Fewer Epistemological Challenges for Connectionism

Artur S. d'Avila Garcez*

Dept. of Computing
School of Informatics
City University
London EC1V OHB, UK
aag@soi.city.ac.uk

Abstract. Seventeen years ago, John McCarthy wrote the note *Epistemological challenges for connectionism* as a response to Paul Smolensky's paper *On the proper treatment of connectionism*. I will discuss the extent to which the four key challenges put forward by McCarthy have been solved, and what are the new challenges ahead. I argue that there are fewer epistemological challenges for connectionism, but progress has been slow. Nevertheless, there is now strong indication that neural-symbolic integration can provide effective systems of expressive reasoning and robust learning due to the recent developments in the field.

1 Introduction

This paper is about the integration of neural networks and symbol processing; it is about how to represent, learn, and compute expressive forms of symbolic knowledge using neural networks. I believe this is the way forward towards the provision of an integrated system of expressive reasoning and robust learning. The provision of such a system, integrating the two most fundamental phenomena of intelligent cognitive behaviour (i.e. the ability to learn from experience and the ability to reason from what has been learned) has been recently identified by Leslie Valiant as a key challenge for computer science [25]. The goal is to produce biologically plausible models with integrated reasoning and learning capability, in which neural networks provide the inspiration and the machinery necessary for cognitive computation and learning, while logics provide practical reasoning and explanation capabilities to the models, facilitating the interaction between them and the outside world.

In what follows, I will briefly review my recent work (joint with Luis Lamb and Dov Gabbay) on how to integrate logic and neural networks [8, 9]. I will then address the open question of how to represent variables effectively in neural networks, which emerges from my recent work (joint with Dov Gabbay) on how to

* I'm grateful to Luis C. Lamb for useful discussions. This work was partly supported by The Nuffield Foundation UK.

combine neural networks in a principled way [7]. Throughout, I will try and put the recent advances on neural-symbolic integration in the context of John McCarthy's note *Epistemological challenges for connectionism* [17], written as a response to Paul Smolensky's paper *On the proper treatment of connectionism* [22]. Briefly, McCarthy identifies four knowledge representation problems for neural networks: the problem of *elaboration tolerance* (the ability of a representation to be elaborated to take additional phenomena into account); the *propositional fixation* of neural networks (based on the assumption that neural networks cannot represent relational knowledge); the problem of how to make use of any available *background knowledge* as part of learning, and the problem of how to obtain domain *descriptions* from trained networks as opposed to mere discriminations.

I will start by giving examples of how we represent propositional modal logic (and thus relational knowledge) in neural networks, pointing the reader to the papers in the area. I will then discuss our proposal for combining (fibring) neural networks, and how it may allow us to represent variables. In what regards the challenges put forward by McCarthy, in a nutshell, the problem of elaboration tolerance may be resolved by having networks that are fibred in a hierarchy (this is similar to the idea of using self-organising maps [12], e.g., for language processing, in which the lower levels of abstraction are used for the formation of concepts that are then used at the higher levels of the hierarchy); in the case of the so-called propositional fixation of neural networks, connectionist modal logic shows that, as a matter of fact, neural networks can encode relational knowledge (in the form of accessibility relations) [9]; as for learning with background knowledge, this can be achieved by translating symbolic rules into the initial architecture of a neural network; whereas problem description can be obtained by rule extraction from trained neural networks. In the past decade, a number of such translation algorithms [8, 9, 13, 23] and knowledge extraction algorithms [1, 3, 19, 24] has been proposed.

Nevertheless, there are still challenges ahead, particularly in what regards the effective integration of expressive reasoning and robust learning. In this case, we cannot afford to lose on the learning capability side as we add reasoning capability to neural networks. This means that we cannot depart from the idea that neural networks are composed of simple processing units organised in a massively parallel way (and allow for some *clever* neurons to perform complex symbolic computation). We also would like our models to be biologically plausible, not as a principle but in a pragmatic way. There have been recent advances in brain imaging, which offer us data we can make use of to get insight into new forms of representation. Human beings are quite extraordinary at performing practical reasoning as they go about their daily business. *There are cases where the human computer, slow as it is, is faster than Artificial Intelligence systems. Why are we faster? Is it the way we perceive knowledge as opposed to the way we represent it? Do we know immediately which rules to select and apply? We must look for the correct representation in the sense that it mirrors the way we perceive and apply the rules* [10]. Ultimately, Neural-Symbolic integration is about asking and trying to answer these questions.

