SVM(Support Vector Machines)의 하드웨어 설계 및 구현

진종렬*, 김동성*, 박종서*

*한국항공대학교 컴퓨터공학과

e-mail: {jycpt, dskim, jspark}@mail.hangkong.ac.kr

The Hardware Design and Implementation of the Support Vector Machines

Jong Ryul Jin*, Dong Seong Kim*, Jong Sou Park*

*Department of Computer Engineering, Hankuk Aviation Univ.

요 약

본 논문에서는 SVM의 효과적인 학습 알고리즘인 SMO(Sequetial Minimal Optimization)를 하드웨어적으로 설계하고 구현하는 방법을 제시한다. SVM은 Vapnik에 의한 제안된 기계학습 방법으로 음성인식, 문자인식, BT, 보안 등 다양한 응용분야에서 기존의 신경망보다 우수한 성능을 나타내었다. 그러나 SVM은 계산량이 많아 연산속도가 느려지는 단점을 가진다. 이를 개선하기 위해 본 논문에서는 SVM의 학습 알고리즘인 SMO의 핵심인 지수함수와 실수 연산기를 VHDL로 설계하고 Mentor의 ModelSim을 이용하여 시뮬레이션 하고 Synopsys의 Design Analyzer를 이용하여 합성하였다. 구현된 칩은 시뮬레이션 결과 약 50MHz의 속도로 동작 하며,이는 소프트웨어적으로 구현된 SMO보다 약 10~20배 빠른 성능을 나타내었다.

Ⅰ. 서 론

본 논문에서는 Support Vector Machine(SVM)의 성능 개선을 위한 하드웨어 구현 방안을 제시한다. SVM은 1995년 Vapnik에 의해 제기된 학습 알고리즘으로 복잡한 패턴 인식, 분류 문제에 뛰어난 성능을 발휘하여 문자, 얼굴 및 물체 인식 등의 실제분야에 성공적으로 적용되고 있으며, 특히 이진 분류 문제를 최적으로 해결한다[3]. SVM은 top-down 방식으로 문제를 접근하는 학습방법으로 기존의 bottom-up 방식의 여러 학습 방법에 비해 다소 계산 량이 적어 속도가 빠르며 대용량의 데이터에 대해서도 처리가 가능하다. SVM은 우수한 성능을 보이지만 계산량이 많은 단점을 지니며 이를 개선하기 위한 방법으로 SMO(Sequential Minimal Optimization) 방식이 가장 우수한 성능을 나타낸 것으로 알려져 있다[1]. 그러나 SMO 방식 또한 소프트웨어적으로 동작할 때 그 속도에 한계가 있다. 따라서 본 논문에서는 SMO를 하드웨어적으로 설계하고 구현하였다.

Ⅱ. Support Vector Machines

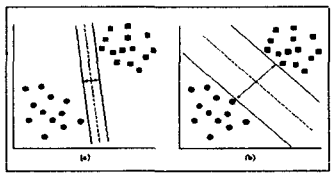

그림 1. 선형 고차원 평면[5]

2. 1 선형 SVM

다음과 같은 Hyperplane을 고려하자.

$$H : y = w \cdot x + b = 0 \qquad (2.1)$$

w는 단위길이를 갖고 Hyper-plane에 직교하는 가중치 벡터를 나타낸다. 이때 원점에서 Hyper-plane과의 수직 거리는 $\frac{|b|}{|w|}$가 된다.

또한 식 (2.1)과 평행인 Hyper-plane은 다음과 같다.

$$H_1 = y = w \cdot x + b = +1 \qquad (2.2)$$

$$H_2 = y = w \cdot x + b = -1 \qquad (2.3)$$

이 두 Hyper-plane 간에는 객체가 존재해서는 안 된다. 또한 식 (2.2), (2.3)은 각각 분리 Hyper-plane에서 가장 가까운 객체를 지나게 되므로 H_1 과 H_2간의 거리(margin)은 $\frac{2}{|w|}$가 되며 선형 SVM의 경우 이 거리를 최대로 하는 분리 Hyper-plane을 구하는 문제로 귀납된다. 즉, 다음과 같은 최적화문제가 된다.

