocity.

The performance of the second level networks can be no better than the performance of the first level classifier network. That is to say, if the first level network were to misclassify a truck, then the wrong networks would be selected from the second level in the system, and thus the truck attribute estimations will be invalid. The second level network has been shown in earlier work to provide an acceptable degree of accuracy assuming that the first level network correctly classified the truck. The emphasis of this study was, therefore, on ensuring correct first level classification, and to see if an alternative networking system could improve on the performance of the second level in the system.

Of the two types of networking system evaluated for the second level in the system, the

[i] Misclassifications were measured from an early test using the EHAM system, but are characteristic of all the experiments performed with EHAM and RGIN for truck classification.

RGIN approach was found to significantly outperform the GDR. A major problem with GDR was that it was very slow to train, often taking several days (see Section 4.3 below), and often did not converge on a solution at all. Where GDR did converge, its performance at truck attribute estimation was considerably below that of RGIN. For RGIN, typical results provided estimates that, for 90% of the test cases, had absolute errors within 6.4kN for axle loads, 0.78m for axle spacings and 6.2km/h for truck velocity. These results are encouraging, though ways of further improving performance, especially removing outlying results, are being considered.

4.3 TRAINING SPEED

A major advantage of the proposed EHAM classifier system is the speed with which it can develop a network. For example, 0.024 seconds was the average processing time required to develop one hidden neuron in a network (typically 1,000 hidden neurons were required to provide an accurate classifier), using a set of 4,131 training patterns[ii]. In contrast, the RGIN was approximately 430 times slower in training for the classification problem, taking an average of 10.33 seconds to train each hidden neuron for the same set of training patterns (taking 2.87 hours to train the entire network).

Moreover, the RGIN training algorithm was found to operate several orders of magnitude faster than the GDR training algorithm for the truck attribute estimation problem. In the experiments performed here, the GDR took several days to train on the truck attribute estimation problem if it converged on a solution at all.

[ii] Processing speed experiments were performed on an IBM Aptiva 300MHz, 64Mb RAM, computing system programmed using Borland's Turbo Pascal.

TABLE I Range of Parameter Settings used for Establishing Training/Testing Patterns

Type of Truck	Axle Loads (kN)						Axle Spacings (m)					No. of training patterns
	1	2	3	4	5	6	1-2	2-3	3-4	4-5	5-6	
2S	13.3-53.4	8.8-80.1	-	-	-	-	2.74-6.10	-	-	-	-	27
3S	13.3-53.4	8.8-80.1	8.8-80.1	-	-	-	2.74-6.10	1.22	-	-	-	27
4S	13.3-53.4	8.8-80.1	8.8-80.1	8.8-80.1	-	-	2.74-5.49	1.22	1.22	-	-	27
2S-1	13.3-53.4	8.8-80.1	8.8-80.1	-	-	-	2.74-4.88	5.49-1.6	-	-	-	243
2S-2	13.3-53.4	8.8-80.1	8.8-80.1	8.8-80.1	-	-	2.74-5.49	6.10-11.6	1.22	-	-	243
3S-1	13.3-62.3	8.8-71.2	8.8-71.2	8.8-80.1	-	-	2.74-6.10	1.22	6.10-11.6	-	-	243
3S-2	13.3-53.4	8.8-71.2	8.8-71.2	8.8-80.1	8.8-80.1	-	2.74-6.10	1.22	6.10-11.6	1.22	-	243
3S-3	13.3-53.4	8.8-71.2	8.8-71.2	8.8-80.1	8.8-80.1	8.8-80.1	2.74-6.10	1.22	6.10-11.6	1.22	1.22	243
2S-1-2	13.3-53.4	8.8-80.1	8.8-80.1	8.8-80.1	8.8-80.1	-	2.74-5.49	5.49	3.05	5.49	-	81

TABLE II Performance of Neural Networks for Truck Classification Problem

FHWA TRUCK CLASS	% CORRECT CLASSIFICATIONS	
	PROPOSED BINARY NETWORK	RADIAL-GAUSSIAN NETWORK
2S	92.3%	93.1%
3S	89.6%	90.2%
4S	88.2%	84.3%
2S-1	91.0%	89.4%
2S-2	88.0%	87.8%
3S-1	86.1%	85.3%
3S-2	88.2%	81.3%
3S-3	89.7%	86.4%
2S-1-2	76.1%	79.2%

