Fibring Neural Networks

Artur S. d'Avila Garcez[δ] and Dov M. Gabbay[γ]
[δ]Dept. of Computing, City University London, EC1V 0HB, UK aag@soi.city.ac.uk
[γ]Dept. of Computer Science, King's College London, WC2R 2LS, UK dg@dcs.kcl.ac.uk

Abstract

Neural-symbolic systems are hybrid systems that integrate symbolic logic and neural networks. The goal of neural-symbolic integration is to benefit from the combination of features of the symbolic and connectionist paradigms of artificial intelligence. This paper introduces a new neural network architecture based on the idea of fibring logical systems. Fibring allows one to combine different logical systems in a principled way. Fibred neural 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 by allowing the activation of neurons in one network (A) to influence the change of weights in another network (B). Intuitively, this can be seen as training network B at the same time that one runs network A. We show that, in addition to being universal approximators like standard feedforward networks, fibred neural networks can approximate any polynomial function to any desired degree of accuracy, thus being more expressive than standard feedforward networks. **Keywords**: Neural-Symbolic Integration, Fibring Systems, Recursion.

Introduction

Neural-Symbolic integration concerns the use of symbolic knowledge in the neurocomputing paradigm of Artificial Intelligence (AI) (d'Avila Garcez, Broda, & Gabbay 2002; 2001). Its goal is to benefit from the integration of the symbolic and connectionist paradigms of AI, by providing either a logical characterisation of a connectionist system, a connectionist, massively parallel implementation of a logic, or a hybrid system bringing together features from neural networks and symbolic AI (Cloete & Zurada 2000; d'Avila Garcez, Broda, & Gabbay 2002; Holldobler, Kalinke, & Storr 1999). Towards this end, efficient, parallel and distributed reasoning and learning capabilities should be at the core of any Neural-Symbolic system and, one may argue, of any AI system. Ultimately, our goal should be to produce an effective AI system with added reasoning and learning capabilities, as recently pointed out by Valiant (Valiant 2003) as a key challenge for computer science.

Neural-Symbolic systems that use simple neural networks, such as single hidden layer feedforward or recurrent networks (Haykin 1999), typically only manage to represent and reason about propositional symbolic knowledge or *if then else* rules (Boutsinas & Vrahatis 2001; d'Avila Garcez, Broda, & Gabbay 2002; Fu 1994; Pinkas 1995; Towell & Shavlik 1994). On the other hand, Neural-Symbolic systems that are capable of representing and reasoning about more expressive symbolic knowledge, such as modal logic and first order logic, normally are less capable of learning new concepts efficiently (Holldobler, Kalinke, & Storr 1999; Sun & Alexandre 1997; Shastri 1999; Kijsirikul, Sinthupinyo, & Chongkasemwongse 2001). There is clearly a need to strike a balance between the reasoning and learning capabilities of Neural-Symbolic systems. Either the simple networks to which, for example, the efficient *backpropagation* learning algorithm is applicable to (Rumelhart, Hinton, & Williams 1986; Werbos 1990) must be shown to represent languages more expressive than propositional logic, or the complex connectionist systems that are capable of representing first order and higher order logics, such as for example CHCL (Holldobler & Kurfess 1992), must have efficient learning algorithms developed for them. This is necessary because real-world applications such as failure diagnosis, engineering and bioinformatics applications, will require the use of languages more expressive than propositional logic. Bioinformatics, in particular, very much requires the ability to represent and reason about relations as used in first order logic (Angelopoulos & Muggleton 2002).

In this paper, we adopt the approach of extending simple networks that use *backpropagation* in order to allow for higher expressive power. We do so by following Gabbay's Fibring methodology (Gabbay 1999), in which several different systems such as logical systems of space and time, neural networks and Bayesian networks (Williamson & Gabbay 2004; d'Avila Garcez 2004; d'Avila Garcez & Lamb 2004), may be put to work together in a co-ordinated man-

ner to solve a particular problem.[1] To this end, we know that a fundamental aspect of symbolic computation lies in the ability to implement recursion. As a result, to make neural networks behave like logic, we need to add recursion to it by allowing networks to be composed not only of interconnected neurons but also of other networks. Figure 1 exemplifies how a network (B) can be embedded recursively into another network (A). Of course, the idea of fibring is not only to organise networks as a number of sub-networks (A, B, etc). In Figure 1, for example, hidden neuron X of Network A is expected to be a neural network (Network B) in its own right. The input, weights and output of Network B may depend on the activation state of neuron X, according to what is known as a fibring function φ. One such function may be to multiply the weights of Network B by the input potential of neuron X.

