Practical Assessment of Neural Network Applications

Ian T Nabney* Mickael J S Paven* Richard C Eldridge† Clive Lee†

Abstract

This paper reports the initial results of a joint research project carried out by Aston University and Lloyd's Register to develop a practical method of assessing neural network applications. A set of assessment guidelines for neural network applications were developed and tested on two applications. These case studies showed that it is practical to assess neural networks in a statistical pattern recognition framework. However there is need for more standardisation in neural network technology and a wider takeup of good development practice amongst the neural network community.

1 Introduction

Neural computing is a form of *inductive programming*: a task is performed by a general model which is trained using data that represents the task. Such an approach is particularly appropriate when applied to problems that involve modelling complex systems. As the use of neural networks becomes more common, with many live systems now commercially available, the question of how to assess and certify neural computing applications is becoming more important. In particular, if neural networks are to be used in safety related systems, it is essential for there to be an assessment methodology that is sound and accepted by regulatory authorities, end users, and developers. Even if neural networks are implemented in software on conventional computers, they correspond to a very different way of viewing computer programs, so it is not obvious that the classical methods used to develop and assess software are applicable to them. This paper reports the results of a joint research project carried out by the Neural Computing Research Group at Aston University and Lloyd's Register to develop a practical method of assessing neural network applications. Lloyd's Register provides commercial safety and quality assessment, and have been active in the field of software assessment and certification for many years. They provide the necessary experience and knowledge of assessment of conventional software.

The work carried out on the project so far has necessarily been limited in scope. We have only addressed certain sorts of problems and certain neural network architectures, and we have not looked at hardware issues. In particular, we have considered classification or regression problems that are tackled using multi-layer perceptron (MLP) or radial basis function (RBF) networks (see (2) for a good survey of statistical pattern recognition and neural networks). Approximately 70–80% of applications in the process modelling, monitoring and control industries (which are those of most interest to Lloyd's Register, since they represent the bulk of safety critical applications) are of this type.

In this paper we describe a practical method for assessing neural network applications based on current best practice. Neural computing is a field of very active research, so while we expect most of the *aims* and *principles* we describe here to remain valid, the *means* by which the aims are achieved may change in the future. Our work has also highlighted some new areas where further research is needed to provide developers with credible and *quantitative* tests suitable for assessment. During the project, we developed a set of guidelines and supporting technical information to allow people

*Neural Computing Research Group, Aston University, Birmingham, B4 7ET. Correspondence to first author at this address or by email at i.t.nabney@aston.ac.uk

†Safety Integrity and Risk Management, Lloyd's Register of Shipping, Lloyd's Register House, 29 Wellesley Road, Croydon CR0 2AJ

trained in the assessment of conventional software systems to assess neural network applications as well. Two case studies (involving real neural computing applications: one of which is live, and the other of which is at the prototype stage) were used to test the assessment method. To ensure conformance with assessment practice for conventional software systems, the case studies were carried out with the involvement of software assessors from Lloyd's Register.

Earlier papers on this subject (6; 8) have raised some of the important issues in the development process, but have not considered recent theoretical developments in the field that allow us to measure the dependability of neural network outputs. They have also been addressing the problem from the developer's, rather than the assessor's, point of view.

2 Neural Computing

While this paper is concerned with how to assess neural network applications and not with how to develop them, it is nevertheless important to consider some of the key issues in application development. This is because assessment is concerned with process ('was the system built using sound engineering principles?') as well as product ('does this system perform to specification?'). The emphasis in this section is on the principal differences between neural computing and conventional software engineering.

It is clear that neural computing represents a very different approach from the conventional view of software development, where an algorithmic solution can be specified in advance of writing the software. Because the performance and precision of the model cannot be determined in advance, the use of neural computing is best confined to applications where efficient algorithmic solutions are impossible or impractical. Such applications are typically complex, poorly understood, and imprecise. Understanding speech, reading hand-written documents, and modelling and controlling non-linear systems are all domains where neural computing and other statistical techniques outperform algorithmic methods.

