論文2002-39SD-9-5

학습된 신경망 설계를 위한 가중치의 비트-레벨 어레이 구조 표현과 최적화 방법

(Bit-level Array Structure Representation of Weight and Optimization Method to Design Pre-Trained Neural Network)

林 國 贊 *, 郭 又 榮 *, 李 顯 洙 **

(Guk-Chan Lim, Woo-Young Kwak, and Hyon-Soo Lee)

요 약

학습된 신경망(Pre-trained neural network)은 고정된 가중치(weight)를 갖는다. 이 논문에서는 이러한 특성을 이용하여 신경망의 효과적인 디지털 하드웨어의 설계방법을 제안한다. 이를 위해 신경망의 PEs(Processing Elements)연산은 행렬-벡터 곱셈으로 표하고 고정된 가중치와 입력 데이터의 관계를 비트-레벨 어레이(array) 구조로 표현하여, 노드 소거와 가중치 비트 패턴에 따른 공유 노드 설정을 통한 최적화로 연산에 필요한 노드를 최소화한다. FPGA 시뮬레이션 결과, 완전한 정확성에 기반한 하드웨어를 설계하는 경우, 하드웨어 비용을 상당부분 줄였고 동작 주파수가 높다는 것을 확인하였다. 또한, 제안한 설계방법은 한정된 공간 내에서 많은 수의 PEs 구현이 가능함으로, 큰 신경망 모델에 대한 온-칩(on-chip) 구현이 가능하다.

Abstract

This paper proposes efficient digital hardware design method by using fixed weight of pre-trained neural network. For this, arithmetic operations of PEs(Processing Elements) are represented with matrix-vector multiplication. The relationship of fixed weight and input data present bit-level array structure architecture which is consisted operation node. To minimize the operation node, this paper proposes node elimination method and setting common node depend on bit pattern of weight. The result of FPGA simulation shows the efficiency on hardware cost and operation speed with full precision. And proposed design method makes possibility that many PEs are implemented to on-chip.

Key Words : 신경망, 구현, 어레이 구조, 최적화

I. 서 론

신경망(Neural network)은 음성 및 영상인식, 예측, 적응적 처리 및 최적화 문제를 해결하기 위한 다양한 해법을 제공한다. 이를 위해 신경세포의 인공 모델인 PE(Processing Element)들은 각각의 응용에 맞는 다양한 형태로 연결되어 적절한 연산과정을 수행하게 된다. 이러한 다양한 네트워크 모델은 최적의 구현 수단의 부재와 방대한 연산과정으로 인한 구현의 어려움 때문에 실제 구현이 이뤄지지 않고 시뮬레이션 레벨의 활용에 그치고 있다.

* 正會員, LG電子 WLL端末硏究所
(WLL Mobile Lab. LG Electronics Inc.)
** 正會員, 慶熙大學校 電子計算工學科
(Dept. of Computer Engineering Kyunghee Univ.)
接受日字:2001年11月15日, 수정완료일:2002年8月19日

현재 신경망 모델을 효율적으로 구현하기 위한 다양한 연구가 진행되고 있으며, 범용 PC을 이용한 방법, 아날로그 뉴런 칩(analog neuron chip), 디지털 하드웨어(digital hardware), 하이브리드 하드웨어(hybrid hardware)구현과 광학 디바이스(optical device)를 이용한 방법 등이 있다.[1,2]

범용 PC를 이용한 소프트웨어 구현은 실시간 응용에 부적합하고 고비용의 일반 컴퓨터가 요구되며 세밀한 병렬구조(fine granularity) 처리를 거의 지원하지 못한다. 아날로그 기술은 덧셈과 곱셈 등의 구현이 쉽고 높은 동작속도를 갖지만, 정확성과 안정성이 떨어지고 잡음과 파라메터(parameter) 영향에 의한 민감성 문제를 가질 수 있으므로 엄격하고 경험적인 조립 및 설계 방법이 필요하다. 광학 디바이스를 이용한 구현은 일찍이 실험적으로 시도되었다.[3] 디지털 하드웨어 구현은 정확성에 대한 조정이 가능하고 높은 유연성을 가질 뿐만 아니라 외부영향에 따른 문제가 거의 발생하지 않지만, 신경망이 갖는 복잡한 산술연산 때문에 하드웨어 비용이 높고 동작속도가 낮다는 문제점이 있다.