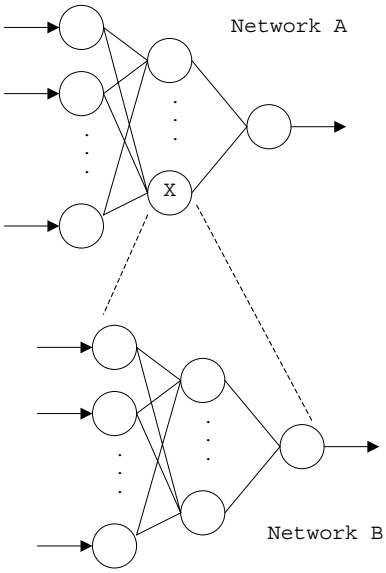

Figure 1: Fibring Neural Networks

Most of the work on how to implement recursion in neural networks has concentrated on the use of recurrent auto-associative networks and symmetric networks to represent formal grammars (Elman 1990; Touretzky & Hinton 1988; Smolensky 1990; 2000; Pollack 1990). In general, the networks learn how to simulate a number of recursive rules by similarity with a set of examples, and the question of how such rules are represented in the network is treated as secondary. In this paper, we give a different treatment to the subject, looking at it from a Neural-Symbolic integration perspective (d'Avila Garcez, Broda, & Gabbay 2002). The idea is to be able to represent and learn expressive symbolic rules, such as rules containing embedded implication of the form $(a \rightarrow b) \rightarrow c$, where $(a \rightarrow b)$ would be encoded into network B, and then $X \rightarrow c$, with $X = (a \rightarrow b)$, would be encoded into network A so that the fibred network represents $(a \rightarrow b) \rightarrow c$.

In what follows, we introduce and define fibred neural networks (fNNs), and show that, in addition to being universal approximators[2], fNNs can approximate any polynomial function in an unbounded domain, thus being more expressive than standard feedforward networks. Briefly, this can be shown by noting that fibred neural networks compute, e.g., the function $f(x) = x^2$ exactly for any given input x in $\mathbb{R}$, as opposed to feedforward networks which are restricted to compact (i.e. closed and bounded) domains (Cybenco 1989; Hornik, Stinchcombe, & White 1989). Intuitively, fibring neural networks can be seen as the running and training of neural networks at the same time. In Figure 1, for example, at the same time that we run network A, we perform a kind of learning in network B because we allow the weights of B to change according to the fibring function. In other words, object-level network running and meta-level network training are occurring simultaneously in the same system, and this is responsible for the added expressiveness of the system.

This paper is organised as follows. Firstly, we introduce and exemplify fibred neural networks. Then, we define the architecture and dynamics of fibred networks precisely, and show that fibred networks approximate polynomials. Finally, we conclude and discuss directions for future work.

Examples of Fibring

The main idea behind fibring neural networks is to allow single neurons to behave like entire embedded networks according to a fibring function φ. This function qualifies the function computed by the embedded network so that the embedded network's output depends on φ. For example, consider Network A and its embedded network (Network B) in Figure 1. Let $\mathbf{W}_A$ and $\mathbf{W}_B$ be the set of weights of Network A and Network B, respectively. Let $f_{\mathbf{W}_A}(\mathbf{i}_A)$ be the function computed by Network A, and $g_{\mathbf{W}_B}(\mathbf{i}_B)$ be the function computed by Network B, where $\mathbf{i}_A$ and $\mathbf{i}_B$ are the input vectors of Networks A and B, respectively. If Network B is embedded into neuron X of Network A with fibring function φ, the function computed by Network B becomes $g_{\mathbf{W}'_B}(\mathbf{i}_B)$, where $\mathbf{W}'_B = \varphi(\mathbf{W}_B)$, and then the output of neuron X becomes the output of Network B, as the following example illustrates.

