第 32 卷 第 21 期 计 算 机 工 程 2006 年 11 月
Vol.32 No. 21 Computer Engineering November 2006
·博士论文· 文章编号：1000—3428(2006)21—0018—04 文献标识码：A 中图分类号：TP 311

BP 算法的脉动阵列结构在 FPGA 上的实现

郝智泉[1,2]，王贞松[1,2]
(1. 中国科学院计算技术研究所，北京 100080；2. 中国科学院研究生院，北京 100039)

摘 要：提出了一种用于实现 BP 神经网络的串行输入串行输出的脉动阵列结构，在 FPGA 上实现了基于该阵列结构的用于进行“A－Z”的印刷体字符识别系统。文中对 FPGA 中运算部件的微结构进行了讨论。实验结果表明，与软件实现相比用 FPGA 实现神经网络算法能够极大地提高 BP 网络的学习和分类速度。

关键词：神经网络；脉动阵列结构；BP 算法；FPGA

Realization of BP Algorithm in FPGA Based on Systolic Array Architecture

HAO Zhiquan[1,2], WANG Zhensong[1,2]
(1. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080;
2. School of Graduate, Chinese Academy of Sciences, Beijing 100039)

【Abstract】This paper presents a kind of systolic array architecture which is used to realize BP algorithm. It designs a characters recognition system in FPGA based on this systolic array architecture. The micro architectures of the computing components in FPGA are analyzed. Experiment results show that realizing BP algorithm in FPGA can improve the study and recognition speed.

【Key words】 Neural networks; Systolic array architecture; BP algorithm; FPGA

神经网络算法在模式识别、信号处理、时间序列分析等领域有着广泛的应用。目前用软件实现神经网络算法，速度慢，所使用的计算机体积庞大，不适于嵌入式场合的应用。专用的神经网络芯片常常只适用于特定的算法，使用起来不够灵活。随着 FPGA 的规模不断扩大，它能够同时满足系统的速度和灵活性两方面的要求，可以作为硬件实现神经网络算法的理想载体。

本文提出了一种适于实现多层感知器网络的串行输入串行输出的脉动阵列结构。并在这一结构的基础上以 FPGA 为平台实现了一个带有学习功能的字符识别系统。该系统用于识别“A－Z”的印刷体字符。与基于 PC 机的软件实现相比，FPGA 内部能够实现更多神经元单元，将算法进行充分并行化，极大地提高了识别和训练速度。

1 BP 算法和字符识别

神经网络是一个由简单处理单元构成的规模宏大的并行分布式处理器[1]。这些简单的计算单元称为“神经元”。使用 BP算法作为学习算法的多层感知器网络以其强大的非线性映射能力和学习功能，已经成为目前在模式识别、语音处理等领域使用最为广泛的神经网络模型。

1.1 BP 算法的基本原理

反向传播算法（BP 算法）是目前应用很广的神经网络算法。BP（Back-Propagation）算法的主要思想是从后向前（反向）逐层传播输出层误差，以间接计算出隐层误差，再根据计算所得的误差修正神经元之间的连接权值。算法分为两个阶段：第 1 阶段（正向过程）输入信息从输入层经隐层逐层计算各单元的输出值；第 2 阶段（反向传播过程）输出误差逐层向前算出隐层各单元的误差，并用此误差修正前层权值。

