제 18 회 한국정보처리학회 추계학술발표대회 논문집 제 9 권 제 2 호 (2002.11)

신경회로망의 고속 구현 방법에 관한 연구

김병근, 김두식, 이상호
한국전자통신연구원 우정기술연구센터

e-mail : iros@etri.re.kr

A Study on Tools for Implementing High-speed Neural Network

Pyong-Kun Kim, Doo-Sik Kim, Sang-Ho Lee
Automatic Sorting and Processing Research Team,
Electronics and Telecommunications Research Institute

요 약

신경회로망은 문자인식, 자동제어 등의 여러 분야에 널리 쓰이는 방식이다. 그러나 신경회로망을 구현하는데는 연산량이 많아서 실시간으로 구현하기에 어려움이 많이 따른다. 본 논문은 신경회로망을 구현하는데 필요한 연산을 살펴보고 그 연산을 구현하는 방법을 비교 분석하였다. 신경회로망을 구현하기 위해 DSP(Digital Signal Processor), PC의 FPU(Floating Point Unit), Intel사의 Pentium 계열 프로세서에서 지원하는 SIMD(Single Instruction Multiple Data) 기술을 사용하여 결과를 비교 분석 하였다. 신경회로망의 핵심인 MLP(Multi Layer Perceptron) 연산에 대해 실험한 결과 SIMD 기술을 이용하는 방법이 다른 방법에 비해 2배이상 좋은 결과를 나타내었다.

1. 서론

신경회로망은 문자인식, 자동제어 등의 여러 분야에 널리 쓰이는 방식이다[2-3]. 그러나 신경회로망은 구현하는데 필요한 연산량이 많아서 실시간으로 구현하기에 어려움이 많다. 본 논문에서는 신경회로망을 구현하는데 쓰이는 연산의 종류에 대하여 알아보고 그 각각의 연산을 구현함에 있어서 PC의 FPU(Floating Point Unit)와, DSP(Digital Signal Processor), 그리고 Intel사의 Pentium 계열 프로세서에서 지원하는 SIMD(Single Instruction Multiple Data) 기술을 이용하여 그 성능을 비교분석한다.

2. MLP의 구조

MLP는 단층 퍼셉트론의 선형 분리 문제를 해결하기 위해 나온 모델이다[1]. 단층 퍼셉트론의 경우 출력 유닛이 선형 분리가 가능한 패턴들만을 분류할 수 있다. 선형 분리 가능이란 패턴 클래스가 하나의 직선에 의해 두개의 영역으로 나뉘어지는 것을 말하는데 XOR(Exclusive-OR) 함수는 선형 분리가 가능하지 않은 패턴분류 문제의 대표적인 예이다.

MLP는 입력층과 출력층 사이에 하나 이상의 중간층이 존재하는 신경망으로 그림 1에 나타낸 것과 같은 계층 구조를 갖는다.

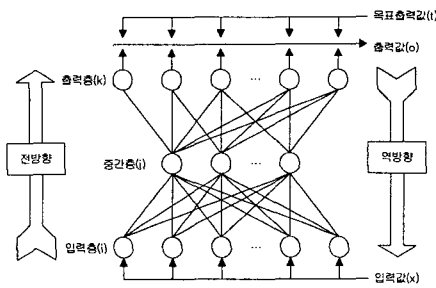

그림 1. MLP의 구조

입력층의 신호는 네트워크의 연결 강도에 의해 변화되어 중간층의 입력값이 된다. 중간층은 이 값을 활성화 함수를 이용하여 자신의 출력값으로 사용한다.

중간층의 출력은 다음 단계의 중간층의 입력값을 구하는데 사용된다. 중간층은 여러 개 둘 수 있으며 문제의 종류에 따라 그 개수가 결정된다. 마지막 층은 출력층이 되며 학습이 진행될수록 출력값은 목표값에 근사하게 된다.

본 연구는 실시간 문자인식 시스템에서 활용되어야 할 신경회로망의 처리속도 향상을 위해 진행되었다. 실시간 문자인식에서 주요한 부분은 전방향 MLP 연산을 빨리 수행하는 것이다. 역방향 연산은 신경망의 학습에 사용되는데 이 부분은 실시간으로 처리 될 필요가 없으므로 수행 속도가 그다지 중요하지 않다.