본 논문에서는 학습된 신경망을 디지털 하드웨어로 구현시, 가장 문제가 되는 곱셈연산을 최소화 할 수 있는 방법에 대해 다룬다. 이는 학습된 신경망이 고정된 가중치를 갖는다는 점을 이용하여 PE들의 연산형태를 집합된 행렬-벡터 곱셈으로 표현하고, 이를 비트-레벨(bit-level)에서 완전한 병렬계산이 이뤄지도록 하는 방법이다. 즉, 고정된 가중치와 입력데이터의 관계를 비트-레벨 어레이(array) 구조로 표현하고 결과에 영향을 주지 않는 노드 소거와 가산-트리 표현, 그리고 가중치의 비트-패턴(bit-pattern)에 따른 최적화를 통하여 노드를 공용하게 함으로써, 전체 연산에 필요한 노드를 최소화하였다.

본 논문에서는 완전한 정확성을 고려한 FPGA 시뮬레이션 결과, 분산연산(distributed arithmetic)[6] 및 CSD(Canonical Signed Digit)[7]과 같은 기존의 방법으로 구현하는 경우와 비교하여 하드웨어 비용을 상당부분 줄였으며, 높은 동작 주파수를 갖는 설계가 가능함을 확인하였다. 본 논문에서 제안한 설계방법은 신경망 설계에 필요한 복잡한 연산과정을 단순화하여 한정된 공간 내에서 많은 수의 PE구현이 가능케 함으로써, 복잡도가 큰 신경망 모델에 대한 온-칩(on-chip)구현이 가능함을 확인하였다.

Ⅱ. 학습된 신경망의 구조 및 기존의 설계 방법

1. PE의 구조

PE는 뉴런(neuron)의 인공적인 모델로, 유닛(unit) 또는 셀(cell)등으로 불리는데 대부분 신경망에서는 다-입력 1-출력 소자가 사용된다. 그림 1은 PE의 예이다. 여기서, I_n은 PE의 입력으로 뉴런과 뉴런사이의 시냅스(synapse) 연결을 나타내고, $w_{n,m}$은 각 입력에 대한 결합하중 즉, 가중치다. n은 PE의 입력 인덱스이고 m은 각 PE를 나타내는 인덱스이다. O_m는 입력의 총합이고 f는 출력응답함수이다. y_m은 비선형 함수인 f에 의해 변형되어 나오는 PE의 출력 값으로, 이 관계를 식 (1)과 같이 정의 할 수 있다. f는 PE의 응답특성을 결정한다.

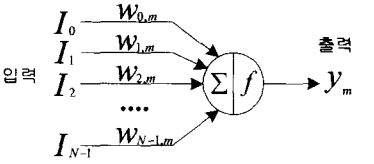


그림 1. PE의 구조
Fig. 1. Architecture of PE.

$$y_m = f(O_m) = f\left(\sum_{n=0}^{N-1} w_{n,m} I_n\right) \quad (1)$$

2. 학습된 신경망

신경망은 가중치 값의 고정 여부에 따라, 가중치가 동적인 적응적 네트워크(adaptive network)와 고정된 정적 네트워크(static network) 즉, 학습된 신경망이 있다. 적응적 네트워크는 환경에 따라 가중치가 변화하는 신경망으로 대부분 메모리를 이용한 구현 방법[4,5]이 있다. 반면에, 학습된 신경망은 학습을 통하여 가중치가 미리 정해져있지만, 식 (1)처럼 각 PE마다 입력벡터와 가중치 벡터의 곱셈연산이 필요하다. 이는 하드웨어 비용을 높이는 직접적인 문제로 이를 해결하기 위한 효과적인 알고리즘 및 설계방법이 필요하다. 여기에는 MAC(Multiply Accumulator), Wallace Tree, Baugh-Wooley, 분산연산, CSD, MCM(Multi- Constant

Multiplication[8]과 같이 DSP(Digital Signal Processing)분야에서 효율성을 높이기 위한 다양한 곱셈기의 구조[9]가 적용될 수 있다. 여기서 몇몇 알고리즘은 동작 속도를 높이고 세부 병렬구조를 표현하기 위해 비트-레벨에서 구현된다.

식 (1)에서 입력값 I_n은 식 (2)와 같이 비트-레벨로 표현할 수 있다. 이를 식 (1)에 대입하면 식 (3)과 같이 표현된다.

$$I_n = \sum_{t=0}^{T-1} I_{n,t} \cdot 2^{-t} \tag{2}$$

$$y_m = f(O_m) = f\left(\sum_{n=0}^{N-1}\sum_{t=0}^{T-1}(w_{n,m}I_{n,t})2^{-t}\right) \tag{3}$$

식 (3)을 연산하기 위한 기존의 MAC구조는 그림 2와 같다. 하나의 PE을 구성하기 위하여 N개의 MAC과 $\lceil \log_2 N \rceil$ 개의 덧셈기가 필요하므로 하드웨어 오버헤드(overhead)가 커지는 문제점이 있다.

