ESANN'2000 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 26-28 April 2000, D-Facto public., ISBN 2-930307-00-5, pp. 383-388

Simplified neural architectures for symmetric boolean functions

Bernard Girau

INRIA, France & School of Computer Science, McGill University
3480 University street, Montreal, Quebec H3A 2A7, Canada
email: bgirau@cgm.cs.mcgill.ca

Abstract. The theoretical and practical framework of *Field Programmable Neural Arrays* has been defined to reconcile simple hardware topologies with complex neural architectures: FPNAs lead to powerful neural models whose original data exchange scheme allows to use hardware-friendly neural topologies. This paper addresses preliminary results in the study of the computation power of FPNAs. The computation of symmetric boolean functions is taken as a textbook example. The FPNA concept allows successive topology simplifications of standard neural models for such functions, so that the number of weights is greatly reduced with respect to previous works.

1. Introduction

Various fast parallel implementations of neural networks have been developed (see [2]). The very fine-grain parallelism of neural networks uses many information exchanges, so that hardware implementations are more likely to fit neural computations. Nevertheless, digital hardware implementations of neural networks either handle simplified neural computations or simple neural architectures, or they limit themselves to few well-fitted neural architectures. An upstream work is preferable: neural computation paradigms may be defined to counterbalance the main implementation problems, and the use of such paradigms naturally leads to neural models that are more tolerant of hardware constraints ([2]). Since the main implementation difficulties are linked to area-greedy operators and complex neural architectures, two kinds of *hardware-adapted paradigms of neural computation* may be found. Several models, such as bit-stream neural networks([5]), allow to handle area-saving neural computations, whereas FPNAs (Field Programmable Neural Arrays defined in [1]) lead to complex neural processings based on simplified topologies.

The practical study of FPNAs shows that they lead to very efficient hardware implementations of neural networks. The theoretical study of their computation power shows that FPNAs appear as more powerful than standard multilayer models for the exact computation of discrete functions (i.e. FPNAs require less neural resources). Conversely, FPNAs are less powerful than

ESANN'2000 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 26-28 April 2000, D-Facto public., ISBN 2-930307-00-5, pp. 383-388

standard multilayer models for the exact computation of continuous functions (virtual weights constrained in a subspace). As for the problem of approximate computing, [1] shows that the topological simplifications of FPNAs do not infer a significant loss of approximation capability.

This paper shows how FPNAs allow simplified neural architectures to compute discrete functions. The case of symmetric boolean functions is studied. The textbook case of the parity problem is detailed. Section 2. shortly describes the FPNA computation paradigm. Section 3. shows how this paradigm applies to symmetric boolean functions: previous results are recalled, subsection 3.1. shows how the FPNA concept allows to get rid of any shortcut link, and subsection 3.2. finally describes how a FPNA with $\mathcal{O}(\sqrt{d})$ weights may replace the quasi-optimal shortcut perceptron of [6] that has $\mathcal{O}(d\sqrt{d})$ weights.

2. Field Programmable Neural Arrays

A FPNA is a set of resources with configurable interactions (inspired by FPGA, Field Programmable Gate Arrays). These autonomous resources are: *neurons* that apply standard neural functions to a set of input values, and *links* that behave as independent affine operators. The links connect the nodes of a directed graph, each node contains one neuron. The specificity of FPNAs is that a link may be connected or not to the local neuron *and to the other local links*. Direct connections between affine links appear, so that the FPNA may compute numerous composite affine transforms, i.e. numerous *virtual neural links*. A FPNA is formally defined as:

- a directed graph $(\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is a finite set of nodes, and $\mathcal{E}$ is a set of directed edges without loop: for each node n, $Pred(n)$ (resp. Succ(n)) is the set of the direct predecessors (resp. successors) of n, the set of the input nodes is $\mathcal{N}_i = \{n \in \mathcal{N} \mid Pred(n) = \emptyset\}$,
- a set of affine functions $(x \mapsto W_n(p)x + T_n(p))_{(p,n)\in\mathcal{E}}$: each link is associated with an affine operator,
- a set of neurons $((\theta_n, i_n, f_n))_{n\in\mathcal{N}-\mathcal{N}_i}$, where $\theta_n \in \mathbb{R}$, i_n is a function from $\mathbb{R}^2$ to $\mathbb{R}$, and f_n is a function from $\mathbb{R}$ to $\mathbb{R}$: for each (non-input) node, one neuron resource handles any neuron computation in a sequential way,
- for each node n, an integer $a_n \geq 0$: expected number of received values, or number of global inputs sent by n if $n \in \mathcal{N}_i$,
- for each node n, several binary values: $\forall\, p \in Pred(n)$, $r_n(p)$ set to 1 iff link (p, n) is connected to the neuron in n; $\forall\, (p, s) \in Pred(n) \times Succ(n)$, $R_n(p, s)$ set to 1 iff links (p, n) and (n, s) are connected; $\forall\, s \in Succ(n)$, $S_n(s)$ set to 1 iff the neuron in n is connected to link (n, s).

