DIVISION OF COMPUTER SCIENCE

Applications of Neural Networks in Telecommunications

Simon Field

Technical Report No 219

February 1995

Applications of Neural Networks in Telecommunications

Simon Field

Department of Information Sciences
University of Hertfordshire,
Hatfield, Hertfordshire, UK, AL10 9AB
E-mail: S.D.Field@herts.ac.uk

Abstract

The Teaching Company Scheme (TCS) I am participating in involves investigating the potential for using neural networks in the development of complex software products.

Recently neural networks have enjoyed a resurgence in popularity in academia and industry (cf. DTI scheme), and it has become apparent that there are a number of ways in which the technology can be applied to real-world problems. Neural networks have a statistical basis, and they can be viewed as making these powerful statistical techniques available and accessible to non-statisticians.

This paper will report on a number of projects I have been involved in over the last year which have all used neural networks in the development of new software products. It will include details of:

- a tool that detects cloned, or copied, software,
- a project to evaluate the complexity of software,
- a structured methodology for the design of neural network systems,
- a project which shows that the neural network approach is not always applicable.

The difficulties in transferring this particular research technology to computer-based products, and the advantages conferred by so doing, will also be summarised.

BNR, and its involvement with Neural Nets

I work at BNR (Bell–Northern Research) Europe Ltd. in the Software & Systems Engineering (S&SE) department, which is part of their Advanced Technology Centre. BNR is the Research and Development organisation for Northern Telecom — a Canadian–based global telecommunications equipment company, the fourth largest in the world.

To give an idea of the size of the NT/BNR group, there are approximately 58,000 people working for NT/BNR, and it makes sales to more than 90 countries, with revenues of over $8 Billion. Currently, approximately $1 Billion p.a. is invested in research and development.

The S&SE mission is 'To provide a European source of expertise in the use of advanced techniques.... in particular to improve software quality and development interval, while reducing development costs'.

Software technology goes through three phases in reaching these goals:

1. Technology investigations
   — Investigate the applicability of developments in software technologies (University and market place) for use in BNR, triggered by a business problem or new opportunity, and investigate possible software technologies to apply to the problem.
2. Pilot studies
   — Validating and tailoring the technology for use to solve specific problems, and examining the possibility for use in products.
3. Introducing the technology
   — Education, applying expertise in product development groups, product component production, etc.

Normally a software technology passes through each of these phases until it is introduced as a standard BNR technology (or rejected if it is not applicable).

Examples of current software technology studies include neural networks, virtual reality, genetic algorithms, and knowledge based systems. It is the Neural Networks Study that I am working with.

The Neural Network (hereafter referred to as NN) Study started in late 1992, with 1.5 full–time people. The original aim was to investigate the subject, focus on an area of software engineering to which the technology could be applied, and demonstrate the benefits to BNR by the end of 1993. If the technology could be seen to be worth the investment of resources then a full–blown programme of work would be drawn up.

With the help of a Senior Lecturer from the University of Hertfordshire, working for BNR under the SAIS scheme, the initial area chosen to investigate (which will be described in more detail later) was turned into a prototype. This prototype proved to be so successful under testing that it is currently being productised for use throughout NT/BNR, the idea was patented, and there are now 5 full–time people and two part–time visiting academics working on the team. An additional result, owing to the success of the SAIS, was the instigation of the TCS which myself and another newly appointed TCA are working on.

The project now has a number of key objectives:

- Can the technology of NNs be used elsewhere within NT/BNR today? What needs to be put in place if not?
- What properties do NNs have and how might these be useful either embedded within current or future products, or be used in the process of constructing products?
- There are many types of NNs; what sorts of cost/benefit differences are there between them?
- The project will produce concept demonstration systems to illustrate the potential of the technology; where the promise is good, full–scale prototypes will be built.
- If NNs are to be useful within the company, a systematic method of constructing systems which incorporate the technology is needed — a so–called 'methodology'.

The role of the TCS is to develop the methodology, and also a number of spin–off applications. The tasks should work hand–in–hand, with the method development being tested through the production of a number of applications, and the development of the applications being aided by the existence of a methodology to follow.

An Introduction to Neural Networks

There is currently a lot of talk and hype surrounding neural networks (NNs), but just how much of it is based on fact? They are talked about as 'artificial brains' and 'programs that can think'. Neither of these points are true, but the truth about what you can do with them is almost as grand. For instance, on 30 May 1994 the national media in Britain reported that several of William Shakespeare's early plays were in fact written by another playwright, Christopher Marlowe, and then adapted by Shakespeare. Researchers at Aston University made this discovery using a neural network to analyse each author's writing style, characterising it by word frequency and context.

