

Fibring Neural Networks

A. S. d'Avila Garcez and D. M. Gabbay

Department of Computing

City University

Technical Report Series

TR/2003/SEG/03

ISSN 1364-4009

Fibring Neural Networks

A. S. d'Avila Garcez^δ and D. M. Gabbay^γ

^δDept. of Computing
City University, London EC1V 0HB, UK
aag@soi.city.ac.uk

^γDept. of Computer Science
King's College, London WC2R 2LS, UK
dg@dcs.kcl.ac.uk

Abstract:

This paper introduces a new neural network architecture based on the idea of fibring logical systems. Fibring allows one to combine different systems (neural networks) in a principled way. Fibred neural networks may be composed not only of interconnected neurons but also of other networks in a recursive architecture. A fibring function then defines how this recursive architecture must behave by defining how the networks relate to each other (typically by allowing the activation of one network to influence on the change of the weights of another). We show that, in addition to being universal approximators, fibred networks can approximate any polynomial function to any desired degree of accuracy, thus being more expressive than standard feedforward neural networks.

1 Introduction

The goal of Neural-Symbolic integration is to benefit from the symbolic and the connectionist paradigms of Artificial Intelligence (AI) [CZ00, dGBG02]. Towards this end, efficient, parallel and distributed learning capability should be at the core of any Neural-Symbolic system and, one may argue, of any AI system. Neural-Symbolic systems that use simple neural networks, such as single hidden layer feedforward or recurrent networks [Hay99], typically only manage to represent and reason about propositional symbolic knowledge or *if then else* rules [BV01, dGBG02, Fu94, Pin95, TS94]. On the other hand, Neural-Symbolic systems that are capable of representing and of reasoning about more expressive symbolic knowledge, such as modal logic and first order logic, normally are less capable of learning new concepts efficiently [HKS99, SA97, Sha99, KSC01]. 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 or its variations can be applied to [RHW86, Wer74, Wer90] must be shown to represent languages more expressive than propositional logic, or the complex connectionist systems that are capable of representing first order logic, such as for example CHCL [HK92], must have efficient learning algorithms developed for them. This is necessary because real-world applications such as failure diagnosis, engineering and bioinformatics applications, do require the use of languages more expressive than propositional logic. Bioinformatics, in particular, very much depends on the ability to represent and reason about relations as used in first order logic [AM02].

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 [Gab99], in which several different systems such as logical systems of space and time, neural networks and bayesian networks [WG03], can be put to work together in an co-ordinated manner to solve a particular problem.¹ To this end, we know that a fundamental aspect of symbolic computation lies on the ability to do 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 can be embedded into another. Of course, the idea of fibring is not only to organise networks as a number of sub-networks. In Figure 1, for example, the hidden neuron of Network A is expected to be a neural network (Network B) in its own right, and the input, weights and output of Network B may depend on the activation values of neurons in Network A, according to the fibring function used. For example, a fibring function may be to multiply the weights of Network B by the input potential of Network's A output neuron.

Most of the work on how to do recursion in neural networks has concentrated on the use of recurrent auto-associative networks and symmetric networks to represent formal grammars [Elm90, TH88, Smo90, Smo00, Pol90]. In general, the network learns how to simulate a number of recursive rules by similarity, 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 [dGBG02]. The idea is to be able to represent and learn symbolic rules of

¹For example, a robot's motion control system requires a logic of space, a logic of time and a visual, pattern recognition system.

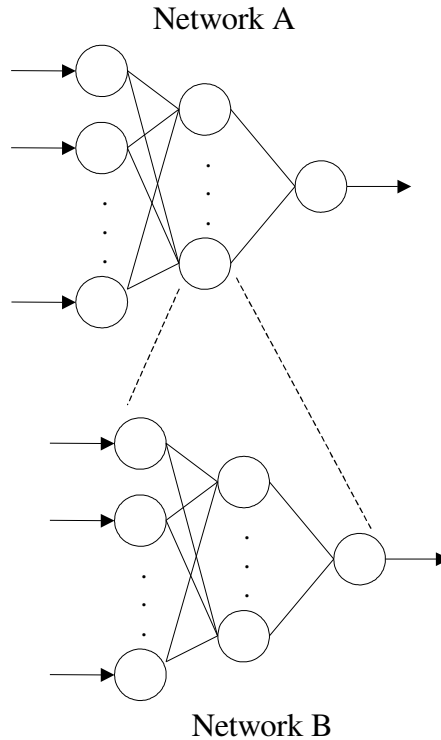


Figure 1: Fibring Neural Networks

the form $a \rightarrow (b \rightarrow c)$, where $(b \rightarrow c)$ would be encoded into network B and then $a \rightarrow (b \rightarrow c)$ would be encoded into the fibred network containing networks A and B.

