Neural Networks and Notations

0201991

Abstract

Neural Networks are primarily concerned with the modelling of biological systems within a mathematical or computer based context. The fundamental component of a network, the neuron, has been the subject of modelling since the middle of the last century. In this paper we explore a new approach to modelling based on integrating the mathematical basis of a model with the rich, interactive processes found in the Empirical philosophy. We introduce a simple notation, N3, for creating models and demonstrate how it can be used to model neural networks through the definition of a simple bottle-neck network. We conclude with a discussion of our approach and ask whether the Empirical approach has benefits when modelling neural networks.

1 Introduction

Neural Networks are concerned primarily with the modelling of parts of the human brain and nervous system within a mathematical or computer based context. Through the study of biological processes such as vision, perception and memory, complex networks have been created to allow computer vision, natural language processing and efficient organisation of information. Although many of these tasks appear very simple when observed in the day to day living of biological organisms, the models of these processes reveal how incredibly complex these systems are.

The fundamental component of the neural network is the neuron, found biologically in the brains and nervous systems of organisms. One of the earliest studies in the modelling of a neuron, which ultimately set the context of the field, was conducted by McCulloch and Pitts (1942) who argued that a neuron could be modelled as some threshold function applied to the sum of its weighted inputs. Since the development of McCulloch and Pitts' TLU neuron, a vast number of models (Hopfield, 1982; Hebb, 1949; Kohonen, 1990) have been proposed which seek to expand the applications of neural networks and more faithfully represent the underlying biological mechanism.

In order to permit any kind of modelling, Neural networks are often developed as a highly abstracted representation of an underlying biological system. Even though the biological behaviour may not typically be deterministic, the language of representation for Neural Networks is often mathematics as it permits concise descriptions of a system with the ability to reason formally about its behaviour.

In this paper we seek to investigate the applications of an Empirical philosophy with respect to the modelling of Neural networks. We begin the paper with a brief description of a new notation, N3, for defining network models within the EDEN framework. This is followed by a discussion of a bottleneck-network model which learns the average luminance of pixels within an image. We conclude with a review of the techniques and tools used and ask whether the modelling of neural networks within the Empirical framework offers the richness of experiment and experience that the users of other models enjoy.

2 Modelling Neural Networks

In this section we discuss the Neural Network Notation (N3) language with which modellers can construct simple neural networks and how the definitions in the model are converted to EDEN dependencies. As Neural Network models are often represented by mathematics, N3 is closely akin to mathematics in its appearance with the intention of permitting highly expressive models unhindered by the complexity of representing objects such as vectors and matrices within the EDEN language.

2.1 Neural Network Notation (N3)

The N3 (Neural Network Notation) notation is a simple notation developed to allow modellers to quickly define neural models in a language similar to that of conventional mathematics. The notation itself composes sections of code written using the Agent-Oriented Parser (AOP) (Brown, 2001; Harfield, 2003) and traditional EDEN.

The main motivation for the use of N3 is to greatly simplify the process of developing mathematically

based network models within EDEN. As models are usually defined in terms of vectors and matrices, a modeller would usually need to resort to managing list data structures to achieve a behaviour similar to that of the mathematical definition. Although this is certainly possible, the experience for the modeller is improved by removing this complexity and permitting a richer input language.

The basic data structures of an N3 model are the vector, the matrix and the scalar. If defined in the model, these variables are parsed by the N3 notation into EDEN scripts as either a list, a list of lists or a float type respectively. Once the variables have been defined, N3 permits users to develop relationships between the variables using operators highly similar to that of mathematics. Figure 1 demonstrates how the conversion process from an N3 model to EDEN happens at runtime. In this simple model, the user defines their mathematical objects x and y which are then given default initial values within the EDEN code. Once this N3 script has been input to the system the variables defined are available in standard EDEN, thus giving the modeller the opportunity to assign values with either EDEN or N3 or to build further dependencies within their model by using the variables as the basis of a dependency relationship in EDEN, SCOUT or other notations.