2.1 A Comparison with Conventional Software Engineering

We can compare neural computing with a conventional approach to software development by considering a concrete example: the problem of developing a system model for a marine engine.

A conventional approach would involve determining the physical processes governing the engine, analysing these to generate mathematical equations that represent the system, and then programming some software to simulate these equations and their solution. In principle, this method could be used without the use of a physical engine.

An inductive approach involves gathering data from an engine and training a model (a neural network, for example) to reproduce the same relationships as are present in the data and determine a good approximation to the underlying function that has generated the data. The training process consists of adjusting some variable parameters in the model using the data ('parameter estimation' in statistical terminology). In our view, this process is best studied from the statistical pattern recognition point of view, placing neural networks in the framework of linear regression, time series models, and other statistical methods.

A consequence of the way in which neural networks are trained is that the parameters in the model are the only part of the model which is specific to a given application. The interpretation by humans of these parameters is considerably more difficult and less precise than the interpretation of algorithmic high level source code. This implies that the assessment of neural computing systems is necessarily statistical in nature. As safety cases are typically written in terms of limiting the probability of failure, this may actually be an advantage.

In principle, inductive learning involves no software development, as the software to run and train the model is independent of the application and data. In some respects, this software is analogous to a compiler in conventional software development, with the training process similar to compilation, and the parameters of the trained model comparable to machine code. This software is entirely algorithmic, and so can be certified for use in safety related applications with existing assessment methods. It is rather hard to develop high integrity compilers for languages of reasonable size (see (9) for an example); by comparison, neural network software is comparatively small and straightforward, so that it should be relatively straightforward to assess such software by conventional means.

In practice, the distinction between the two development approaches may not be so clear cut. When modelling any complex real world system, some data from the actual system is nearly always required, if only for model calibration and validation. Most of the issues discussed in this paper are relevant whenever real world data is used for such purposes. Equally, neural computing may only represent part of the solution to a problem: for example, conventional programming may be needed to pre-process the data before fitting a model.

Although there is little or no software development in a neural computing application, there are other important tasks to be carried out which have no real equivalent in conventional software engineering.

- Data collection. Adequate quantities of relevant data are essential.
- Data pre-processing. It is rare that the best performance is achieved when the data is presented in its raw form.
- Network training. This has to be carefully monitored to ensure that a good solution is reached.
- Performance assessment. The key question is not how well the network performs on the data it was trained on, but how well it generalises to unseen data.

The assessment guidelines have as their focus sound practice and testing for these aspects of development.

2.2 A Lifecycle Model

Most phases of a neural computing application development lifecycle will be familiar from those used for conventional software engineering. The main differences arise for three reasons:

- a precise functional specification is not possible at the start of development;
- the use of data means that extra tasks must be carried out;
- development is *necessarily* iterative.

Although there is no definitive lifecycle model (just as for conventional software engineering), the following is a principled approach which has been successfully used in practice.

1. Problem Definition: what are the aims? How can you measure success? This document is key to credible performance assessment.
2. Data collection: data sampled from the system to be modelled, cost measures, prior information (for example, how smooth the function should be, operational constraints on variables). In neural computing, particularly with a Bayesian approach, we attempt to quantify and

formalise the assumptions (or prior knowledge). This makes it easier to test whether these assumptions are valid, and is an important part of verifying the performance of the trained system.

3. Preliminary data analysis: visualisation to understand the data, feature extraction, missing and corrupt data, pre-processing.

4. Model development: model design is based on prior knowledge and data analysis. A range of models should be trained: simple models as benchmarks and more complex models to attempt to capture more features of the problem and to improve accuracy.