In what follows, we introduce and define the fibred neural network (fNN) architecture, and show that, in addition to being universal approximators, fNNs can approximate any polynomial function, 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 \mathfrak{R} , as opposed to feedforward networks which are restricted to compact (i.e. closed and bounded) domains [Cyb89, HSW89]. Intuitively, fibring neural networks can be seen as running and training 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. Section 2 introduces and exemplifies fibred neural networks. Section 3 defines the architecture and dynamics of fibred networks precisely, and shows that fibred networks approximate polynomials. Section 4 concludes and discusses directions for future work.

2 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 Network A with fibring function φ , the function computed by Network B becomes $g_{\varphi(\mathbf{W}_B)}(\mathbf{i}_B)$, and then the function computed by Network A becomes $f_{\mathbf{W}_A, g_{\varphi(\mathbf{W}_B)}(\mathbf{i}_B)}(\mathbf{i}_A)$, as the following example illustrates.

Consider the two simple networks (**A** and **B**) 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 [HSW89]. We use bipolar inputs $i_j \in \{-1, 1\}$, $W_{jk} \in \mathfrak{R}$, and outputs $o_k \in (-1, 1)$. The output of Network **A** is $o_1^{\mathbf{A}} = W_3^{\mathbf{A}} \cdot h(W_1^{\mathbf{A}} i_1^{\mathbf{A}} + W_2^{\mathbf{A}} i_2^{\mathbf{A}})$, and the output of Network **B** is $o_1^{\mathbf{B}} = W_3^{\mathbf{B}} \cdot h(W_1^{\mathbf{B}} i_1^{\mathbf{B}} + W_2^{\mathbf{B}} i_2^{\mathbf{B}})$. Now, let Network **B** be embedded into Network **A** as shown in Figure 2. This indicates that the input potential of **A**’s output neuron will influence **B** according to fibring function φ . Let us refer to the input potential of **A**’s output neuron as $\mathbf{I}(o_1^{\mathbf{A}})$.² In addition, this indicates that the output of **B** ($o_1^{\mathbf{B}}$) will influence **A** (in this example, only the output of **A**). Suppose $\varphi(\mathbf{W}_B) = \mathbf{I}(o_1^{\mathbf{A}}) \cdot \mathbf{W}_B$, where $\mathbf{W}_B = [W_1^{\mathbf{B}}, W_2^{\mathbf{B}}, W_3^{\mathbf{B}}]$. Let us use $\bar{o}_1^{\mathbf{A}}$ and $\bar{o}_1^{\mathbf{B}}$ to denote the outputs of networks **A** and **B** respectively, after they are fibred. $\bar{o}_1^{\mathbf{B}}$ is obtained by applying φ to \mathbf{W}_B and calculating the output of such a network, as follows: $\bar{o}_1^{\mathbf{B}} = (\mathbf{I}(o_1^{\mathbf{A}}) \cdot W_3^{\mathbf{B}}) \cdot h((\mathbf{I}(o_1^{\mathbf{A}}) \cdot W_1^{\mathbf{B}}) i_1^{\mathbf{B}} + (\mathbf{I}(o_1^{\mathbf{A}}) \cdot W_2^{\mathbf{B}}) i_2^{\mathbf{B}})$.

²Note that, in this particular example, $\mathbf{I}(o_1^{\mathbf{A}}) = o_1^{\mathbf{A}}$ due to the use of the identity as activation function in the output layer.

$\bar{o}_1^{\mathbf{A}}$ is obtained by taking $\bar{o}_1^{\mathbf{B}}$ as the output of the neuron in which Network \mathbf{B} is embedded. In this example, $\bar{o}_1^{\mathbf{A}} = \bar{o}_1^{\mathbf{B}}$. Notice how network \mathbf{B} is being trained (when φ changes its weights) at the same time that network \mathbf{A} 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 Back-propagation [RHW86]). Networks \mathbf{A} and \mathbf{B} of Figure 2, for example, could have been trained separately before being fibred. Network \mathbf{A} could have been trained, e.g., with a robot's visual system, while network \mathbf{B} 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.

