Symbolic Encoding of Neural Networks using Communicating Automata with Applications to Verification of Neural Network Based Controllers

Li Su, Howard Bowman and Brad Wyble
Centre for Cognitive Neuroscience and Cognitive Systems, University of Kent,
Canterbury, Kent, CT2 7NF, UK
{ls68,hb5,bw5}@kent.ac.uk

Abstract

This paper illustrates a way for applying formal methods techniques to specifying and verifying neural networks, with applications in the area of neural network based controllers. Formal methods have some of the characteristics of symbolic models. We describe a communicating automata [Bowman and Gomez, 2005] model of neural networks, where the standard Backpropagation (BP) algorithm [Rumelhart *et al.*, 1986] is applied. Then we undertake a verification of this model using the model checker Uppaal [Behrmann *et al.*, 2004], in order to predict the performance of the learning process. We discuss broad issues of integrating symbolic techniques with complex neural systems. We also argue that symbolic verifications may give theoretically well-founded ways to evaluate and justify neural learning systems.

1 Introduction

This work is an initial step towards integrating symbolic and neural computation. Our motivations are justified from a cognitive point of view on the one hand, and an engineering application point of view on the other hand.

1.1 Cognitive Viewpoint

[Barnard and Bowman, 2004] suggested that one of the motivations of integrating symbolic and sub-symbolic computation is to specify and justify behavior of complex cognitive architectures in an abstract and suitable form. Firstly, numerous theories assume that mental modules, or their neural substrates, are processing information at the same time. Hence, any realistic architecture of the mind must be concurrent at some level. Secondly, the control of concurrent processing can be distributed. So, cognition should be viewed as the behavior that emerges from interaction amongst independently evolving modules. Most traditional AI approaches fail to acknowledge the requirements of concurrent and distributed control. This is because they tend to be prescriptive and are built around centralized memory and control algorithms. Finally, [Fodor and Pylyshyn, 1988] argued that hierarchical decomposition is needed in order to reflect the characteristics of the mind.

Although connectionist networks are commonly regarded as concurrent and distributed, they are typically limited to only one level of concurrently evolving modules. The primitive elements of neural networks are neurons but not neural networks. To some degree, it is hard to construct and understand large architectures without hierarchical structuring. In certain respects, modeling based on neural networks is low-level in character, i.e. it is hard to relate to primitive constructs and data structures found in high-level notations preferred by the symbolic modeling community.

For these reasons, [Barnard and Bowman, 2004] provided an illustration of modeling a high-level cognitive architecture using formal methods. Their model contains a set of top-level modules that are connected by communication channels. Modules interact by exchanging data items along channels. Control is distributed and each module evolves independently. They also suggested encoding low-level neural networks using the same method, and formally relating models at different levels of abstraction. In this paper, key constructs within neural networks are encoded at two levels of description, which have characteristics of symbolic systems. The low-level descriptions use neural networks encoded in communicating automata. We argue that this formalism sits between classical forms of symbolic systems arising from programming languages such as Lisp and Prolog, and connectionist networks. We will explain these models in section 2. The high-level descriptions contain a set of properties, which are expressed in logical formulae. They are abstract descriptions of global properties, which do not prescribe internal details of how those properties are realised. We will explain these models in section 3. Computer scientists have developed a number of theories and tools to automatically justify the relationship between different levels of description within a formal framework. Our models prescribe low-level internal structure, which we hypothesis can be used to explore complex interactions within neural networks and to justify whether high-level properties can emerge from low-level constructs.

1.2 Application Viewpoint

Symbolic systems are good for manipulating, explaining and reasoning about complex data structures, but neural networks are good at dealing with complex highly non-linear systems, especially in handling catastrophic changes or gradual degradations. It is argued by [Schumann *et al.*, 2003] that neural networks can be applied to extending traditional controllers, which are ineffective in some systems, including aircrafts, spacecrafts, robotics and flexible manufacturing systems. Neural network based controllers have demonstrated a superior ability to control adaptive systems. However, the correctness of adaptive systems must be guaranteed in safety/mission critical domains. This is because it is not possible to adapt toward controllable behaviours when the system has changed beyond a critical point. This is also because the system has to dynamically react to changes within short periods of time. So, this requires that the learning processes converge before a pre-specified deadline.

Unfortunately, the slow speed of learning is one of the greatest limitations of current learning algorithms. For example, the standard BP algorithm often requires the training patterns to be presented hundreds or thousands of times in order to solve a relatively simple task. Furthermore, connectionist networks rarely provide any indication of the accuracy and reliability of their predictions. As long ago as 1988, [Fodor and Pylyshyn, 1988] pointed out that the neural networks approach remained almost entirely experimental. Although a great deal of mathematical work has been done, it is still not sufficient from the analytical point of view to justify that certain configurations of neural networks and their mechanisms are reliable.

