NEURONS OR SYMBOLS
Why does OR Remain Exclusive?

Ekaterina Komendantskaya
School of Computer Science, University of St Andrews, U.K.

Keywords: Computational logic in neural networks, Neuro-symbolic networks, Connectionism, Hybrid networks.

Abstract: Neuro-Symbolic Integration is an interdisciplinary area that endeavours to unify neural networks and symbolic logic. The goal is to create a system that combines the advantages of neural networks (adaptive behaviour, robustness, tolerance of noise and probability) and symbolic logic (validity of computations, generality, higher-order reasoning). Several different approaches have been proposed in the past. However, the existing neuro-symbolic networks provide only a limited coverage of the techniques used in computational logic. In this paper, we outline the areas of neuro-symbolism where computational logic has been implemented so far, and analyse the problematic areas. We show why certain concepts cannot be implemented using the existing neuro-symbolic networks, and propose four main improvements needed to build neuro-symbolic networks of the future.

1 INTRODUCTION

The joint efforts of researchers in many areas have given many insights on how logic and neuroscience can relate[1]: Boolean (binary) networks can compute logical connectives (McCulloch and Pitts, 1943; Aleksander and Morton, 1993); binary threshold networks can simulate Finite Automata, (Universal) Turing machines can be simulated by neural networks with rational weights (Siegelmann, 1999).

The natural question that arises is how neural networks can cope with logical theories and calculi. The traditional approach developed by neuro-symbolic community is based upon McCulloch and Pitts binary neural networks and encodes the *semantics* of first-order logic theories in neural networks. The books (d'Avila Garcez et al., 2002; d'Avila Garcez et al., 2008) and papers (Bader et al., 2008; Domingos, 2008; Hölldobler et al., 1999; Lange and Dyer, 1989; Hitzler et al., 2004; Wang and Domingos, 2008) are good examples of this approach. We discuss here why this kind of view has not yet resulted in a neuro-symbolic system that successfully implements algorithms of computational logic.

[1]The title of this position paper is inspired by the book **"Neurons and Symbols"**, (Aleksander and Morton, 1993), that claimed, rather convincingly, that neural and symbolic approaches to computation and information processing can be considered as two complementary sides of one machine.

Throughout this paper, we make one subtle distinction between the levels of abstraction one uses when speaking about relations between logic and neural networks. The most fundamental results relating logic and neural networks, such as papers by McCulloch and Pitts (McCulloch and Pitts, 1943), Kleene (Kleene, 1956), books by von Neumann (Neumann, 1958) and Minsky (Minsky, 1954; Minsky, 1969), considered neural networks at the level of automata and computing machines, showing that the neural networks, taken as a hardware, can simulate computations performed by digital computers.

This line of research has been successfully developed. There has been series of publications concerning computational complexity of neural networks and their relation to Turing computability. Pollack (Pollack, 1987) showed that a particular type of heterogeneous processor network is Turing universal. Siegelmann and Sontag (Siegelmann and Sontag, 1991) showed the universality of homogeneous networks of first-order neurons having piecewise-linear activation functions. Their results were generalised by Kilian and Siegelmann (Kilian and Siegelmann, 1996) to include various sigmoidal activation functions. The number of neurons required for universality with first-order neurons was estimated at 886 (Siegelmann and Sontag, 1995), and in the later papers was reduced to 96 (Koiran et al., 1994) and down to 25 (Indyk, 1995). In (Siegelmann and Margenstern, 1999), it was

Komendantskaya E. (2009).
NEURONS OR SYMBOLS - Why Does OR Remain Exclusive?.
In *Proceedings of the International Joint Conference on Computational Intelligence*, pages 502-507
DOI: 10.5220/0002334805020507
Copyright © SciTePress

shown that there is a universal neural network with nine switch-affine neurons which is Turing universal. A good summary can be found in (Siegelmann, 1999).

The paper by McCulloch and Pitts (McCulloch and Pitts, 1943) started another line of research into the relation between logic and neural networks. Namely, *Connectionism*, and one of its branches - *Neuro-Symbolic Integration* (Güsgen and Hölldobler, 1992; Bader and Hitzler, 2005; Hammer and Hitzler, 2007) took up this problem at another level of abstraction. Neural networks were taken as a ready-to-use tool in which one could implement some aspects of logical deduction. Unlike the theoretical line of research we described above, this approach did not necessarily involve general translation of automata or Turing machines into Neural networks. On the contrary, it often explored the ways of developing neural networks architectures that could be suitable for a particular task in computational logic. The generality of the theoretical level was sacrificed for convenient and efficient implementations. We give a survey of this approach in Section 2.

