How to Improve the Reliability of Artificial Neural Networks

Amanda J.C. Sharkey and Noel E. Sharkey
Department of Computer Science,
University of Sheffield, U.K.

Abstract

Although Artificial Neural Networks (ANNs) have been successfully applied to a wide variety of complex problems, they rarely exhibit 100% accuracy on independent Tests. If they are to be employed effectively in real-world industrial applications, particularly in the safety-critical industries, their reliability must be assured. This paper reports research on the use of the software engineering concept of *reliability through redundancy (diversity)* to develop reliable systems by combining diverse ANNs. Four different types of diversity are identified and discussed. Type 1 and Type 2 guarantee reliability with a simple majority vote. Type 3 improves performance and, with selection strategies, may create reliable ANN systems. Type 4 enhances performance, but cannot lead to fully reliable systems. The Type classification is used to assess two new *Transformation* methodologies in three applications studies. The Type 1 and Type 2 diversity exhibited in these studies demonstrate the promise of the methodologies for the creation of reliable ANN software.

Keywords: Artificial Neural Networks, reliability, redundancy, diversity, transformations.

Dr Amanda Sharkey, Department of Computer Science, Regent Court, Portobello Street, University of Sheffield, Sheffield, S1 4DP, United Kingdom. Tel: 0114 2825568 Email: amanda@dcs.shef.ac.uk Fax: 0114 2780972

Professor Noel Sharkey, Department of Computer Science, Regent Court, Portobello Street, University of Sheffield, Sheffield, S1 4DP, United Kingdom. Tel: 0114 2825550 Email: noel@dcs.shef.ac.uk Fax: 0114 2780972

Software engineers make use of redundancy in their efforts to improve the reliability of software programs. The idea of reliability through redundancy originates in investigations of N-version programming, where the approach is related to the Triple Modular redundancy employed in hardware development. Essentially alternative versions of a software program are developed, and harnessed together. The versions can then be executed in parallel, and either checked for agreement before proceeding (fail-safe operation), or passed to a voter which determines the output. N-version programming is used in real systems such as the Airbus Industry A310 aircraft which makes use of dual programming in the slat and flap control system [1]. Similarly, dual programming is employed for point switching, signal control and traffic control in the Gothenburg area by Swedish State Railways.

The aim of N-version programming, is to produce program versions which show a minimum number of coincident failures. Ideally, when an input fails on one version, it will succeed on other versions, and the correct output will still be obtained. The question then is how this situation may best be achieved. It used to be assumed that good results would be obtained through an insistence on the independent development of programs [2]. The idea here was that although a programmer will tend to make the same or similar errors when working on different versions of a program, this tendency will be reduced if the programs are developed by independent programmers. This, however, is not necessarily the case; Knight and Leveson [3] report a study in which independently developed programs exhibit shared errors: even when working independently, people tend to make the same mistakes when solving a difficult intellectual problem. For this reason, Littlewood and Miller [4] advocate the use of diverse methodologies in the development of software programs, where the aim is to reduce the number of coincident failures.

An approach that is analogous to N-version programming has been adopted for the development of more reliable Artificial Neural Networks (ANN) systems. Rather than providing one neural net solution to a problem, a set of nets can be combined to form a neural net *system* that performs better than any one of the nets on its own [5][6]. However, it is still the case that the mere existence of redundancy does not guarantee an increase in reliability. In order to use redundancy to improve the performance of ANNs, we need to know (i) what kind of diversity will lead to improved performance, and (ii) what is the best way of creating a set of nets that show this kind of diversity.

Four different types of diversity for ANNs are identified and discussed in this paper. Type 1 and Type 2 guarantee reliability with a simple majority vote. Type 3 improves reliability and, with a sophisticated selection strategy, may create reliable ANN systems. Type 4 can improve reliability, but cannot lead to fully reliable ANN systems. This discussion is followed by the presentation of two new *transformation* methodologies for creating diverse ANNs. These are demonstrated in the context of three case studies; Engine Health Monitoring; 'The Launch Interceptor Problem'; and 'Compare Length', a function for comparing the length of a line (as specified in (x,y) coordinates) with a varying length parameter.

1 Diversity and Artificial neural networks

Neural computing involves a very different approach to that taken in traditional symbolic computing. Ideally, an ANN is trained on a function f from a set of inputs, V, to a set of outputs, O, such that it assigns to each $v \in V$ a unique output value $f(v) \in O$. This is not difficult, for example, using a universal approximator such as backpropagation in a multilayer net [7], provided that the total set, P, of ordered pairs for f is used for training the net, where $P = \{(v, o) \mid v \in V, o \in O\}$.