연산의 측면에서 MLP의 한 계층은 크게 네트워크의 연결 강도에 의한 연산과 활성화 함수의 연산으로 이루어져 있다.

$$net_j = \sum_i w_{ji} x_i$$

수식 1. 네트워크 연산

$$f(net_j) = \frac{1}{1+e^{-net_j}}$$

수식 2. 활성화 함수

네트워크 연산은 부동소수의 곱셈의 합(Sum Of Product)으로 이루어져 있고, 활성화 함수는 지수 함수의 계산과 부동소수의 나눗셈으로 구성된다. 연산을 수행하는 프로세서에서는 주로 부동소수의 곱셈과 나눗셈이 다른 연산에 비해 많은 수행 시간을 필요로 하므로, 수행 시간 측면에서 신경회로망은 네트워크 연산의 SOP, 활성화 함수에서 사용되는 지수 함수, 그리고 활성화 함수에서 사용되는 나눗셈 연산 3부분으로 구분된다.

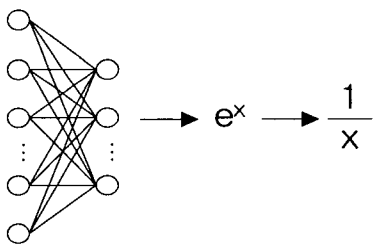

그림 2. MLP 연산 구조

다음의 코드 1은 MLP연산을 C로 구현한 것이다.

```
int i, h;
for( h=0; h<m_nHiddenNodes; h++ )
{
        net = wgt1[h][m_nInNodes];    // bias
        wptr = wgt1[h];
        for( i=0; i<m_nInNodes; i++ )
        {
                net+= (*(wptr++))*(*(input+i));
        }

        net=exp(-net);
        m_pHiddenNodes[h]=1/(1+net);
}
```

코드 3. MLP의 C언어 구현

3. DSP

DSP는 Digital Signal Processor의 약자로 디지털 신호처리용으로 개발된 빠르고 강력한 마이크로 프로세서의 일종이다. DSP는 크게 고정소수 연산용과 부동소수 연산용, 2가지로 나뉘어지는데 본 논문에서는 TI(Texas Instruments)사에서 개발된 부동소수 연산용 TMS320C6711 프로세서를 이용하여 MLP의 연산 수행 시간을 측정하였다[7]. TMS320C6711은 TI사의 제품 중 가장 빠른 부동소수 연산 능력을 가진 프로세서이다. 그 사양은 표 1과 같다.

표 1. C6711 의 사양

Frequency	150MHz
Cycle Time(ns)	6.7ns
Cache	32Kbits L1D Data Cache 32Kbits L1P Program Cache 512Kbits L2 Cache
DMA	16(EDMA)

성능 테스트를 위해 사용할 MLP는 ETRI에서 개발한 한글 주소 인식 시스템(EKAR)에 사용되는 문자인식 모듈의 Forward 함수에서 추출하였다. 입력층의 개수는 100개이며 편의상 중간층의 개수도 100개로 하였다. 시간 측정은 DSP 통합 개발환경에서의 프로파일러를 사용하였다[6]. MLP의 한 계층을 활성화 함수 출력까지 구한 결과는 표 2와 같다.

표 2. DSP 연산 결과

Function	연산 시간(ms)	점유율(%)
SOP	1.01	65.2
Exponential	0.33	21.3
Division	0.21	13.5
총합	1.55	100

4. FPU

PC의 CPU에서는 고정소수 연산과 부동소수 연산을 다르게 처리한다. 고정소수 연산은 일반 명령어와 마찬가지로 ALU(Arithmetic Logic Unit)에서 처리하나 부동소수 연산은 FPU(Floating Point Unit)에서

처리한다. ALU에서의 고정소수 연산은 지연없이 1~2 cycle내에 처리가 되지만 FPU는 스택을 이용하여 연산을 수행하기 때문에 수행 속도가 느리게 된다.

PC에서 작동되는 프로그램의 속도를 측정하는 방법은 내적인 방법과 외적인 방법이 있다. 내적인 방법은 작성하는 프로그램의 소스 코드에 시간 측정을 위한 코드를 넣고 프로그램의 실행후에 그 값을 확인하는 방법이다. 이 방법으로는 C library의 clock 함수를 사용하는 방법과 time 함수를 사용하는 방법, struct_timeb 구조체를 사용하는 방법과 Windows API의 QueryPerformanceCounter 함수를 사용하는 방법, 그리고 SYSTEMTIME 구조체를 사용하는 방법이 있다. 외적인 방법으로는 Intel사에서 나온 VTune 소프트웨어를 사용하는 방법이 있다. 어느 방법을 써도 결과는 비슷하나 VTune 소프트웨어의 경우 전체 프로그램의 처리 시간뿐만 아니라 각 모듈당 처리 시간도 알려주기 때문에 본 논문에서는 VTune을 사용하였다.