2 Neural-Symbolic Integration

For neural-symbolic integration to be effective in complex applications, we need to investigate how to represent, reason, and learn expressive logics in neural networks. We also need to find effective ways of expressing the knowledge encoded in a trained network in a comprehensible symbolic form.

There are two ways to move forward and benefit from neural-symbolic integration. The first is to take standard neural networks and try and find out which logics they can represent. The other is to take well established logics and concepts (e.g. recursion) and try and encode them in a neural network architecture. This needs to be carried out in a systematic way. Whenever we show that a particular logic can be represented by a particular neural network, we need to show that the network and the logic are in fact equivalent (a way to do this is to prove that the network computes the semantics of the logic). Similarly, if we develop a knowledge extraction algorithm, we need to make sure that it is correct in the sense that it produces rules that are encoded in the network, and that it is complete in the sense that it produces rules that increasingly approximate the exact behaviour of the network.

In the past twenty years, a number of models for neural-symbolic integration has been proposed. Broadly speaking, researchers have made contributions to three main areas. Neural-symbolic systems provide either: (*i*) a logical characterisation of a connectionist system; (*ii*) a connectionist implementation of a logic; or (*iii*) a hybrid system bringing together advantages from connectionist systems and symbolic artificial intelligence [15]. Key contributions to the area were given by Ron Sun [23], Lokendra Shastri [20], and Steffen Hölldobler [14] on the knowledge representation side, by Jude Shavlik [21] on learning with background knowledge, and by Sebastian Thrun on knowledge extraction [24], among others. The reader is referred to [6] for a general presentation of the subject of neural-symbolic integration, and to [4] for a more advanced collection of papers on the subject.

Neural-symbolic systems [6] contain six main phases: (1) *symbolic knowledge insertion*; (2) *inductive learning with examples*; (3) *massively parallel deduction*; (4) *theory fine-tuning*; (5) *symbolic knowledge extraction*; and (6) *feedback* (see Figure 1). In phase (1), symbolic knowledge is translated into the initial architecture of a neural network with the use of a *Translation Algorithm*. In phase (2), the neural network is trained with examples by a neural learning algorithm, which revises the theory given in phase (1) as *background knowledge*. In phase (3), the network can be used as a massively parallel system to compute the logical consequences of the theory encoded in it. In phase (4), information obtained from the computation carried out in phase (3) may be used to help fine-tuning the network to better represent the problem domain. This mechanism can be used, for example, to resolve inconsistencies between the background knowledge and the training examples. In phase (5), the result of training is explained by the extraction of revised symbolic knowledge. As with the insertion of rules, the *Extraction Algorithm* must be provably correct, so that each rule extracted is guaranteed to be encoded in the network. Finally, in phase (6), the knowledge extracted may be analysed by an expert to decide if it should feed the system once again, closing the learning cycle. A typical application of Neural-Symbolic Systems is in safety-critical domains, e.g. power plant fault diagnosis, where the neural network can be used to detect a fault quickly, triggering safety procedures, while the knowledge extracted from it can be used to explain the reasons for the fault later on. If mistaken, this information can be used to fine tune the learning system.