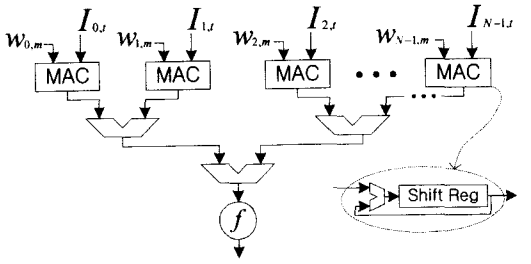


그림 2. MAC을 이용한 PE 설계
Fig. 2. PE design using MAC unit.

학습된 신경망은 가중치가 고정되어 있다. 이를 이용하여 보다 효율적인 하드웨어 구성 방법으로 분산연산과 CSD을 이용한 설계가 있다.

분산연산은 가중치가 고정되어 있으므로 입력 값에 의하여 출력 값이 결정된다. 따라서, 출력 값을 미리 계산하여 ROM에 저장하는 방법이다. 여기서, ROM의 주소는 2^N이고 최대 비트 폭(bit width)은 '$w_{n,m}$의 비트 폭 + $\log_2 N$'이므로, 구현에 필요한 ROM의 크기는 '2^N * ($w_{n,m}$의 비트 폭 + $\log_2 N$)' 비트가 된다. 분산연산이 N에 의해 지수적으로 증가하는 ROM이 필요하기 때문에, PE의 상호 연결이 많은 신경망 구현에는 비효율적이다. 그림 3은 ROM을 이용한 분산연산의 구조이다.

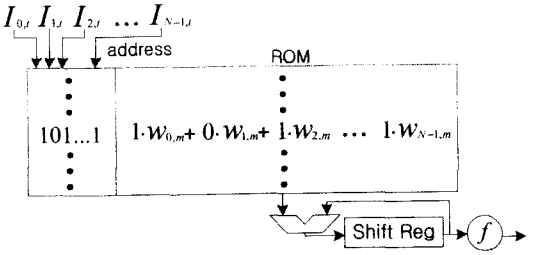


그림 3. 분산연산을 이용한 PE 설계
Fig. 3. An PE design using distributed arithmetic.

입력(I_n)이 아닌 가중치($w_{n,m}$)를 식 (4)과 같이 비트-레벨로 확장하여, 식 (5)와같이 나타내면 계수 값이 '0'인 부분에 대한 덧셈 연산을 소거할 수 있다. 또한, 가중치를 CSD로 표현하면 비트가 '1'인 부분의 수를 $L/2$ 이하로 줄일 수 있으므로 더욱 효율적인 구현이 가능하다.

$$w_{n,m} = \sum_{l=0}^{L-1} w_{n,m,l} \cdot 2^{-l} \tag{4}$$

$$y_m = f(O_m) = f\left(\sum_{n=0}^{N-1}\sum_{l=0}^{L-1}(w_{n,m,l}I_n)2^{-l}\right) \tag{5}$$

그림 4는 CSD 코딩을 이용한 PE 설계의 예를 나타낸다. $w_{n,m}$이 '15'인 경우로 6비트의 2진수와 CSD로 표현하면, 각각 '001111'과 '01000-1'이다. 그림 4처럼, 3개의 덧셈기가 필요한 연산을 CSD 코딩을 이용하여 하나의 감산기로 구현할 수 있으므로 구현에 필요한 하드웨어를 줄일 수 있다.

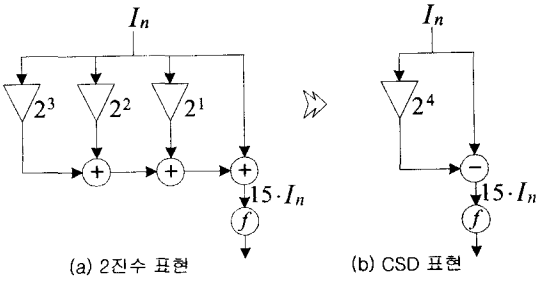


그림 4. CSD 코딩을 이용한 PE 설계
Fig. 4. An PE design using CSD coding.

그림 4와 같은 설계 방법은 적절한 파이프라인(pipeline)설계가 어렵고 입력 값이 워드(word)단위이므로 덧셈연산에 소요되는 시간이 길어지는 단점이 있다.

Ⅲ. 가중치의 비트-레벨 어레이 구조 표현 및 최적화

학습된 신경망을 구현하기 위한 효율적인 방법으로 학습 완료된 PE의 가중치를 비트-레벨 어레이 구조로 표현하고 이를 최적화하여 구현에 필요한 하드웨어를 최소화하고 동작속도를 높이기 위한 효율적인 파이프라인 적용에 관하여 살펴본다.