5.0 CONCLUSIONS AND RECOMMENDATIONS

The project has demonstrated the currently most viable ways of using artificial neural networks to determine the attributes of trucks in motion, to within an acceptable degree of accuracy, using a two-layered artificial neural network. In particular, the EHAM method was shown to provide results equally as good as RGIN in terms of its ability to classify trucks, and to outperform RGIN in terms of the speed with which it can develop a working model for a bridge. However, results from an attempt to improve classification accuracy (and thus ultimately the accuracy of estimates of truck attributes such as axle loads and spacings) by allowing a SORG network to develop its own classification system for trucks were inconclusive. The project has generated interest from industry and an international consortium has requested a meeting in April to look into the possibility of adopting and implementing this technology.

Several recommendations are made for future work, aimed at further improving the performance of the system. Firstly, the work here focussed on simply supported steel bridges with negligible skew. It is recommended that the technique be applied to other bridge configurations, such as skewed and pre-stressed concrete structures. Ways of further improving the accuracy with which the neural networks classify trucks and estimate truck attributes, are also being considered. This includes, the use of larger sets of training patterns, which in turn requires the use of more powerful computing facilities. Another possible way of improving accuracy (suggested by the experiments undertaken in this project) involves classifying trucks within two tiers of neural networks: the first tier would identify the number of axle clusters on a truck, and the second would refine this classification to a specific FHWA category. Finally, further work is underway trying alternative neural networking system for the self-organized approach to truck classification. If a self-organized network can be found that derives its own meaningful classification of truck types, it is likely that a neural network classifier based on this would make fewer misclassifications of trucks – the accuracy of estimates of truck attributes, such as axle loads and space, would also improve as a direct consequence.

6. BIBLIOGRAPHY

[1] Chao, L.C., and Skibniewski, M.J. (1994). "Estimating Construction Productivity: Neural Network-Based Approach." Journal of Computing in Civil Engineering, 8(2), ASCE, New York, NY, 234-251.

[2] Daniels, J.H., Wilson, J.W., Yen, B.T., Lai, L.Y., and Abbaszadeh, R. (1987). "Weigh-In-Motion and Response Study of Four In-service Bridges." Report FHWA/RD-86/045, Federal Highway Administration, McLean, Virginia.

[3] Fahlman, S.E. and Lebiere, C. (*1990*). The Cascaded-Correlation Learning Architecture", *Rep.* CMU-CS-90-100, Carnegie Mellon University, Pittsburgh, PA.

[4] FHWA, "States Successful Practices Weigh-In-Motion Handbook", FHWA, 1997.

[5] Flood, I. and Worley, K., (1995). "An Artificial Neural Network Approach to Discrete-Event Simulation", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* 9, Cambridge University Press, pp. 37-49.

[6] Gagarin, N., and Albrecht, P. (1992). "Advances in Weigh-in-Motion Using Pattern Recognition and Prediction of Fatigue Life of Highway Bridges". Vol. 1: Final Report, Report No. FHWA-RD-92-046, Federal Highway Administration, McLean, Virginia, September, 98 p.

[7] Gagarin, N., and Albrecht, P. (1992)."Advances in Weigh-in-Motion Using Pattern Recognition and Prediction of Fatigue Life of Highway Bridges," Vol. 2: Data Report, Report No. FHWA/RD-92/045, Federal Highway Administration, McLean, Virginia, September.

[8] Gagarin, N., Flood, I., and Albrecht, P., (1992). "Weighing Trucks in Motion Using Gaussian-Based Neural Networks", *Proceedings of the International Joint Conference on Neural Networks,* IEEE and INNS, Baltimore.

[9] Gagarin, N., Flood, I., and Albrecht, P., (1994). "Computing Truck Attributes with Artificial Neural Networks", *Journal of Computing in Civil Engineering*, ASCE, 8 (2), pp. 179-200.