When it is not practical to train an ANN on all $p \in P$ in a reasonable time it is trained on a sample, $S \subset P$. However, any sample, S, may be a subset of at least one other set of ordered pairs P' that are the range and domain of another function g, that is, $S \subset (P \cap P')$. Thus, the ANN may actually induce the function g from S. The problem then is to determine how well g approximates f. This is the *generalisation* problem.

The worst case for a trained ANN occurs when $S = P \cap P'$. Then the ANN is said to have memorised S and does not exhibit generalisation with respect to f. If $^{\#}(S) < {}^{\#}(P \cap P')$, where $^{\#}(S)$ is the cardinality of the set S, then the ANN is said to exhibit generalisation on f. The degree of this generalisation can be determined exactly by the size of the intersection, $P \cap P'$, relative to the size of P,

$$\kappa = \frac{^{\#}(P \cap P')}{^{\#}(P)} \times 100 \tag{1}$$

However, if $^{\#}(P)$ and $^{\#}(P \cap P')$ are unknown and it is not practical to calculate them, κ must be estimated. This is done by substituting a known Test set, $T \subset P$, where $T \cap S = \emptyset$, for P in Equation 1[1].

[1] Such estimates must be treated with care since it is possible, for example, to obtain

The generalisation performance of an ANN is rarely 100% correct for interesting functions. However, it has recently been shown that it is possible to improve generalisation by combining a number of different trained ANNs, e.g. [5]. Such improvement relies on exploiting the diversity inherent in a set of ANNs. In what follows, we define four types of diversity, and assess their usefulness both for the development of reliable ANN systems, and for the improvement of generalisation performance. As will become apparent, the extent to which reliability can be improved by combining neural net solutions, depends on the *type* of diversity exemplified in a set of nets.

We define four Types of diversity in terms of whether any or all of the 3 conditions below are met by a collection of ANNs and that the collection meets the minimum requirement for diversity (see below). In all cases it is assumed that $P_i \neq P,\ i = 1, 2, ..., n$, where n is the number of ANNs. The conditions are,

$\mathcal{C}$ 1. **P is covered by the P_i where coverage is defined as,**

$$P \subset \bigcup_{i=1}^{n} P_i$$

$\mathcal{C}$ 2. **there are no coincident failures**

$$\bigcap_{i=1}^{n} P \setminus P_i = \emptyset$$

where "\" is the set-theoretic difference operator

$\mathcal{C}$ 3. **the majority of the ANNs is always correct**

$$D_{max} < n/2$$

where

$$D_{max} = \max_{p_j \in P} \ {}^{\#}(\{P \setminus P_i \mid p_j \in P \setminus P_i\})$$

In other words, D_{max} is the maximum number of coincident errors for any one p. i.e. the set of sets for which p_j is in the difference $P \setminus P_i$.

The four different Types of diversity are defined in Table 1. The rows show the presence or absence of the three conditions and the columns define the Types.

a generalisation correct performance of 100% when, $P \cap P' = T \cup S$. The choice of Test set is thus crucial.

Conditions	Diversity Types 1	2	3	4
$P \subset \bigcup_{i=1}^{n} P_i$ *P is covered by the P_i*	$\surd$	$\surd$	$\surd$	X
$\bigcap_{i=1}^{n} P \setminus P_i = \emptyset$ *there are no coincident failures*	$\surd$	X	X	XX
$D_{max} < n/2$ *the majority is always correct*	$\surd\surd$	$\surd$	X	XX

Table 1: The four types of diversity, where $\surd$ indicates the presence of a condition, X indicates the absence of a condition, $\surd\surd$ and XX indicate that the presence or absence of a condition, respectively, necessarily follows from the presence or absence of the other conditions in the Type.

A Minimum Requirement: Of course, if $\bigcup_{i=1}^{n} P \cap P_i = S$, assuming that all nets are trained on the same sample, S, then there would be no diversity. On this point, Partridge and Griffith [9] provide a definition of what they call *useful diversity* that can be applied here as a definition of *minimum diversity.* Translated, for terminological consistency, the minimum requirement for diversity is,

$$\#\left(\bigcup_{i=1}^{n} P \cap P_i\right) \quad > \quad \#(P_{max})$$

where $\#(P_{max})$ is the largest set resulting from intersecting P with each of the P_is. In other words, there are a least two ANNs such that there is at least one correct input generalisation in each that is not shared by the other.