그림 5는 학습된 신경망의 예를 나타낸다. 입력에서 출력으로 단일 방향인 계층형 네트워크로 입력층 노드와 출력층 노드로 구성되며 $w_{n,m}$은 학습을 통하여 얻어진 가중치다. 그림 5에서 점선 부분에 해당하는 각 PE의 Σ값 즉, O_m을 얻기 위해서는 식 (6)과 같은 계산이 요구된다. 여기서는 비선형함수, f를 제외한 가중치($w_{n,m}$)와 입력(I_n)의 연산결과인 출력(O_m) 관계만을 생각한다. 이 관계를 행렬연산으로 나타내면 식 (7)과 같다.

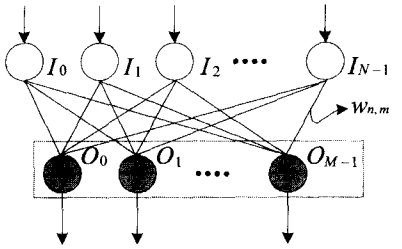


그림 5. 학습된 신경망의 예
Fig. 5. An Example of Pre-trained neural network.

$$O_m = \sum_{n=0}^{N-1} w_{n,m} \cdot I_n \tag{6}$$

$$\begin{bmatrix} O_0 \\ O_1 \\ \cdots \\ O_{M-1} \end{bmatrix} = \begin{bmatrix} w_{0,0} & w_{1,0} & \cdots & w_{N-1,0} \\ w_{0,1} & w_{1,1} & \cdots & w_{N-1,1} \\ \cdots & \cdots & \cdots & \cdots \\ w_{0,M-1} & w_{1,M-1} & \cdots & w_{N-1,M-1} \end{bmatrix} \cdot \begin{bmatrix} I_0 \\ I_1 \\ \cdots \\ I_{N-1} \end{bmatrix} \tag{7}$$

1. 비트-레벨 어레이 구조

식 (6)에서 가중치와 입력을 각각 식 (2)와 식 (4)을 이용하여 비트-레벨로 표현하면 식 (8)과 같다. 식 (8)에서 가중치는 그림 6과 같이 m, n, l의 3개의 인덱스를 갖는 육면체 형태로 확장할 수 있다.

$$\begin{aligned} O_m &= \sum_{l=0}^{L-1}\left(\sum_{n=0}^{N-1}\sum_{t=0}^{T-1}(w_{n,m,l} \cdot I_{n,t})2^{-t}\right)2^{-l} \\ &= \sum_{t=0}^{T-1}\left(\sum_{l=0}^{L-1}\sum_{n=0}^{N-1}(w_{n,m,l} \cdot I_{n,t})2^{-l}\right)2^{-t} \end{aligned} \tag{8}$$

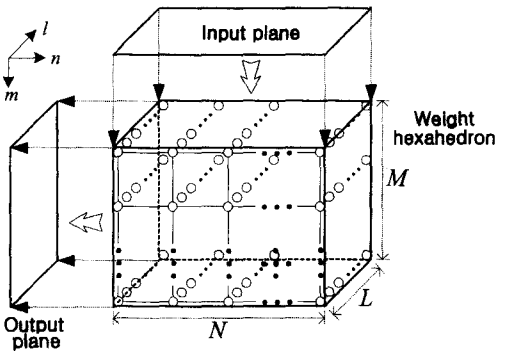


그림 6. 가중치의 비트-레벨 확장
Fig. 6. Bit-level extraction of weight.

그림 7은 병렬성을 고려한 그림 6의 세부 구조이며 회색 노드는 최종 출력을 갖는 노드를 나타낸다. 그림 4의 (a)는 'Input plane'의 형태이다. 인덱스는 n, t를 갖지만 'weight hexahedron'의 입력을 맞추기 위해 l방향으로 확장된다. 각 클럭마다 t는 증가하고, 이때 해당되는 $I_{n,t}$값이 l방향으로 확장되어 n, l의 'Input plane'을 만들어 'weight hexahedron'에 입력된다. LSB(Least Significant Bit)가 가장 먼저 입력되고 MSB(Most Significant Bit)는 마지막 클럭($T-1$)에 입력된다.

그림 7의 (b)는 'weight hexahedron'의 세부 구조이다. 각 노드는 $I_{n,t} \cdot w_{n,m,l}$을 계산하며 이전에 계산된 $I_{n-1,t} \cdot w_{n-1,m,l}$와 덧셈을 수행한다. 따라서 최종 노드에는 $\sum_{n=0}^{N-1}(I_{n,t} \cdot w_{n,m,l})$의 값이 출력되며, 각 노드는 하나의 덧셈기와 AND 게이트로 구성된다.

