The application of neural networks in systems where their failure can result in loss of life or property, or financial damage, must be backed up with techniques to minimise these undesirable effects. Furthermore, to meet the requirements of many statutory bodies such a system must be certified. This paper investigates some of the problems in using neural networks in safety critical applications and highlights similarities and differences with conventional computer systems in how their behaviour might be guaranteed. We believe that neural networks offer a chance to design hardware with inherent reliability properties.

1. Introduction

The purpose of this paper is to highlight a number of problems which currently bar the way towards artificial neural networks (ANN) being used in many applications. These are applications which are in some way critical such that the failure of the (computer) system containing the ANN may result in damage to people or property. Our intention is not to provide solutions to these problems but to indicate that they exist and that work needs to be done. Hence the paper is more of a discussion than a survey of techniques or presentation of results.

The requirements for safe operation in a computer system are orthogonal to its architecture. Therefore, whenever a new architecture is devised it is usually interesting to examine its behaviour when one or more of its components fail. Although not new, neural architectures have largely escaped this scrutiny. This is in part because it has been assumed (incorrectly) that their operational similarity with biological neural systems extends to a degree of tolerance to faulty components. Artificial neural networks can be designed with a degree of fault-tolerance, but it can not be assumed. The related issue of the often erratic behaviour of naturally occurring neural networks has been ignored. This paper considers the behaviour of a system which contains a neural network as an integral component.

Conventional computer hardware is designed without regard to desirable safety critical properties such as simplicity of reasoning for behavioural arguments and tolerance to hardware faults during service. The latter is added afterwards (by redundancy). The
The former is currently a large research area with the application demands (code size and complexity) easily outstripping technology’s ability to prove correct. Ideally we would like to be able to design some computer (or neural network) hardware in such a way that useful safety critical properties arise as a natural consequence of the design. Hence we believe that work on neural networks which are to be included in some sort of safety critical system should address this problem now.

Clearly such a system has desirable and undesirable properties. We would like it to continue to operate correctly. We would not like it to do something unexpected or inconsistent with its operational requirements. In some cases a legal requirement exists that the system be certified. This means that some official body must be satisfied that the system conforms to certain standards as regards safety, and will then take legal responsibility for that system. The various aviation authorities are a good example of this.

Certifying aircraft which contain safety critical computer systems is a complex activity. However, certification where the computer system contains an ANN is almost undressed. As such systems are now being developed it is important that certification issues are addressed.

This paper is divided into four sections in which we briefly consider the following topics: specification, implementation, verification and validation, and reliability. We consider them to be related in the following way. The system specification is a functional description of what it should do. This is transformed (through a number of refinement stages) into an implementation. In order to achieve some confidence that the implementation does what it is supposed to it might be checked for conformance with the original specification and high level requirements. This is verification and validation. Once in operation the system should reliably provide the required service. Acceptable verification and validation procedures with regard to either legal statutes or some official body then lead to certification. The details of the certification procedures and requirements vary considerably between authorities.

2. Specification

The role of the specification is to provide a description of the behaviour of the system to be used as a reference in its design and implementation. In the context of conventional computer systems both formal and informal techniques are used. Different methods tend to be used for hardware and software. So, initially, we might find the tools most suitable for neural networks by asking which (of hardware and software) do neural networks most closely resemble?
The answer to this is far from clear. In both hardware and software a hierarchy of abstractions is involved in transforming the specification into the realisation. The highest level might take the form of a formal notation such as Z, or an informal system such as Jackson System Development. In each case the gross functionality of the system is identified. This is then repeatedly decomposed into simpler units until something simple enough to be implemented arises.

Formal techniques rely on a concrete mathematical basis thereby allowing the system to be expressed as a theory in some calculus. As the theory is refined towards an implementation one can (hopefully) prove that the refinement is equivalent to the higher level. Formal languages are slowly becoming more accepted for system specification. Tools such as CADiZ help in making sure that the specification is consistent with itself. However, most proof efforts are so large that they have to be automated and can involve the production of thousands intermediate proofs and lemmas. This calls into doubt the proof tools. After all, they themselves are large and unwieldy pieces of software. This, however, is a recursive argument and stops when we reach a tool which we trust sufficiently.