为论述方便，采用图 1 所示的神经元符号约定。

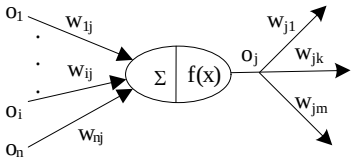

图 1 神经元的符号约定

图 1 中 j 用于表示当前神经元的编号，i 用于表示前一层的神经元，k 表示下一层的神经元。

当输入某个样本时，从前到后每个神经元依次计算：

$$u_j = \sum_i \omega_{ij} O_i \tag{1}$$

$$O_j = f(u_j) \tag{2}$$

对输出层而言，$\hat{y}_j = O_j$ 为实际输出，y_i为理想输出值，样本误差可以表达为

$$E = \frac{1}{2}\sum_j (y_j - \hat{y}_j)^2 \tag{3}$$

为使式子简化，定义局部梯度

$$\delta_j = \frac{\partial E}{\partial u_j} \tag{4}$$

考虑权值 ω_{ij} 对误差的影响，有

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial u_j}\frac{\partial u_j}{\partial \omega_{ij}} = \delta_j O_i \tag{5}$$

基金项目：国家自然科学基金资助项目(60303017)
作者简介：郝智泉(1978－)，男，博士，主研方向：实时信号处理和芯片设计；王贞松，教授、研究员、博导
收稿日期：2005-12-06 **E-mail：**haozhq@ict.ac.cn

权值修正应该使误差最快减小，修正量为

$$\Delta\omega_{ij} = -\eta_\omega \delta_j O_i \tag{6}$$

$$\omega_{ij}(t+1) = \omega_{ij}(t) + \Delta\omega_{ij}(t) \tag{7}$$

其中 η_ω 为修正的步进量。

进一步推导分两种情况：

(1)如果节点是输出单元，则

$$O_j = \hat{y}_j \tag{8}$$

$$\delta_j = \frac{\partial E}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial net_j} = -(y_j - \hat{y}_j) f'(net_j) \tag{9}$$

(2)如果节点不是输出单元，O_j对所有的后层节点都有影响。因此

$$\delta_j = \frac{\partial E}{\partial net_j} = \sum_k \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial O_j}\frac{\partial O_j}{\partial net_j} = \sum_k \delta_k \omega_{jk} f'(net_j) \tag{10}$$

对于 Sigmoid 函数

$$y = f(x) = \frac{1}{1+e^{-x}} \tag{11}$$

有

$$f'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = y(1-y) \tag{12}$$

分别代入到式(9)和式(10)中：

输出节点单元

$$\delta_j = -(y_j - \hat{y}_j)\hat{y}_j(1-\hat{y}_j) \tag{13}$$

非输出节点单元

$$\delta_j = O_j(1-O_j)\sum_k \delta_k \omega_{jk} \tag{14}$$

1.2 实现“A－Z”字符识别的神经网络结构

对于“A－Z”的字符识别，实际上是对输入的表征某个字符的一组特征向量进行准确的分类，根据该向量所属的类别判定它具体表征哪一个字符。

首先将字符的二值图像规格化，即将图像分割为等大小的 16×8 个子图像，以子图像中白色像素点与整个子图像中像素点数的比值作为 BP 神经网络的一个输入。16×8 个子图像的规格化结果作为 BP 网络的输入向量。整个网络是一个如图 2 所示的三层 BP 网络。

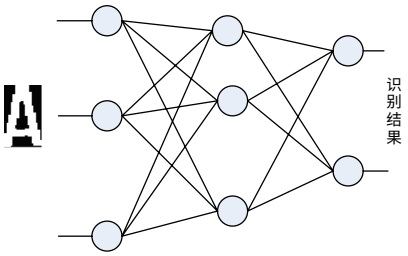

图 2 三层 BP 网络

网络具有 128 个输入节点，32 个隐层节点，26 个输出层节点。

在 BP 网络的结构确定以后，需要选取一定数量的样本对 BP 进行训练，即确定神经元之间的连接权值。完成训练以后，网络就可以用来进行字符识别了。训练过程包括一次正向过程、一次误差反向传播过程和一次权值更新过程；识别过程就是一次正向计算过程。