Type 1 Diversity is the ideal for reliable software. The function is covered by the collection of ANNs and none of the failures are coincident. It follows that the majority of nets will always be correct since only 1 will ever fail on a randomly chosen input. This type requires $n > 2$.

Type 2 Diversity does not meet all of the criteria of Type 1, i.e. it allows coincident failures. However, since the majority is always correct, Type 2 will also lead to reliable software. This type requires $n > 4$. It is possible that a Type 2 system is *upwardly mobile* in that removing some of the ANNs could eliminate the coincident failures and lead to a Type 1.

Type 3 Diversity makes it more difficult to construct reliable software, since a simple Majority vote will not always result in the correct answer. However, the collection of ANNs do, between them, cover the function, and it might be possible for a sophisticated voter to weight the outputs of different nets in such a way that either the correct answer was always obtained, or at least, generalisation was improved. It is possible that a set of ANNs that exhibits Type 3 Diversity may contain subsets of ANNs with either Type 1 or Type 2 Diversity and thus, like Type 2, Type 3 may be upwardly mobile.

Type 4 Diversity can never be reliable since P is not entirely covered by the ANNs and thus there are failures, $\mathcal{F}$ that are shared by all of the ANNs, where $\mathcal{F} = \{p \in P \mid p \notin \bigcup_{i=1}^{n} P \cap P_i\}$. However, although Type 4 can never be upwardly mobile, it can still be used to improve generalisation. The different degrees of usefulness for Type 4 can be seen by its division into three subtypes paralleling Types 1 to 3 above: **4.1** (i) there are no coincident failures outside of $\mathcal{F}$; (ii) when $\mathcal{F}$ is excluded, the majority of ANNs are always correct. **4.2** (i) there are coincident failures outside of $\mathcal{F}$, but (ii) when $\mathcal{F}$ is excluded, the majority of ANNs are always correct. **4.3** is the same as 4.2 except that even when $\mathcal{F}$ is excluded, the majority are not always correct.

One important proviso in discussing the Types is that with real-world tasks, it can be extremely difficult (and often impossible) to precisely determine the Type of Diversity of an ANN system (apart from Type 4). This is because, as discussed earlier, it is usually not possible to test all of P within a reasonable time period. Thus we must rely on an estimation of Type by using a Test set T, where estimated Types are defined by substituting T for P in $\mathcal{C}$ 1 through $\mathcal{C}$ 3. Regardless, it is still better to have an estimated Type 1 than an estimated Type 3 on the same Test. Reference to types in the following all refer to estimated types unless otherwise stated.

2 Transformation methodologies for diversity

Previous studies on the creation of ANN diversity have discussed the manipulation of a number of different factors including training parameters [5], initial weight configurations [8], network architecture [9], training samples [6], and contrasting measures [10]. These methods are all examples of extensional programming, as opposed to conventional programming [11], and they have achieved a mixed degree of success in the improvement of generalisation although none have been shown to achieve an estimated Type 1 or Type 2 diversity.

One of the more promising of these methodologies for creating diversity is the use of different samples for training different ANNs. This increases the chances of diversity by relaxing the training constraint that all of the sets of ordered pairs for the g_is must intersect because they all contain S as a subset (in the extreme, it is possible that $\bigcap_{i=1}^{n} P_i = \emptyset$). However, although this method shows potential, it has not been shown to achieve Type 1 or 2 Diversity.

A related methodology, contrasting measures, has recently been reported [10], in which the ANNs were trained on different interpretations of the input states from a ship's engine, rather than on different training samples. Two contrasting measures were taken from each engine state, *temperature* and *pressure*, on each time cycle and used as the training input to different groups of ANNs. From the perspective of the ANNs, they were effectively being trained on different functions. However, from the perspective of the system, the measurement may be though of as an extra processing step after input. That is, the state of the engine, v, is preprocessed by the measurement method, $m(v)$, and then passed as input to the ANN, $g(m(v))$. In comparisons with training sample variation and initial weight variation, it was found [10] that the most statistical independence among solutions was developed by the contrasting measures methodology. More recently, another study [12] has used the interaction of varying sample set and contrasting measures to create a Type 1 system.

The two new transformation methodologies reported here represent a more general development of contrasting measures and sampling variation. In the first methodology, the input elements, v, in the sample set are transformed, $t(v)$, in different ways for training each ANN. The ordered pairs for the training samples for the ANNs are now (t(v),o) and the functions extracted by the ANNs are, $g_i(t_i(v)),\ i = 1, 2, ..., n$. Figure 1 illustrates the method with one normal and two transformed input routes to the ANNs.