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 \mathbf{B} into an input neuron of Network \mathbf{A} (say, the one with input i_1). In this case, outputs $\bar{o}_1^{\mathbf{B}}$ and $\bar{o}_1^{\mathbf{A}}$ would have been $\bar{o}_1^{\mathbf{B}} = \varphi(W_3^{\mathbf{B}}) \cdot h(\varphi(W_1^{\mathbf{B}})i_1^{\mathbf{B}} + \varphi(W_2^{\mathbf{B}})i_2^{\mathbf{B}})$, where φ is a function of \mathbf{W}_B (say, e.g., $\varphi(\mathbf{W}_B) = i_1 \cdot \mathbf{W}_B$), and $\bar{o}_1^{\mathbf{A}} = W_3^{\mathbf{A}} \cdot h(W_1^{\mathbf{A}}\bar{o}_1^{\mathbf{B}} + W_2^{\mathbf{A}}i_2^{\mathbf{A}})$.

Let us now consider an even simpler example that, nevertheless, illustrates the power of fibring neural networks. Consider two networks \mathbf{A} and \mathbf{B} , both with a single input neuron ($i^{\mathbf{A}}$ and $i^{\mathbf{B}}$, respectively), a single hidden neuron and a single output neuron ($o^{\mathbf{A}}$ and $o^{\mathbf{B}}$, 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 $o^{\mathbf{A}} = f(W_2^{\mathbf{A}} \cdot f(W_1^{\mathbf{A}} \cdot f(i^{\mathbf{A}}))) = i^{\mathbf{A}}$ and $o^{\mathbf{B}} = f(W_2^{\mathbf{B}} \cdot f(W_1^{\mathbf{B}} \cdot f(i^{\mathbf{B}}))) = i^{\mathbf{B}}$, where $W_1^{\mathbf{A}}$ and $W_2^{\mathbf{A}}$ are the weights of network \mathbf{A} , and $W_1^{\mathbf{B}}$ and $W_2^{\mathbf{B}}$ are the weights of network \mathbf{B} . Now, if we embed network \mathbf{B} into the input neuron of network \mathbf{A} , we obtain $\bar{o}^{\mathbf{B}} = f(\varphi(W_2^{\mathbf{B}}) \cdot f(\varphi(W_1^{\mathbf{B}}) \cdot f(i^{\mathbf{B}})))$ and $\bar{o}^{\mathbf{A}} = f(W_2^{\mathbf{A}} \cdot f(W_1^{\mathbf{A}} \cdot \bar{o}^{\mathbf{B}}))$. Since $f(x) = x$, we have $\bar{o}^{\mathbf{B}} = \varphi(W_2^{\mathbf{B}}) \cdot \varphi(W_1^{\mathbf{B}}) \cdot i^{\mathbf{B}}$ and $\bar{o}^{\mathbf{A}} = W_2^{\mathbf{A}} \cdot W_1^{\mathbf{A}} \cdot \bar{o}^{\mathbf{B}}$. Now, let our fibring function be $\varphi(\mathbf{W}_A, \mathbf{i}_A, \mathbf{W}_B) = i^{\mathbf{A}} \cdot \mathbf{W}_B$, where $\mathbf{W}_B = [W_1^{\mathbf{B}}, W_2^{\mathbf{B}}]$.³ Since $W_1^{\mathbf{A}}, W_2^{\mathbf{A}}, W_1^{\mathbf{B}}, W_2^{\mathbf{B}}$ are all equal to 1, we obtain $\bar{o}^{\mathbf{B}} = i^{\mathbf{A}} \cdot i^{\mathbf{A}} \cdot i^{\mathbf{B}}$ and $\bar{o}^{\mathbf{A}} = \bar{o}^{\mathbf{B}}$. This means that if we fix $i^{\mathbf{B}} = 1$, the output of network \mathbf{A} (fibred with network \mathbf{B}) will be $i^{\mathbf{A}} \cdot i^{\mathbf{A}}$. Finally, assume that the following sequence is given as input to \mathbf{A} fibred with \mathbf{B} : $n, 1/n, n+1, 1/(n+1), n+2, 1/(n+2), \dots$ for $n \in \mathfrak{R}$. The corresponding

³In practice, the fibring function φ must be defined depending on the problem domain.