In N3, operators are defined as procedures which execute the appropriate operation for the definition based on the types of the inputs. This approach allows the model a considerably higher degree of flexibility. By defining the intended operation of the user as a procedure we can enforce alternative behaviours based on the current state of the model thus freeing the user to experiment with varying data types without needing to change the expressions already entered into the system. We take as example the code in Figure 1, in this model the current type of the variable y is a vector. Hence $multiply$ is defined as the multiplication of each element of the vector by the scalar x. If the user were to experiment by entering $y = 0$ into the system the type of y would change to be scalar, hence v would be recalculated to be a scalar.

2.2 Dependency in an N3 Model

In N3 we take a definitive approach to the definition of variables as described by Beynon and Russ (1991). Instead of a variable definition such as $x = y * z$ assuming a procedural approach to evaluation where x is assigned the value of the corresponding multiplication of y and z, we assign x a definition in a similar manner to that of a spreadsheet cell. When either y or

N3 Statement:

```
%n3
vector y<2>
scalar x

v = x * y
```

Translated EDEN Statements:

```
%eden
y = [0,0];
x = 0;
v is multiply(x,y);
```

Figure 1: Translation of N3 definitions to EDEN

z are changed in some way, the value of x is changed also.

When assignments in N3 are parsed, a dependency is created between the variable being assigned to and the right hand side of the definition. An example of the parsing process is shown in figure 1. The creation of dependencies rather a list of expressions to be evaluated changes the modelling process to one whereby the model can be created and refined interactively as behaviour is explored from a process of iteratively defining, executing and observing the outputs of a model.

2.3 Agency in an N3 model

At the heart of Empirical Modelling lies the concept of an agent, which can be thought of as "an entity that plays a role in the representation and transformation of system state" (Beynon, 1994). A model written using only N3 represents only the definitions of the behaviour of the system and as such contains no autonomous agents capable of changing the system state with surprise to the user. In this regards we would categorise the behaviour of an N3 model as being that of a 1-agent (Beynon, 2005) system whereby the behaviour is entirely specified by that of the user in their role as the system super-agent. We would also argue that this is an important aspect of a notation such as N3 where the aim is create a model with which the user can gain a complete understanding of the behaviour of the network through interaction. As the model functions in a manner similar to a spreadsheet, the user has complete control over their experience of the model and can experiment in a highly consistent and clear manner.

3 Modelling a Bottleneck Network

A bottleneck network in its simplest form is a network of inputs and outputs both of size x linked by several 'hidden' middle layers with the property that the inner layers have a size $\leq x$. Bottleneck networks have a wide variety of uses in neural network systems particularly in the field of image and signal encoding and noise reductions. A good description of multi-layer networks can be found in Gurney (1997). An example bottleneck network diagram is shown in Figure 2.

The network implemented in the model accompanying this paper is a simple multi-layer network which seeks to 'learn' the average luminance value of pixels in a small input image with a view to being used as a basis for compressing and decompressing highly similar images. The biological system of interest is that of vision whereby regions of colour are may be communicated in relatively low detail to save the optical nervous connections from bandwidth overload. The model works by showing the input image to the learning routine on each iteration of the training schedule. Within an iteration, the values of a weight vector are compared to the current input and where these differ a small change (defined by a learning parameter) is made in the weight vector to move elements of the vector closer to the input. Over successive iterations, the values of the weight vector tend to the average of the images input to the system. Once a weight vector has been learned by the system it can be used to compress images by averaging the luminance of a group of pixels, communicating this average and then decompressing by reapplying the weights to the average value. The compression factor of the model defines how many pixels are grouped into a single value, in terms of our network, this compression value also defines the fan-in of the input elements to the midlayer of the network.