In the second methodology, the ANN is first trained on a sample of the target function. Then diversity is achieved by transforming the input space through different angles of rotation. This is particularly appropriate for geometric problems such as that described in Section 2.1.3.

The results from three studies using these new methodologies are reported in the next section of the paper. All studies employ standard Back-Propagation training [13]. The evidence suggests that these are good candidates for providing an effective means of creating systems of ANNs that exhibit either Type 1 or Type 2 diversity.

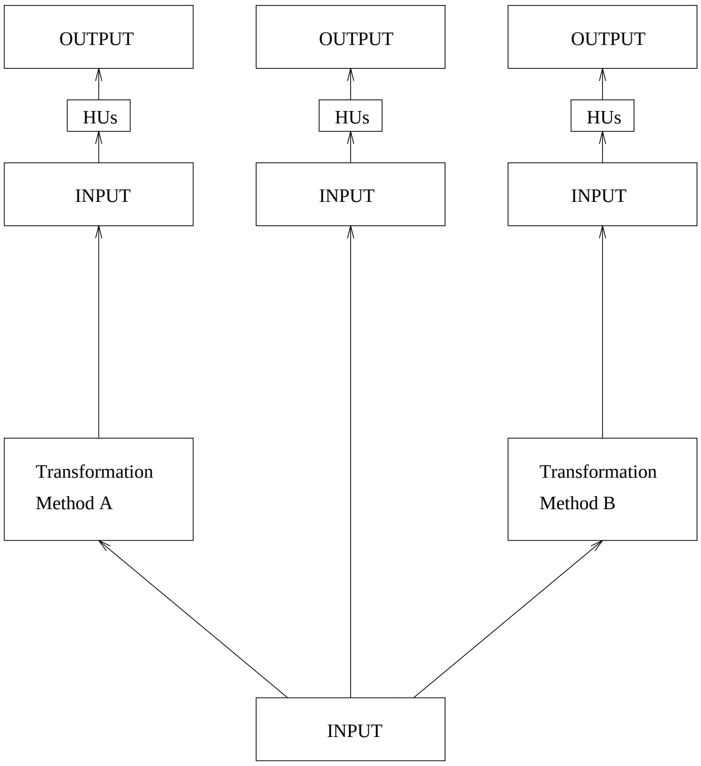

Figure 1: Creating diversity through transformations. (HUs = Hidden units)

2.1 Three case studies

2.1.1 Engine Health Monitoring

In the first case study, the transformation methodology was employed to investigate the possibility of creating a Type 1 system, using 3 ANNs, for online detection and diagnosis of combustion faults in the cylinder of a marine engine. The raw data collection for this problem was undertaken by Gopinath O.Chandroth and is reported in [10]. The ANNs were trained to classify pressure data generated by the MERLIN[2] engine simulator, as examples of (a) Ideal combustion, (b) Advanced injection of fuel and (c) Retarded injection of fuel. During each engine cycle, the simulator provided a measure of pressure in the engine cylinder at fixed intervals of crank angle rotation.

The same sample of 150 training pairs (50 examples of each fault type) was used for training 3 ANNs. Each pair consisted of a 52 element input vector and a 3 element output vector. 414 pairs were reserved as a Test set. All tests were conducted with an error tolerance of 0.1. For one of the ANNs, the input set consisted of the raw pressure data, and for the other two, the input data was preprocessed using two different kinds of transformation, $t_A(v)$ and $t_B(v)$. Thus the trained ANNs would have functions, $g_1(t_A(v))$, $g_2(t_B(v))$, and $g_3(v)$.