2 BP 网络的脉动阵列实现

自从 1978 年 H.T.Kung 提出了脉动阵列的概念以后，脉动阵列在数字信号处理领域得到了广泛的应用。脉动阵列是一种有节奏地计算并通过系统传输数据的处理单元网络。它的实现主要是通过将线性映射技术用于规划依赖图（Dependence Graph DG）来实现的。BP 算法的每一层神经元核心运算是向量矩阵乘法运算，向量矩阵乘法是一种典型的适合用脉动阵列结构来实现的运算。

2.1 向量和矩阵的乘法脉动阵列结构

向量和矩阵的乘法的 DG 图如图 3 所示，向量 $\vec{a}$ 由阵列左端输入，计算结果 $\vec{u}$ 从阵列的下方输出。

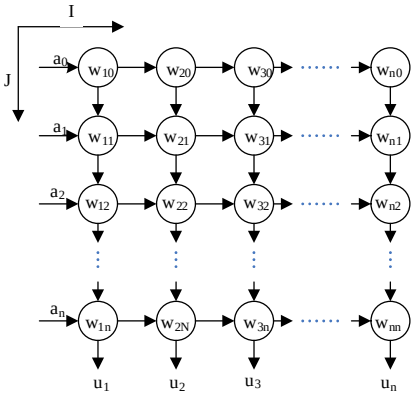

图 3 向量和矩阵的乘法的 DG 图

对于这种规则的二维DG，可以将它通过投影的方法映射成为一维的信号流图(Signal Flow Graph, SFG图)[2]。如果按 $\vec{d} = (i\ \ j) = (0\ \ 1)$ 的方向进行投影，得到的一维SFG如图 4。

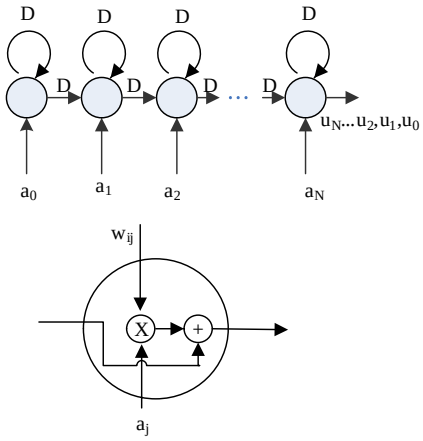

图 4 (0, 1)方向投影

向量并行输入到每个计算节点，运算结果在最后一个节点串行输出。如果按 $\vec{d} = (1\ \ 0)$ 方向进行投影，得到的一维SFG 如图 5 所示。

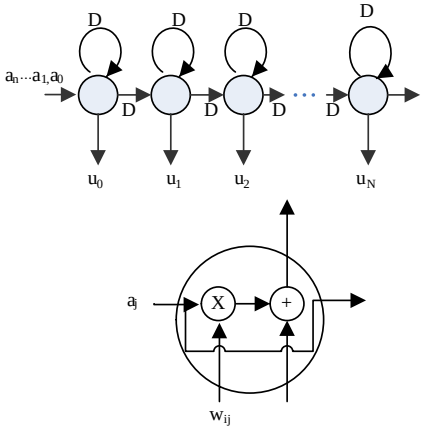

图 5 (1, 0)方向投影

输入按顺序输入，并被广播到所有计算元节点，在完成

乘法累加操作后，结果并行输出。

2.2 BP 算法的脉动阵列结构

S.Y.Kung 和 J.N.Hwang 在文献[3]中给出了一种实现神经网络的脉动阵列结构。这种脉动阵列结构是并行输入，并行输出，且要求每一层的神经元数量相同，对于各层数量不相同的神经网络结构，需要插入相应数量的空操作神经元。由于进行字符识别的 BP 网络的输入数量很大，如果使用并行输入方式，将极大地耗费 FPGA 宝贵的 I/O 资源。本文提出可通过改变投影方向的方法，实现了一种串行输入，串行输出的脉动阵列结构。

对隐层网络按 $\vec{d}=(1\quad 0)$ 投影，对于输出层网络按 $\vec{d}=(1\quad 0)$ 方向进行投影，可以得到如图 6 所示的一个格型的三层 BP 神经网络脉动阵列实现的拓扑结构。