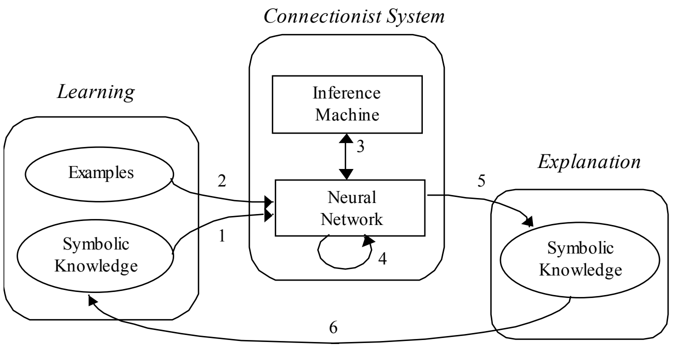

Fig. 1. Neural-Symbolic Learning Systems

In this paper, we focus on knowledge representation (phase (1) above). First, let us see how the *Translation Algorithm* works in the case of general logic programs[1]. Before we proceed, let us define the type of neural network used here. An artificial neural network is a directed graph. A unit in this graph is characterised, at time t, by its input vector $I_i(t)$, its input potential $U_i(t)$, its activation state $A_i(t)$, and its output $O_i(t)$. The units (neurons) of the network are interconnected via a set of directed and weighted connections. If there is a connection from unit i to unit j then $W_{ji} \in \Re$ denotes the *weight* associated with such a connection. The input potential of neuron i ($U_i(t)$) is obtained by applying the *propagation rule* of neuron i (g_i) such that $U_i(t) = g_i(I_i(t), W_i)$, where $I_i(t)$ is the input vector $(x_1(t), x_2(t), ..., x_n(t))$ to neuron i at time t, and W_i denotes the weight vector $(W_{i1}, W_{i2}, ..., W_{in})$ to neuron i. In addition, θ_i (an extra weight with input always fixed at 1) is known as the *threshold* of neuron i.

The *activation state* of neuron i ($A_i(t)$) is a bounded real or integer number given by its *activation rule* (h_i). In general, h_i does not depend on the previous activation state of the neuron, and the propagation rule g_i is a weighted sum such that $A_i(t) = h_i(\sum_j((W_{ij} \cdot x_j(t)) - \theta_i))$. Finally, in general, the output is given by the identity function, and thus $O_i(t) = A_i(t)$.

The units of a neural network can be organised in layers. A *n-layer feedforward network* N is an acyclic graph. N consists of a sequence of layers and

[1] A general clause is a rule of the form $L_1, ..., L_k \rightarrow A$, where A is an atom and L_i $(1 \leq i \leq k)$ is a literal. A general logic program is a finite set of general clauses.

connections between successive layers, containing one input layer, $n-2$ hidden layers and one output layer, where $n \geq 2$. When $n = 3$, we say that N is a *single hidden layer network*. When each unit occurring in the *i-th* layer is connected to each unit occurring in the $i+1$-*st* layer, we say that N is a *fully-connected network*.

Now, let $\mathcal{P}$ be a general logic program, and let $\mathcal{N}$ be a single hidden layer feedforward neural network. Each clause (r_l) of $\mathcal{P}$ can be mapped from the input layer to the output layer of $\mathcal{N}$ through one neuron (N_l) in the single hidden layer of $\mathcal{N}$. Intuitively, the *Translation Algorithm* from $\mathcal{P}$ to $\mathcal{N}$ implements the following conditions: (**C1**) The input potential of a hidden neuron (N_l) can only exceed N_l's threshold (θ_l), activating N_l, when all the positive antecedents of r_l are assigned the truth-value *true* while all the negative antecedents of r_l are assigned $false$; and (**C2**) The input potential of an output neuron (A) can only exceed A's threshold (θ_A), activating A, when at least one hidden neuron N_l that is connected to A is activated.

Example: Consider the logic program $\mathcal{P} = \{B \wedge C \wedge \neg D \rightarrow A; E \wedge F \rightarrow A; B\}$. The *Translation Algorithm* derives the network $\mathcal{N}$ of Figure 2, setting weights (W) and thresholds (θ) in such a way that conditions (**C1**) and (**C2**) above are satisfied. Note that, if $\mathcal{N}$ ought to be fully-connected, any other link (not shown in Figure 2) should receive weight zero initially.

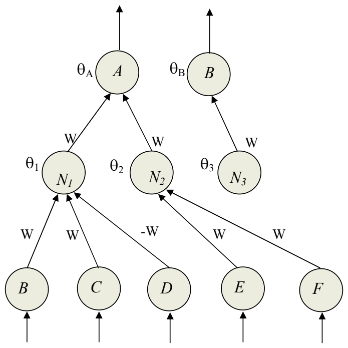

Fig. 2. Neural Network for Logic Programming

Note that, in the above example, each input and output neuron of $\mathcal{N}$ is associated with an atom of $\mathcal{P}$. As a result, each input and output vector of $\mathcal{N}$ can be associated with an interpretation for $\mathcal{P}$. Note also that each hidden neuron N_l corresponds to a rule r_l of $\mathcal{P}$. In order to compute the stable models of $\mathcal{P}$, output neuron B should feed input neuron B such that $\mathcal{N}$ is used to iterate $T_{\mathcal{P}}$, the fixed-point operator of $\mathcal{P}$. $\mathcal{N}$ will eventually converge to a stable state which is identical to a stable model of $\mathcal{P}$ [6].