When a FPNA resource receives values, it applies its local operator(s), and it sends the result to all neighboring resources to which it is locally connected (a

ESANN'2000 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 26-28 April 2000, D-Facto public., ISBN 2-930307-00-5, pp. 383-388

link may handle several values, and it may directly send them to other links). The following sequential computation illustrates the basic FPNA principles. This computation algorithm handles a list of tasks $\mathcal{L}$ that are processed according to a FIFO scheduling. Each task $[(p,n),x]$ corresponds to a value x sent on a link (p,n). Each node n in $\mathcal{N}-\mathcal{N}_i$ has got local variables c_n and x_n, initially set as $c_n = 0$ and $x_n = \theta_n$.

Initialization: For each input node n in $\mathcal{N}_i$, a_n values $\left(x_n^{(i)}\right)_{i=1..a_n}$ are given (global inputs of the FPNA), and the corresponding tasks $[(n,s), x_n^{(i)}]$ are created for all s in $Succ(n)$ such that $S_n(s) = 1$.

Sequential processing: (while $\mathcal{L}$ is not empty)

1. remove the first element $[(p,n),x]$ from $\mathcal{L}$
2. compute $x' = W_n(p)x + T_n(p)$
3. for all $s \in Succ(n)$ such that $R_n(p,s) = 1$, create $[(n,s),x']$
4. if $r_n(p) = 1$ (the neuron in n "receives" the value of task $[(p,n),x]$)
   - update c_n and x_n : $c_n = c_n + 1$, $x_n = i_n(x_n, x')$
   - if $c_n = a_n$ (the local neuron computes its output)
     i. $y = f_n(x_n)$, $c_n = 0$, $x_n = \theta_n$
     ii. for all $s \in Succ(n)$ such that $S_n(s) = 1$, create $[(n,s),y]$

3. Symmetric boolean functions by FPNAs

The neural computation of symmetric boolean functions has been a widely discussed problem. The quasi-optimal results of [6] answer a question that was posed as early as in [3]. A boolean function $f : \{0,1\}^d \to \{0,1\}$ is said symmetric if $f(x_1,\ldots,x_d) = f(x_{\sigma(1)},\ldots,x_{\sigma(d)})$ for any permutation σ of $\{1,\ldots,d\}$. An example is the d-dimensional parity problem: it consists in classifying vectors of $\{0,1\}^d$ as *odd* or *even*, according to the number of non zero values among the d coordinates. This problem may be solved by d-input multilayer perceptrons (MLP) or shortcut perceptrons. A MLP consists of several ordered layers of sigmoidal neurons. Two *consecutive* layers are fully connected. A layer which is not the input layer nor the output layer is said *hidden*. A shortcut perceptron also consists of several ordered layers of sigmoidal neurons. But a neuron in a layer may receive the output of any neuron in *any* previous layer.

The search for optimal two-hidden layer shortcut perceptrons in [6] has led to solve the d-dimensional parity problem with only $\sqrt{d}(2 + o(1))$ neurons, thanks to an iterated use of a method introduced in [4]. The shortcut links and the second hidden layer are essential in this work, though there is no shortcut link towards the output neuron. This neural network uses $d(2\sqrt{d} + 1 + o(1))$ weights. The results of [6] apply to any symmetric boolean function. Figure 1(a) shows the topology of the optimal shortcut network

ESANN'2000 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 26-28 April 2000, D-Facto public., ISBN 2-930307-00-5, pp. 383-388

of [6] for the 15-dimensional parity problem. In such a neural network, the first hidden layer may contain $\left\lceil \sqrt{d} \right\rceil$ neurons such that the i-th neuron of this layer computes $y_{i,1} = \sigma\left(\sum_{j=1}^{d} x_j + \Theta_i\right)$, where $\sigma(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$. The second hidden layer contains at most $\left\lceil \frac{d+1}{\lceil \sqrt{d} \rceil} \right\rceil$: its i-th neuron computes $y_{i,2} = \sigma\left(\sum_{j=1}^{\lfloor \sqrt{d} \rfloor} w_{i,j,2} y_{j,1} + (-1)^i \sum_{j=1}^{d} x_j\right)$. Then $y = \sigma\left(\sum_{j=1}^{\sqrt{d}} w_{i,j,3} y_{j,2} + \Theta\right)$ is computed by the only output neuron.