3.1 Modelling the Bottleneck Network with N3

The first stage in developing our model is to build the basic bottleneck network which takes an input vector (the pixels of the input image) and communicates this input information to the output layer. This is done by defining a vector $input$ to contain the colour values of the pixels in the input image. Secondly, we create another vector $midlayer$ which serves as the middle, hidden layer of our network. For now we set $midlayer = input$. Executing this assignment in N3

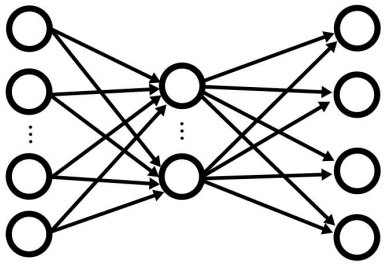

Figure 2: A Simple Bottleneck Network - the bottleneck occurring in the middle layer of the network

creates a dependency for the $midlayer$ variable in the values of vector input. Finally we create a third vector $output$ and set this to equal $midlayer$. Our basic bottleneck network is now complete although it contains no learning function and no processing of input values.

3.2 Extending the model to include learning

The learning function used in our model is a highly simplified version of the Generalized Hebbian Algorithm (GHA) (Haykin, 1999, p. 419). The GHA has a vector of weights w which is altered on each iteration of the learning cycle by a vector Δw. Δw is calculated as a function of w, the inputs to the network and a parameter η which controls the rate at which Δw is converged to the input.

As the N3 notation is fairly close to mathematical definitions we can introduce the learning component of our network by defining our two vectors $weight$ and $weightdelta$ and our learning parameter $learning$. We define $weightdelta$ in terms of the input vector, $weight$ and a scale factor to ensure the values remain with luminance. The N3 code for building the learning component of our model is shown in figure 3.

We cannot however, define $weight$ in terms of $weightdelta$ as this could create a cyclical dependency. Instead, changes to weight must be made explicitly, so a we introduce a $trainnetwork$ procedure which when called updates the values of $weight$ by the $weightdelta$ vector.

```
scalar learning
scalar compression
scalar scalefactor
vector input<64>
vector weight<64>

tempy = elementmultiply(weight, input)
y = sum(tempy)
tempyinput = y * input
tempyweight = y * weight
tempsub = tempyinput - tempyweight
weightdelta = scalefactor *
  (learning * tempsub)
```

Figure 3: Defining the learning function in N3

3.3 Increasing Accessibility through the use of SCOUT

The founding aim of the EDEN toolkit was to provide an improved user experience through the ability to engage in a rich modelling experience by dynamically defining and altering a model. As the EDEN language developed additional graphical notations were added to permit windowing capabilities and graphical drawing. These notations are now manifested in SCOUT and DoNALD (Yung, 1992) for drawing two dimensional graphics and windows and Sasami for developing three dimension models.

In the model accompanying the paper, the user experience of the bottleneck network was limited to altering variables within EDEN or N3 to certain values and then observing these changes through traditional EDEN querying. This experience was poor as it provided no graphical opportunity to view the results of one's modelling actions. As such, a graphical representation of the learning experience was developed through a SCOUT interface. This is shown in figure 4.

On the left hand side of the screen is an input image consisting of a 16 x 16 grid of pixels, in the centre are a series of labels displaying observables about the model and on the right the effect of compressing an image using our 'learned' weights. There is also a small palette of colours at the bottom of the screen the user can use to draw an image onto the left hand display. At launch, the compression factor of the model is set to 1, that is no compression is taking place. A value of 2 would indicate the bottleneck fan-in is by a factor of 2.