Consider the two simple networks (C and D) of Figure 2. Let us assume, without loss of generality, that input and output neurons have the identity as activation function, while hidden neurons have $h(x) = \tanh(x)$ as activation function (Hornik, Stinchcombe, & White 1989). We use bipolar inputs $i_j \in \{-1, 1\}$, $W_{jk} \in \mathbb{R}$, and outputs $o_k \in (-1, 1)$. The output of Network C

[1] For example, a robot's motion control system may require a logic of space, a logic of time, and a visual pattern recognition, neural networks-based system.

[2] Universal approximators, such as feedforward neural networks, can approximate any (Borel) measurable function *in a compact domain* to any desired degree of accuracy.

is $o_C = W_{3C}.h(W_{1C}.i_{1C} + W_{2C}.i_{2C})$, and the output of Network D is $o_D = W_{3D}.h(W_{1D}.i_{1D} + W_{2D}.i_{2D})$. Now, let Network D be embedded into Network C as shown in Figure 2. This indicates that the input potential of neuron Y will influence D according to fibring function φ. Let us refer to the input potential of Y as $\mathbf{I}(Y)$.[3] In addition, this indicates that the output of D (o_D) will influence C (in this example, only the output of C). Suppose $\varphi(\mathbf{W}_D) = \mathbf{I}(Y) \cdot \mathbf{W}_D$, where $\mathbf{W}_D = [W_{1D}, W_{2D}, W_{3D}]$, i.e. φ multiplies the weights of D by the input potential of Y. Let us use $\overline{o}_C$ and $\overline{o}_D$ to denote the outputs of networks C and D, respectively, after they are fibred. $\overline{o}_D$ is obtained by applying φ to $\mathbf{W}_D$ and calculating the output of such a network, as follows: $\overline{o}_D = (\mathbf{I}(Y).W_{3D}) \cdot h((\mathbf{I}(Y).W_{1D}).i_{1D} + (\mathbf{I}(Y).W_{2D}).i_{2D})$. $\overline{o}_C$ is obtaining by taking $\overline{o}_D$ as the output of neuron Y. In this example, $\overline{o}_C = \overline{o}_D$. Notice how network D is being trained (as φ changes its weights) at the same time that network C is running.

Clearly, fibred networks can be trained from examples in the same way that standard feedforward networks are (for example, with the use of *backpropagation* (Rumelhart, Hinton, & Williams 1986)). Networks C and D of Figure 2, for example, could have been trained separately before being fibred. Network C could have been trained, e.g., with a robot's visual system, while network D would have been trained with its planning system. For simplicity, we assume for now that, once defined, the fibring function itself should remain unchanged. Future extensions of fibring neural networks could, however, consider the task of learning fibring functions as well.

Notice 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. The networks of Figure 2, for example, could have been fibred by embedding Network D into an input neuron of Network C (say, the one with input i_{1C}). In this case, outputs $\overline{o}_D$ and $\overline{o}_C$ would have been $\overline{o}_D = \varphi(W_{3D}) \cdot h(\varphi(W_{1D}).i_{1D} + \varphi(W_{2D}).i_{2D})$, where φ is a function of $\mathbf{W}_D$ (say, $\varphi(\mathbf{W}_D) = i_{1C} \cdot \mathbf{W}_D$), and then $\overline{o}_C = W_{3C}.h(W_{1C}.\overline{o}_D + W_{2C}.i_{2C})$.