그림 7의 (c)는 'Output plane'의 구조로 m, l의 인덱스를 갖는다. 'weight hexadedron'의 출력 값을 입력받아 l방향으로 덧셈을 행하게 된다. 이때, l방향은 2^{-l}로 확장되었기 때문에, 노드의 이전 값은 1-비트 오른쪽으로 시프트된 후, 덧셈을 행하게 되며 최종 노드의 출력 값은 $\sum_{l=0}^{L-1}\sum_{n=0}^{N-1}(I_{n,t} \cdot w_{m,n,l})2^{-l}$이 된다. 이 값을 그림 8같이 시프트 덧셈기(shift adder)를 구성하여 클럭에 따라 $I_{n,t}$을 입력시키면, 식 (6)을 계산할 수 있다.

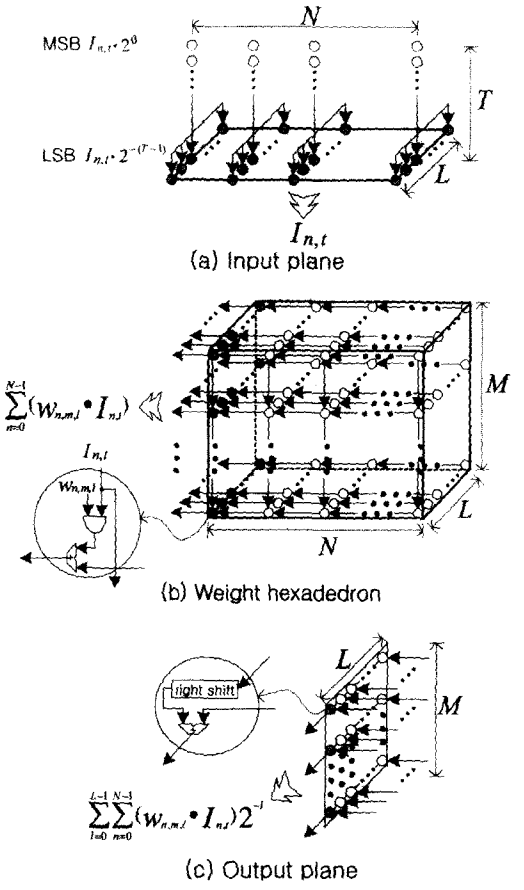


그림 7. 그림 6의 세부 구조
Fig. 7. Detailed architecture of Fig 6.

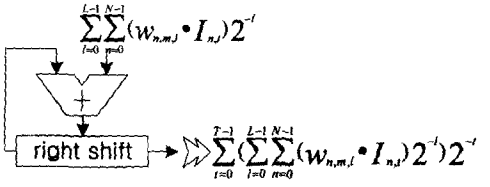


그림 8. 비트-레벨 덧셈을 위한 시프트 덧셈기의 구조
Fig. 8. The architecture of shift adder for bit-level addition.

2. 노드의 최적화

(1) 노드의 소거

그림 7의 (b)에서 하나의 O_m이 출력을 얻기 위해서는 모든 n과 l에 대한 관련연산이 요구된다. 즉, 그림 9와 같이 n, l의 인덱스를 갖는 하나의 'n, l-weight plane'의 모든 노드가 하나의 O_m을 계산하기 위해 연산되어야 한다.

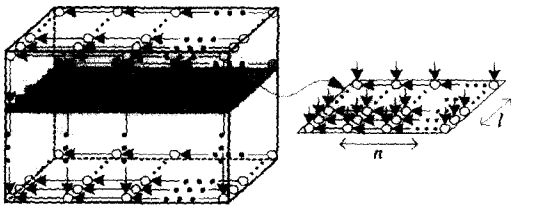


그림 9. n, l-weight plane의 구조
Fig. 9. The architecture of n, m-weight plane.

그림 9에서 각 노드는 하나의 AND 게이트와 덧셈기 그리고 $w_{n,m,l}$을 저장하기 위한 FF(Flip-Flop)로 구성된다. 여기서 $w_{n,m,l}$은 고정되어 있으므로, 그 값이 '0'인 경우에는 $I_{n,t}$에 상관없이 노드의 입력 값을 그대로 패스(pass)하게 된다. 즉, 동작을 하지 않기 때문에 생략이 가능하다는 의미이다. 또한, $w_{n,m,l}$이 '1'인 경우에는 $I_{n,t}$값이 AND의 결과 값이 되므로 각 노드의 AND 게이트와 FF 또한 소거할 수 있다. 그림 10은 노드 소거에 대한 예를 나타낸다. 노드 밑의 숫자는 해당 노드가 갖는 $w_{n,m,l}$ 비트 값이다.