But how many people know what a neural network actually is? (Some see them merely as a ruse to get lots of research money!) A neural network is quite simply just a collection (network) of interconnected processing units (neurons). Individually, the neurons can each receive an input and perform a simple mathematical instruction, such as addition or multiplication, in order to produce the output. Working together in a network, however, artificial neurons can perform surprisingly complex non–linear functions. The human brain is also made up of these simple, adaptive neurons, but contains approximately 10,000,000,000 of them. It is this huge number of nodes which enables humans to carry out the complex tasks that we do.

The idea of building an intelligent machine out of artificial neurons has been around for many years. Some early results on brain–like mechanisms were achieved by McCulloch and Pitts in the 1940's. At first, research was centred on trying to resolve problems such as theorem proving and chess, as these were thought to require the essence of intelligence. Tasks such as vision and natural language understanding were thought to be simple because even a child could do them. The reality is somewhat different. Today we have many expert chess programs, but there are no programs that can match the basic perceptual skills of an infant. Humans can do what appear to be very simple tasks like walking, talking, and common–sense reasoning, whilst computers can carry out complex arithmetic calculations in nanoseconds, without any errors, but cannot carry out the tasks easy for humans. It seems that the structure of the brain is suited to some tasks, and not suited to other tasks such as high–speed arithmetic calculation.

NNs are biologically inspired; that is the nodes perform in a manner that is analogous to the most elementary functions of the biological neuron. They are then organised in a way that may or may not be related to the anatomy of the brain. Despite this very superficial resemblance, NNs exhibit a surprising number of the brains characteristics. For example, they learn from examples and thus make available experiential knowledge, generalise from previous examples to new ones, and abstract essential characteristics from inputs containing irrelevant data. NNs have a statistical basis, and they can be viewed as making these powerful statistical techniques available and accessible to non–statisticians.

NNs are trained to respond to the inputs they receive. This requires the user to spend some time up front providing them with data so that they can be coaxed into performing a useful function — this is referred to as training. Some NNs need to be supervised during this learning phase, whilst others can learn features unsupervised, and others can even learn as they perform the task, i.e.

they adapt as new data arrives in a live situation. NNs do not compute an algorithm as such, however, NNs can synthesize any mathematical function to some desired degree of precision.

NNs are resilient to problems such as noisy (slightly corrupt) data. Some will degrade gracefully, i.e. they do not just fall over if presented with noisy data, they will perform 'nearly right'. This may or may not be a useful feature to have.

At BNR, the NN Study team has just completed a project to detect cloned software, i.e. software that has been copied from one part of a program to another, and then had some changes made to it in order to alter its functionality. The neural network trained on a section of code containing 1,700 procedures needed only 400 nodes to detect the clones. The next version will use 2500 nodes and will thus be able to be trained on much larger sections of code.

Other areas where NNs are being applied at BNR include: management of a telecommunications network; and more specifically fault correlation (more on this later); routing; and traffic trends analysis. These three areas *seem* to match the capabilities and features of NNs. Fault correlation is concerned with trying to solve the problem of 'seeing the wood for the trees' in large telecommunications networks under failure conditions. A large number of alarm events can be generated from one or more real problems happening, but these alarms can then cause further alarms to be generated — an explosion of information results. The NN can be used to filter out redundant information. Route finding is concerned with trying to find paths through a telecommunications network with various constraints such as capacity, quality of service, priority, etc. NNs can complement, or be used in addition to, existing routing algorithms. Traffic trends analysis is a diverse area concerned with spotting trends in the utilisation of a network, or other patterns over time, such as failure patterns. Such trends can be used to optimise the configuration of the network and predict faults.

As can be seen, NNs are being used in real applications today. They are not research curiosities. Thus, in addition to developing NN solutions to problems, the team at Harlow is also producing a systematic design method for the development of neural network systems. The overall aim is to develop and evaluate a structured methodology to aid the design, development, and implementation of hybrid artificial neural network systems. This will make it possible to 'engineer' NN components, rather than just hack them together as now.