[10] Green, M.F. and Cebon, D., (1994), Dynamic Response of Highway Bridges to Heavy Vehicle Loads: Theory and Experimental Validation, *Journal of Sound and Vibration*, 170, **1**, 51-78.

[11] Hecht-Nielsen, R. (1990). *Neurocomputing*. Addison-Wesley, 1990.

[12] Kohonen, T. (1989). *Self-Organizing and Associative Memory,* Third Edition, Springer-Verlag, Berlin.

[13] Moody, J and Darken, C J, (1989), "Fast Learning in Networks of Locally Tuned Processing Units", *Neural Computation,* 1, 281-294.

[14] Moses, F., and Kriss, M. (1978). "Weigh-in-Motion Instrumentation." Report FHWA/RD/-78/81, Federal Highway Administration, McLean, VA.

[15] Moses, F. and Ghosn, M. (1983). "Instrumentation for Weighing Trucks-In-Motion for Highway Bridge Loads." Report FHWA/OH-83/001, Federal Highway Administration, McLean, Virginia.

[16] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Internal Representations by Error Propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel Distributed Processing, exploring the microstructure of cognition,* 1 (pp. 318-362) Cambridge, MA: MIT Press.

APPENDIX I: Radial-Gaussian Networks with Incremental Learning (RGIN)

I.1 Networks of Radial-Gaussian Neurons

The RGIN networking system is based on a simple feedforward architecture (such as shown to the right of Figure 1) comprising a layer of input neurons that perform a normalizing function, a single layer of hidden neurons each of which implements a radial-Gaussian function, and a layer of output neurons that act as simple summers of incoming values. The operation of these networks can be formalized by the following equation:

$$o_y = \sum_{h=1}^{H} v_{h,y} e^{-s_h \sum_{j=1}^{J} (\alpha_j i_j - c_{j,h})^2} \qquad \text{...(I.1)}$$

where: o_y is the value generated at the y-th output neuron; $v_{h,y}$ are weights on the links to the output neuron defining the amplitude of the *h-th* radial-Gaussian function at the *y-th* output neuron; s_h is a squeezing parameter defining the spread of the *h-th* radial-Gaussian function; $c_{j,h}$ are offsets on the connections to the *h-th* hidden neuron defining the position of the radial-Gaussian function in the *j-th* input dimension ; α_j is a normalizing parameter; and i_j is the value input to the *j-th* input neuron.

I.2 RGIN Training Algorithm

The training algorithm proceeds by constructing a solution surface out of radial-Gaussian functions, one at a time. The addition of each hidden neuron is guaranteed to increase the closeness of fit of the network function to the training data (see Step 3). It is thus possible to train the network to a state in which it provides a literal interpretation of the training patterns.

Each training pattern provides a mapping from a vector of inputs, $\vec{i}$, to a corresponding vector of targeted outputs, $\vec{t}$. Each radial-Gaussian function is focused on the general area of the problem where the residual target (that is, the portion of the target surface yet to be learned) is greatest. Final values, or at least close approximations, are determined directly for all neuron parameters except for the widths of the radial-Gaussian functions which are learned using a fast error gradient descent technique. Each successive hidden neuron is in effect trained on the component of the problem its predecessors failed to learn.

Step 1: A network is first set-up with the number of input and output neurons required for the problem at hand, but with no hidden neurons. Normalizing parameters are then established for the input neurons. Usually, the values are selected so that the range of activations generated at each input neuron measured over the set of training patterns is equal to 1.0, thus: $\alpha_j = 1/(i_j^{\max} - i_j^{\min})$. This ensures that each radial-Gaussian function bears equal significance to each input variable.

Step 2: The set of training patterns is scanned to see which one deviates from the input plane by the greatest amount; that is, the pattern $\hat{p}$ is selected where $\sum_{y=1}^{Y} |t_{p,y}|$ is the greatest. A hidden neuron is then added to the network. The parameters on the input and output links of the hidden neuron are set equal to the values of the normalized input vector and the output vector respectively of training pattern $\hat{p}$, that is, $\vec{c} = \vec{\alpha}_{\hat{p}} \vec{i}_{\hat{p}}$ and $\vec{v} = \vec{t}_{\hat{p}}$. This forces the network to produce a radial-Gaussian function at each output neuron, centered near the point where the amplitude of the target surfaces is greatest. The amplitude of the radial-Gaussian function produced at each output will be equal to the amplitude of the corresponding target surface at the point under consideration.