We also note here, that this second approach to developing neuro-symbolic networks has been influenced by the fact that the hardware development of digital computers took over from neuro computers; and the development of neural networks became more a question of software than of a hardware. And this justifies the changes in the style of neuro-symbolism.

However, neuro-symbolic networks have not yet become a useful tool in computational logic. Instead, neuro-symbolism has found its place somewhere on the intersection of artificial intelligence, cognitive science, and linguistics. See (Markus, 2001; Smolensky and Legendre, 2006) for very successful examples. In Section 3, we formulate four principles that are essential for implementing algorithms of computational logic. We conclude in Section 4.

2 PROPERTIES OF EXISTING NEURO-SYMBOLIC NETS

Many of the neuro-symbolic networks rely on the representation of logical connectives by McCulloch and Pitts (McCulloch and Pitts, 1943). We start with a simple example concerning a very popular approach to implementing semantic operators for logic programs in neural networks.

Logic programs consist of clauses of the form $A \leftarrow B_1, \ldots, B_n$, where $A, B_1, \ldots, B_n$ are atomic (first-order) formulae. For a given program P, the Herbrand Base B_P is a set that contains all possible ground instances of atomic formulae appearing in P. Given a logic program P, one can define a semantic operator T_P that computes the *least Herbrand model* of P: $T_P(I) = \{A \in B_P : A \leftarrow B_1, \ldots, B_n$ is a ground instance of a clause in P and $\{B_1, \ldots, B_n\} \subseteq I\}$, where I is an interpretation of a program given by a set of propositions that are true. See (Lloyd, 1987) for formal description.

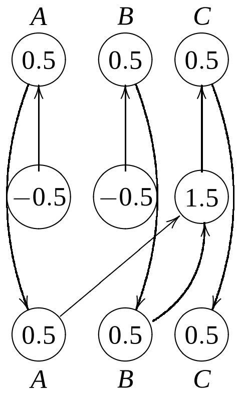
Theorem 1. *(Hölldobler and Kalinke, 1994; Hölldobler et al., 1999) For each propositional logic program P, there exists a 3-layer recurrent neural network built of binary threshold units that computes T_P.*

We will call these networks *T_P-neural networks* and illustrate them below. The theorem extends to function-free first-order logic programs, and other subclasses of first-order logic programs that have finite models. First-order logic programs in their full-generality may have infinite models, and these cases would require infinite neural networks, see (Hitzler et al., 2004; Bader et al., 2008) for discussions and solutions.

Example 1. *Consider the ground logic program:*

$$\begin{aligned} B &\leftarrow \\ A &\leftarrow \\ C &\leftarrow A, B. \end{aligned}$$

The neural network below computes $T_P \uparrow 2 = \{A, B, C\}$, that is, the least model of P, in two iterations. Each of the atoms A, B, C is represented by a neuron in the output and input layers. Connections between the hidden layer and both outer layers are set in a particular way that reflects the structure of clauses. The weights of the recurrent connections between the output and the input layer are set to 1*. That is, all the connections* $\longrightarrow$ *on the following diagram have weights set to* 1*. Numbers* 0.5, −0.5*, etc. show the thresholds (biases) of the neurons. The activation functions are binary, that is, if the neuron receives a signal greater than its threshold, it outputs* 1*; and it outputs* 0 *otherwise.*



There are **three characteristic properties** that distinguish T_P-neural networks. We summarise them as follows.

1. For a given program P, the number of neurons in the input and output layers is the number of atoms in the Herbrand base B_P.
2. Signals of T_P-neural networks are binary, and this provides the computations of truth value functions $\wedge$ and $\leftarrow$ that are used in program clauses.
3. As a consequence of the property (3), first-order atoms are not represented in the neural network directly, and only truth values 1 and 0, that are the same for all the atoms, are propagated.

These three properties arise from a more general **Principle** universally applicable to neuro-symbolic networks of other kinds: **When processing a logic theory, the neural networks process truth values of the ground instances of formulae, and compute the models of the theory as a result. Moreover, each ground instance of an atom from the given theory should be represented by at least one neuron**.