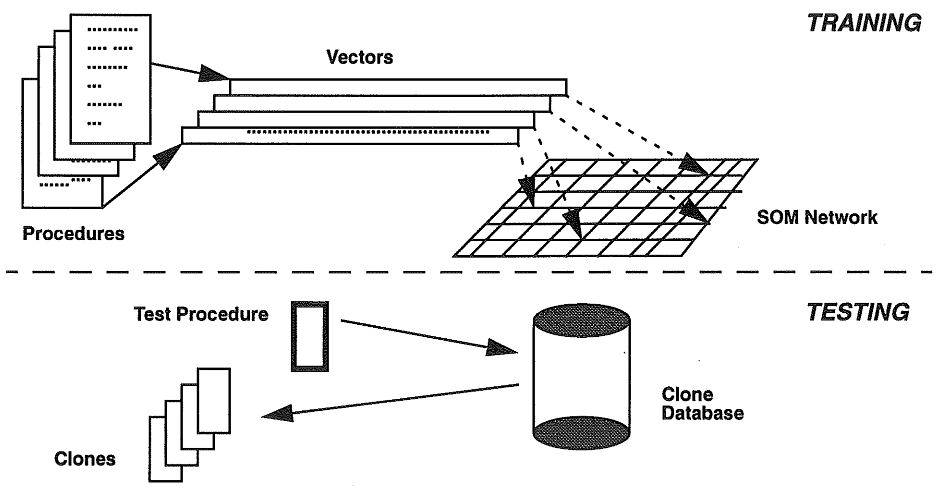
The above are just a small number of examples of the use of NN technology in the telecommunications world. Examples from other industries include medical diagnosis, monitoring of trading on the stock exchanges, process control, noise filtering from time–series data, loan application scoring, credit card transaction monitoring, etc.

Neural Networks have become yet another tool to add to the set an engineer can use everyday, as you will see with the following examples of products developed with the aid of NNs.

The Clone Detector

When the NN Study was formed at BNR, the team needed to learn a lot more about the basic technology of NNs, and it was decided to try to use this new technique to solve a realistic problem. The problem area chosen was that of detecting cloned software, i.e. software that has been copied from one part of a program to another. This is a potential problem for companies that are producing software products that contain large quantities of code because a common method for producing new code is to copy existing code that is known to work, and then alter it slightly to gain some new functionality. The task of identifying sections of code that have been copied is becoming more important as the amount of code increases, because if a problem is discovered with one piece of code it is necessary to correct *all* occurrences of that code.

In the training phase, the blocks of code (procedures) that are to be analysed are converted to a format that the NN can understand — vectors. In each vector particular features of a procedure are recorded, including the number of programming language keywords that appear in the code and how the code is laid out (the indentation pattern).



The NN chosen to classify these vectors was a Self Organising Map (SOM). A SOM transposes the input vectors into a 2–D representation, which simply means that the vectors will be organised in such a way that similar vectors are placed in the same location, and a number of groupings are thus formed.

The final state of the network is recorded in the 'clone database', which is simply a file of all the procedures and which groups they are in. To find clones the user then interacts with the database by specifying a test procedure and then the tool displays a list of potential clones of that procedure, by looking up the group it belongs to in the database.

The prototype clone detector seemed so potentially useful that it was offered for trial around the company in seventeen different groups. The results have been unanimous support, and the team are now converting the prototype into a full–blown tool, ready to be called up from within the standard BNR development environment.

Evaluating Software Complexity[1]

Software complexity measures are used to determine how difficult a program is to comprehend and work with, especially when maintaining it. As software takes an ever more prominent role in systems engineering, then the need to be able to ensure the quality of the software becomes even more important.

Numerous papers have been published identifying various metrics that can be used to transform what is a qualitative issue into a quantitative measure. This project was based on a piece of work carried out by Sheppard & Simpson who chose a subset of twelve of all the possible metrics that could be used. They developed a program to analyse 'C' source code and then used a simple competitive learning NN to identify features from this data. Competitive learning (CL) uses a form of unsupervised learning to develop a set of feature detectors, from which the authors identified three classes of code which they then categorised as *Standard*, *Marginal* and *Non–Standard.* They considered *Standard* to have a complexity that was acceptable, whilst the other two classes were deemed to be more complex than perhaps is necessary. It was felt that the approach they had taken in reaching these decisions was somewhat flawed in that the simple competitive learning net is prone to making a small number of classifications, and we wished to see if indeed this was the case.