[2] Engine simulation software developed by Lloyds Register of Shipping, London

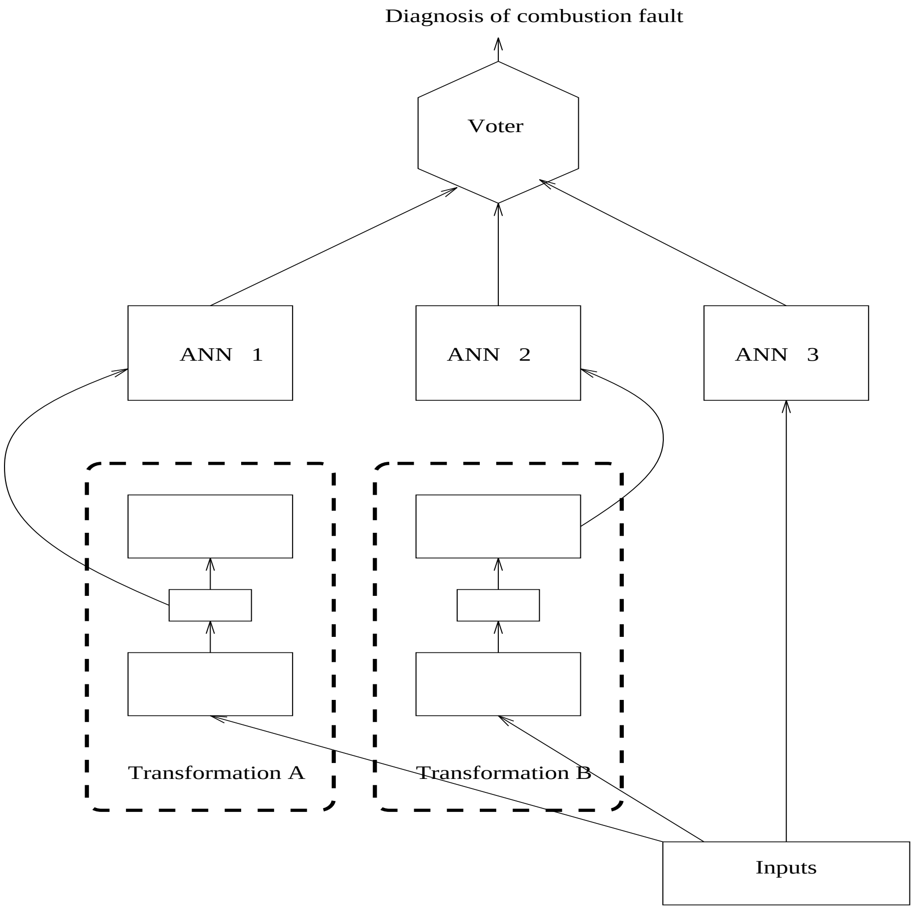

Figure 2: Health monitoring in a ship's engine

Transformation A**:** The pressure data was transformed for input to ANN_1 by using another ANN with 2 layers of weights, called the Transformation Net (TN); shown as Transformation A in Figure 2. In order to develop the nonlinear transformation, the TN was pretrained to autoencode the pressure data by reproducing the 52 element input vector as output in a 52-2-52 ANN. Once trained, only the input layer of weights in the TN was used to transform the input data, as shown in Figure 2. Thus ANN_1 was trained on the classification task with the TN hidden unit activations as input (with dimensions 2-2-3). After training, ANN_1 exhibited 98.1% correct generalisation on the transformed Test set.

Transformation B**:** For ANN_2, the TN was an untrained 52-2-52 ANN with two matrices of random weights as shown as Transformation B in Figure 2. This time the input to ANN_2 was the output from the TN (rather than the hidden unit activation as in Transformation A). ANN_2 was trained with 2 hidden units (52-2-3), and after training, exhibited 95.7% correct generalisation on the transformed Test set.

Original set: ANN_3 was trained on the original 52 element input vector of pressure data with no preprocessing (as shown in Figure 2) (in a net with dimensions 52-2-1). After training, ANN_3 exhibited 99.3% correct generalisation on the Test set.

Effect on Diversity: Although all 3 ANNs exhibited high generalisation performance, none were at 100%. However, when combined into the system shown in Figure 2, they covered the function (at least the Test set), there were no coincident failures, and thus the majority of ANNs were always correct. Thus, it has been shown that this transformation methodology can lead to an estimated Type 1 Diversity.

2.1.2 Launch Interceptor Problem

The second case study concerns the Launch Interceptor (LI) problem that has been employed by software engineers in investigations of diversity (see [3] for a specification). A software implementation of the problem was used as a source of data, to which the transformation method could be applied. In the original specification, the problem is described as part of an anti-missile system, in which a radar image, together with a set of conditionals, is classified as one to which the correct response is either a 'launch' or 'no launch' decision. It was designed to provide an example of a realistically complex safety critical problem, and is a problem with many parts, as shown in Figure 3. In the current investigation, the LI problem was implemented as a Hybrid system, with most of the work being carried out by parts of a standard Gold program. The ANNs were trained on a small part of the problem by taking the fifteen element 'final unlocking vector' produced by the gold program, and training nets to classify this as an example of either a launch, or no launch. See Figure 3 for a diagram of the whole system.