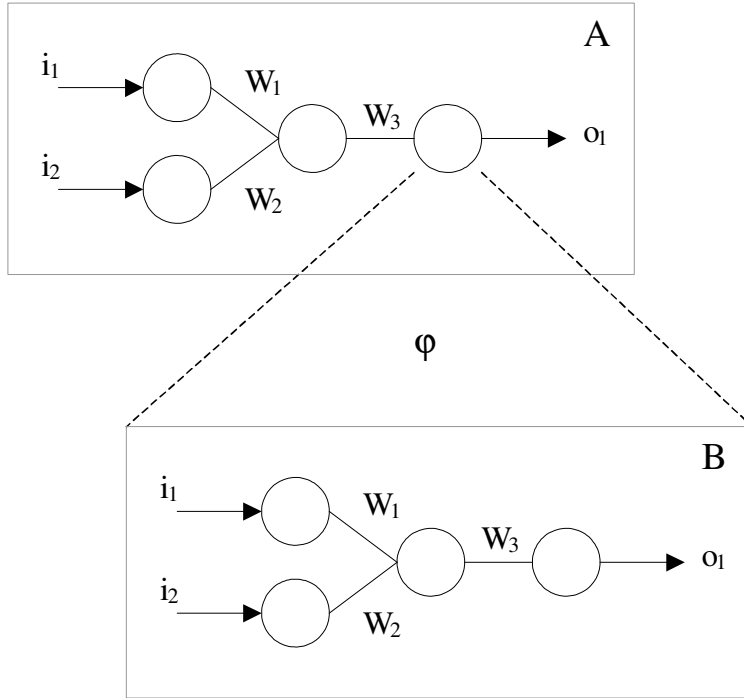


Figure 2: Fibring two simple networks

output sequence of **A** will be: $n^2, 1, (n+1)^2, 1, (n+2)^2, 1, \dots$. 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 \mathfrak{R} . This illustrates an important feature of fibred neural networks, namely, their ability to approximate functions in an unbounded domain [Hen02, Hin96]. This results from the recursive characteristic of fibred networks as indicated by the function $f_{\mathbf{W}_1, g_{\mathbf{W}_2}(i_2)}(i_1)$ computed by the network, and will be discussed in more detail in the following section.

⁴Note that, since the fibring function changes the weights of the embedded network, we use $1/n, 1/n+1, 1/n+2\dots$ to *reset* the weights back to 1 in the sequence computation.

3 Fibred Neural Networks

In this section, we define fibred neural networks (fNNs) precisely, we define the dynamics of fNNs, and we show that fNNs can approximate unbounded functions.

3.1 Fibring Definition

For the sake of simplicity, we restrict the definition of fibred networks to feedforward, single output neuron networks. We also concentrate on networks with linear input and 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 called 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. Let us denote the weight from the input neuron to the hidden neuron of network $x \in \{A, B, C\}$ by W_x^h , and the weight from the hidden neuron to the output neuron of x by W_x^o . Assume we embed network C into the output neuron of network B , and embed the resulting*

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

network into the output neuron of network A (according to Definition 2), as depicted in Figure 3. 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 $\varphi_B = i_A^o \cdot \mathbf{W}_B$ and $\varphi_C = i_B^o \cdot \mathbf{W}_C$, where i_A^o is the input potential of A 's output neuron given input x , i_B^o is the input potential of B 's output neuron given inputs x and 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 $a \in \mathfrak{R}^+$, 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 , the input potential of A 's output neuron will be $x\sqrt{a}$. 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 $o = ax^2$ for network B and then A . Since network C is also embedded into the system, given input y , fibring function φ_C will be used to update the weights of network C , according to the input potential of B 's output neuron. Thus, given $y = 1$, the input potential of B 's output neuron will be ax^2 , and the weights of network C will change to $W_C^h = ax^2$ and $W_C^o = ax^2$. Finally, assume $z = 1$. The output o of networks C , B and A will be a^2x^4 . This illustrates the computation of polynomials in fNNs. The computation of odd degree polynomials and of negative coefficients could be achieved with the addition of more hidden layers to the networks, as we will see in the sequel.