3.1. Removing the shortcut links

The construction in [6] implies that for any (i, j), $(-1)^i$ and $w_{i,j,2}$ have opposite signs. This property may be used so that all shortcut links (between the input and the second hidden layer) are virtually replaced by some direct connections between incoming and outgoing links in the first hidden layer of a FPNA. This FPNA has got $d\sqrt{d}(1 + o(1))$ weights, instead of $d\sqrt{d}(2 + o(1))$ weights in [6].

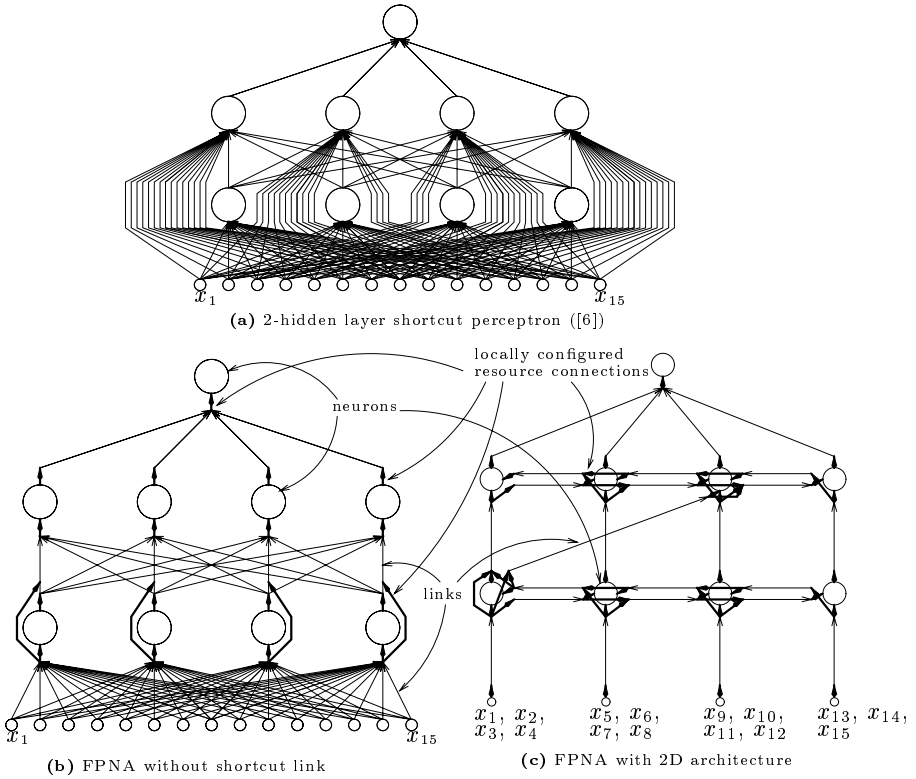
More precisely, the architecture of the FPNA is the same as the shortcut perceptron of [6], *without all shortcut links*. The weights of the links between both hidden layers are as in [6]. Each neuron is fully connected to all incoming and outgoing links ($\forall\, (p, n)\ r_n(p) = 1$ and $\forall\, (n, s)\ S_n(s) = 1$). If n is the i-th node of the first hidden layer, and if s is the i-th node of the second hidden layer, then for any $p \in Pred(n)$, there is a direct connection between (p, n) and (n, s) (i.e. $(R_n(p, s) = 1)$. If n is the i-th node of the first hidden layer, then for any $p \in Pred(n)$, $W_n(p) = -\frac{1}{w_{i,i,2}}$ and $T_n(p) = 0$. If n is the i-th node of the first hidden layer, then $\theta_n = -\frac{\Theta_i}{w_{i,i,2}}$. Figure 1(b) sketches the architecture of such a FPNA for a 15-dimensional symmetric boolean function.

3.2. Towards a simplified 2D architecture

Even without shortcut links, the FPNA of figure 1(b) still does not have a hardware-friendly architecture. A more drastic simplification of the architecture is expected. The full connection between consecutive layers may be virtually replaced by the use of sparse inter-layer and intra-layer FPNA links.