5. Integration: interface with other software.

6. Operation and maintenance: ongoing model validation and retraining.

Note that data collection and pre-processing can take up to 40% of the development time (4). This lifecycle model (see Figure 1) is quite general, and is similar to others in the literature (see, for example, (8)). However, there are two important points to note about the lifecycle defined above and that are often omitted. Firstly, data is not just collected from the system to be modelled. Prior information can and should be used to structure and constrain the solution. Secondly, during operation, the model must be validated and retrained when necessary. Techniques recently developed allow us to measure, and therefore monitor, the quality of neural network performance during operation. For example error bars (see (3) amongst others) allow us to assign a *confidence* in the neural network prediction.

2.3 Training and Generalisation

Learning for a neural network means adjusting the parameters (usually called 'weights') to approximate an unknown function, based on a data set sampled from that function. The weights are adjusted during the training process by minimising an error (or cost) function. This error function is a global measure of the discrepancy between the target values and the values predicted by the neural network.

The error function is often very complex (since for neural networks it depends on the weights in a highly non-linear way). Thus finding the minimum is not easy. Most algorithms are based on the fact that the global minimum of a function is a point where all the partial derivatives of the function equal zero. However, this is a necessary but not sufficient condition. These points can be local minima (i.e. at this point the function value is a minimum for a small region around the point), local maxima, or other 'flat' regions. Being trapped in a bad local minimum implies that the solution is sub-optimal, which may result in non-compliance to the specification. Most optimisation algorithms find a local minimum close to their starting point: thus it is important to carry out multiple training runs with different random starting points.

Often the data used for training is corrupted by noise: for example, owing to the imprecision of measuring instruments. The aim is to avoid learning the noise but to learn the underlying structure so that the model *generalises* to previously unseen inputs. Thus if a neural network learns the training data and fits it perfectly, it is said to *overfit* the training data. This usually leads to poor generalisation performance, as can be seen by testing the model on an independent set of data. Often overfitting is associated with a complex model for which the computed function may vary greatly between the training data points. We usually have a prior expectation that the function should vary in a relatively smooth fashion and this can be incorporated into the model training (by *regularisation* techniques) and usually improves the generalisation of the trained model.

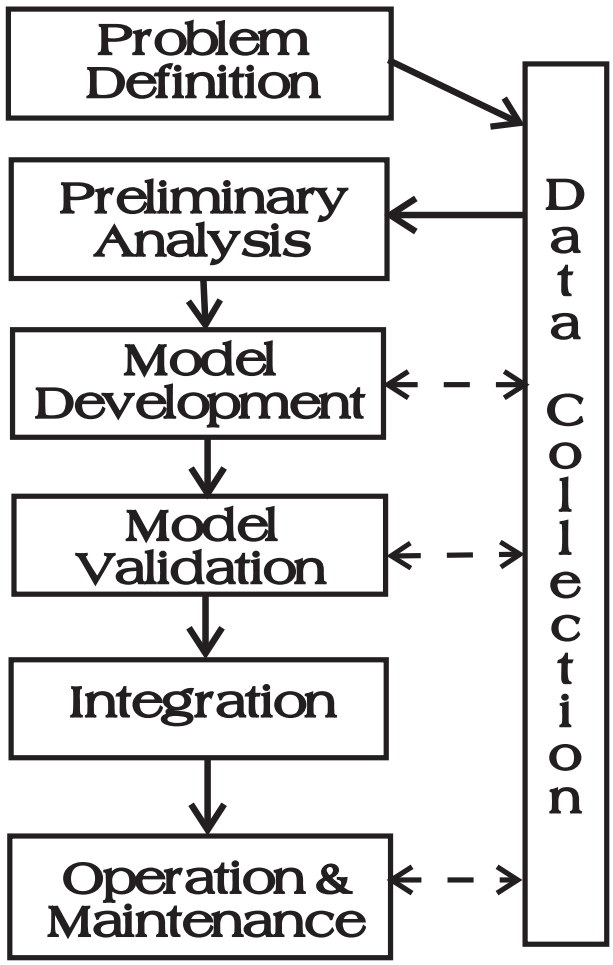

Figure 1: Lifecycle Model. The solid line denotes the main path through the lifecycle; dotted lines denote points at which which more data may be required and the development may return to an earlier stage.