This display has been used to extend the model further, the input vector, $input$ has been made a dependency of all of the background colours of the input

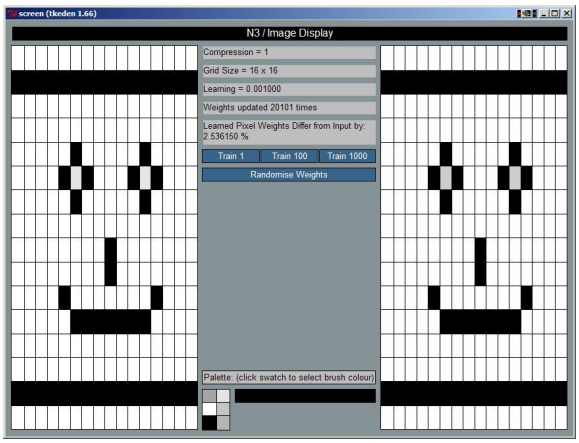

Figure 4: N3 Image Display - Input Panel, Information display and Output Panel

image. This dependency relationship means that as users draw onto the input image the $midlayer$ and $output$ variables are updated automatically, this is often reflected in the output image changing slightly depending on the success of the training. The value of the display labels in the central panel have also been made through dependencies on the $input$ and $weight$ vectors in the model. For instance, the percentage difference label is a dependency on the values of the input and the current values of the weight vector.

4 Experiencing the Model

We have now defined our bottleneck network in terms of its N3 definition and an associated graphical element to increase the richness of the modelling experience. As it stands the model is basically a blank canvas where the user can experience by drawing images onto the input and viewing how the learning function adapts to 'learn' the input. The user is now able to alter and experiment with the model definitions to examine how the network will behave in an alternative context or if different definitions are input.

The main activities that can be experimented with are changes to the $trainnetwork$ procedure that alter the method in which the network learns, by changing this procedure the user can observe the behaviour of different learning strategies. The user may also change the $compression$ variable to determine how aggressively the network will compress the information containing within the input. The user is also free to experiment with changing the mathematical model.

The wide variety of tasks that can be used as the basis for experimentation show that although the no-

tation is very simple and indeed the accompanying model is simple, the opportunities for exploration are many. This is perhaps a clear demonstration that modelling Neural Networks with Empirical tools can lead to a richer experience and understanding than simply the study of a models mathematical properties. Although the mathematics can tell us how the network behaves, we argue that the experimentation permitted by our notation and model achieves a better understanding of how the definitions actually works within their specific context, a view we believe correlates with that of James (1979).

5 Technical Limitations of N3

One of the major technical limitations of the N3 notation is that in a usual scenario as assignment of a vector element to an expression, say `v<1> = y * 2` would be translated into an assignment of the first element of the list `v` to be a dependency of `y`. However, this dependency creation is not permitted in EDEN due to the highly dynamic nature of list structures and as such assignments to elements of a vector in N3 behave in much the same way as a conventional C++ or Java statement. The *%edensl* notation is a prototype language that has been developed to overcome some of the dependency issues arising from the use of lists within Empricial Modelling, particularly with respect to the individual dependency of elements within a list. For more information on *%edensl* the reader should consult Ward (2004).

A second technical limitation is the limited number of implemented mathematics operations for vectors and matrices. In particular, there are no methods for raising these structures to a power or conducting advanced operations such as matrix convolutions or correlations. If N3 were to be developed further, extra operators to allow these operations may be required to formulate larger and more complex neural network models.

6 Model Perspectives

Throughout this paper we have discussed the concept of richness in terms of modelling a neural network, ultimately, this is key to successful model. A rich user experience combined with an insightful model encourages the user to experiment and learn through their interaction. We would argue that N3 is successful in raising the richness of the experience by providing a familiar notation combined with the empirical approach to permitting modelling using entirely dynamic definitions. Given that the notation may permit a rich experience, we must also ask how can this experience be useful and what are the benefits of taking this dynamic, empirically based approach over say conventional mathematical models such as Mathematica? The answer, we claim, is quite simple. In Mathematica, the language is pure mathematics. The user inputs mathematical equations and executes them with respect to particular inputs. In this model the user never experiences what this actually means in the context of a Neural network, instead their experience is limited to the mathematics, the evaluation, the symbols that make up a neural network definition. The empirical approach differs from this because although by using the N3 notation the modeller is working with a language similar to mathematics, their experience is different. Instead of entering formula after formula and evaluating these, the user can experiment with particular inputs and change and adjust small parts of the system little by little, the user is not required to re-evaluate the whole adjusted equation instead they can see the effect of their experimentation as they conduct it. We would also claim that through the integration of richer graphical elements the experience is widened further, the user has the ability to define the model in mathematics but can experience how the mathematics works when the model is applied to the context of study.