Step 3: The squeezing parameter, s, is the only remaining parameter to be learned (though the offsets can be allowed to drift slightly from their initial values as described in Step 4.b).

This is performed in two stages. In the first stage, s is set to 0.0 and then gradually increased in steps of size η_s until the total error:

$$E = \sum_{p=1}^{P} \sum_{y=1}^{Y} \left| t_{p,y} - o_{p,y} \right| \qquad \text{...(I.2)}$$

is less than the threshold value:

$$T = \sum_{p=1}^{P} \sum_{y=1}^{Y} \left| t_{p,y} \right|. \qquad \text{...(I.3)}$$

At this stage the neuron's contribution to modeling the target surface will exceed the errors it is introducing. Unless there are ambiguous training data (that is if there are two or more training patterns with the same input vector but different output vectors), it is guaranteed that E can always be reduced below T. That is:

given that $v_y = t_{\hat{p},y}$ from Step 2

and thus $E = \sum_{p=1}^{P} \sum_{y=1}^{Y} \left| t_{p,y} - t_{\hat{p},y} e^{-s\sum_{j=1}^{J}(\alpha_j i_j - c_j)^2} \right|$

then as $s \to \infty$ so $E \to \sum_{p=1}^{P} \sum_{y=1}^{Y} \left| t_{p,y} \right| - \sum_{y=1}^{y} \left| t_{\hat{p},y} \right| = T - \sum_{y=1}^{y} \left| t_{\hat{p},y} \right|$

In other words, as s is increased the radial-Gaussian functions it produces at the outputs become increasingly localized until $\hat{p}$ is essentially the only pattern on which they have any influence. Typically, E will drop below T before the radial-Gaussian function becomes localized to training pattern $\hat{p}$. Experience indicates that, generally, the performance of the training algorithm is not very sensitive to the starting value of the parameter η_s.

Step 4a: The second stage training of s is based on a gradient descent technique. The error gradient is given by:

$$\frac{\partial E}{\partial s} = \sum_{p=1}^{P} \sum_{y=1}^{Y} \pm - o_{p,y} \tau_p \qquad \text{...(I.4)}$$

where the sign of the expression $-o_{p,y}\tau_p$ is reversed if $o_{p,y} < t_{p,y}$.

The parameter s is increased by η_s if $\frac{\partial E}{\partial s}$ is negative, and is decreased by η_s if $\frac{\partial E}{\partial s}$ is positive. Each occasion the sign of $\frac{\partial E}{\partial s}$ changes, the value of η_s is reduced by the factor β_s (where $0 < \beta_s < 1$; a value of 0.5 was adopted for this paper). When the sign of $\frac{\partial E}{\partial s}$ does not change, the value of η_s can be increased, but by a factor less than $1/\beta_s$ to accelerate convergence (a value of 1.1 was adopted for this paper). Adjustments are made to s in this manner until the relative reduction in error at each iteration $\left| (E_{previous} - E_{current}) / E_{previous} \right|$ goes below a given limit (a value of 0.00005 was adopted for this paper).

Step 4b: Adjustments to the offsets c_j are made simultaneously with the adjustments to s in Step 4a. This enables the radial-Gaussian function to drift slightly from its original position, overcoming the limitation that the centroids of these functions can only be located where there is a training pattern. Adjustments are made using essentially the same error gradient descent technique used in Step 4a. In this case, the error gradient is given by:

$$\frac{\partial E}{\partial c_j} = 2s \sum_{p=1}^{P} \sum_{y=1}^{Y} \pm - (\alpha_j i_{p,j} - c_j) o_{p,y} \qquad \text{...(I.5)}$$

where the sign of the expression $-(\alpha_j i_{p,j} - c_j) o_{p,y}$ is reversed if $t_{p,y} < o_{p,y}$.