Details about the translation and extraction algorithms, their proofs of correctness, and extensions to other types of logic program can be found in [6], together with algorithms to deal with inconsistencies and experimental results in the areas of DNA sequence analysis, power systems fault diagnosis, and the evolution of requirements in software engineering.

3 Connectionist Modal Logic

Let us now consider the case of modal logic programs, which extend logic programs with the *necessity* ($\Box$) and *possibility* ($\Diamond$) modalities, according to an accessibility relation $\mathcal{R}(\omega_i, \omega_j)$ on possible worlds ω_i and ω_j. I will give an example of how neural networks can represent such modalities. The basic idea is simple. Instead of having a single network, if we now allow a number of networks (like the one in Figure 2) to occur in an ensemble, and we label the networks as w_1, w_2, etc, we can talk about having x in w_1 and having x in w_2. In this way, we can see w_1 as a possible world and w_2 as another, and this allows us to represent modal logic programs[2]. This is of interest in connection with McCarthy's conjecture on the propositional fixation of neural networks because there is a well established translation between propositional modal logic and the two-variable fragment of first order logic[3] [26], which indicates that neural-symbolic systems may go beyond propositional logic, thus contradicting McCarthy's conjecture.

Example: Let $\mathcal{P} = \{\omega_1 : r \rightarrow \Box q;\ \omega_1 : \Diamond s \rightarrow r;\ \omega_2 : s;\ \omega_3 : q \rightarrow \Diamond p;\ \mathcal{R}(\omega_1;\omega_2), \mathcal{R}(\omega_1,\omega_3)\}$. The network ensemble $\mathcal{N}$ in Figure 3 is equivalent to $\mathcal{P}$. Take network $\mathcal{N}_1$ (representing ω_1). To implement the semantics of $\Diamond$, output neurons of the form $\Diamond\alpha$ should be connected to output neurons α in an arbitrary network $\mathcal{N}_i$ (representing ω_i) to which $\mathcal{N}_1$ is related. For example, taking $i = 2$, $\Diamond s$ in $\mathcal{N}_1$ is connected to s in $\mathcal{N}_2$. To implement the semantics of $\Box$, output neurons $\Box\alpha$ should be connected to output neurons α in every network $\mathcal{N}_i$ to which $\mathcal{N}_1$ is related. For example, $\Box q$ in $\mathcal{N}_1$ is connected to q in both $\mathcal{N}_2$ and $\mathcal{N}_3$. Dually, taking $\mathcal{N}_2$, output neurons α need to be connected to output neurons $\Diamond\alpha$ and $\Box\alpha$ in every world $\mathcal{N}_j$ related to $\mathcal{N}_2$. For example, s in $\mathcal{N}_2$ is connected to $\Diamond s$ in $\mathcal{N}_1$ via the hidden neuron denoted by $\vee$ in Figure 3, while q in $\mathcal{N}_2$ is connected to $\Box q$ in $\mathcal{N}_1$ via the hidden neuron denoted by $\wedge$. Similarly, q in $\mathcal{N}_3$ is connected to $\Box q$ in $\mathcal{N}_1$ via $\wedge$. The translation terminates when all output neurons have been considered. The translation algorithm defines the weights and thresholds of the network in such a way that it can be shown to compute a fixed-point semantics of the modal logic program associated to it (for any extended modal

[2] An *extended modal program* is a finite set of clauses C of the form $\omega_i : ML_1, ..., ML_n \rightarrow MA$, where ω_i is a label representing a world in which the associated clause holds, and $M \in \{\Box, \Diamond\}$, together with a finite set of relations $\mathcal{R}(\omega_i, \omega_j)$ between worlds ω_i and ω_j in C.