In some applications, the control architecture uses pre-trained networks, which are numerical approximations of a function. The correctness of such systems can be verified [Rodrigues *et al.*, 2001], but their verification does not consider the adaptability of the system. Other control architectures use on-line training of neural networks. This approach is attractive because it is able to handle dynamic adaptation, but it requires a high level of stability and correctness of the learning process. There are existing approaches to evaluate the performance of neural networks, such as [Schumann *et al.*, 2003], who proposed a layered approach to verify and validate neural network based controllers. The limitation of their work is that they only focus on monitoring the on-line adaptation but cannot guarantee stability and correctness at system design stages.

This paper describes a case study, which applies formal methods techniques to evaluating the learning speed using automatic analysis (model checking). Formal methods are strongly based on logic. They have rich tool support and have shown their power in software engineering and various areas where correctness and effectiveness of computer systems need to be guaranteed. So they can, for example, be used in designing distributed systems [Bowman and Derrick, 2001]. In these areas, there are similar problems and requirements in respect of modeling complex interactions among components with distributed control.

2 Communicating Automata Specification

In this section, we introduce a communicating automata specification of neural networks, which may be used to specify components of neural network based controllers. The interested reader is referred to [Bowman and Gomez, 2005] for comprehensive definition of communicating automata. A similar framework was presented by [Smith, 1992] in a general mathematical setting. But his work did not consider automatic simulation or verification.

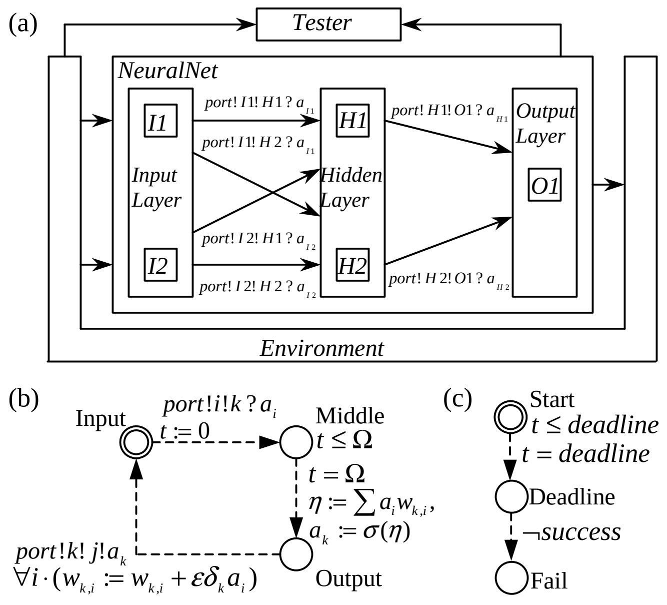

Figure 1: (a) Neural Network that Learns XOR. (b) Example of a Neuron Automaton (c) The Test Automaton, *Tester*.

2.1 Neural Network Description

Communicating automata are Finite State Machines with associated communication channels and mathematical equations. Each neuron is encoded as an automaton, denoted as the smallest square boxes in Figure 1 (a). Automata evolve concurrently and the state of each neuron only depends on the local data structure, which may change as a result of interaction and communication between automata. Activation exchange between neurons is modeled through communication channels. In Figure 1 (a), each arrow denotes a communication channel, such as the channel between neuron *I1* and *H1*:

$$port!I1!H1?a_{I1}$$

Working from left to right, *port* denotes the communication name (which in this case, is shared by all interactions), *I1* denotes the pre-synaptic neuron identity, defining the sender, and *H1* denotes the post-synaptic neuron identity, defining the receiver. The last element a_{I1} denotes the activation passed though the channel.

The system in Figure 1 (a) is described as a hierarchy of components, at the top-level it has three modules: *Environment*, *NeuralNet* and *Tester*. Each of which can be composed from a set of modules. For example, *NeuralNet* itself is composed of *InputLayer*, *HiddenLayer* and *OutputLayer*, each of which is also a module composed of a set of neurons. Neurons are fully connected between adjacent layers and BP learning is applied. The *Environment* automaton provides inputs to and receives outputs from *NeuralNet*. The *Tester* will be used in the verification in section 4.

The BP algorithm is a supervised learning rule widely used in many applications, so study of this algorithm has practical value. We have chosen the XOR problem as our learning task due to its historical position. Although it requires a small number of nodes and connections, it is characteristic of difficult linearly inseparable learning tasks. This simple problem is often used to test the ability of learning algorithms and it has been much discussed. In terms of our larger ambition, analyses of neural network based controllers, this XOR verification serves as a preliminary assessment of our approach, which will be extended to realistic applications in future work.