Let us now consider an even simpler example that, nevertheless, illustrates the power of fibring neural networks. Consider two networks A and B, both with a single input neuron (i_A and i_B, respectively), a single hidden neuron, and a single output neuron (o_A and o_B, respectively). Let all the weights in both networks have value 1, and let the identity ($f(x) = x$) be the activation function of all the neurons (including the hidden neurons). As a result, we simply have $o_A = f(W_{2A} \cdot f(W_{1A} \cdot f(i_A))) = i_A$ and $o_B = f(W_{2B} \cdot f(W_{1B} \cdot f(i_B))) = i_B$, where W_{1A} and W_{2A} are the weights of network A, and W_{1B} and W_{2B} are the weights of network B. Now, assume we embed network B into the input neuron of network A. We obtain $\overline{o}_B = f(\varphi(W_{2B}) \cdot f(\varphi(W_{1B}) \cdot f(i_B)))$, and then $\overline{o}_A = f(W_{2A} \cdot f(W_{1A} \cdot \overline{o}_B))$. Since $f(x) = x$, we have $\overline{o}_B = \varphi(W_{2B}) \cdot \varphi(W_{1B}) \cdot i_B$ and $\overline{o}_A = W_{2A} \cdot W_{1A} \cdot \overline{o}_B$. Now, let our fibring function be $\varphi(\mathbf{W}_A, \mathbf{i}_A, \mathbf{W}_B) = i_A \cdot \mathbf{W}_B$, where $\mathbf{W}_B = [W_{1B}, W_{2B}]$. Since W_{1A}, W_{2A}, W_{1B} and W_{2B} are all equal to 1, we obtain $\overline{o}_B = i_A \cdot i_A \cdot i_B$ and $\overline{o}_A = \overline{o}_B$. This means that if we fix $i_B = 1$, the output of network A (fibred with network B) will be the square of its input. As a result, if the following sequence is given as input to A (fibred with B): $n, 1/n, n+1, 1/(n+1), n+2, 1/(n+2), ...$ for $n \in \mathbb{R}$, the corresponding output sequence of A will be: $n^2, 1, (n+1)^2, 1, (n+2)^2, 1, ...$ Note that, input n changes the weights of B from 1 to n, input $1/n$ changes the weights of B back to 1, input $n+1$ changes the weights of B from 1 to $n+1$, input $1/(n+1)$ changes the weights of B back to 1, and so on.[4] The interest in this sequence lies in the fact that, for alternating inputs, the square of the input is computed exactly by the network for any input in $\mathbb{R}$. This illustrates an important feature of fibred neural networks, namely, their ability to approximate functions in an unbounded domain (Henderson 2002; Hines 1996). This results from the recursive characteristic of fibred networks as indicated by the fibring function, and will be discussed in more detail in the following section. Note that, in practice, the fibring function φ is to be defined according to the problem domain.

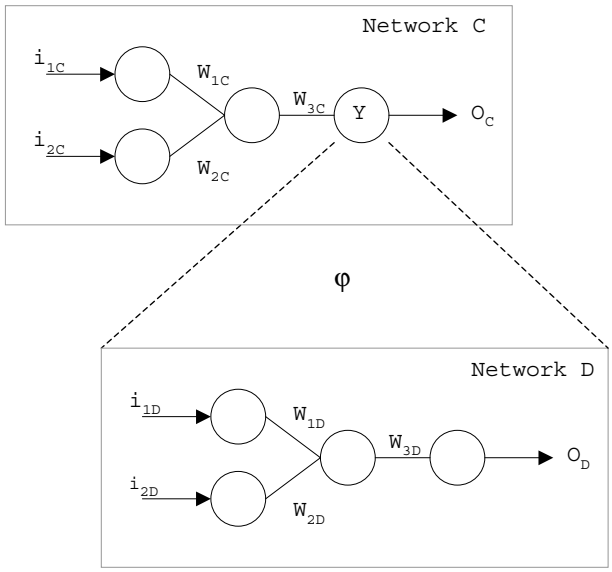

Figure 2: Fibring two simple networks

Fibred Neural Networks

In this section, we define fibred neural networks (fNNs) precisely, we define the dynamics of fNNs, and we show

[3]Note that, in this particular example, $\mathbf{I}(Y) = o_C$ due to the use of the identity as activation function in the output layer.

[4]Since the fibring function changes the weights of the embedded network, we use $1/n$, $1/n+1$, $1/n+2$... to *reset* the weights back to 1 during the computation of the sequence.

that fNNs can approximate unbounded functions.

Fibring Definition

For the sake of simplicity, we restrict the definition of fibred networks to feedforward networks with a single output neuron. We also concentrate on networks with linear input and linear output activation functions, and either linear or sigmoid hidden layer activation function. We believe, however, that the principles of fibring could be applied to any artificial neural network model.[5] In what follows, we allow not only two networks, but any number of embedded networks to be nested into a fibred network. We also allow for an unlimited number of hidden layers per network.