Based on the sign of $\frac{\partial E}{\partial c_j}$, the parameter c_j is either increased or decreased by η_{c_j}. This parameter is reduced (again typically by halving) each occasion the sign of the error gradient changes. When the sign of the error gradient does not change, the parameter c_j can be increased. The values of η_{c_j} are kept small relative to the distances between the training patterns in the input plane - this is to prevent the introduction of large errors that can result if the radial-Gaussian function is allowed to jump a long way from the training pattern around which it was originally focused.

Step 5: Having developed the hidden neuron, the target output vectors of each training pattern are replaced by their corresponding residual target vectors, that is:

$$\vec{t}_p^{\,new} = \vec{t}_p^{\,old} - \vec{o}_p$$

thus producing a new set of training patterns. These revised training patterns infer a new target or error surface at each output, representing the difference between the initial target surface and that produced by the network. The hidden neuron is then temporarily switched off and a new hidden neuron is added to the network. This hidden neuron is then taught on its own using the modified set of training patterns, following steps 2 to 5 inclusive. The process of adding successive hidden neurons as such is repeated until the network has learned the training patterns to within a prescribed degree of accuracy.

Step 6: Once training has been completed, all hidden neurons are switched on - the resultant network will be able to reproduce the training patterns to within the prescribed degree of accuracy.

APPENDIX II: Extended Hamming Networks (EHAM)

II.1 EHAM Network Architecture and Operation

The main part of an EHAM network architecture, illustrated in Figure II.1, comprises an input, output, and single hidden layer of neurons. A layered feedforward configuration is adopted with each output neuron connected to its own set of hidden neurons. The main network provides a direct mapping from a vector of binary inputs, $\vec{i}$, to a vector of binary outputs, $\vec{a}''$. A regulator neuron, feeding information back from the output neurons to the hidden neurons, is included to ensure that only a prescribed number of output neurons fire (usually one).

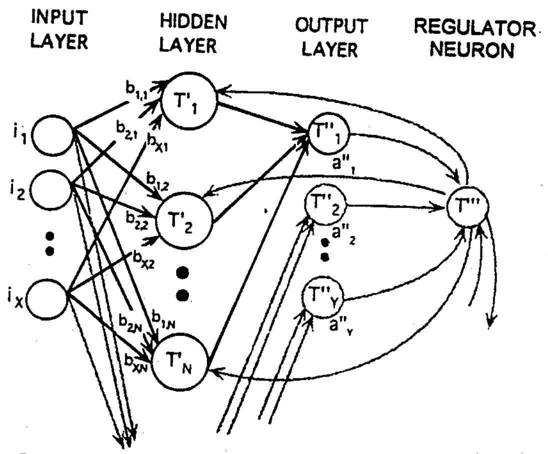

Figure II.1: Architecture of Extended Hamming Network.

Binary values, i_X, presented at each of the input neurons are transmitted across the connections to the hidden neurons. Associated with each of these connections is a binary value $b_{X,n}$ which is applied to the transmitted value in an exclusive-or (XOR) operation. The results of this process over all connections to a hidden neuron are summed, providing the Hamming distance H_n between the input vector $\vec{i}$ and the template vector $\vec{b}_n$. Formally, this can be expressed as:

$$H_n = \sum_{x=1}^{X} (b_{x,n} \ XOR \ i_x) \qquad \text{... (II.1)}$$

If the value H_n is less than or equal to the threshold value T'_n (T'_n is a positive integer in the range 0 to X inclusive where X is the number of inputs to the network), the hidden neuron will fire by outputting binary 1, otherwise it will output binary 0. That is:

$$\begin{aligned} &If\ H_n \le T'_n\ then\ a'_n = 1; \\ &if\ H_n > T'_n\ then\ a'_n = 0. \end{aligned} \qquad \text{... (II.2)}$$

Each output neuron sums the values a'_n generated at the set of hidden neurons to which it is connected:

$$S''_y = \sum_{n=1}^{N} a'_n \qquad \text{... (II.3)}$$

and compares the result with a threshold value T''_y (T''_y can be any positive integer including zero). Firing of an output neuron works the opposite way round to that of the hidden neurons:

$$\begin{aligned} &If\ S''_y > T''_y\ then\ a''_y = 1; \\ &if\ S''_y \le T''_y\ then\ a''_y = 0. \end{aligned} \qquad \text{... (II.4)}$$