2.4 Interpolation/Extrapolation and Data Density

Understanding the distinction between interpolation and extrapolation is fundamental to neural network reliability. Typically neural networks (and other data models) are much more accurate when interpolating than when extrapolating. The usual definitions of these terms (that the new data point lies in the interior or exterior respectively of the region of input space containing the training data) are not accurate or easy to measure in more than one dimension.

Instead, we say that areas of the input space where the training data density is high are interpolation regions, while areas of the input space where the training data density is low are extrapolation regions. Intuitively the idea is that in the areas where the model has a lot of information its behaviour is constrained, while in areas where there is little or no information its functionality is unconstrained and therefore unreliable.

Another way to measure the reliability of the neural network predictions is to generate 'error bars' which give an interval of 'likely' values which take into account possible sources of variation around the predicted output. There are different sorts of error bars corresponding to different source of variation: input noise, output noise, and parameter uncertainty. The wider the error

bars, the less certain the network is about its output. It has been shown in (3) that Bayesian error bars (which take into account the uncertainty of the network weights due to the use of a sample of data in training) are related to the input data density, which links error bars with novelty detection.

We conclude that for safety critical neural network applications, the novelty of the input data should be monitored (as suggested in (1)), so that outputs that are likely to be unreliable can be identified. This would form part of a monitoring system which would be a quantitative way of assessing performance and testing the assumptions made during development on-line. In addition, most neural network systems make the assumption that the data generator is *stationary* (i.e. it does not change with time). This assumption can also be tested by monitoring the novelty of the input data and assessing the accuracy of the network's output.

3 Assessment Guidelines

The basic principles of the assessment guidelines are the same as those in conventional software:

- Check that the neural network function has been developed in a controlled and planned way (i.e. the quality of the processes and methodology used).
- Check that the neural network function has successfully passed the tests and complies with its specification.

Neither of these two aspects, if applied alone, is sufficient to assess a neural system: the two are complementary. Although the principles of conventional software development methods still apply, their practical application may differ. For example, repeatability means that random seeds used in initialising models and splitting data must be recorded so that experimental results can be confirmed at a later date.

The fact that neural systems are data based has many implications for development and assessment. The data becomes part of the 'program', and therefore needs to be subject to the same control as other project documents. More fundamentally, the data needs to be of good quality, and representative of the problem. This can be difficult to test (and therefore assess), as there are few objective statistical tests for these properties. As in any system, certain assumptions are made during development. One advantage of neural networks is that it is possible to quantify many of these assumptions and test them before deployment.

It should be noted that the assessment guidelines are not prescriptive: they do not, in general, mandate particular methods for developing applications. We can compare this with statement 3.3.2.4 from the MISRA guidelines (5): "Many diverse methods may be used to assess stability. No one method may be identified as preferred. It should not be assumed that similar results will be obtained from different methods for a given solution".

Most of the issues in neural network development apply also to many systems already in use. For example, in control theory the stability of linear controllers for linear systems can be mathematically analysed. However, developing a linear controller for a real system requires a system model (involving some parameter estimation from data). This model is not exact (for example, it may be a linear approximation to a non-linear plant, and may ignore system noise) and this inexactness affects the controller's performance: it will not exactly match theory and stability is no longer necessarily guaranteed. Thus practical systems are typically designed with large margins from the boundary conditions predicted by theory. Thus 'rules of thumb' are used now in safety critical systems, and some of the questions raised about the use of neural networks and other new technologies can also be asked of conventional development practice.

4 Case Studies

4.1 Questar

The first case study was based on the Questar product developed by Oxford University in collaboration with Oxford Instruments Ltd. The product monitors EEG traces to detect sleep disorders (7). It tracks the sleep/wake continuum on a 1 second time interval. Prior to the development of this instrument, a *hypnogram* would be constructed by hand from an EEG trace with a 30 second time interval. To analyse a whole night of sleep manually takes a considerable length of time (about 1 hour). Extensive trials have demonstrated that the system correctly automates a labour intensive process and improves the quality of the results (both through improved time resolution and with consistent and repeatable analysis). An analysis of a night of data by Questar takes just 10 seconds. The system is now used as a diagnostic aid for clinicians.