The main **Principle** has **three main consequences**. The resulting network *cannot*:

- directly deal with recursive programs, that is, programs that can have infinitely many ground instances;
- deal with non-ground reasoning that is common in computational logic;
- cover proof-theoretic aspect of logic theories;

One would be surprised to find how often neuro-symbolic systems of different architectures obey the main **Principle** and its three **consequences**. We will consider some examples.

There were attempts to develop an original **Learning Theory** within Neuro-Symbolic integration. In (d'Avila Garcez et al., 2002; d'Avila Garcez et al., 2008), binary threshold units in the hidden layer of T_P networks were replaced by sigmoidal units. Such networks allow back-propagation which can be used to train neural networks. These neural networks obey the general **Principle**, albeit they process signals from the interval $[-1,1]$. They were applied to inductive logic programming, where facts in a database can be assigned some measures of probabilities. The values from the interval $[-1,0]$ were recognised by the output unit as "false", and the values from the interval $[0,1]$ as "true".

The **Markov Logic Networks** (Wang and Domingos, 2008; Domingos, 2008) are another possible modification of T_P-neural networks. Here, the computational power of Markov chains and stochastic (probabilistic) methods was used to give an account of Logics of Probabilities. These networks do not use a semantic operator directly, but the methodology is very close to T_P-networks, in that the networks rely on ground instances of atoms appearing in logic programs, and propagate their truth values. This model has been successfully applied to many practical problems, but it still obeys the main **Principle**.

Propositional Modal Logic Programs (d'Avila Garcez et al., 2007; d'Avila Garcez et al., 2008). Construction of T_P-neural networks was exactly reproduced in this approach, the only difference being that it was adapted to Kripke semantics. That is, in each *possible world* (by Kripke) one could have a separate T_P-neural network computing classical values.

Fibrational Neural Networks. Some research was done on creating networks of T_P-neural networks, they were called *Fibring Neural Networks* in (d'Avila Garcez and Gabbay, 2004).

Many-Valued Logic Programs. This approach was developed in (Komendantskaya et al., 2007; Komendantskaya, 2007). Here, binary neurons represented some atom together with its value, and several additional layers were needed in order to reflect some additional properties of many-valued semantic operators. These networks have the same properties as T_P-networks.

There have been several papers relating **Fuzzy Prolog and Fuzzy Neural Networks**, (Ding, 1995; Nauck et al., 1997; Zadeh, 1992). The resulting neural networks are called *Fuzzy-Logical Neural Networks*. The authors of (Nauck et al., 1997) show that Fuzzy Logic Programs can be simulated by feed-forward neural networks; moreover, they use learning algorithms when working with fuzzy numbers. The *Fuzzy-Logical Neural Nets* are capable of propagating fuzzy signals, and not just 0, 1. If we examine *Fuzzy-Logical Neural Networks* relative to the T_P-neural networks, we will notice that they are not correlated with semantic operators any more, however, they still obey the main **Principle**, in that they work with truth values and functions over truth values, and do not encode first-order atoms directly.

Even with approaches that are not directly tied to logic programs, such as (Pollack, 1990; Shastri and Ajjanagadde, 1993), one can notice that the networks essentially rely upon ground reasoning and binary signal processing.

3 OBSTACLES AND SOLUTIONS

Networks considered in the previous section have been successfully implemented for certain practical or industrial problems. But they did not advance our understanding of how inference algorithms used in computational logic and proof theory could be implemented in neural networks.

In this section, we offer four principles for developing neuro-symbolic networks suitable for simulating the algorithms of computational logic.

1. Implementation of Techniques from Proof Theory should not be Replaced by the Model Theory. There are two reasons why such a replacement cannot lead to successful implementations of computational logic.

In the case of first-order theories, the *semantics* is often developed on the meta-level, not using the *syntax* of the theory. For example, first-order logic programs are executed by the algorithms of unification and SLD-resolution, whereas their semantics is given by semantic (fixed point) operators. Neural-symbolic networks we have surveyed can do the semantics of (a restricted class of) first-order logic programs, but not the syntactic inference. And implementations of the former did not lead to the implementation of the latter. Using logic programming terminology, the neuro-symbolic networks could do declarative semantics, but not the operational one.