3.2 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, \dots, N_n be neural networks. N_1, N_2, \dots, 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, \dots, 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 feed-forward networks. The output o_j of network N_j ($1 \leq j \leq n - 1$) is defined*

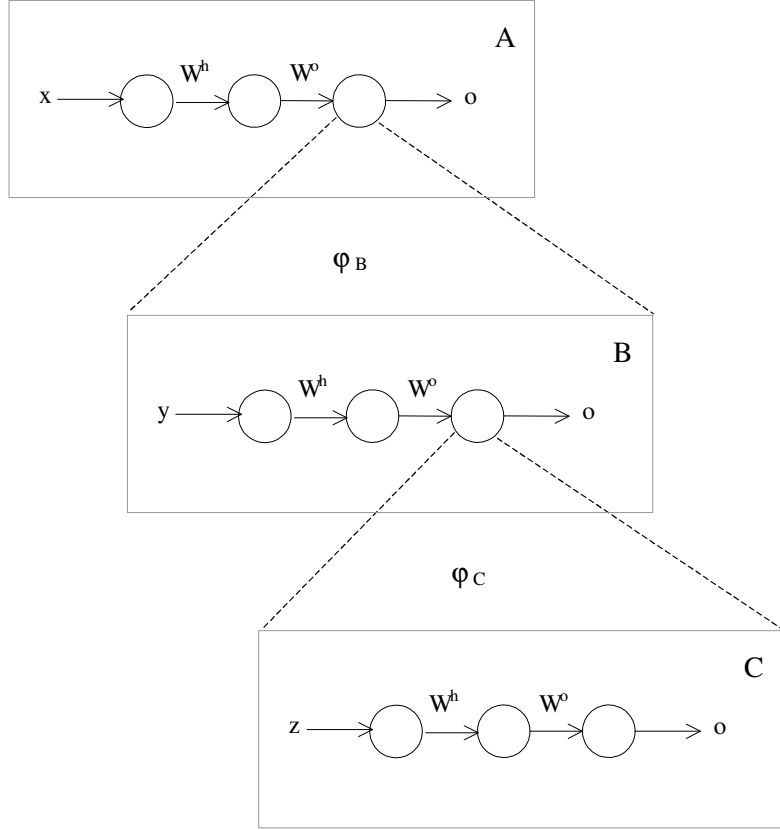


Figure 3: Nesting Fibred Networks

recursively in terms of the output o_{j+1} of network N_{j+1} , as follows:

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

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

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

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

3.3 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 [HSW89], together with the observation that level zero networks are a generalisation of single hidden layer feedforward networks. \square*

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, and its three embedded networks A , B and C at level 1, all containing linear neurons. Let $n + 1$ ($n \in \mathbb{N}$) be the number of input neurons of N , $0 \leq i \leq n$, $a_i \in \mathbb{R}$. We embed $n - 1$ networks into the input neurons of N , each network representing x^2, x^3, \dots, x^n , as indicated in Figure 4 for networks A , B and C , representing x^2, x^3 and x^n , respectively. 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, \dots, 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 such a network computes $a_j x^j$. From Definition 5, given input x to N and $\varphi_j = x \mathbf{W}_j$, 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 Figure 4), completing the proof that fNNs compute $\sum_i a_i x^i$. \square*

4 Conclusions and Future Work

This paper has introduced a new neural network architecture named fibred neural networks (fNNs), which combines a number of standard feedforward