The regulator neuron sums the values generated by the output neurons:

$$S''' = \sum_{y=1}^{Y} a''_y \qquad \text{... (II.5)}$$

and compares the result with a threshold value T''' (where T''' is any integer between 1 and X-1). Firing of the regulator neuron operates as follows:

$$\begin{aligned} &\textit{If } S''' < T''' \quad \textit{then } a''' = -1; \\ &\textit{if } S''' = T''' \quad \textit{then } a''' = 0; \\ &\textit{if } S''' > T''' \quad \textit{then } a''' = 1. \end{aligned} \qquad \ldots \text{(II.6)}$$

The value generated at the regulator neuron is transmitted back to the hidden neurons where it is added to the previous summed input, modifying Equation II.1 so that:

$$H_{n,j+1} = H_{n,j} + a_j''' \qquad \ldots \text{(II.7)}$$

The network operates in this recursive manner until the activation at the regulator neuron, a''', either becomes equal to 0 (at which point the number of output neurons firing will be equal to T''') or flips from -1 to +1.

II.2 Training Algorithm

The EHAM training algorithm falls into the supervised class, making use of a set of P training patterns each comprising an input vector $\vec{i}_p$ and corresponding target output vector $\vec{t}_p$. Training operates on the main section of the network, exclusive of the regulator neuron. Since each output neuron has its own set of hidden neurons, training can be more conveniently described for the one output neuron case. If there are two or more outputs the training for each is performed in parallel but is otherwise the same as in the single output neuron case.

Training proceeds by developing one hidden neuron at a time. Each successive hidden neuron is trained on the component of the problem its predecessors failed to learn, that is, the remaining error. Hidden neurons are added as such until the remaining error is zero. For a network comprising one output neuron, the procedure is as follows:

Step 1: Initialize the network.

The network is first set-up with the required number of input neurons, an output neuron, but with no hidden neurons. The threshold T'' on the output neuron is set to 0.

For the network to output the correct result for the $p-th$ training pattern, the value generated for S_p'' should be greater than the threshold T'' if the target output is 1, or it should be less than or equal to T'' if the target output is 0. An error vector $\vec{\varepsilon}$ is established to measure the distance from these objectives for each training pattern:

$$\varepsilon_p = t_p + T'' - S_p''. \qquad \ldots \text{(II.8)}$$

The objectives then become:

$\varepsilon_p \leq 0$ if $t_p = 1$; and

$\varepsilon_p \geq 0$ if $t_p = 0$.

Since the network starts with no hidden neurons and $T'' = 0$, each element of the error vector is initialized to $\varepsilon_p = t_p$. The total error in the network is measured as:

$$E = \sum_{p \subset \Pi_\varepsilon} |\varepsilon_p| \qquad \ldots \text{(II.9)}$$

where Π_ε is the set of all training patterns for which the network produces an incorrect output.

Step 2: Determine whether most errors are for training patterns with a target of 1 or a target of 0.

A count is made of the number of training patterns P_{high} in the set Π_{high} (comprising all training patterns that have a target of 1 but the network produces 0) and the number of training patterns P_{lo} in the set Π_{low} (comprising all training patterns that have a target output of 0 but the network produces a 1). This is undertaken by checking how many of the elements in the error vector $\vec{\varepsilon}$ do not satisfy the $\varepsilon_p \leq 0$ and $\varepsilon_p \geq 0$ requirements respectively (see Step 1). A hidden neuron is then added to the network, as described in the following steps, to reduce the number of patterns in either Π_{high} or Π_{low} whichever is the largest set.

Step 3: Add a hidden neuron to the network and establish the values of its template vector $\vec{b}_n$.
If the number of training patterns in Π_{high} is greater than or equal to the number in Π_{low} then follow Step 3a otherwise follow Step 3b.