7 Conclusions

The motivation for the creation of the N3 notation was to create a mechanism whereby modellers could create neural network definitions in a language with which they are already familiar but increase the richness of their modelling experience by engaging in an interactive modelling process as opposed to a largely declarative one. We would argue that the principle achievement of N3 is to provide a mechanism for exploration through bringing the dynamic properties of EDEN to a language similar to that of mathematics. In doing so, the complexity of actually programming the neural model is reduced when compared to alternative procedural languages. We are also of the opinion that through the integration of SCOUT and DONALD the experience of the model can be extended further to a point where the user can actually experience the mathematical definition of their model within the context of its operation. As an example of this we constructed a bottleneck network where the user is actually capable of observing how the network behaves with respect to visible inputs as opposed to a series of numbers. This experience is a dramatic im-

provement over tools such as Mathematica, whereby inputs are numbers and definitions instead of rich, visible inputs as in our model.

The main contributions of this paper are to demonstrate a potential for modelling neural networks within the Empirical framework. We began by stating that many neural networks are highly abstracted from the biological system being reasoned about. In this way, network modellers are consistently seeking to find more faithful machine representations of the system of study. The Empirical approach allows us a much higher degree of freedom to explore our models, to engage in alterations and refinements as we are developing our models. With this in mind, we conclude that Empirical Modelling does have a lot to offer in the modelling of Neural Networks furthermore, the ability of the philosophy to allow interaction is, we claim, richer than conventional modelling tools.

Acknowledgements

The author would like to thank both Dr. Steve Russ and Dr. Meurig Beynon for providing a valuable introduction to the world of Empiricial Modelling and also the PhD tutors of the module for help inside and outside of lab sessions. Thanks must also be given to Professor Roland Wilson for a fascinating introduction to Neural Networks.

References

W.M. Beynon. Agent-oriented Modelling and the Explanation of Behaviour. *Shape Modelling Parallelism, Interactivity and Applications*, pages 54–63, September 1994.

W.M. Beynon. Concurrent Systems Modelling: Agentification, Artefacts, Animation. *MSc Lecture Series*, November 2005.

W.M. Beynon and S.B. Russ. The Development and Use of Variables in Mathematics and Computer Science. *The Mathematical Revolution inspired by Computing IMA Conf Series 30*, pages 285–95, 1991.

C. Brown. Agent-based Parsing System in EDEN. May 2001.

K. Gurney. *An Introduction to Neural Networks*. University College London Press, 1997.

A. Harfield. Agent-Oriented Parser. May 2003.

S. Haykin. *Neural Networks - A Comprehensive Foundation*. Pearson Education, second edition, 1999.

D.O. Hebb. *The Organization of Behaviour*. John Wiley and Sons, 1949.

J.J. Hopfield. Neural networks and physical systems with emergent collective computational properties. In *Proceedings of the National Academy of Sciences of the USA*, volume 79, pages 2554–2588, 1982.

W. James. *Some Problems of Philosophy*. Harvard University Press, second edition, 1979.

T. Kohonen. The Self-Organizing Map. In *IEEE*, volume 7, pages 1464 – 1480, 1990.

W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 7:115–133, 1942.

A. Ward. *Interaction with Meaningful State: Implementing Dependency on Digital Computers*. PhD thesis, University of Warwick, 2004.

S. Yung. *Definitive Programming - A Paradigm for Exploratory Programming*. PhD thesis, University of Warwick, October 1992.