A typical neural network specification might be very informal in nature, such as "recognise faulty chromosomes". The refinement might then involve getting a lot a pictures of faulty and healthy chromosomes and using these to train the network. At what point training and specification become separate is not clear. Certainly, example inputs and target outputs are unlikely to constitute, in any trusted sense, a proper specification of the network. This might be contrasted with the more precise specification such as "implement this control law" which one might encounter in the case of a conventional computer system. In general, problem domains in which neural networks are applied consist of ill-defined requirements as given above. This is because their great advantage over conventional computer systems lies in providing a self-adaptive technique capable of determining the necessary configuration of a neural architecture to solve such problems.

3. Implementation

There is currently one major method for implementing neural networks: emulation on a conventional computer system. Other implementations do exist and are more interesting because they exploit the inherent potential for massive parallelism. For example, devices based upon continuous voltage and pulse train analogue models exhibit greater simplicity without detriment to the required computation. Weightless systems can be implemented using RAMs, counters and other logic devices. Each of the above methods of implementation might fail in different ways. These need to be assessed in order to gain an understanding of which might be the most useful in safety critical systems.
In the case of systems based upon analogue (either pulse train or continuous voltage) technologies, there will always be uncertainties in the values of resistors and capacitors. These will also vary with time. One would therefore expect such systems to be tolerant of a certain amount of uncertainty in the network otherwise they would not work at all!

A further benefit is accrued from the massive parallelism in such systems. Because there are many very simple nodes it is expected that some will fail, and probably sooner rather than later. If the network can be trained to deal with such failures as they occur then one would hope that a more reliable system will result.

Weightless systems can exist similar properties. Large digital RAMs are relatively frequently found to be in error due to radiation and other sources. However, we have found that suitable training and care regarding the information density in the memory can have a marked effect upon behaviour in the presence of hardware faults.

We will now consider the general purpose computer implementation of an ANN in software because it is currently the most widely used. The emulation is characterised by a single processor with a nested loop kernel. Inputs are read, preprocessed in some way and fed into the network. The network’s outputs are again processed before going to the output device. Hence we need to consider three main software components: the input preprocessor, the neural network, and the output postprocessor. Firstly, the neural network itself.

Programs to emulate commonly used ANNs tend to be based upon a nested iterative loop structure. Hence there is little complexity in the algorithm. Further, the code is highly reusable. These are rather obvious points that result from viewing the behaviour of the network arising from its connectivity rather than the operation of a single neuron. If effect, this emulation gives us one neuron which we move over the entire network.

This approach means that, because the kernel of the algorithm is small, we can successfully use existing techniques to make sure that its design is correct and highly trustworthy. This can be compared with conventional algorithmic approaches to problem solving where many thousands of lines of code are written, with little hope of their behaviour being accurately known.

We will consider the input and output processes together. These are less regular in structure than the simple nested loop kernel. However, they are likely to be fairly simple, easy to define, and hence relatively simple to build using conventional software engineering techniques. Further, the processing performed commonly employs well-known data manipulation algorithms (e.g. Fourier transform) where trustworthy implementations have
already been documented. Therefore, it would appear to be relatively simple to gain confidence in the software used to emulate an ANN.

The use of TMR and the software techniques outlined above emphasise the point made earlier that the reliability requirement for a safety critical system is met post hoc rather as a natural result of the design.

4. Verification and Validation

Validation is concerned with showing that the implemented system containing the ANN meets some high level abstract requirements. This may be the real reason the system exists, such as "build an aeroplane which is economical and doesn’t have too many accidents". Verification is a process by which correct operation of the system is assured by proving that the implementation is a refinement of the specification. That is, it implements the specification. Hence we are concerned that the specification is consistent with itself, and that it is complete. Attempts to verify conventional computer systems tend to focus on a formal methods as outlined above, or testing.

The former technique assumes that you have a formal language with which to define your ANN. It is not immediately obvious how current formal languages would be useful in this task. An abstraction of the network in terms of its input and output sets would be relatively simple. However, refinement of the specification, which starts to approach the functionality of the network, is more of a problem. This probably reflects the applications into which ANNs are placed, because these are generally not amenable to conventional solution (pattern recognition, for example). In short, we use ANNs because we don’t know how to write the program, and we can’t specify the network for the same reason. Therefore, it is hard to verify formally an ANN using these techniques.

Verification by testing is much more closely allied to the way in which ANNs are currently developed. Typically, the ANN is taught using one set of data and then a separate set, which exhibits the same properties as the training set (eg. the same face but in different positions), is used to evaluate the network’s performance. Considerable theoretical work has been done in an attempt to quantify the extent to which the ANN can be trusted to perform its task correctly. However, it is not clear that the high reliability values required in safety critical systems can be reached using these techniques because of the uncertain relationship between the training/test data and the real data presented to the "live" network.