The problem is expressed as a classification problem (with 3 classes) which is mapped to a wakefulness score in the range $[-1, 1]$ which is the clinical users' preferred format. RBF networks were used and compared with linear models, over which they showed a significant improvement.

The main finding of the assessment was that although the development had generally been carried out in a principled way, the documentation was not as complete as could be desired. The Functional Specification had been written early in the project: however the performance targets were only determined at the end of the project. The developers agreed that it would have been useful to set concrete targets earlier in the development. The users and scope of the system were not explicitly defined: although end users had been consulted, particularly concerning the way in which the results should be presented, these reviews were not documented.

Testing was commendably thorough, involving comparisons with hypnograms (using 7,200 test samples) and diagnostic tests under clinical conditions. It has been shown to correlate with a by-hand analysis as well as one carried out by a second expert. The risks (i.e. misclassification costs) have not been elicited from clinicians: this information is important to make optimal choices in the decision theoretic framework.

Confidence measures, such as error bars, were constructed for the models, but they proved not to be robust (i.e. the confidence intervals themselves did not generalise well). A known problem with the system is that novel data tends to be classified as 'wakefulness', but there is no specific monitoring for novel data in the operational system. The assessors also had some concerns about how representative the data is. For example, the effect of different types of EEG recorder is unknown, and the impact of any variations has not been analysed. There was a good use of visualisation and a considerable body of prior knowledge from 30 years of clinical experience to understand the data and select relevant features. A systematic search over the order of the pre-processing AR model and the size of the network means that we can have confidence that the selected architecture is near optimal. Good use was made of script files so that all experiments are repeatable.

4.2 Engine Management System

The second case study is an ongoing project carried out by Aston University in collaboration with a company that manufactures engine management systems. At the time when the case study was carried out, the project was only in its early stages: that is, its feasibility had been shown on a sub-problem.

In normal engine operation, away from idle speed, the ignition timing and fuel injection volume is determined from a set of look up tables as a function of several variables, such as load, speed, engine temperature, etc. These look up tables are obtained on the basis of labour intensive experiments

which involve tuning engine parameters until the engineer is satisfied that the engine is running optimally in a steady state. Typically, the input variables are quantised into 16 bands, and the resulting matrix is quite sparsely populated with experimental data. The criteria for 'optimal' values are complex, involving tradeoffs between performance, emissions, economy and driveability. There are many such look up tables in modern engine management systems governing all parts of an engine operating envelope. Usually a simple linear interpolation method is used to estimate values away from the measured data, which gives rise to unsmooth (non-differentiable) control surfaces. The aim of this project was to replace this interpolation scheme by a neural network.

Overall there was a good level of conformance with the guidelines especially after taking into account the early stage of development. The technical standard of the work was generally high, with a particularly thorough exploration of suitable error bars. The main findings were:

- Specification. The performance requirements were not fully specified and although core objectives had been identified, there was no priority order. This is important since it strongly influences the choice of cost function.

- Documentation. The activities and findings from the initial data collection and analysis were distributed in notebooks and could usefully have been summarised in a report. This report would form part of the system specification. It was also difficult to trace design features to the specification requirements.

- Testing of assumptions. An assumption of constant variance Gaussian noise was made (leading to a sum of squares error function) but the validity of this assumption and the sensitivity of the neural network to it had not been investigated.

5 Discussion

As with conventional software, providing a formal proof of the correctness of a neural network application is in general impractical. However, both case studies suggest that if a neural network application is developed in a controlled and methodical way and properly tested, then it should be possible to validate and verify it with an effectiveness comparable to conventional software. In both cases the technology had been applied in a principled and well engineered way. We note that assessment of such applications requires a good understanding of the basic properties of neural networks.