2.2 Neuron Automaton

We define the neuron automaton based on a set of functions, which describe the network updating dynamics. An example of a neuron automaton is shown in Figure 1 (b), where circles denote locations of the automaton, circles with a smaller circle inside denote initial locations, and arrows with dotted lines denote transitions between two locations. *k* denotes the identity of this neuron, *i* and *j* denote the pre-synaptic and post-synaptic neuron identities respectively. Note that neuron identities are assumed to be unique.

Briefly, the neuron automaton begins at the *Input* location. When all pre-synaptic activations have been received from input channels, it moves to the next location, *Middle*. Then it evaluates the net input η and the activation a_k. σ is a sigmoid function. At the *Output* location, it sends its activation via output channels, and weights are updated. ε is the learning rate and δ_k denotes the extent to which neuron *k* is in error. It is evaluated externally, an explanation of which is beyond the scope of this paper. For simplicity of presentation, we show a neuron automaton with just one input and one output, but a more general form can be defined.

The timing constraints in this application are the following. $t \leq \Omega$ is an invariant of the *Middle* location, and *t* is a local clock. Invariants are timing conditions, and automata can only stay in locations while the condition holds. $t = \Omega$ is a guard, which is a condition allowing the transition to be taken. To summarize the time course of the neuron, it stays at the *Input* and *Output* locations while communication is completing, but it stays at the *Middle* location for exactly Ω units of time, which we assume represents the time required to update net input and activation. In this case, Ω is 5 units of time. This assumption is made only for analytical reasons and is not based on neuron physiology.

3 Requirements Language

The high level behaviour of neural networks is described using a requirements language allowing logical formulae to be expressed. The network of automata evolves through a series of states, which form several paths. The system can evolve through different paths. The requirements language consists of state formulae, which describe individual states and path formulae, which quantify over paths of the model. Assuming φ is a state formula, $\forall$, $\exists$, $\Diamond$ and $\Box$ are operators of path formulae. The property $\forall\Box\varphi$ requires that all states satisfy φ, $\exists\Diamond\varphi$ requires that at least one reachable state satisfies φ, $\exists\Box\varphi$ requires that at least along one path all states satisfy φ and $\forall\Diamond\varphi$ requires that along all paths at least one state eventually satisfies φ.

In this paper we are interested in learning. There is a set of properties, which we want the learning system to satisfy. These properties fall into three categories: Reachability, Safety and Liveness [Behrmann *et al.*, 2004].

- **Reachability Properties**
  These ask whether there eventually exists a state in which something will happen. For example, the formula *success* is *true* when all the output neurons get their activations on the correct side of 0.5. Thus, $\exists\Diamond\, success$ checks if the learning process could eventually allow the network to output the correct answer. These properties validate the basic behaviour of the model, but do not guarantee the correctness of adaptive systems.

- **Safety Properties**
  These ask whether something "bad" will ever happen. For example, the property *deadlock* evaluates to *true* at a state without successors. Thus, $\forall\Box\neg deadlock$ justifies that the system is free from such situations. Safety properties can always be expressed as reachability properties, such as $\neg\exists\Diamond\, deadlock$. Assuming approperate formulation of properties, neural networks, cognitive models or any dynamic systems will never reach "bad" states if they satisfy safety properties.

- **Liveness Properties**
  These ask whether the system eventually does something useful. So, we could check liveness properties such as $\forall\Diamond success$, which justifies that the system can meet our requirements along all paths. By verifying these properties over learning algorithms, we can justify if they will eventually converge at desired situations. In order to understand learning algorithms or adaptive systems in general, we are also interested in whether the models can perform something infinitely often.

Properties can be expressed by nesting different types of operators, such as $\forall\Diamond\forall\Box\, success$. This property describes a complex behaviour that we require the learning process to possess. This behaviour is that the adaptive system starts with no knowledge of the task. At this time, *success* does not hold. During training with examples, it may sometimes show correct answers, i.e. *success* holds, but it may also show incorrect answers. When this happens, the system has found some solutions to the task but these solutions are not stable during further training. However, starting from some states during the training, it is able to show correct answers invariantly. Thereby, the above property is satisfied.