Definition 1 *(*Fibring Function*) Let A and B be two neural networks. A function $\varphi_n : \mathbf{I} \rightarrow \mathbf{W}$ is called a fibring function from A to B if $\mathbf{I}$ is the input potential of a neuron n in A and $\mathbf{W}$ is the set of weights of B.*

Definition 2 *(*Fibred Neural Networks*) Let A and B be two neural networks. We say that B is embedded into A if φ_n is a fibring function from A to B, and the output of neuron n in A is given by the output of network B. The resulting network, composed of networks A and B, is said to be a fibred neural network.*

Note that many networks can be embedded into a single network, and that networks can be nested so that network B is embedded into network A, network C is embedded into network B, and so on. The resulting fibred network can be constructed by applying Definition 2 recursively, e.g., first to embed C into B and then to embed the resulting network into A.

Example 3 *Consider three identical network architectures (A, B and C), each containing a single linear input neuron, a single linear hidden neuron, and a single linear output neuron, as depicted in Figure 3. Let us denote the weight from the input neuron to the hidden neuron of network N, $N \in \{A, B, C\}$, by W_N^h, and the weight from the hidden neuron to the output neuron of N by W_N^o. Assume we embed network C into the output neuron (Y) of network B, and embed the resulting network into the output neuron (X) of network A (according to Definition 2), as shown in the figure. Let φ_B denote the fibring function from A to B, and φ_C denote the fibring function from B to C. As usual, let us define $\varphi_B = I(X) \cdot \mathbf{W}_B$ and $\varphi_C = I(Y) \cdot \mathbf{W}_C$, where $I(X)$ is the input potential of neuron X, $I(Y)$ is the input potential of neuron Y, $\mathbf{W}_B$ denotes the weight vector $[W_B^h, W_B^o]$ of B, and $\mathbf{W}_C$ denotes the weight vector $[W_C^h, W_C^o]$ of C. Initially, let $W_A^h = \sqrt{a}$, where $a \in \mathbb{R}^+$, and $W_A^o = W_B^h = W_B^o = W_C^h = W_C^o = 1$. As a result, given input x to A, we have $I(X) = x\sqrt{a}$. Then, φ_B will be used to update the weights of network B to $W_B^h = x\sqrt{a}$ and $W_B^o = x\sqrt{a}$. If we had only networks A and B fibred, input $y = 1$, for example, would then produce an output $o_B = ax^2$ for network B, and the same ($o_A = ax^2$) for network A. Since network C is also embedded into the system, given input y to network B, fibring function φ_C will be used to update the weights of network C, using $I(Y)$ as a parameter. Thus, if $y = 1$, we have $I(Y) = ax^2$, and the weights of network C will be changed to $W_C^h = ax^2$ and $W_C^o = ax^2$. Finally, if $z = 1$, the output of network C (and then of networks B and A as well) will be a^2x^4. This illustrates the computation of polynomials in fNNs. The computation of odd degree polynomials and of negative coefficients can be achieved by adding more hidden layers to the networks, as we will see in the sequel.*

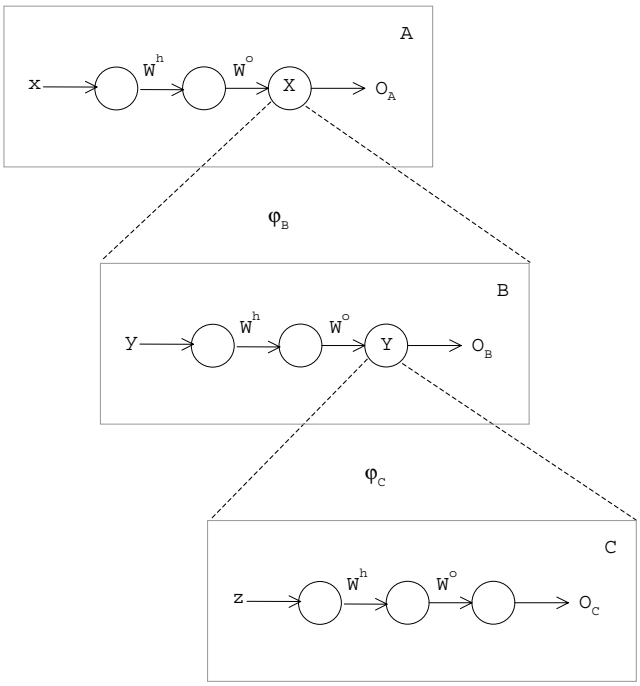

Figure 3: Nesting Fibred Networks

Fibring Dynamics

Example 3 also illustrates the dynamics of fibred networks. Let us now define such a dynamics precisely.