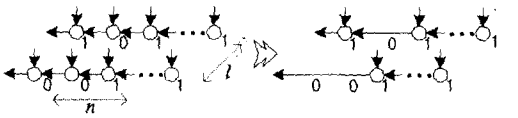


그림 10. 노드의 소거
Fig. 10. Elimination of node.

(2) 가산-트리

하나의 'n, l-weight plane'에서 오른쪽 행 단위로 진행 될수록, 노드에 있는 덧셈기는 이전의 결과 값을 계속 더해야 함으로 더 큰 용량의 덧셈기가 필요할 뿐만 아니라 인접 노드간에 더 많은 신호선(최대 $\lfloor \log_2 N \rfloor$ 개)을 포함해야한다. 또한 임계 패스(critical path)는 N개 노드를 통과하는 시간이 되므로 매우 길어지게 된다. 이러한 문제를 해결하기 위하여 그림 11과 같은 가산-트리를 구성한다. 가산-트리는 'n, l-weight plane'의 한 행에 하나씩 매핑 된다.

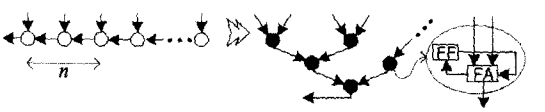


그림 11. 가산-트리의 구조
Fig. 11. Architecture of add-tree.

그림 11과 같은 가산-트리는 각 노드가 하나의 FA(Full-Adder)와 FF로 구성됨으로 모든 노드간은 1-비트 폭으로 연결된다. 또한 임계 패스는 가산트리의 깊이, 즉 $\lfloor \log_2 N \rfloor$ 만큼의 FA가 된다. 각 노드에서 발생하는 캐리는 노드의 FF에 저장되어 다음 클럭에 더해진다. 여기서, 다음 클럭에 들어오는 입력은 2^1만큼 증가된 값이므로, 이전 클럭에 저장된 캐리와 더해져 정확한 연산을 수행하게 된다. 반면에 최종 출력을 얻기 위해서는 각 노드에 저장되어 있는 캐리를 모두 출력해야 함으로 $\log_2 N$만큼의 클럭이 더 필요하다.

(3) 공유 노드 설정

하나의 'n, l-weight plane'에서 그림 11과 같은 가산-트리가 행 단위로 매핑 되므로 M개의 가산-트리가 형성된다. 이 M개의 가산-트리는 동일한 $I_{n,l}$을 입력받기 때문에 $w_{n,m,l}$의 비트 패턴이 동일한 곳은 같은 결과를 출력하게 된다. 그러므로 패턴이 동일한 부분은 하나의 가산-트리로 표현하여 결과 값을 공유하면, 구현에 필요한 노드 수를 줄일 수 있다. 그림 12는 l 값이 다른 두 가중치의 비트 패턴이 동일 할 때, 가산-트리 공유에 대한 예를 나타낸다.

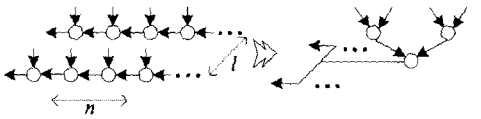


그림 12. 공유 노드 설정
Fig. 12. Setting of sharing node.

결과적으로 학습이 완료되어 고정된 가중치 값을 갖는 신경망은 가중치를 비트-레벨로 표현하고 앞에서 제안한 노드 소거, 가산-트리 표현, 공유 노드 설정을 통하여 연산에 필요한 노드를 최적화 시킬 수 있다. 이렇게 최적화된 노드는 최소의 하드웨어로 구현 될 수 있다.

Ⅳ. 설계 및 시뮬레이션

본 논문에서는 식 (9)과 같은 임의의 가중치 값을 갖는 신경망을 대상으로 설계하였다. 여기서 입력층(I_n)과 출력층(O_m)의 뉴런 수는 각각 4개이고(N, M=4) 가중치와 입력데이터는 양의 정수로만 구성되며 비트 폭은 각각 8(L, T=8)로 가정하였다.

$$\begin{pmatrix} O_0 \\ O_1 \\ O_2 \\ O_3 \end{pmatrix} = \begin{pmatrix} 105 & 75 & 38 & 117 \\ 99 & 65 & 50 & 162 \\ 19 & 87 & 68 & 85 \\ 162 & 43 & 78 & 15 \end{pmatrix} \cdot \begin{pmatrix} I_0 \\ I_1 \\ I_2 \\ I_3 \end{pmatrix} \tag{9}$$

표 1은 비트-레벨 어레이 구조 표현에서부터 최종 공유 노드 설정까지 'weight hexadedron'의 연산에 필요한 노드 수 및 노드를 구성하는데 필요한 요소를 나타낸다. 완전 연결된 계층형 신경망에서 '1. 비트-레벨 어레이 구조'의 노드 수는 '$N*M*L$'으로 128개가 된다. '2. 노드 제거'는 가중치의 이진수 표현에서 값이 '1'인 부분에 해당하는 노드이며 '3. 가산-트리'에서의 노드 수는 '1'인 부분의 연산을 가산-트리로 표현했을 때의 구성되는 노드 수이다. 마지막으로 '4. 공유 노드 설정' 노드 수는 그림 12와 같이 가중치의 비트 패턴이 동일한 부분의 노드를 공유하도록 설정한 후 요구되는 노드의 수이다.