[3] In [26], Vardi states that "(propositional) modal logic, in spite of its apparent propositional syntax, is essentially a first-order logic, since the necessity and possibility modalities quantify over the set of possible worlds... the states in a Kripke structure correspond to domain elements in a relational structure, and modalities are nothing but a limited form of quantifiers". In the same paper, Vardi then proves that propositional modal logics correspond to fragments of first-order logic.

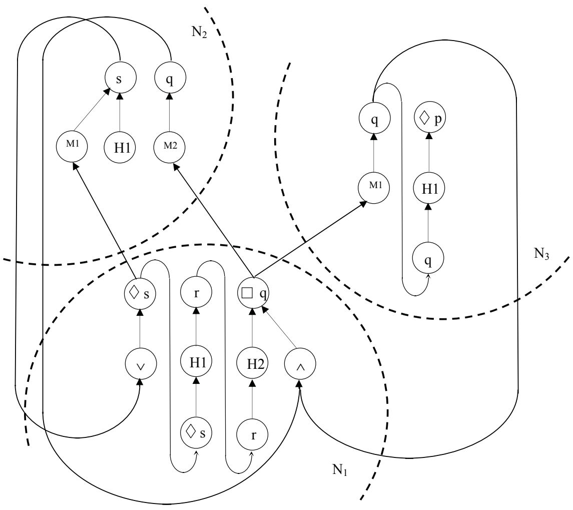

Fig. 3. Connectionist Modal Logic

program $\mathcal{P}$ there exists an ensemble of feedforward neural networks $\mathcal{N}$ with a single hidden layer and semi-linear neurons, such that $\mathcal{N}$ computes the modal fixed-point operator $MT_{\mathcal{P}}$ of $\mathcal{P}$). The proof and details about the algorithm can be found in [9]. Finally, as we link the neurons in the output layer to the corresponding neurons in the input layer of each network $\mathcal{N}_i$, the ensemble can be used to compute the modal program in parallel. In this example, we connect output neurons $\Diamond s$ and r to input neurons $\Diamond s$ and r, respectively, in $\mathcal{N}_1$, and output neuron q to input neuron q in $\mathcal{N}_3$. The ensemble converges to a stable state containing $\{\Diamond s, r, \Box q\}$ in ω_1, $\{s, q\}$ in ω_2, and $\{q, \Diamond s\}$ in ω_3.

4 Fibring Neural Networks

In Connectionist Modal Logic (CML), one needs to create copies of certain concepts. As a result, CML cannot deal with infinite domains, since this would require infinitely many copies. An alternative is to map the instances of a variable onto the reals, and then use real numbers as inputs to a neural network as a way of representing variables. This has been done in [15], in which a theorem shows that the fixed-point semantics of first order logic programs can be approximated arbitrarily well by neural networks very similar to the one depicted in Figure 2. However, the question of which neural network approximates a given first order program remained, since no translation algorithm has been introduced in [15]. Recently, we have followed the idea of representing variables as real numbers, and proposed a translation algorithm from first order acyclic programs to neural network ensembles [2]. The algorithm makes use of *fibring* of neural networks [7], which we discuss in the sequel. Briefly, the idea is to use a neural network to iterate a global counter n. For each clause C_i in the logic program, this counter is combined (fibred) with another neural network, which determines whether C_i outputs an atom of level n for a given interpretation I. This allows us to translate programs having an infinite number of ground instances into a finite neural network structure (e.g. $\neg even(x) \rightarrow even(s(x))$ for $x \in \mathbb{N}, s(x) = x + 1$), and to prove that indeed the network approximates the fixed-point semantics of the program. The translation is made possible because fibring allows one to implement a key feature of symbolic computation in neural networks, namely, recursion.