Definition 4 *(*Nested fNNs*) Let N_1, N_2,..., N_n be neural networks. N_1, N_2,..., N_n form a nested fibred network if N_i is embedded into a neuron of N_{i-1} with a fibring function φ_i for any $2 \leq i \leq n$. We say that $j - 1$ $(1 \leq j \leq n)$ is the level of network N_j.*

Definition 5 *(*fNNs Dynamics*) Let $N_1, N_2, ..., N_n$ be a nested fibred network. Let φ_i be the fibring function from N_{i-1} to N_i for $2 \leq i \leq n$. Let $\mathbf{i}_j$ denote an input vector to network N_j, $\mathbf{W}_j$ the current weight vector of N_j, $\mathbf{I}_n(\mathbf{i}_j)$ the input potential of N_j's neuron n_j into which N_{j+1} is embedded given input vector $\mathbf{i}_j$, $\mathbf{O}_{n_j}$ the output of neuron n_j, and $f_{\mathbf{W}_j}(\mathbf{i}_j)$ the function computed by network N_j given $\mathbf{W}_j$ and $\mathbf{i}_j$ as in the standard way for feedforward networks. The output o_j of network N_j*

[5] Particularly interesting would be to consider fibring recurrent networks (i.e. networks with feedback connections).

$(1 \leq j \leq n-1)$ is defined recursively in terms of the output o_{j+1} of network N_{j+1}, as follows:

$$\mathbf{W}_{j+1} := \varphi_{j+1}(\mathbf{I}(\mathbf{i}_j), \mathbf{W}_{j+1}), 1 \leq j \leq n-1$$

$$o_n = f_{\mathbf{W}_n}(\mathbf{i}_n)$$

$$o_j = f_{\mathbf{W}_j}(\mathbf{i}_j, \mathbf{O}_{n_j} := o_{j+1})$$

where $f_{\mathbf{W}_j}(\mathbf{i}_j, \mathbf{O}_{n_j} := o_{j+1})$ denotes the function computed by N_j by substituting the output of its neuron n_j by the output of network N_{j+1}.

Fibring Expressiveness

Now that fNNs have been defined, we proceed to show that, in addition to being universal approximators, fNNs can approximate any polynomial function, and thus are more expressive than standard feedforward neural networks.

Proposition 6 *Fibred neural networks can approximate any (Borel) measurable function in a compact domain to any desired degree of accuracy (i.e. fNNs are universal approximators).*

proof: *This follows directly from the proof that single hidden layer feedforward neural networks are universal approximators (Hornik, Stinchcombe, & White 1989), together with the observation that level zero networks are a generalisation of single hidden layer feedforward networks.* □

Proposition 7 *Fibred neural networks can approximate any polynomial function to any desired degree of accuracy.*[6]

proof: *Consider the level zero network N of Figure 4. Let $n+1$ $(n \in \mathbb{N})$ be the number of input neurons of N, $0 \leq i \leq n$, $a_i \in \mathbb{R}$. Now, embed $n-1$ networks into the input neurons of N, all at level 1, as indicated in Figure 4 for networks A, B and C, such that network A is embedded into neuron A of network N, network B is embedded into neuron B of N, and network C is embedded into neuron C of N. Each of the $n-1$ embedded networks will be used to represent $x^2, x^3, ..., x^n$. In Figure 4, A represents x^2, B represents x^3, and C represents x^n. In the ensemble, all networks, including N, contain linear neurons. A network N_j that represents x^j $(2 \leq j \leq n)$ contains two input neurons (to allow the representation of $a_j \in \mathbb{R}$), $j-1$ hidden layers, each layer containing a single hidden neuron (let us number these $h_1, h_2, ..., h_{j-1}$), and a single output neuron. In addition, let $a_j/2$ be the weight from each input neuron to h_1, and let 1 be the weight of any other connection in N_j. We need to show that N_j computes $a_j x^j$. From Definition 5, given input x to N and $\varphi_j = x \cdot \mathbf{W}_j$, the weights of N_j are multiplied by x. Then, given input $(1,1)$ to N_j, neuron h_1 will produce output $a_j x$, neuron h_2 will produce output $a_j x^2$, and so on. Neuron h_{j-1} will produce output $a_j x^{j-1}$, and the output neuron will produce $a_j x^j$. Finally, by Definition 2, the neuron in N into which N_j is embedded will present activation $a_j x^j$, and the output of N will be $\sum_j a_j x^j$. The addition of $a_1 x$ and a_0 is straightforward (see network N in Figure 4), completing the proof that fNNs compute $\sum_i a_i x^i$.* □

[6] Recall that, differently from functions in a compact domain, polynomial functions are not bounded.