본 논문에서는 표 1에서와 같이, 비트-레벨 어레이 구조 표현에서 128개로 표현된 노드를 최종 17개 노드로 연산되도록 설계하였으며 VHDL(Very high speed Hardware Description Language)로 기술하여 동작을 검증하였다.

표 1. 단계별 설계에 필요한 노드 수 및 노드 구성요소
Table. 1. Required numbers and component of node to design each step.

	128	덧셈기,FF,ANDgate
	57	덧셈기,FF
	41	FF,FA
	17	FF,FA

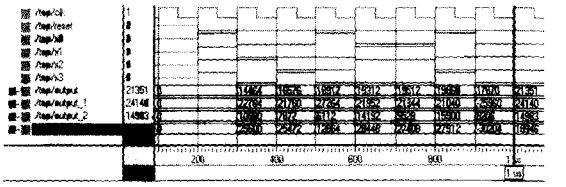


그림 13. 시뮬레이션 결과 파형
Fig. 13. Wave of simulation result.

그림 13은 시뮬레이션 파형으로 입력(I_n)이 각각 73, 49, 51, 69에 대한 결과이다. 1 μs에 정확한 결과 값이 출력됨을 알 수 있다. 그림 14는 Xilinx 9500XL의 9536xlPC44 디바이스로 합성한 그림이다.

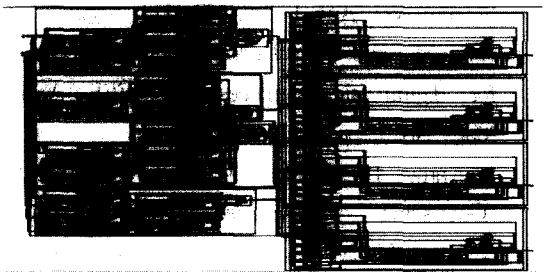


그림 14. Xilinx 9500XL을 이용한 합성 결과
Fig. 14. Result of synthesis using Xilinx 9500XL.

표 2. 'Arrival time'과 'Required clock'의 비교
Table. 2. Comparison 'Arrival time' and 'Required clock'.

	47.0	1 (1)
	41.5	8 (T)
	42.0	1 (1)
	25.5	8 (T)
	28.5	*10* (T+ $\log_2 N$)

표 3. Xilinx 9500XL 합성 결과 비교
Table. 3. Comparison of synthesis result using Xilinx 9500XL.

	17438	0	160	72
	7822	480	163	72
	5208	0	32	72
	656	106	6	72
	573	*81*	*6*	*72*

표 2는 임계경로(critical path)에 소요되는 시간인 'Arrival time'과 최종 출력을 얻기 위해 필요한 클럭수인 'Required clock'에 대한 비교이다. 'Combinational logic'과 'CSD coding'은 입력데이터가 워드 단위이므로 'Required clock'이 1인 반면, 'Sequential logic (MAC)'과 'Distributed arithmetic'은 비트-시리얼 (bit-serial) 입력이므로 8(T)이다. 본 논문에서 제안한 설계방법은 가산-트리에 저장된 캐리의 출력 값까지 모두 출력해야 하므로 2클럭($\log_2 N$)이 더 필요하다. 그러나, 결과 값의 'Arrival time'이 빠르기 때문에 높은 주파수로 동작이 가능하다.

표 3은 Xilinx 9500XL로 합성한 결과 비교이다. 'Combinational logic'과 'Sequential logic(MAC)'은 입력단자가 두 개로 I와 w을 각각 입력받고, 'CSD coding', 'Distributed arithmetic', 본 논문에서 제안한 방법은 입력단자가 하나로, w은 하드웨어상에 구현 되고 I만 입력받는다. 여기서 'PIs'의 차이는 입력 데이터 수 및 직/병렬 입력 형태의 차이로 생성되는 입력 버퍼들을 나타낸다.

표 3에서 알 수 있듯이, 본 논문에서 제안한 방법은 구현에 필요한 'Area(gates)' 및 'FFs'가 다른 설계 방법에 비교하여 매우 우수함을 알 수 있다.