The idea of fibring neural networks is simple. Fibred networks may be composed not only of interconnected neurons but also of other networks, forming a recursive architecture. A fibring function then defines how this recursive architecture must behave by defining how the networks in the ensemble relate to each other. Typically, the fibring function will allow the activation of neurons in one network (A) to influence the change of the weights in another network (B) (e.g. by allowing the activation state of a neuron in A to be multiplied by the weights of neurons in B). Intuitively, this may be seen as training network B at the same time that one runs network A. Interestingly, albeit being a combination of simple and standard neural networks, fibred networks can approximate any polynomial function in an unbounded domain, thus being more expressive than standard feedforward networks (which are universal approximators of functions in compact, i.e. closed and bounded, domains only) [7]. For example, fibred networks compute $f(x) = x^2$ exactly for $x \in \mathbb{R}$.

Figure 4 exemplifies how a network (B) can be fibred into a network (A). Of course, the idea of fibring is not only to organise networks as a number of subnetworks (A, B, etc). In Figure 4, for example, the output neuron of A is expected to be a neural network (B) in its own right. The input, weights, and output of B may depend on the activation state of A's output neuron, according to the fibring function φ. One such function may be simply to *multiply* the weights of B by the activation state of A's output neuron. Fibred networks can be trained from examples in the same way that standard networks are (for

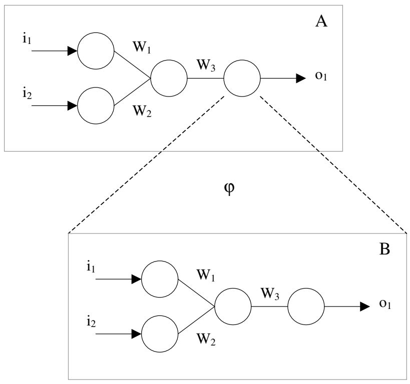

Fig. 4. Fibring Neural Networks

example, with the use of *backpropagation* [18]). Networks A and B above, e.g., could have been trained separately before having been fibred. Notice also that, in addition to using different fibring functions, networks can be fibred in a number of different ways as far as their architectures are concerned. Network B above, e.g., could have been fibred into a hidden neuron of network A.

I believe that fibring can contribute to the solution of the problem of elaboration tolerance by offering a principled and modular way of combining networks. Network A could have been trained, e.g., with a robot's visual system, while network B would have been trained with its planning system, and fibring would serve to perform the composition of the two systems (along the lines of Gabbay's methodology for fibring logical systems [11]). Of course, a lot of work still remains to be done in this area, particularly in what regards the question of how one should go about fibring networks in real applications.

5 Concluding Remarks

I see CML as an example of how neural networks can contribute to logic, and I see fibring as an example of how logic can bring insight into neural networks. CML offers a parallel model of computation to modal logic that, at the same time, can be integrated with an efficient learning system. Fibring is a clear example of how concepts from symbolic computation may help in the development of new neural network models (this does not necessarily conflicts with the concept of biological plausibility, e.g. fibring functions can be understood as a model of *presynaptic weights*, which play an important role in biological neural networks).

Together with its algorithms for learning from examples and background knowledge [6] and for knowledge extraction from trained neural networks [5] (which I have neglected in this paper), I believe that neural-symbolic integration finally starts to address all the challenges put forward by McCarthy. On the other hand, there are new challenges now, which arise directly from our goal of integrating reasoning and learning in a principled way, as put forward by Valiant [25].

In my opinion, the key challenges ahead are: how to get a *constructive translation* of variables into simple neural networks, and how to have a *sound and complete extraction* method that is also efficient for large-scale networks. Let me try and explain what I mean by a constructive translation. In the propositional case, when we look at a neural network, we can see the literals and their relationship with other literals explicitly represented as neurons and their connections with other neurons in the network. In the same way, we would like to be able to look at a first order neural network and see the variables and their relationship with other variables explicitly represented in the network. Although fibring allows us to translate first order programs into neural networks, the current translation algorithm does not produce one such constructive view of the network. As a result, we still do not know how to learn first order programs using neural networks, and I believe that a constructive translation would help shed light into this learning problem. Due to the success of the propositional case, I am convinced that such a representation would allow for effective learning if it kept the networks simple. This is still a big challenge. Of course, we will need to be much more precise as we develop this work, and we will need to keep an eye on the recent developments in the area of logic and learning [16].