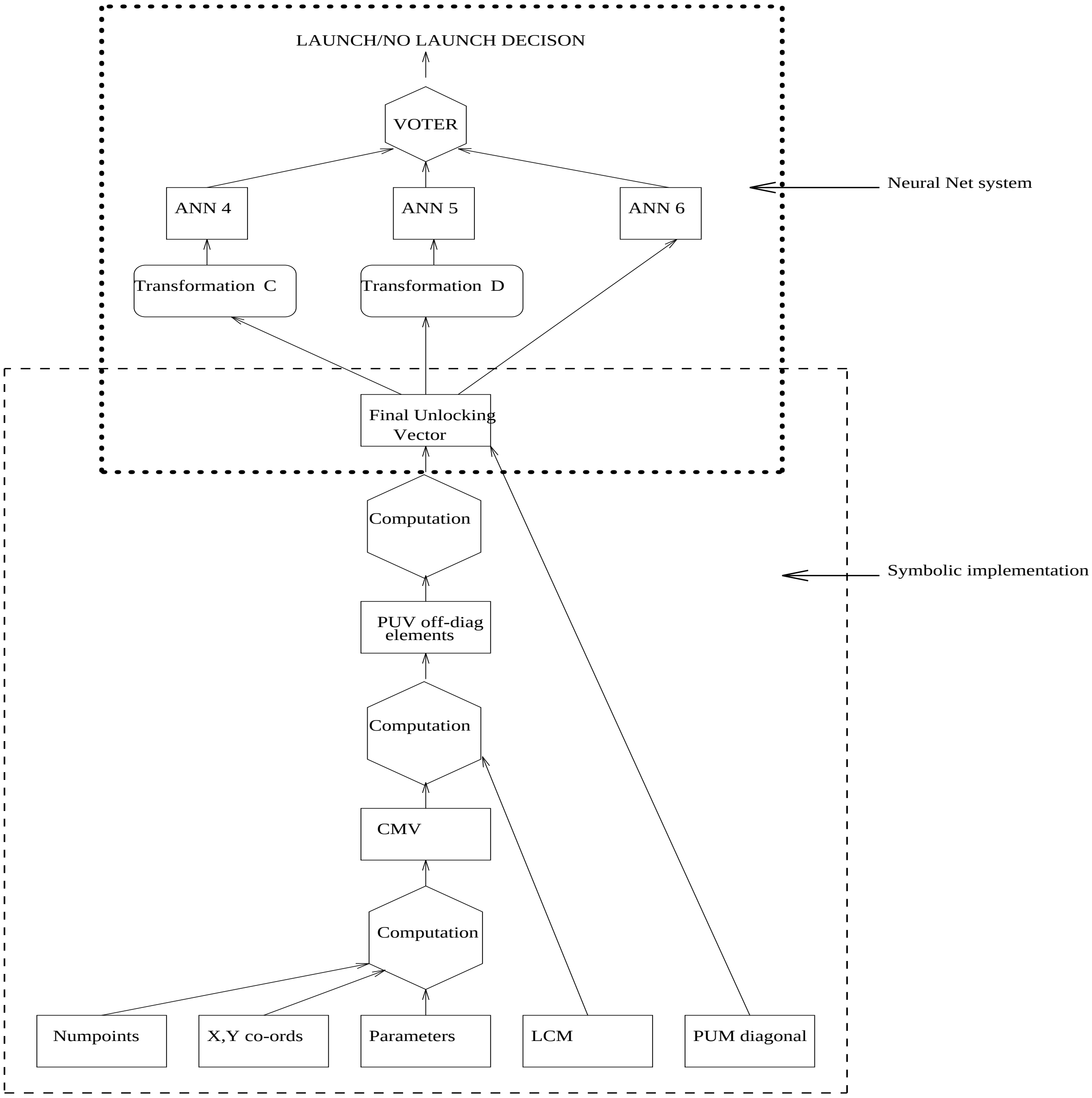

Figure 3: Launch Interceptor Problem

The same training sample of 500 patterns (500 examples of a fifteen bit binary input vector, and a one bit output unit that classifies the data as either a *Launch* or *No Launch* example), was used to train a set of 3 ANNs. 5000 examples were reserved as a Test set, and all tests were conducted to a tolerance of 0.1. For one ANN, the input consisted of the raw 15 bit input vector. For the other ANNs, the input data was preprocessed using two transformations, $t_C(v)$ and $t_D(v)$.

Transformation C: The training sample was transformed for input to ANN_4 by pretraining a 15-10-1 TN (Transformation C, in Figure 3) to classify each input in terms of whether or not it contained at least one example of the binary string, '10101'. Once trained, the input weights for the TN were used to transform the original inputs for training ANN_4, on the launch/no launch decision. After training with dimensions 10-2-1, ANN_4 exhibited a level of 99.9% correct generalisation to a similarly transformed version of the Test set.