Another obstacle is that the traditional Neuro-Symbolic Networks, owing to their dependence on truth value assignments and first-order semantics, do not adapt well to higher-order logics and type theory. Moreover, in typed theories, such as typed lambda calculus or the calculus of constructions, the very distinction between syntax and semantics is erased, and manipulations with types of expressions are performed using the syntax of the theory, and this, in its turn, justifies correctness of computations.

Because the existing neuro-symbolic networks do not adapt well to the two major trends in computational logic – automated proving (given by operational semantics of first-order logic programs) and Type theory, – they have not yet found their proper niche in the area, and are normally conceived as a branch of AI rather than of computational logic.

2. Direct Processing of Logic Terms should be Preferred to Processing of their Truth Values. This can be achieved by using a one-to-one numerical encoding of the symbols of the formal alphabet, such as Gödel numbering. The main difficulty here is to represent strings of such numbers.

Logic formulae are effectively strings of symbols of a given alphabet. However, according to a general convention, neurons do not process strings, or ordered sequences, or any other structured data. Every neuron can accept only a scalar as a signal, and output a scalar in its turn. This general convention has been developed through decades of discussion, and different views on it are best summarised in (Aleksander and Morton, 1993; Smolensky and Legendre, 2006).

However, some order is innate to neural networks: and this order is imposed by position of neurons in a given layer, and by positions of layers in a network. So, although each neuron accepts only a scalar number as an input, a layer of neurons accepts a vector of such numbers, and the whole network can accept a matrix of numbers.

Therefore, a vector of neurons in a layer mirrors the structure of a string, and the matrix that describes a many-layered neural network mirrors the structure of a tree. This is all we need to encode logical terms (strings of symbols of an alphabet), and term trees - in case we use a tree-like representation of terms in parallel algorithms. See (Pollack, 1990; Smolensky and Legendre, 2006) for related discussion.

Given a first-order formula $F = P(t)$ built from the symbols of the alphabet A, and having a one-to-one numerical encoding of A given by a function η, the vector $[P^{\eta};\ (^{\eta};\ t^{\eta};\ )^{\eta}]$ can be directly used as an input or weight vector in a neural network. Here, we denote the result of applying η to a symbol x by x^{η}.

3. Levels of Abstraction should not be Mixed. Two levels of abstraction we mentioned in the introduction can provide two different solutions to the problem of simulating symbolic logic in neural networks. On the low level, we can build a network that simulates Turing machines, and then use it for symbolic computations. This approach can be bulky, and for this reason logic algorithms are commonly written on a higher level of abstraction than Turing machines. We suggest to look for direct representations of these high-level algorithms in Neural networks, ignoring the low-level theoretical correspondence between neural networks and Turing machines.

This direct approach may seem to lack generality; nevertheless, we claim that it can be very useful in practice. Consider, for example, the algorithm of first-order unification due to (Robinson, 1965), that is used as one of the basic and essential algorithms in proof theory and computational logic. It is P-complete and can be performed by a Turing machine, however, its definition in terms of Turing machines can be bulky. Its traditional description is as follows.

Suppose we have two first-order formulae $P(a)$ and $P(x)$. The algorithm of unification will inspect each symbol in the formulae, and pick two symbols that disagree - x and a. Then it will substitute a for x in $P(x)$, and thus, the two formulae will be unified. This can be done using a conventional error-correction learning in neural networks. We simply need to take numerical vector representation of the symbolic terms; let $[P^{\eta};\ (^{\eta};\ a^{\eta};\ )^{\eta}]$ and $[P^{\eta};\ (^{\eta};\ x^{\eta};\ )^{\eta}]$ denote the numerical encodings. Take a layer of 4 neurons. Let the weight vector be $[P^{\eta};\ (^{\eta};\ x^{\eta};\ )^{\eta}]$ and the bias vector be $[P^{\eta};\ (^{\eta};\ a^{\eta};\ )^{\eta}]$. Set the activation function to be linear, and the target to be $[0;\ 0;\ 0;\ 0]$. Send the signal 1 to the network, it will output $[0;\ 0;\ (x^{\eta} - a^{\eta});\ 0]$. Thus, the error – the difference between the desired response and the output – will be $[0;\ 0;\ -(x^{\eta} - a^{\eta});\ 0]$. The error-correction algorithm will amend the weight vector, and, on the next iteration, the weight vector will be $[P^{\eta};\ (^{\eta};\ a^{\eta};\ )^{\eta}]$, and the error will be zero. See (Komendantskaya, 2009b).