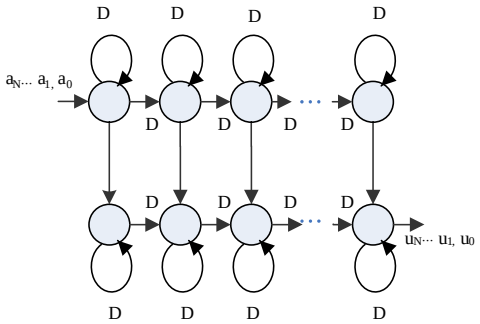

图 6 神经网络脉动阵列

(1)输出层神经元的处理过程

正向过程：输入信号并行输入，在计算节点完成输入和权值的乘法运算，然后再和前一个计算节点送来的结果相加，累加和送入后一个计算节点。最终的运算结果由最后一个计算节点串行输出。反向过程：对于输出层神经元，串行输入 δ_k，δ_k 被广播到每个神经元，每个神经元同时计算 $\sum_k \delta_k \omega_{jk}$，运算结果并行反向回传到隐层网络。权值调整过程：权值调整过程中计算 $\Delta\omega_{jk}=-\eta_\omega \delta_k O_j$，并更新权值。权值更新过程可以和误差的方向传播过程同时进行。

(2)隐层神经元的处理过程

正向过程：信号从网络的左端顺序输入，运算结果同时从每一个神经元输出。由于误差不需要向输出层方向传播，在隐层只需要进行权值调整。对于权值调整过程，隐层的每个神经元并行地接收从输出层反传回来的 $\sum_k \delta_k \omega_{jk}$，按式 (14) 计算出 δ_j，再进行权值更新。

3 计算节点的实现

由于脉动阵列的结构规整，节点间的数据传输按照固定的时序进行，计算节点可以根据目前的时钟拍数进行相应的操作，节点间的握手信号十分简单。

3.1 输出层节点的微结构

由于输出层是串行输出，因此可以将 Sigmoid 函数的实现由单独一个计算节点进行。输出层计算节点的微结构如图 7 所示。

正向传播过程：计算节点j从隐层接收输出O_i，与从RAM中读出的 ω_{ij} 相乘，再与从节点j-1 输出的ACC_{j-1}相加，结果输出到ACC_j。

误差反向传播过程：每个节点同时接收到计算节点送来的 δ_k，在节点内部完成 $\sum_k \delta_k \omega_{jk}$ 操作，然后将结果输出至隐层神经元。反向传播过程可以和正向过程使用相同的运算功能单元，实现功能单元的复用。

权值更新过程：权值更新过程可以和误差的方向传播过程同时进行。每个节点同时接收计算节点送来的 $\eta\delta_k$，从 RAM 中读出 ω_{ij}，计算 $\omega_{jk}(t+1)=\omega_{jk}(t)-\eta\delta_k O_j$。