Transformation D: For the input to ANN_5, a TN was pretrained to classify the inputs according to whether or not there were more than 9 bits on in the input vector. The input weights of the TN (15-2-1) were used to transform the inputs for training ANN_5 on the launch/no launch decision. The trained ANN_5 (dimensions 2-4-1) yielded 98.5% correct generalisation on the transformed Test set.

Original set: ANN_6 (dimensions 15-2-1) was trained, without input preprocessing, on the launch/no launch decision. When tested on the Test set, it showed a 99.1% correct generalisation.

The effect on diversity: All three ANNs showed a level of generalisation that was better than 98%; but all made some errors on the Test set. When combined to form the system shown in Figure 3, they made no coincident errors, and they covered the function. Therefore a majority voter produced the correct answer for all the examples in the Test set. This is another example of Type 1 diversity.

2.1.3 Compare-Length Problem

The third and final case study is of the Compare-Length function which is a 'scaled-down' [3] version of one of the components of the LI problem known as Launch Interceptor Condition 1 (LIC 1) (See also [9]). In this scaled-down version, the net takes as input, two pairs of (x,y) coordinates, and a length

[3] This function differs from the original LIC 1 specification in that (i) it only operates in the range between 0 and 1 - the original was over all real numbers; (ii) Compare-Length makes only one comparison at a time, whereas LIC 1 returns True only if there is at least one Greater-Than out of a set of up to 100 (x,y) coordinates; (iii) LIC 1 used a function REAL COMPARE to make comparisons to 6 significant digits - this was to avoid differences attributable to machine precision.

parameter, LENGTH. The output is 1 or 0 depending on whether the line specified by the coordinates is longer than LENGTH.

Training: An ANN was trained on 9408 patterns. 2,352 patterns were created by varying the x and y coordinates in steps of 0.13 along the x and y axis; the number of patterns was increased to 9408 by pairing these with 4 different length parameters (0.25, 0.5, 0.75 and 1). The ANN, of 5-6-1 dimensions, was trained to a tolerance of 0.4.

Transforming and testing: A Test set of 8280 patterns was created by varying the coordinates in steps of 0.17 along the x and y axes, and the length parameter in steps of 0.1, up to 1.4 (and removing any pairs of points which had a distance between them that was closer to the length parameter than any of the examples used in the training set). The ANN exhibited 99.8% correct generalisation for the Test set.

Since this is a geometrical problem that does not depend on orientation within the input space, rotating the space does not make any difference to the length of a line. Diversity was thus created by rotating the input space through 6 different angles about the point (0.5,0.5) on presentation of each member of the Test set. The 6 angles of rotation and their percentage correct generalisations on the Test set were: 50° (99.49%); 100° (99.65%); 150° (99.52%); 200° (99.43%); 250° (99.65%); 300° (99.75%).

Effect on diversity: The diversity created by rotating the Test set was of estimated Type 2. Although there were coincident failures, there was coverage of the function, and the majority of ANNs was always correct. In a real parallel system, the implementation would use 7 copies of the ANN (one for each rotation and one for the original). It should be noted that post-hoc testing indicated that 17 different Type 1 systems (with 3 rotations in each) could be extracted from amongst the rotations. However, given that these are all estimated Types, it is difficult to tell whether a 7-net estimated Type 2 system would outperform an 3-net estimated Type 1 system in the long run.

3 Discussion and Conclusions

The advantages of ANN training over more conventional types of programming can most often be seen in real-world problems for which a functional specification is difficult. With a learning rule and examples, an ANN can induce complex interacting rules that cannot always be readily identified.

The disadvantage is that, due to the nature of their training, ANNs rarely exhibit 100% generalisation performance. The starting assumption of this paper was that an important way of improving the performance of ANNs is to combine them to form an ANN system. Such improvement relies on an exploitation of the diversity amongst the ANNs.

Previous research [9] had defined the notion of *useful diversity*. Taking this to be a minimum requirement, a new classification for four main Types of diversity was presented here. The aim was to enable identification of the kind of diversity that obtains for any collection of ANNs. From this information, decisions can be made about how to best combine the ANNs. For example, for Types 1 and 2 a majority vote is sufficient, whereas for Type 3 a more sophisticated selection strategy [9] or some sort of intelligent voter would be required.