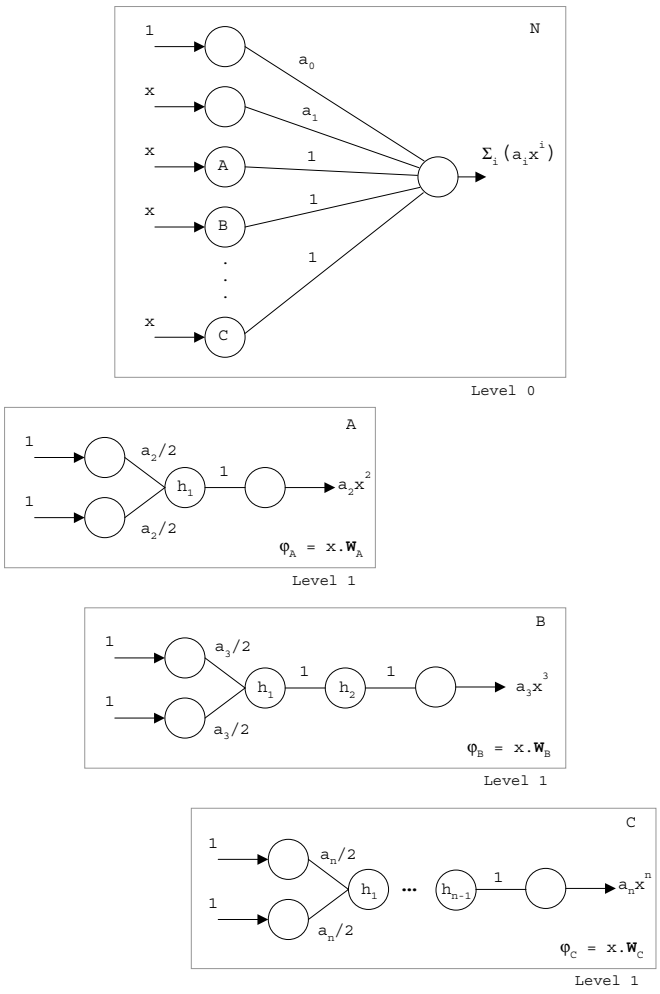

Figure 4: Computing polynomials in fibred networks

Conclusion and Future Work

This paper has introduced a new neural network architecture named fibred neural networks (fNNs), which combines a number of standard feedforward neural networks (that can be trained using *backpropagation*) with the use of a fibring function. We have shown that, in addition to being universal approximators, fNNs can approximate any polynomial function, therefore being more expressive than standard feedforward neural networks.

The question of which logics could be represented in fNNs is an interesting open question. Our next step is to use the recursive, more expressive architecture of fNNs to perform symbolic computation, giving fNNs a Neural-Symbolic characterisation. We expect to be

able to use fNNs to represent variables and to learn and reason about relational knowledge.

Another interesting work to pursue would be to define how recurrent neural networks could be fibred. Recurrent networks already possess a limited ability to compute unbounded functions (Henderson 2002). A comparison of the computational capabilities of these two architectures would be highly desirable.

Finally, the questions of how different networks should be fibred and which fibring functions should be used is a very important one when it comes to practical applications of fNNs. This is clearly domain dependent, and an empirical evaluation of fNNs in comparison with standard neural networks would also be required.

Acknowledgments

We are grateful to Stefan Rueger for very useful discussions. Artur Garcez is partly supported by the Nuffield Foundation.

References

Angelopoulos, N., and Muggleton, S. H. 2002. Machine learning metabolic pathway descriptions using a probabilistic relational representation. *Electronic Transactions in Artificial Intelligence* 6. MI-19.