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

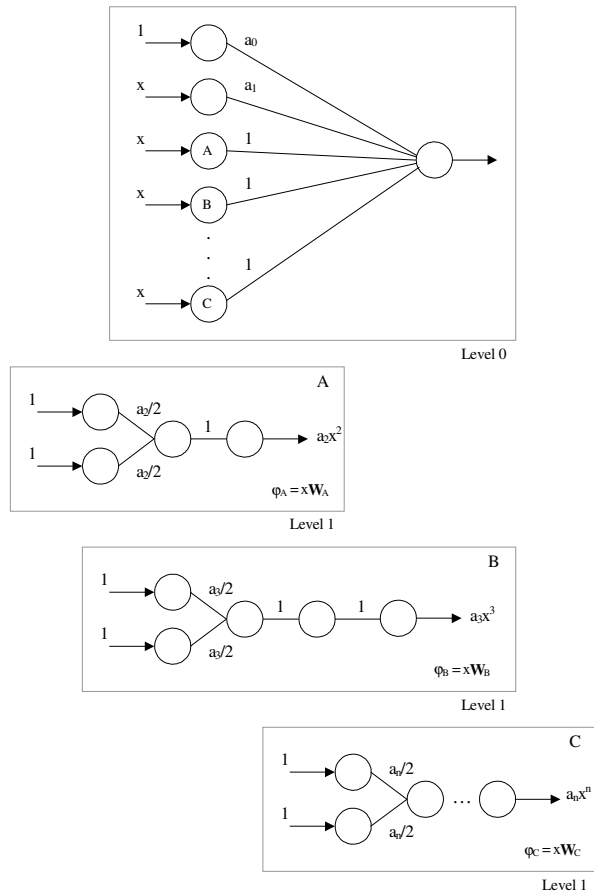


Figure 4: Computing polynomials in fibred networks

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 lim-

ited ability to compute unbounded functions [Hen02]. A comparison of these two architectures' capabilities 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 be required.

5 Acknowledgments

We are grateful to Stefan Rueger for useful discussions.

References

- [AM02] N. Angelopoulos and S. H. Muggleton. Machine learning metabolic pathway descriptions using a probabilistic relational representation. *Electronic Transactions in Artificial Intelligence*, 6, 2002. MI-19.
- [BV01] B. Boutsinas and M. N. Vrahatis. Artificial nonmonotonic neural networks. *Artificial Intelligence*, 132:1–38, 2001.
- [Cyb89] G. Cybenko. Approximation by superposition of sigmoidal functions. In *Mathematics of Control, Signals and Systems 2*, pages 303–314. 1989.
- [CZ00] I. Cloete and J. M. Zurada, editors. *Knowledge-Based Neurocomputing*. The MIT Press, 2000.
- [dGBG02] 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.
- [Elm90] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.
- [Fu94] L. M. Fu. *Neural Networks in Computer Intelligence*. McGraw Hill, 1994.
- [Gab99] D. M. Gabbay. *Fibring Logics*. Oxford University Press, 1999.
- [Hay99] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.
- [Hen02] J. Henderson. Estimating probabilities of unbounded categorization problems. In *Proceedings of European Symposium on Artificial Neural Networks*, pages 383–388, Bruges, Belgium, April 2002.

- [Hin96] J. Wesley Hines. A logarithmic neural network architecture for unbounded non-linear function approximation. In *Proceedings of IEEE International Conference on Neural Networks*, Washington, USA, June 1996.
- [HK92] S. Holldobler and F. Kurfess. CHCL: A connectionist inference system. In B. Fronhofer and G. Wrightson, editors, *Parallelization in Inference Systems*, pages 318–342. Springer, 1992.
- [HKS99] S. Holldobler, 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*, 11(1):45–58, 1999.
- [HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.
- [KSC01] B. Kijssirikul, S. Sinthupinyo, and K. Chongkasemwongse. Approximate match of rules using backpropagation neural networks. *Machine Learning*, 43(3):273–299, 2001.
- [Pin95] G. Pinkas. Reasoning, nonmonotonicity and learning in connectionist networks that capture propositional knowledge. *Artificial Intelligence*, 77:203–247, 1995.
- [Pol90] J. B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1):77–105, 1990.
- [RHW86] 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.
- [SA97] R. Sun and F. Alexandre. *Connectionist Symbolic Integration*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1997.
- [Sha99] 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*, 11:79–108, 1999.
- [Smo90] P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist networks. *Artificial Intelligence*, 46:159–216, 1990.
- [Smo00] P. Smolensky. Grammar-based connectionist approaches to language. *Cognitive Science*, 23:589–613, 2000.
- [TH88] D. Touretzky and G. Hinton. A distributed connectionist production system. *Cognitive Science*, 12(3):423–466, 1988.

- [TS94] G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1):119–165, 1994.
- [Wer74] P. J. Werbos. *Beyond Regretion: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA, 1974.
- [Wer90] P. J. Werbos. Backpropagation through time: what does it mean and how to do it. In *Proceedings of the IEEE*, volume 78, pages 1550–1560, 1990.
- [WG03] J. Williamson and D. Gabbay. Recursive causality in bayesian networks and self-fibring networks. Technical report, Department of Computing, King’s College London, UK, 2003.