VTune의 Sampling 기능을 이용하여 MLP 구현 프로그램의 성능을 살펴보면 표 3과 같다. 사용한 CPU는 Pentium 4 2GHz 프로세서이다.

표 3. FPU 연산 결과

Function	연산 시간(ms)	점유율(%)
SOP	0.0303	69.3
Exponential	0.0073	16.7
Division	0.0061	14.0
총합	0.0437	100

5. SIMD

SIMD(Single Instruction Multiple Data) 기술은 Intel 사에서 대용량 멀티미디어 자료를 처리하기 위해서 나온 기술이다[8]. MMX(Multi Media Extension)라고 명명된 64비트 고정소수 연산 기술이 Pentium 프로세서에서 최초로 적용되었다. MMX는 8 bits 고정소수 8개, 16비트 고정소수 4개, 32비트 고정소수 2개를 동시에 처리할 수 있는 구조이다. 그러나 MMX는 고정소수만을 처리할 수 있을 뿐이고 부동소수는 처리할 수 없었다. 또한 프로세서의 구조상 MMX와 FPU를 동시에 사용할 수 없다는 한계도 단점으로 작용하였다.[4]

Pentium Ⅲ에 적용된 SSE(Streaming SIMD Extensions) 기술은 MMX 기술의 부동소수 확장이었다. SSE 기술을 이용하면 32비트 부동소수 4개를 동시에 처리할 수 있었다. 또한 SSE와 FPU는 프로세서 구조상 분리되어 동시에 사용할 수 있었다.

Pentium 4에 적용된 SSE2기술은 SSE를 발전시킨 기술이다. SSE2 기술을 사용하면 8비트 고정소수 16개를 동시에 처리할 수 있다. 즉 MMX기술에서 동시에 처리할 수 있는 고정소수의 개수를 2배 향상시킨 것이다. 그리고 64비트 부동소수 2개를 동시에 처리할 수 있는 기능도 추가되었다.

한글 주소 인식 시스템에 사용된 신경회로망은 32비트 부동 소수를 사용하여 구현되어 있으므로 본 논문에서는 SSE 기술을 사용하여 동일한 신경회로망을 구현하였다. SIMD를 사용하는 방법으로는 어셈블러를 사용하는 방법과 Intel사 및 Microsoft사에서 제공하는 내재 함수를 사용하는 방법, 2가지가 있다. 성능은 어셈블러를 사용하는 편이 좋지만 사용의 편의성 및 호환성 문제로 인하여 본 논문에서는 내재 함수를 사용하여 MLP를 구현하였다. 구현한 예는 코드 2에 나타나있다.

```
_m128 a, b, s;
float t[4];
float net, *wptr, *inptr;
int i, h;

for( h=0; h<m_nHiddenNodes; h++ )
{
        wptr = (float *)(w1_row+h);
        inptr=input;
        s=_mm_setzero_ps();
        for( i=0; i<m_nInNodes; i+=4 )
        {
                a=_mm_load_ps(wptr);
                b=_mm_load_ps(inptr);
                a=_mm_mul_ps(a, b);
                s=_mm_add_ps(s,a);
                wptr=wptr+4;
                inptr=inptr+4;
                net += (*(wptr++)) * (*(input+i));
        }
        _mm_store_ps(t, s);
        net=w1_row[h].bias; // bias
        net=net+t[0]+t[1]+t[2]+t[3];

        net=exp(-net);
        m_pHiddenNodes[h]=1/(1+net);
}
```

코드 2. MLP의 SIMD 구현

SSE구조에서의 연산은 xmm 레지스터를 이용한다. xmm 레지스터는 128비트로 이루어져 있으며 8개가 있다. 명령어에 따라 xmm 레지스터는 4개의 32 bits 부동소수를 구성하기도 하고 16개의 8 bits 고정소수를 구성하기도 한다. 내재 함수를 사용하는 경우에는 *xmm* 레지스터를 *_m128* 변수형으로 정의해서 사용한다. 코드 2에서 사용된 내재 함수의 기능을 표 4에 설명하였다.