In conventional systems, testing can be aided by the use of structured techniques. These involve isolating a part of the system and then subjecting it to many tests and reviews with the objective of finding anything wrong with it. Whilst these can work well in the
conventional highly hierarchical design it is again unclear how a monolithic ANN could subjected to this process. Part of the problem is that an ANN may have a very large input space thus rendering impractical an exhaustive test. However, due to the way in which an ANN classifies we can regard the input space of neurons or layers of neurons as consisting of a number of regions. Therefore we might be able to reduce the size of the test set by choosing points in the input space which emphasise the regions rather test the whole system.

Also, because the ANN cannot be broken down into smaller components (other than the neuron, trivially and uselessly) with respect to their function we cannot reduce the input space size easily.

5. Reliability

Once the system has been designed and installed it is desirable that it should continue to operate correctly for a specified length of time. The reliability is usually specified as the probability of the system operating correctly over a given period of time. For example, an oft quoted figure is a probability of failure of $10^{-9}$ over any one hour period, this figure being roughly commensurate with the airframe of an aircraft.

Once in service a correct system is subject to failure due to wear in hardware components. In order to mitigate against this effect replicated resources are put in place with some sort of data consistency system to mask erroneous data. The most well known example of this is von Neuamnn’s Triple Modular Redundancy (TMR) in which three processors execute the same program and vote upon the results.

Much work has been done on these types of system and they provide perhaps the best developed aspect of safety critical systems. As most ANNs are emulated using conventional computers and software we expect that the transformation into a more reliable redundant system should be fairly straightforward. However, where other implementation techniques are used (such as binary weights, pulse trains, etc.) one would expect to have to develop substantially new techniques.

Additionally, the computational nature of neural networks can, with suitable training techniques, lead to a degree of inherent tolerance to the effect of certain component defects. These will typically be memory errors in the case of an emulated neural network.
6. Conclusions

This short paper has pointed out more problems than it has solved. This was intentional and represents the current lack of trusted techniques which will allow ANNs to be used in safety critical systems. However, some useful points have arisen. There is a lack of rigid specification methods for ANNs. The choice of *rigid* here is deliberate. We mean "not open to misinterpretation". Research in formal methods is currently very fashionable. Informal methods are well established in industry. However, in both cases it is not clear to what extent a trusted numerical value can be placed upon the reliability of the component produced, or even the reliability improvement. However, no similar methods exist for neural networks. It is tempting to believe that, should a rigid specification method be devised for neural networks, that proving conformance to the specification should be relatively easy when compared with software systems. The justification for this remark is based in the simplicity of the ANN compared to software, and the availability of appropriate statistical methods.

For example, computational learning theory allows us to calculate a bound on the number of examples with which an ANN must be presented given the network’s computational capacity and the problem’s computational complexity. No comparable work exists for conventional computer systems. We can’t predict how much code or how much an algorithm or design will take simply by looking at them and applying some equation. Also, we tend to find that problems with a more variable (i.e. very unstructured) input space result in better ANNs. This is because more variability generates instances of more examples to constrain.

It is clear that hierarchical methods for the design of ANNs are not sufficiently advanced. These are highly useful in conventional systems because they allow an initially highly complex design to be partitioned into components which can be specified, implemented and verified independently. Although an embedded ANN may be considered as a component it is difficult to decompose it into useful subcomponents. Therefore it may remain highly complex and difficult to analyse.

Providing a system where ANN implementation is performed using emulation has been seen to be relatively straightforward since existing design techniques can be successfully employed due to the very simple structure of ANN execution and training algorithms.

The nature of problem domains in which neural networks are applied tend to have ill-defined solutions with respect to formal descriptive techniques, and hence existing verification methods are unlikely to be successful. Instead, more complete methods of testing will be required, i.e. techniques to generate test examples with a large degree of
coverage over the input domain.

Reliability can be attained via standard methods where an ANN is emulated. Additional protection at the computational level is also possible given suitable ANN architectures, hardware and training techniques.

Further information on safety critical systems and the processes involved in their certification can be found in the Software Engineer’s Reference Book and in an overview document on licensing safety critical systems. Also, some of the documents in the bibliography can be obtained via anonymous ftp from ftp.cs.york.ac.uk.