Both case studies have shown that during the development of a neural network application, three issues seem to be generally neglected: documentation, specification and testing of assumptions. For instance, both applications no quantitative targets were defined in the specification, and a standard noise model (leading to a sum of squares error function) was used without examining how appropriate this was for the data.

The main reasons for this neglect are probably:

1. The iterative nature of the development lifecycle and the fact that system performance cannot be predicted in advance mean that it is often inappropriate to define a concrete specification at the start of the project. Lower limits on performance can be derived from safety arguments and cost/benefit analysis. After the feasibility stage the specification *should* be reviewed to make it more precise, but this often does not happen.

2. As neural networks correspond to a novel way of viewing software, no standards currently exist. For instance, there is no clear definition of what should appear in a specification for a neural network application.

3. Although neural network technology is currently moving from research to products, many neural network applications are developed by academics. While they generally have a very good understanding of the technology, they usually do not have the same objectives and experience in software development as commercial software houses.

However, we believe that these problems are temporary and are due to the relative youth of the technology. Furthermore, there are many conventional software applications that are poorly specified and documented, so these problems are by no means unique to neural networks.

As was the case in the early days of conventional software development, there is a need for standardisation for neural networks. There are several motivations for standardisation:

1. Providing guidelines to develop successful neural network applications makes the technology more accessible.

2. Application development is easier to control if it is done in a systematic way. Moreover, better control over development is (usually) synonymous with higher dependability.

3. A standardised development method enables verification and validation to be standardised as well. This is not completely achievable (even for conventional software), but is a goal to aim for.

The two lead engineers on the case studies found the assessment useful and the procedure convincing. As these are two of the leading neural network application developers in the UK, this suggests that the process we have proposed would meet with widespread acceptance in the technical community.

Our current work is addressing four areas:

- Quantitative results. For some rules of good practice there is no standard technique to test their correct application in a quantitative way. So, for some aspects of the current guidelines, their assessment involves making sure that 'rules of thumb' and accepted good practice have been applied. This is not desirable for applications requiring the highest levels of integrity, although it is tolerated by some existing standards (e.g. (5)).

- Data quality and characterisation. This is essential for successful applications, but there are few, if any, useful tests for determining any weaknesses in this regard.

- Safety integrity levels. In principle, since neural networks are statistical models that make probabilistic predictions, they should be well suited to incorporation into safety cases. We shall investigate how this could be done and what the implications for specification of neural network systems are.

- Neural controllers. Some applications use neural networks as part of a closed loop control system. To train a neural network to perform this task is quite different from the usual supervised training regime we have considered up to now, and also raises questions of stability.

6 Acknowledgements

We are grateful to Lionel Tarassenko and James Pardey of Oxford University and David Lowe and Chris Zapart of Aston University for their assistance with the case studies.

References

[1] C. M. Bishop. Novelty detection and neural network validation. *IEE Proc.-Vis. Image Signal Process.*, 141:217–222, 1994.

[2] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[3] C. M. Bishop, C. Qazaz, C. K. I. Williams, and H. Zhu. On the relationship between bayesian error bars and the input data density. In *4th IEE Conference on Artificial Neural Networks*, pages 160–165, 1995.

[4] DTI. *Best Practice Guidelines for Neural Computing Applications*, 1994.

[5] MISRA Report 4. *Software in Control Systems*, 1994.

[6] G. Morgan and J. Austin. Safety critical neural networks. In *4th IEE Conference on Artificial Neural Networks*, pages 212–217. IEE, 1995.

[7] J. Pardey, S. Roberts, L. Tarassenko, and J. Stradling. A new approach to the analysis of the human sleep/wakefulness continuum. *J. Sleep Res.*, 5:201–210, 1996.

[8] D. Partridge and W. B. Yates. Engineering reliable neural networks. In *4th IEE Conference on Artificial Neural Networks*, pages 352–357. IEE, 1995.

[9] S. Stepney. *High Integrity Compilation: A Case Study.* Prentice Hall, 1993.