References

1. R. Andrews, J. Diederich, and A. B. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based Systems*, 8(6):373–389, 1995.
2. S. Bader, A. S. d'Avila Garcez, and P. Hitzler. Computing first order logic programs by fibring artificial neural networks. In Proceedings of International FLAIRS Conference, Florida, USA, AAAI Press, 2005.
3. G. Bologna. Is it worth generating rules from neural network ensembles? *Journal of Applied Logic, Special issue on Neural-Symbolic Integration*, A. S. d'Avila Garcez, D. Gabbay, S. Hölldobler, J. G. Taylor (eds.), 2(3):325–348, 2004.
4. I. Cloete and J. M. Zurada, editors. *Knowledge-Based Neurocomputing.* The MIT Press, 2000.
5. A. S. d'Avila Garcez, K. Broda, and D. M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125:155–207, 2001.
6. A. S. d'Avila Garcez, K. Broda, and D. M. Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications.* Perspectives in Neural Computing. Springer-Verlag, 2002.
7. A. S. d'Avila Garcez and D. M. Gabbay. Fibring neural networks. In *Proceedings of 19th National Conference on Artificial Intelligence AAAI'04*, pages 342–347, San Jose, California, USA, AAAI Press. July 2004.
8. A. S. d'Avila Garcez and L. C. Lamb. Reasoning about time and knowledge in neural-symbolic learning systems. In S. Thrun, L. Saul, and B. Schoelkopf, editors, *Advances in Neural Information Processing Systems 16*, Proceedings of the NIPS 2003 Conference, pages 921–928, Vancouver, Canada, MIT Press. December 2004.
9. A. S. d'Avila Garcez, L. C. Lamb, K. Broda, and D. M. Gabbay. Applying connectionist modal logics to distributed knowledge representation problems. *International Journal of Artificial Intelligence Tools*, 13(1):115–139, 2004.
10. D. M. Gabbay. *Elementary Logics: a Procedural Perspective.* Prentice Hall, 1998.
11. D. M. Gabbay. *Fibring Logics.* Oxford Univesity Press, 1999.
12. S. Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall, 1999.
13. P. Hitzler, S. Hlldobler, and A. Seda. Logic programs and connectionist networks. *Journal of Applied Logic, Special issue on Neural-Symbolic Integration*, A. S. d'Avila Garcez, D. Gabbay, S. Hölldobler, J. G. Taylor (eds.), 2(3):245–272, 2004.
14. S. Hölldobler. Automated inferencing and connectionist models. Postdoctoral Thesis, Intellektik, Informatik, TH Darmstadt, 1993.
15. S. Hölldobler, Y. Kalinke, and H. P. Storr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence Journal, Special Issue on Neural Networks and Structured Knowledge*, F. Kurfess (ed.), 11(1):45–58, 1999.
16. J. W. Lloyd. *Logic for Learning: Learning Comprehensible Theories from Structured Data.* Springer-Verlag, 2003.

17. J. McCarthy. Epistemological challenges for connectionism. *Behavior and Brain Sciences*, 11(1):44, 1988.
18. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, 1986.
19. R. Setiono. Extracting rules from neural networks by pruning and hidden-unit splitting. *Neural Computation*, 9:205–225, 1997.
20. L. Shastri. Advances in SHRUTI: a neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. *Applied Intelligence Journal, Special Issue on Neural Networks and Structured Knowledge*, F. Kurfess (ed.), 11:79–108, 1999.
21. J. W. Shavlik. An overview of research at Wisconsin on knowledge-based neural networks. In *Proceedings of the International Conference on Neural Networks ICNN96*, pages 65–69, Washington, DC, 1996.
22. P. Smolensky. On the proper treatment of connectionism. *Behavioral and Brain Sciences*, 11:1–74, 1988.
23. R. Sun. Robust reasoning: integrating rule-based and similarity-based reasoning. *Artificial Intelligence*, 75(2):241–296, 1995.
24. S. B. Thrun. Extracting provably correct rules from artificial neural networks. Technical report, Institut fur Informatik, Universitat Bonn, 1994.
25. L. G. Valiant. Three problems in computer science. *Journal of the ACM*, 50(1):96–99, 2003.
26. M. Y. Vardi. Why is modal logic so robustly decidable. In N. Immerman and P. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *Discrete Mathematics and Theoretical Computer Science*, pages 149–184. 1997.