计算结果写回到 ω_{ij} 所在的 RAM 单元。

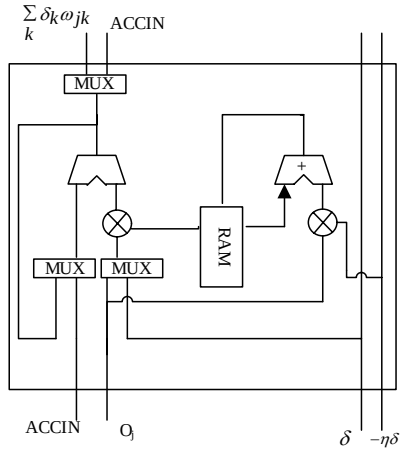

图 7 输出层节点的微结构

3.2 隐层节点的微结构

隐层节点的微结构如图 8 所示。

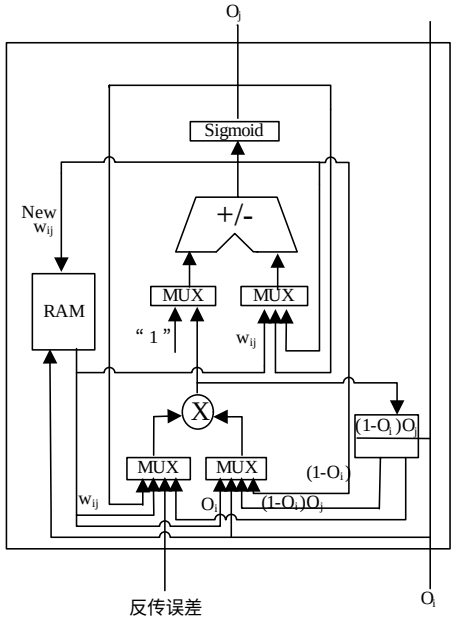

图 8 隐层节点的微结构

由于隐层节点的前一层是输入层，因此在隐层节点中不需要将误差进行反向传播。在正向过程中计算该节点的输出，而在反向过程中进行权值更新。

硬件乘法器是 FPGA 中很宝贵的资源，为节约资源，在隐层节点中只使用一个乘法器，前向操作和反向操作对它进行分时复用。此外，在正向操作完成后，等待反向操作开始的间隙，隐层节点完成 $\delta_j=O_j(1-O_j)\sum_k \delta_k \omega_{jk}$ 中 $O_j(1-O_j)$ 的运算，使功能单元得到更充分的利用。

3.3 sigmoid 函数的实现

Sigmoid 函数包含除法运算和指数运算，这两类运算都不适合用 FPGA 直接实现。为了简化 FPGA 中的硬件逻辑结

构和提高运算速度，使用查找表的方法实现 Sigmoid 函数。

对 Sigmoid 函数求微分的结果如图 9 所示，在[-4, 4]区间内函数变化较快，而在这个区间以外，函数值变化很缓慢。为了满足计算精度的前提下尽量减少实现查找表所耗费的 PPGA 片内的 RAM 资源，当 Sigmoid 函数的输入不在[-4, 4]区间内时，将 Sigmoid 函数的结果赋为定值，当结果在[-4, 4]内时，查表得到 Sigmoid 函数的输出。

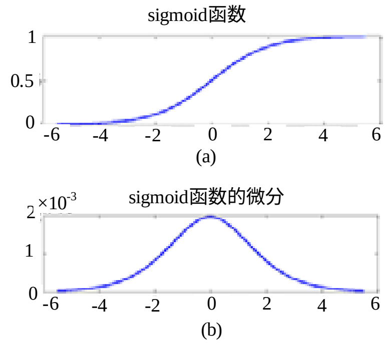

图 9 Sigmoid 函数及其微分

Sigmoid 函数的实现结构如图 10 所示，查找表由 FPGA 内的 Block RAM 实现。RAM 中存储的是 Sigmoid 函数的输出值，Sigmoid 函数的输入作为查找表的地址索引。函数的输入作为地址送到 RAM 的同时也送到比较器，比较的结果控制 MUX 选择哪一路数据。当如输入小于-4 时将其直接赋值为 5 ;当如输入大于 4 时将其直接赋值为 250 ;当输入在[-4，4]时直接将输入作为 RAM 的地址，读出 RAM 中的值作为 sigmoid 函数的输出。

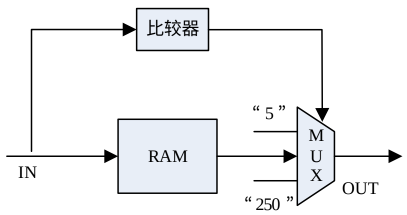

图 10 sigmoid 函数的实现

4 FPGA 上的实现结果

我们采用 16×8 大小网格图像作为神经网络的输入，输入节点 128 个，隐层节点 32 个，输出层节点 26 个。映射到 FPGA 中则，在隐层有 32 个计算节点，输出层有 32 个计算节点。系统使用的逻辑资源如表 1 所示。