Another advantage of the classification is that it can help to identify systems that are candidates for promotion from one type to another. This was shown in the Compare-Length study where a Type 2 system was found to contain several Type 1 systems. Clearly, it would be more beneficial to promote Type 3 or 4 systems. For a Type 4 system, the best that can be expected is an improvement in generalisation. However, it may be possible to promote a system within Type 4 (e.g. from 4.3 to 4.1) by excluding some of the ANNs from the combination. It may even be possible to promote a Type 4 system by finding other means to complete the coverage of the target function.

The possibility of achieving Type 1 and Type 2 diversity in practice, is demonstrated through three case studies. The generalisation results for all of the ANNs in the studies was impressive, ranging from 95.7% to 99.9% (all were $> 98\%$ except for ANN_2). Nonetheless, it was still possible to improve on their performance through the use of the Transformation methodologies. When sets of ANNs were combined into ANN systems, 100% correct performance was achieved on all of the Test sets. The non-linear transformations of the input space used for both the LI and Engine monitoring applications maximised the differences between the failures of the ANNs.

The general Transformation technique is clearly one that could be extended to many other tasks. The particular transformations employed here were chosen fairly arbitrarily and further work could be undertaken to establish a method for matching the types of transformation with the particular task requirements. The investigation of rotational transformations for the geometric problem in Case Study 3 has made a good start in this direction.

In conclusion, the present research testifies to the benefits of applying the software engineering concept of *reliability through redundancy* to Neural Computing. The results from the three studies report here suggest that a promising way of improving the reliability of ANNs is to employ a transformation methodology to create diverse ANNs which can be combined into a systems that exhibits either Type 1 or Type 2 Diversity.

Acknowledgements

This research was funded by EPSRC/DTI grant no.GR/H85427 IED4/1/9301 on the Safety Critical Systems initiative. We thank Gopinath O. Chandroth for his work on diversity, and his permission to use the data for the Health Monitoring of a ship's engine; data he collected for his MSc thesis; Lloyds Register of Shipping, London, for allowing use of their MERLIN engine simulator software; and John Neary, an RA on the EPSRC/DTI project, for coming up with the idea of rotations for the Compare-Length function, and for helpful discussions.

References

[1] Martin, D.J. (1983) Dissimilar software in high integrity applications in flight controls," In *Software for Avionics* (AGARD Conf. Proc. 330), Jan, 1983, pp36-1-36-9.

[2] Avizienis, A. & Kelly, J.P.J. Fault diagnosis by design diversity: Concepts and experiments. IEEE Comput. 17, 8, 67-80.

[3] Knight, J.C. & Leveson, N.G. (1986) An experimental evaluation of independence in multiversion programming. *Trans on Software Eng*, Vol SE-12 no 1.

[4] Littlewood, B. & Miller, D.R. (1989) Conceptual modeling of coincident failures in multiversion software. *IEEE Trans. on Software Engineering*, 15,(12).

[5] Sharkey, N.E. & Partridge, D.P. (1992) The statistical independence of network generalisation: an application in software engineering. In P.G. Lisboa & M.J. Taylor (Eds) *Neural Networks: Techniques and Applications* Chichester, UK: Ellis Horwood.

[6] Partridge, D. & Sharkey, N.E. (1994) Neural computing for software reliability. *Expert Systems*, 11, 3, 167-175.

[7] White, H. (1992) *Artificial Neural Networks: Approximation and Learning Theory*. Blackwell, Oxford, UK.

[8] Sharkey, N.E., Neary, J. & Sharkey, A.J.C. (1995) Searching weight space for backpropagation solution types. In L. Niklasson, M. Boden (Eds) *Current Trends in Connectionism*, 103-121, Lawrence Erlbaum Associates: Hillsdale, New Jersey.

[9] Partridge, D., and Griffith, N. (1995) Strategies for improving neural network generalisation. *Neural Computing and Applications*, **3**, 27-37.

[10] Sharkey, A.J.C., Sharkey, N.E. and Gopinath, O.C. (1995) Diversity, Neural Nets and Safety Critical Applications. In L. Niklasson, M. Boden (Eds) *Current Trends in Connectionism*, 165-178, Lawrence Erlbaum Associates: Hillsdale, New Jersey

[11] Sharkey, A.J.C., and Sharkey, N.E., (in press) Cognitive Modelling: Psychology and Connectionism. In (Ed.) M.A. Arbib *The Handbook of Brain Theory and Neural Networks*, Bradford Books/MIT Press.

[12] Sharkey, A.J.C., Sharkey, N.E., and Chandroth, O.G. (1995) Neural nets and Diversity. In Proceedings of SafeComp '95. Springer-Verlag.

[13] Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986) Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland (Eds) *Parallel Distributed Processing*, vol 1, MIT Press, Cambridge, MA.