This example illustrates that some algorithms of computational logic have direct analogy with the learning algorithms of neurocomputing. This direct use of neural networks for implementations of computational logic should be further exploited.

This discussion of the unification by error-correction learning leads us to the last thesis:

4. Attitude to Learning Functions should be More Liberal, and Allow Symbolic Components. Coming back to the previous example, it is impossible to get a general-purpose network that performs unification *algorithm* without adding a certain symbolic description into the definition of the learning function. For example, we need to make sure that only a can be substituted for x, and not x for a. In case we deal with function symbols in the language, we may need to substitute $f(y)$ for x, and then the network will need to be enlarged. Also, the occur check should be performed, e.g., we should make sure that $f(x)$ is not substituted for x.

Among the advantages of having such hybrid networks would be that simply switching between conventional and pseudo-symbolic learning functions one can achieve multi-functionality of the resulting neural network. Another advantage is parallelism.

The algorithm of unification is not the only algorithm of computational logic that yields a simple and direct representation as a learning process. As another example, parallel (term)-rewriting can be shown to correspond to a form of Hebbian learning. That is, given a string [1 2 3 1 2 3], a 6-neuron layer can perform a rewriting operation for the rewriting rule $x \rightarrow 3x$ using a Hebbian learning rule. For this, we simply need to set the weight vector to be [1 2 3 1 2 3], set the rate of learning to be 2, and send the input signal 1. At the next iteration, the weight will be rewritten to [3 6 9 3 6 9]. This example, moreover, shows that neural networks are capable of *parallel* rewriting, which will bring a much desired speed up to the computations; (Komendantskaya, 2009a).

In order for this scheme to cover a more symbolic form of rewriting — a *term-rewriting* — one must allow a symbolic component into the conventional learning function to deal with the symbolic subtleties that arise when we rewrite and substitute first-order terms.

4 CONCLUSIONS

We separated two levels of abstraction at which the relations between "neurons" and "symbols" can be considered. We restricted our attention to the implementational level. We analysed common properties of existing neuro-symbolic networks, and showed that they obey one general **Principle**. This **Principle** and its **three main consequences** cause the disjunction "*Neurons OR Symbols*" to remain exclusive — that is, they prevent implementations of the *symbolic* (*proof-theoretic*) aspects of computational logic in neural networks. Finally, we proposed four principles for building neuro-symbolic nets of the future.

ACKNOWLEDGEMENTS

The work was sponsored by EPSRC PF research grant EP/F044046/1.

REFERENCES

Aleksander, I. and Morton, H. (1993). *Neurons and Symbols*. Chapman and Hall.

Bader, S. and Hitzler, P. (2005). Dimensions of neural symbolic integration - a structural survey. In Artemov, S., editor, *We will show them: Essays in honour of Dov Gabbay*, volume 1, pages 167–194. King's College, London.

Bader, S., Hitzler, P., and Hölldobler, S. (2008). Connectionist model generation: A first-order approach. *Neurocomputing*, 71:2420–2432.

d'Avila Garcez, A., Broda, K. B., and Gabbay, D. M. (2002). *Neural-Symbolic Learning Systems: Foundations and Applications*. Springer-Verlag.

d'Avila Garcez, A. and Gabbay, D. M. (2004). Fibring neural networks. In *Proceedings of the 19th National Conference on Artificial Intelligence*, pages 342–347. AAAI Press/MIT Press, San Jose, California, USA.

d'Avila Garcez, A., Lamb, L., and Gabbay, D. (2007). Connectionist modal logic: Representing modalities in neural networks. *Theoretical Computer Science*, 1-2(371):34–53.

d'Avila Garcez, A., Lamb, L. C., and Gabbay, D. M. (2008). *Neural-Symbolic Cognitive Reasoning*. Cognitive Technologies. Springer-Verlag.

Ding, L. (1995). Neural prolog - the concepts, construction and mechanism. In *Proceedings of the 3rd Int. Conference Fuzzy Logic, Neural Nets, and Soft Computing*, pages 181–192.

Domingos, P. (2008). Markov logic: a unifying language for knowledge and information management. In *CIKM*, page 519.