It was therefore decided to replicate and extend this work with BNR's code. The data was extracted from BNR's code as a vector containing the twelve metrics and then input to the competitive learning net. The initial results with this approach showed that BNR's code could be separated into four classes, and it also became apparent that there was a difference in the type of code produced by different teams in the company. It appears that the code written by newer graduate employees seems to be 'richer' (short but meaningful) than that written by other teams. This was an unexpected result, and could lead to a separate study of it's own.

The data was then input to other NNs to see if a different classification could be achieved. The CL NN is limited in the details it provides because of the fact that it will increase the chance of a node that has been activated previously being activated again, and this precludes new clusters from forming. When a SOM architecture was used, we were able to see many more clusters. Using the data with another NN architecture, the FuzzyARTMAP, also resulted in more clusters, confirming the fact that code cannot be categorised by just three, or four, classes of complexity.

A further stage with this work is to carry out analysis of the clusters found with the SOM and FuzzyARTMAP architectures in order to determine whether there is any correlation between clusters formed by the different architectures. The results of this work will be presented at the IWANNT 95 Conference.

1. See "Using Neural Networks to Analyse Software Complexity", *University of Hertfordshire Technical Report*, and also "A Complexity Analysis of Telecommunications Software Using Neural Networks", in *Proceedings of International Workshop on Applications of Neural Networks in Telecommunications* (IWANNT) 1995.

Neural Networks Methodology

The main aim of this project, which forms a key part of the TCS, is to produce a structured design methodology to aid the design, development, and implementation of neural network systems. The method will be developed in successive stages and validated at each stage by using it with real applications of the technology. This will result in the production of a number of prototype systems, developed for the telecommunications industry.

Many artificial neural network applications have demonstrated that this technique provides a more effective way of addressing complex problems than conventional techniques. Unfortunately, the current literature on NNs provides very few guidelines as to how to select or develop an NN for a particular problem. Few studies make explicit the base decisions and assumptions that are made during the development of a solution. The strong impression given to most newcomers is of a 'black art'. Because of the lack of a systematic method, the successful development of an NN solution is generally being carried out by experts only, applying heuristics that they have developed through their own experiences.

The project will target the following key areas in the development of a neural network solution to a problem:

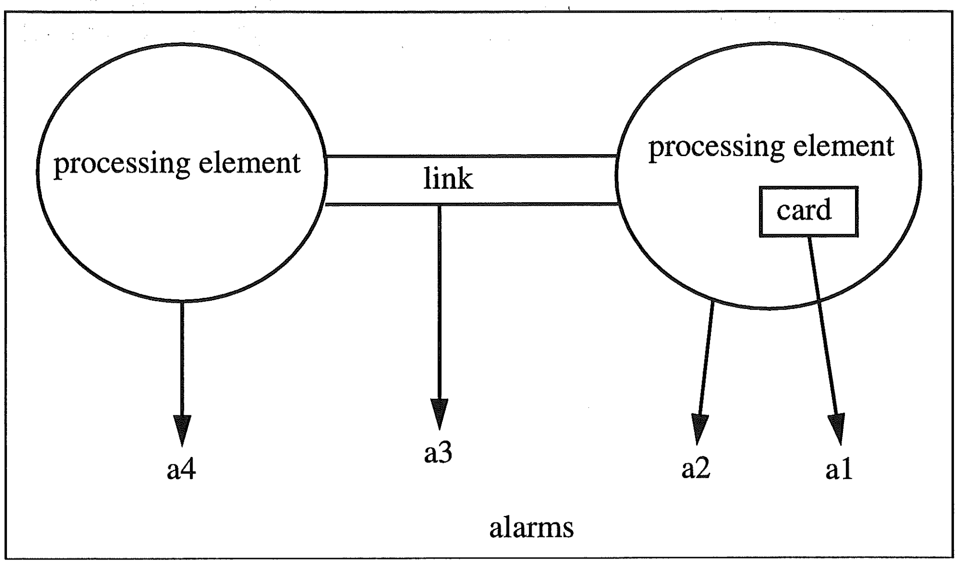
1. The applicability of neural networks to the task
2. Which network architecture to use
3. How to train the network
4. What pre/post processing of the data will be necessary

A number of papers have been published which report poor results for NNs applied to particular problems, but lack the detail required for the reader to ascertain whether the authors in fact applied NNs incorrectly, or whether NNs in general are not suited to that problem area. A report currently being completed by the author investigates the applicability of NNs to a task. A thorough review of published papers is being undertaken, to establish what sorts of tasks NNs are being used for, which architectures are chosen, whether the applications are successful, and also investigating whether any comparative work with non–NN methods was carried out.