The construction of a FPNA in section 3.1. does not depend on the symmetric boolean function. On the contrary, the determination of a hardware-friendly FPNA for symmetric boolean functions takes advantage of function-dependent weight similarities in [6]. Such successful determinations have been performed for various symmetric boolean functions and input dimensions: it appears that for any d and for any symmetric boolean function f, a FPNA with the same number of neurons as in [6], but with only $\mathcal{O}(\sqrt{d})$ weights computes f *exactly* as in [6]. Nevertheless, this assertion has not yet been formally proved. The

ESANN'2000 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 26-28 April 2000, D-Facto public., ISBN 2-930307-00-5, pp. 383-388



(a) 2-hidden layer shortcut perceptron ([6])

(b) FPNA without shortcut link

(c) FPNA with 2D architecture

Figure 1: Neural architectures for the parity problem ($d = 15$)

parity problem may be taken as an example: in [6], the weight of the link between the i-th neuron of the first hidden layer and the j-th neuron of the second hidden layer only depends on $(-1)^j$ when $i \neq 1$. This property may be used so as to build a hardware-friendly FPNA as follows.

The number of nodes in each hidden layer is the same as the number of neurons in [6]. The number of input nodes is the number of nodes in the first hidden layer. Each input node sends up to $\left\lceil \sqrt{d} \right\rceil$ inputs.

The inter-layer links are: for any i, one link from the i-th input node towards the i-th node of the first hidden layer, and one link from the i-th node of the first hidden layer towards the i-th node of the second hidden layer, and one link from the i-th node of the second hidden layer towards the output node. Moreover, for any $j > 1$, there is a link between the first node of the first hidden layer and the $(2j-1)$-th node of the second hidden layer.

The intra-layer links are: in both hidden layers, for any i, one link from the

ESANN'2000 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 26-28 April 2000, D-Facto public., ISBN 2-930307-00-5, pp. 383-388

i-th node tow ards the $(\ell+1)$-th node, another one tow ards the $(\ell-1)$-th node.

The $S_n(s)$, $r_n(p)$ and $(R_n(p,s)$ parameters are set so as to ensure a virtual full connection scheme between consecutive layers. Moreover the $(R_n(p,s)$ parameters are set so that any virtual shortcut link involv es the first node of the first hidden layer. See [1] for more details and for the weight determination.

This FPNA (for any number d of inputs) is easy to map onto a 2D hardware topology, whereas the equivalent shortcut perceptron in [6] rapidly becomes too complex to be directly implemented when d increases. Figure 1(c) shows the arc hitecture of the FPNA for the 15-dimensional parity problem.

Moreover, the theoretical study of [1] shows that the above FPNAs satisfy several conditions that ensure a computation time proportional to the number of weights as in standard multila yer models. Therefore, a $\mathcal{O}(\sqrt{d})$ computation time is achieved thanks to the topological simplifications of the FPNAs.

4. Conclusion

The FPNA framework is a neural computation paradigm that has been defined to fit digital hardware devices. This paper shows how FPNAs allow successive topological simplifications of the standard neural arc hitectures that compute symmetric boolean functions. The shortcut links and then the full inter-la yer connections are removed: they are replaced by virtual links based on numerous multicast composite connections. This work allows to compute d-dimensional symmetric boolean functions by neural models with $\mathcal{O}(\sqrt{d})$ weights, instead of $\mathcal{O}(d\sqrt{d})$ w eigh ts in the best previous works. The results of [6] are proved to be quasi-optimal in the number of neurons. The question of the above FPNAs being quasi-optimal in the number of weigh ts is now posed.

References

[1] B. Girau. *Du parallélisme des modèles connexionnistes à leur implantation parallèle*. PhD thesis n° 99ENSL0116, ENS Lyon, 1999.

[2] B. Girau. Neural netw orks on FPGAs: a survey. In *Proc. Neural Computation*, 2000. T o be published.

[3] W. Kautz. The realization of symmetric switc hingfunctions with linear-input logical elements. *IRE Trans. Electron. Comput.*, EC-10, 1961.

[4] R. Minnick. Linear-input logic. *IEEE T ans. Ele ct on. Comput.*, EC-10, 1961.

[5] V. Salapura. Neural netw orks using bit-stream arithmetic: a space efficient implementation. In *Proc. IEEE Int. Conf. on Circuits and Systems*, 1994.

[6] K. Siu, V. Roychowdhury , and T. Kailath. Depth-size tradeoffs for neural computation. *IEEE Trans. on Computers*, 40(12):1402–1412, 1991.