표 4. 내재 함수의 설명

_mm_setzero_ps()	_m128 변수를 0으로 만듦
_mm_load_ps()	float 변수 4개를 _m128 변수로 복사
_mm_mul_ps()	_m128 변수끼리 곱하여 그 결과를 처음의 변수에 저장

_mm_add_ps()	_m128 변수끼리 더하여 그 결과를 처음의 변수에 저장
_mm_store_ps()	_m128 변수를 float 변수에 저장

코드 2를 사용하여 만든 프로그램을 VTune으로 측정한 결과가 표 5에 나타나있다.

표 5. SIMD 연산 결과

Function	연산 시간(ms)	점유율(%)
SOP	0.0137	55.0
Exponential	0.0062	24.9
Division	0.0050	20.1
총합	0.0249	100

6. 비교

앞의 3가지 방법의 결과를 비교하면 아래의 그림3과 같다.

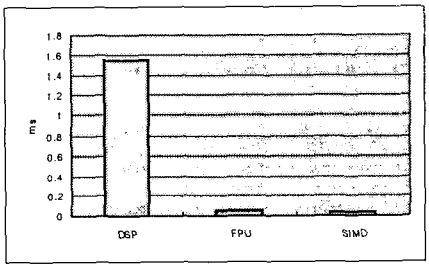

그림 3 연산 시간 비교

각 방법별로 최대 62 배가 차이가 난다. DSP 가 가장 느리고 SIMD 방법이 가장 빠르다. DSP 가 느린 주된 원인은 동작 주파수가 느리기 때문이다. DSP 의 경우 150MHz 로 동작하는 것에 비해 FPU 나 SIMD 는 13 배이상 빠른 2GHz 에서 동작한다. 또한 DSP 는 2 단계 슈퍼스칼라 구조이나 Pentium4 는 이보다 뛰어난 NetBurtst 마이크로 구조로 되어 있다[5]. 또한 SIMD 기술은 이론상 FPU 의 4 배의 성능을 낼 수가 있는데 이러한 것들이 모여 그림 3 의 결과를 나타낸 것이라 할 수 있다.

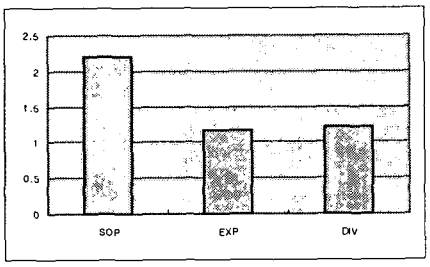

그림 4 SIMD 대한 FPU의 비율

그림 4는 SIMD의 성능을 알아보기 위해 FPU의 결과와의 비를 나타낸 것이다. 지수함수나 나눗셈 연산의 경우보다 SOP 연산에서의 성능 향상이 뚜렷하게 나타난다. 즉 SIMD 방법을 사용함으로써 SOP 연산의 성능을 2배이상 올릴 수 있고 전체적으로는 약 1.7배의 성능 향상을 얻을 수 있다.

7. 결론

MLP를 구현함에 있어서 이용가능한 3가지 방법중에서 SIMD를 사용하는 것이 가장 빠른 결과를 얻을 수 있었다. 연산 속도가 빠르면 실시간 처리가 가능할 뿐만 아니라 MLP의 크기를 크게함으로써 인식율의 향상을 꾀할 수도 있다. 연산 속도의 중요성은 이점에 있다. CPU의 발달은 DSP의 그것을 능가함으로, 당분간은 SIMD 기술을 사용하여 MLP를 구현하는 방법이 가장 빠른 결과를 얻는 방법이라 판단된다.

참고문헌

[1] D.E. Rumelhart, G.E. Hitton and R.J. Williams, "*Learning representations by back-propagating error*," Nature, vol.332, pp.533-536, 1986.

[2] An HMM-MLP hybrid system to recognize handwritten dates Morita, M.; Oliveira, L.S.; Sabourin, R.; Bortolozzi, F.; Suen, C.Y. Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on , 2002 Page(s): 867 -872 vol.1

[3] An hybrid MLP-SVM handwritten digit recognizer Bellili, A.; Gilloux, M.; Gallinari, P. Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on , 2001 Page(s): 28 -32

[4] David Bistry. "The Complete Guide to MMX Technology", McGraw-Hill, 1997

[5] David A. Patterson and John L. Hennessy. "Computer Organization & Design", 2nd Ed. Morgan Kaufmann, 1988 Page(s): 465 -469

[6] "Code Composer User Guide", http://www.ti.com, Ref. No. spru328b

[7] "tms320c6711", http://www.ti.com

[8] "Intel® Architecture Optimization", http://www.intel.com, Ref. No. 24512701