본 논문에서 제안한 설계방법은 하나의 입력 데이터를 처리하는데 필요한 'Required clock'은 다소 길어지지만 'Area' 및 'FF'는 매우 효과적으로 줄일 수 있다. 특히, 제안한 방법은 계산되어야 하는 PE의 수가 많아질수록 공유 노드 설정 확률도 높아지기 때문에 보다 효율적인 하드웨어 구현이 가능하며, 큰 신경망 모델에 적용하는 경우 소규모 칩으로 구현이 가능하다.

V. 결 론

본 논문은 학습된 신경망의 온-칩 구현이 가능한 설계 방법에 대하여 연구하였다. 현재까지 시뮬레이션 레벨의 활용으로만 국한된 다양한 신경망 모델들을 디지털 하드웨어로 구현하기 위하여, 가장 큰 문제점인, PE가 갖는 복잡한 산술 연산 과정을 효율적으로 구현하기 위한 방법을 제안하였다. 이 방법은 고정된 가중치를 비트-레벨 어레이 구조로 표현하고 노드의 소거, 가산-트리 표현, 공유 노드 설정의 단계로 나누어 최적화함으로써, 구현에 필요한 노드 수를 최소화하는 방법이다. 제안한 방법은 합성 결과 다른 알고리즘으로 구현하는 경우 보다도 크기 측면에서 매우 우수하며, 복잡한 신경망의 온-칩 구현이 가능하다는 장점을 갖는다.

참 고 문 헌

[1] Szabo T., Antoni L., Horvath G., Feher B., "A full-parallel digital implementation for pre-

trained NNs", IJCNN 2000, Proc., Vol 2, pp. 49~54, 2000.

[2] Szabo T., Feher B., Horvath G., "Neural network implementation using distributed arithmetic", Proc. KES '98, Vol. 3, pp. 510~518, 1998.

[3] Manferd Glesner, Werner Pochmuller, "Neurocomputers, an overview of neural networks in VLSI", Neural Computing, Chapman & Hall, 1994.

[4] James-Roxby P., Blodget B.A., "Adapting constant multipliers in a neural network implementation", Field-Programmable Custom Computing Machines, IEEE Symposium, pp. 335~336, 2000.

[5] Amin H., Curtis K.M., Hayes-Gill B.R., "Efficient two-dimensional systolic array architecture for multilayer neural network", Electronics Letters, Vol 33, Issue 24, pp 2055~2056, 1997.

[6] Bernie New, "A distributed arithmetic approach to designing scalable DSP chips", EDN Design Feature, Vol. Aug-17, pp. 107~114, 1995.

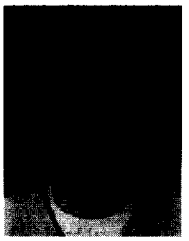
[7] G.K. Ma and F.J. Taylor, "Multiplier policies for digital signal processing", IEEE ASSP Mag., No. 1, pp. 6~10, 1990.

[8] Benyamin D. and Luk W. and Villasenor J., "Optimizing FPGA-based vector product designs", FCCM '99 Proc., pp. 188~197, 1999.

[9] Ma G.-K., Taylor F.J., "Multiplier policies for digital signal processing", ASSP Magazine, Vol 7, Issue 1, pp. 6~20, 1990.

[10] Douglas J. Smith, "HDL Chip Design", Doone Publications, pp. 286~296, 1996.

저 자 소 개

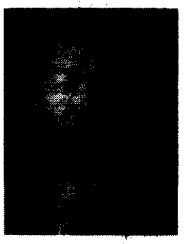


林 國 贊(正會員) 第38卷 SD編 第7號 參照

현재 LG전자(주) WLL단말연구소 주임연구원 재직. <주요관심분야: 병렬처리, FPGA 설계, CDMA>

李 顯 洙(正會員) 第33卷 B編 第10號 參照

현재 경희대학교 전자계산공학과 교수. <주관심분야: VLSI 구조 설계 및 신경망 컴퓨터, 병렬처리>



郭 又 榮(正會員)

1956년 9월 3일생. 1978년 2월 고려대학교 전자공학과 학사. 1992년 8월 고려대학교 산업대학원 전자통신공학과 석사. 1998년 2월 고려대학교 대학원 전자공학과 박사. 1981년~1991년 금성전기 기술연구소 선임연구원. 1992년~1993년 금성통신 연구소 책임연구원. 1994년~1998년 6월 LG전자 미디어통신 연구소 책임연구원. 1998년 7월~1999년 8월 LG정보통신(주) 단말연구소 책임연구원. 1999년 9월~현재 LG전자(주) WLL 단말연구소장, 연구위원/상무. <주관심분야: 초고주파회로, 이동통신, 전파>