Güsgen, H. W. and Hölldobler, S. (1992). Connectionist inference systems. In Fronhöfer, B. and Wrightson, G., editors, *Parallelization in Inference Systems*, pages 82–100. Springer, LNAI *590*.

Hammer, B. and Hitzler, P. (2007). *Perspectives of Neural-Symbolic Integration*. Studies in Computational Intelligence. Springer Verlag.

Hitzler, P., Hölldobler, S., and Seda, A. K. (2004). Logic programs and connectionist networks. *Journal of Applied Logic*, 2(3):245–272.

Hölldobler, S. and Kalinke, Y. (1994). Towards a massively parallel computational model for logic programming. In *Proceedings of the ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pages 68–77. ECCAI.

Hölldobler, S., Kalinke, Y., and Storr, H. P. (1999). Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11:45–58.

Indyk, P. (1995). Optimal simulation of automata by neural nets. In Mayr, E. and Puech, C., editors, *Proc. of the Twelfth Annual Symposium on theoretical aspect of Computer Science*, LNCS, page 337.

Kilian, J. and Siegelmann, H. (1996). The dynamic universality of sigmoidal neural networks. *Information and Computation*, 128(1):48–56.

Kleene, S. (1956). Neural nets and automata. In *Automata Studies*, pages 3 – 43. Princeton University Press.

Koiran, P., Cosnard, M., and M.Garzon (1994). Computability with low-dimensional dynamic systems. *Theoretical Computer Science*, 132:113–128.

Komendantskaya, E. (2007). *Learning and Deduction in Neural Networks and Logic*. PhD thesis, Department of Mathematics, University College Cork, Ireland.

Komendantskaya, E. (2009a). Parallel rewriting in neural networks. In *Proceedings of ICNC'09*.

Komendantskaya, E. (2009b). Unification neural networks: Unification by error-correction learning. *Submitted*.

Komendantskaya, E., Lane, M., and Seda, A. (2007). Connectionist representation of multi-valued logic programs. In Hammer, B. and Hizler, P., editors, *Perspectives of Neural-Symbolic Integration*, Computational Intelligence, pages 259–289. Springer Verlag. To appear.

Lange, T. E. and Dyer, M. G. (1989). High-level inferencing in a connectionist network. *Connection Science*, 1:181 – 217.

Lloyd, J. (1987). *Foundations of Logic Programming*. Springer-Verlag, 2nd edition.

Markus, G. (2001). *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. Cambridge, MA: MIT Press.

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Math. Bio.*, 5:115–133.

Minsky, M. (1954). *Neural Nets and the Brain - Model Problem*. PhD thesis, Princeton University, Princeton NJ.

Minsky, M. (1969). *Finite and Infinite Machines*.

Nauck, D., Klawonn, F., Kruse, R., and F.Klawonn (1997). *Foundations of Neuro-Fuzzy Systems*. John Wiley and Sons Inc., NY.

Neumann, J. V. (1958). *The Computer and The Brain*. Yale University Press.

Pollack, J. (1987). *On Connectionist Models of Natural Language Processing*. PhD thesis, Computer science Department, University of Illinois, Urbana.

Pollack, J. (1990). Recursive distributed representations. *AIJ*, 46:77–105.

Robinson, J. (1965). A machine-oriented logic based on resolution principle. *Journal of ACM*, 12(1):23–41.

Shastri, L. and Ajjanagadde, V. (1993). From associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioural and Brain Sciences*, 16(3):417–494.

Siegelmann, H. (1999). *Neural Networks and Analog Computation. Beyond the Turing Limit*. Birkhauser.

Siegelmann, H. and Margenstern, M. (1999). Nine switch-affine neurons suffice for Turing universality. *Neural Networks*, 12:593–600.

Siegelmann, H. and Sontag, E. (1991). Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80.

Siegelmann, H. and Sontag, E. (1995). On the computational power of neural nets. *J. of Computers and System Science*, 50(1):132–150.

Smolensky, P. and Legendre, G. (2006). *The Harmonic Mind*. MIT Press.

Wang, J. and Domingos, P. (2008). Hybrid markov logic networks. In *AAAI*, pages 1106–1111.

Zadeh, L. (1992). Interpolative reasoning in fuzzy logic and neural network theory. *Fuzzy Systems*, pages 1–20.