Boutsinas, B., and Vrahatis, M. N. 2001. Artificial nonmonotonic neural networks. *Artificial Intelligence* 132:1–38.

Cloete, I., and Zurada, J. M., eds. 2000. *Knowledge-Based Neurocomputing*. The MIT Press.

Cybenco, G. 1989. Approximation by superposition of sigmoidal functions. In *Mathematics of Control, Signals and Systems 2*. 303–314.

d'Avila Garcez, A. S., and Lamb, L. C. 2004. Reasoning about time and knowledge in neural-symbolic learning systems. In Thrun, S.; Saul, L.; and Schoelkopf, B., eds., *Advances in Neural Information Processing Systems 16*, Proceedings of the NIPS 2003 Conference. Vancouver, Canada: MIT Press.

d'Avila Garcez, A. S.; Broda, K.; and Gabbay, D. M. 2001. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence* 125:155–207.

d'Avila Garcez, A. S.; Broda, K.; and Gabbay, D. M. 2002. *Neural-Symbolic Learning Systems: Foundations and Applications*. Perspectives in Neural Computing. Springer-Verlag.

d'Avila Garcez, A. S. 2004. On Gabbay's fibring methodology for bayesian and neural networks. In *Laws and Models in Science*, European Science Foundation (ESF). King's College London.

Elman, J. L. 1990. Finding structure in time. *Cognitive Science* 14(2):179–211.

Fu, L. M. 1994. *Neural Networks in Computer Intelligence*. McGraw Hill.

Gabbay, D. M. 1999. *Fibring Logics*. Oxford Univesity Press.

Haykin, S. 1999. *Neural Networks: A Comprehensive Foundation*. Prentice Hall.

Henderson, J. 2002. Estimating probabilities of unbounded categorization problems. In *Proceedings of European Symposium on Artificial Neural Networks*, 383–388.

Hines, J. W. 1996. A logarithmic neural network architecture for unbounded non-linear function approximation. In *Proceedings of IEEE International Conference on Neural Networks*.

Holldobler, S., and Kurfess, F. 1992. CHCL: A connectionist inference system. In Fronhofer, B., and Wrightson, G., eds., *Parallelization in Inference Systems*, 318–342. Springer.

Holldobler, S.; Kalinke, Y.; and Storr, H. P. 1999. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence Journal, Special Issue on Neural Networks and Structured Knowledge* 11(1):45–58.

Hornik, K.; Stinchcombe, M.; and White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2:359–366.

Kijsirikul, B.; Sinthupinyo, S.; and Chongkasemwongse, K. 2001. Approximate match of rules using backpropagation neural networks. *Machine Learning* 43(3):273–299.

Pinkas, G. 1995. Reasoning, nonmonotonicity and learning in connectionist networks that capture propositional knowledge. *Artificial Intelligence* 77:203–247.

Pollack, J. B. 1990. Recursive distributed representations. *Artificial Intelligence* 46(1):77–105.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning internal representations by error propagation. In Rumelhart, D. E., and McClelland, J. L., eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press. 318–362.

Shastri, L. 1999. 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* 11:79–108.

Smolensky, P. 1990. Tensor product variable binding and the representation of symbolic structures in connectionist networks. *Artificial Intelligence* 46:159–216.

Smolensky, P. 2000. Grammar-based connectionist approaches to language. *Cognitive Science* 23:589–613.

Sun, R., and Alexandre, F. 1997. *Connectionist Symbolic Integration*. Lawrence Erlbaum Associates.

Touretzky, D., and Hinton, G. 1988. A distributed connectionist production system. *Cognitive Science* 12(3):423–466.

Towell, G. G., and Shavlik, J. W. 1994. Knowledge-based artificial neural networks. *Artificial Intelligence* 70(1):119–165.

Valiant, L. G. 2003. Three problems in computer science. *Journal of the ACM* 50(1):96–99.

Werbos, P. J. 1990. Backpropagation through time: what does it mean and how to do it. In *Proceedings of the IEEE*, volume 78, 1550–1560.

Williamson, J., and Gabbay, D. 2004. Recursive causality in bayesian networks and self-fibring networks. In *Laws and Models in Science*, European Science Foundation (ESF). King's College London.