Step 3a: The center of gravity $\vec{\gamma}$ (measured in the input domain) of all training patterns in the set Π_{high} is determined. Each element of the vector $\vec{\gamma}$ is calculated simply as:

$$\gamma_x = \left(\sum_{p \subset \Pi_{high}} i_{p,x} \right) \Big/ P_{high} \qquad \ldots \text{(II.10)}$$

The training patterns in set Π_{high} are then searched to determine which is nearest to the center of gravity $\vec{\gamma}$. The nearest training pattern $\hat{p}$ is that where $\sum_{x=1}^{X} \left| i_{p,x} - \gamma_x \right|$ is smallest. A hidden neuron is added to the network and its template vector is set equal to the input vector of training pattern $\hat{p}$, that is $\vec{b}_n = \vec{i}_{\hat{p}}$.

The rationale behind this step is that the center of gravity will indicate the corner of the input space towards which the erroneous training patterns are clustered, and thus focusing the hidden neuron at a corner near this point facilitates the removal of a significant number of errors.

- OR -

Step 3b: The parameters $\vec{\gamma}$ and $\hat{p}$ are determined as described in Step 3a except that P_{low} and Π_{low} are used in place of P_{high} and Π_{high}. A hidden neuron is added to the network and its template vector is set equal to the logical NOT of the input vector of training pattern $\hat{p}$, that is:

$$\vec{b}_n = NOT\ \left(\vec{i}_{\hat{p}}\right). \qquad \ldots \text{(II.11)}$$

Following this, a value of 1 is added to the output threshold T'' and, so that Equation II.8 is maintained, a value of 1 is added to each component of the error vector $\vec{\varepsilon}$.

The rationale here is essentially the same as that of Step 3a. In this case, however, the erroneous training patterns are causing the output neuron to fire when it should not - the opposite of the situation in Step 3a. The hidden neuron, therefore, needs to be set up in a way that suppresses (rather than promotes) the possibility of the output neuron firing when one of the erroneous training patterns are presented to the network. This is achieved by focusing the hidden neuron on the opposite corner of the input space to that where the erroneous training patterns are clustered (see Equation II.11) and incrementing the output threshold as described.

Step 4: Determine an optimum value for the hidden neuron's threshold T'_n
All possible values for T'_n (from 0 to X inclusive) are checked to see which will provide the greatest reduction in total error E in the network (see Equation II.9). This can be achieved with just one pass through the training patterns.

Each training pattern is examined to see if its contribution to the total error would be reduced or increased by the addition of the hidden neuron. If the $p-th$ training pattern's target output is 1 and $\varepsilon_p > 0$ then its contribution to the total error E will be reduced if the threshold T'_n is set at least equal to the following Hamming distance:

$$H_{p,n} = \sum_{x=1}^{X} \left(b_{x,n} \; XOR \; i_{p,x}\right).$$

A score of 1 is therefore added to a variable, $\Delta_{k=Hb,n}$, associated with this Hamming distance. Conversely, if the $p-th$ training pattern's target output is 0 and $\varepsilon_p \leq 0$ then its contribution to the total error will increase if the threshold T'_n is set at least equal to the Hamming distance $H_{p,n}$. In this case, a score of -1 is added to the variable $\Delta_{k=Hb,n}$.

The cumulative scores for all possible Hamming distances are then calculated:

$$C_{m=0..X} = \sum_{k=0}^{m} \Delta_k$$

Each cumulative score C_m provides a measure of the reduction in error that can be achieved by setting the threshold T'_n to a Hamming distance of m. The value of m at which the highest positive cumulative score is achieved is therefore the value selected for the threshold T_n.

If there are two or more optimum threshold values the choice between them is arbitrary, though in this paper the smallest value is adopted.

Step 5: Adjust the error vector

The values of the error vector $\vec{\varepsilon}$ are adjusted to take account of the contribution of the hidden neuron. This requires each error ε_p to be reduced by a value of 1 if the training pattern causes the hidden neuron to fire, that is:

$$\varepsilon_p^{new} = \varepsilon_p^{old} - a'_{p,n}$$

Further hidden neurons are added to the network by repeating Steps 2 to 5 inclusive until the total error E reaches 0.