4 Verification

We implemented and verified our model in Uppaal, which is a well-known real-time model checker [Behrmann *et al.*, 2004]. So, our models are converted into Uppaal timed automata notation, which is a real-time extension of communicating automata. We assume that the *deadline* is 5000 units of time, the learning rate is 0.05 and all the weights are randomly distributed around 0.2. Note that the *deadline* and Ω are both arbitrary numbers but they can be specified in real applications. We check the system for deadlock freedom using the following Temporal Logic formula:

$$\forall \Box \neg deadlock$$

The result is that the system has no deadlocks for the XOR training set. We also verify the stability of the network using the following Temporal Logic formula, which contains timing constraints:

$$\forall \Diamond_{\leq deadline} \forall \Box\, success$$

Satisfaction of this formula means the system always reaches a successful state before the *deadline*, and *success* holds invariantly from that state. However, Uppaal does not support this formula. Hence, we introduce a test automaton in Figure 1 (c). It begins at the *Start* location and moves to the next location, *Deadline*, when clock *t* equals the parameter *deadline*. When the test automaton is in this location, the patterns are still presented to the neural network. If learning is successful when *deadline* becomes *true* and remains stable during further training, the next location, *Fail*, is unreachable. With this test automaton, we are able to verify the previous property using the following Temporal Logic formula:

$$\forall \Box \neg Tester.Fail$$

The result of the verification is that the system satisfies the above property and is guaranteed to learn XOR according to the required timing constraints. It also guarantees the learning process is eventually stabilised.

5 Discussion and Future Work

In traditional neural network simulations, semantically interpretable elements are patterns of activation. The states of neural networks are expressed in numerical form, such as a landscape, in a multi-dimensional space. However, the states of the neural networks in our models are represented as the locations in the product automaton, which is automatically generated by the model checker. The locations have the characteristics of symbol systems. Model checking is based on symbolic manipulation of the product automaton, which maps to the landscape in the multi-dimensional space.

In this paper, we have specified a neural network that learns the XOR problem using communicating automata. We then verified the model over a set of properties expressed in Temporal Logic. We believe that this approach can provide insight to the field of neural network based controllers. Our models and properties can be regard as symbolic descriptions of the system behaviour at different levels of abstraction. Verifications may give theoretically well-founded ways to evaluate and justify the learning capacity and determine whether cognitive properties can emerge from neural-level architectures.

We argue that most of the properties, which we have verified, are hard to justify by simulation. This is because simulations can only test that something occurs but are unable to test that something can never occur. Simulations are also not able to test if something is (in)valid forever. Therefore, simulations are limited in their capacity to justify safety and liveness properties without explicit mathematical analysis. Our verification approach on the other hand, explicitly and formally describes the system and properties, so safety and liveness properties can be verified.

The next step of research is to explore different configurations of neural networks, e.g. winner-take-all networks, recurrent networks and more biologically plausible implementations of networks. We are also interested in investigating the potential of this approach in integrating symbolic and sub-symbolic computations, enforcing hierarchical compositional structure over neural networks, and building or justifying brain-level models.

References

[Barnard and Bowman, 2004] P. J. Barnard and H. Bowman. Rendering Information Processing Models of Cognition and Affect Computationally Explicit: Distributed Executive Control and the Deployment of Attention, *Cognitive Science Quarterly,* 3(3):297-328, 2004.

[Bowman and Derrick, 2001] H. Bowman and J. Derrick (editors). *Formal Methods for Distributed Processing: a Survey of Object-oriented Approaches*, Cambridge University Press, 2001.

[Bowman and Gomez, 2005] H. Bowman and R. Gomez. *Concurrency Theory - Calculi and Automata for Modelling Untimed and Timed Concurrent Systems*. Springer-Verlag, To Appear, 2005.

[Behrmann *et al.*, 2004] G. Behrmann, A. David and K. G. Larsen. A Tutorial on Uppaal. *SFM-RT'04, LNCS 3185,* Springer-Verlag, 2004.

[Fodor and Pylyshyn,1988] J. A. Fodor and Z. W. Pylyshyn. Connectionism and Cognitive Architecture: A Critical Analysis, *Cognition: International Journal of Cognitive Science*, Vol. 28, 3-71, 1988.

[Rumelhart *et al*., 1986] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning Internal Representations by Error Propagation. *Paralled Distributed Processing. Explorations in the Microstructure of Cognition*. Vol.1: Foundations, 318-362. MIT Press, 1986.

[Rodrigues *et al*., 2001] P. Rodrigues, J. F. Costa and H. T. Siegelmann. Verifying Properties of Neural Networks. *Artifical and Natural Neural Networks*, *LNCS 2084,* Springer-Verlag, 158-165, 2001.

[Schumann *et al*., 2003] J. Schumann, P. Gupta and S. Nelson. On Verification & Validation of Neural Network Based Controllers, *EANN'03*, 2003.

[Smith, 1992] L. S. Smith. A Framework for Neural Net Specification, *IEEE Transactions on Software Engineering*, 18(7): 601 - 612, 1992.