表 1 系统使用的逻辑资源

Number of Slices:	4 480
Number of Slice Flip Flops:	6 138
Number of 4 input LUTs:	7 168
Number of bonded IOBs:	55
Number of TBUFs:	493
Number of BRAMs:	64
Number of MULT18X18s:	98

综合后的系统最高工作频率为 115.63MHz，当系统工作在 100MHz 的时钟频率下，得到表 2 实验数据。

表 2 实验数据

	FPGA	P42.4G CPU
一次正向过程	0.014ms	0.16ms
一次训练过程	0.024ms	0.32ms

实验数据表明与主频 2.4GHz 的 P4 CPU 相比，FPGA 实现相同的神经网络算法，系统的运算速度能够提高 10 倍以上。

本文提出的这种串行输入、串行输出的 BP 神经网络的脉动阵列结构，很切合神经网络的“简单处理单元的大规模并行”这一特点。脉动阵列内部各计算节点并行工作，提高了系统的运算速度。

该结构中计算节点结构相同，节点间的通信机制十分简单，能够在只改变很少的控制逻辑的基础上，通过增减计算节点的数量来实现不同规模的神经网络。计算节点内部结构规整，充分利用了 FPGA 内部内嵌的硬件乘法器和 Block RAM 资源。同时结构串行输入串行输出的 IO 方式明显减少了系统所要占用 FPGA 的 IO 引脚资源。

通过使用 FPGA 来实现神经网络算法能在保证系统灵活性的同时极大地提高系统的运算速度。

参考文献

1 Haykin S. Neural Networks a Comprehensive Foundation[M]. Prentice-Hall, 1999.

2 Kung S Y. Vlsi Array Processors[M]. Prentice- Hall, 1988.

3 Kung S Y, Hwang J N. Digital VLSI Architectures for Neural Networks[C]. Proc. of IEEE International Symposium on Circuits and Systems, 1998: 445-448.

（上接第 2 页）

if R/N ∈condsignal (Expr)

waitsignal (R/N) := waitsignal (R/N) ∪{P};

else

Null;

Verilog 中的 wait 语句是对电平敏感的，当执行到该语句时，如果表达式的值计算为真，则不会阻塞进程的执行；如果为假，那么需要等待，直至表达式的值变为真。因此对于这种电平敏感的时序控制语句，可以使用 VHDL 中的条件语句和 wait 语句来等效表示。

3 结论

本文使用抽象状态机模型对Verilog的形式语义从模拟的角度进行了研究，通过与VHDL比较制定出Verilog向VHDL转换的方法。这些方法已经在我们研制的Verilog—VHDL翻译器中成功应用，经过大量实例测试验证了转换前后功能的一致性，并且较国外同类商业产品[6]的可翻译子集有所扩展。

参考文献

1 Umamageswaran K. Formal Sematics and Proof for Optimizing VHDL Models[M]. Kluwer Academic Publishers, 1999.

2 李勇坚，何积丰，孙永强. Verilog 代数语义研究[J]. 软件学报, 2003, 14(3): 317-327.

3 Boerger E. A Formal Definition of an Abstract VHDL'93 Simulator by EA-Machine [M]. New York: Kluwer Academic Publishers, 1995.

4 Bowen J P. Animating the Semantics of Verilog Using Prolog[R]. United Nations University, Macao, ftp://ftp.iist.unu.erdu/ pub/ techreports/report176.ps.gz.

5 Dimitrov J. Operational Semantics for Verilog[C]. The 8th Asia-pacific Soft Engineering Conference, 2001.

6 Alternative System Concepts Inc.. Product Description[Z]. http://www. ascinc. com/product/verilog2VHDL, 2003.