The various design issues and decisions that are involved in developing NN applications will be extended to form a general methodology which will cover several problem domains, using real applications. The methodology will be applicable across the whole software lifecycle.

NNs do not always provide the answer — Correlating alarm reports in a telephone system

Problem Description



A card may fail causing an alarm, the associated processing element could emit an alarm, the link could transmit an alarm, and a related processing element could emit an alarm. The network manager only wants to see alarm *a1* since it indicates the actual problem — all the others alarms are irrelevant in terms of fault correction.

Neural Network Ideas

Initially this seems like a problem that can be resolved using the pattern matching capabilities of neural networks. The task is to capture the alarm output, and perform some sort of pattern matching in order to decide what problem is occurring. However, it is not quite as simple as that.

The following gives an indication of what makes it a difficult problem to solve with NNs. The alarms can be output in any particular time order, and depending on the order the network manager receives them they can indicate different problems:

a1 + a2 + a3 + a4	= card failure
a2 + a3 + a4	= processing element failure
a4	= different processing element failure
a3 + a4	= link failure

Problems:

- Overlapping alarms. This is the major problem to be overcome. Faults can occur simultaneously for different problems, resulting in a large amount of overlapping/interweaving of the alarms, so we can't get a clear pattern for each problem. The pattern may be so distributed that it will not be recognised by the Neural Net. In a sample taken, there are 33 problem–instances in the first 100 alarms. It is also not clear whether the first 100 alarms contain *all* the alarms for *any* particular problem.
- Deciding the amount of the data to capture and analyse at a time. Different problems have different numbers of alarms (anywhere between one and two hundred). A small 'window' will pick up the problems with few alarms but won't be able to characterise problems with large numbers of alarms. A large 'window' will swamp the input from problems with single alarms.
- Determining whether a new alarm corresponds to the same root cause already flagged by another alarm involves storing all previous information on active alarms within the net. This is impractical as the number of current alarms can keep increasing. The amount of information being fed back to the input layer will completely swamp any new input.
- Alarm order is critical. For the alarm to be correlated into the lowest level problem (root cause) then the alarm which creates the problem instance must come first. Alarms can be added to this problem but will not change it.
- Alarms can repeat and will then occur over a wide time–frame. We would need to filter alarms first, record them, and not admit repeats.
-

It can be seen that simple pattern matching techniques are not enough. If Neural Networks are to be useful then we must find a way of separating out the alarms before presenting them to the net, i.e. to classify them into *related* alarms. To do this it is necessary to know what the root problem is to which they are all related. This is likely to be very domain knowledge intensive — it might be easier to build an expert system.

Summary:

The alarm stream output is too difficult for a NN to interpret due to alarms for a problem being interleaved with alarms for other problems; the large number of alarms; and there being no clearly defined time order in which alarms arrive.

Conclusion

This report has discussed BNR and its involvement with NNs. It has also described NNs and some of their characteristics. Two NN developments at BNR were then reviewed — the Clone Detector tool and the Software Complexity work. One of the main deliverables of the TCS project was then described — producing a development method for NNs in order to make future NN products more "engineered". Finally, an example of where NNs are not applicable was given.

Whilst NNs have many uses in industry, and we have seen some of these here in this paper, they are not a panacea. When NN technology is used correctly it can quickly yield very impressive results. However, without a structured development method to follow it can be all to easy to apply NNs for either the wrong task, or in the wrong manner, e.g. if the wrong NN architecture is chosen for a task then the results will often be poor. This report has also highlighted the following points:

1. NNs can be applied to industrial applications without an enormous lead–up time — the technology is accessible.
2. NNs are particularly useful for finding patterns in large data sets — the clone detector, using unsupervised learning.
3. NNs are not very useful in systems where lots of structured knowledge is needed — the alarm stream problem.
4. NNs are useful for learning implicit rules in large data sets.

One final point that should be noted is that anyone using this technology must not underestimate the amount of time needed to pre– and post–process the data in a NN application — this is the key time consuming area in NN applications development. If the data is not in a format that is to be of use to the NN then it is highly unlikely that any useable results will be obtainable.

To conclude:

- NN technology is accessible, and results can be obtained quickly;
- It is important to consider how the input data is prepared;
- NNs can be used in problem areas where the user does not have any knowledge about the relationship between sets of data — NNs are capable of analysing data and discovering patterns.