$$\min \frac{w^T w}{2} \text{ Subject to } y_i(w \cdot x_i - b) \geq 1$$

Lagrange multiplier($\alpha_1, \alpha_2, \ldots, \alpha_N \geq 0$)를 도입하면 다음과 같은 최적화 문제가 된다.

$$\mathcal{L}(\mathbf{w},b,\alpha) \equiv \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N}\alpha_i y_i(\mathbf{w}\cdot\mathbf{x}_i - b) + \sum_{i=1}^{N}\alpha_i \quad (2.4)$$

(2.4) 식에 Wolfe dual 공식을 적용하면

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0 \quad (2.5)$$

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \quad (2.6)$$

$\alpha > 0$ 및 식 (2.5), (2.6)에서

$\mathbf{w} = \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i$, $\sum_{i=1}^{N}\alpha_i y_i = 0$ 이 된다.

이 두 식을 (2.4)에 대입하면

$$\mathcal{L}_D \equiv \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \quad (2.7)$$

2. 2 비선형 SVM

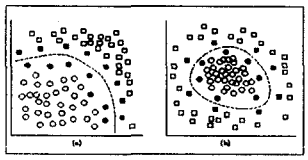

그림 2. 비선형 고차원평면[5]

고차원 공간에서 식 (2.7)는 다음 식과 같이 변형할 수 있다.

$$\mathcal{L}_D \equiv \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j) \quad (2.8)$$

고차원 공간에서 위의 식을 통해 선형적으로 구분이 가능하게 할 수 있다. $\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$와 같이 표현할 수 있으며 $k(\mathbf{x}_i, \mathbf{x}_j)$를 커널 함수라고 하며 커널 함수를 통해 고차원 공간을 2차원의 선형 범주로 구분할 수 있게 하는 함수를 말한다. 커널 함수로는 다음의 Gaussian Kernel 함수가 사용된다.

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-|x_i - x_j|^2/2\sigma^2}. \quad (2.9)$$

2. 3 SMO 알고리즘

SVM은 우수한 성능을 발휘하지만 계산 량이 많은 단점을 가진다. 이를 개선하기 위해 제안된 방법은 Chunking, Decomposition, SMO(Sequential Minimal Optimization) 등이 있으며[1], SMO 방법이 가장 우수한 성능을 발휘하는 것으로 보고 되어 있다. SMO에는 크게 학습 단계와 테스트 단계로 나누어지며 학습단계는 학습 데이터를 이용하여 최적의 고차원 평면을 찾게 되고 이것은 곧 분류기가 된다. 다음은 간단히 C코들 구현된 커널 함수 부분이다[1].

```
float kernel_function(int i1, int i2)
{
        int s;
        s = dot_product_func(i1, i2);
        s += -2;
        s += dot_product_func(i1, i1) +
             dot_product_func(i2, i2);
        return (float)exp(-s/two_sigma_squared)
}
```

그림 3. 커널 함수 정의 부분

II. 설계 및 구현

앞서 유도한 공식을 간단히 살펴보면 HDL(VHDL, Verilog)로 구현하기 위한 핵심 부분은 커널 함수인 가우시안 커널 함수 부분과 실수의 사칙연산부분이다.

3.1 전체 블록도

구현하고자 하는 모듈은 샘플을 학습 시키는 기능의 모듈이다.

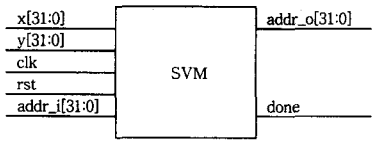

그림 4. 학습 알고리즘 전체블록도

그림 4과 같이 샘플 데이터 x, y 값을 입력하게 되면 SVM 내부 모듈에서 커널 함수 연산 및 실수 계산을 통해 내부 메모리에 학습 데이터가 저장되게 된다. 학습용 module 내부의 주요 핵심 sub-module로는 커널 함수, 실수 사칙 연산기 및 메모리(ram) module이다.

3.2 지수 함수 구현

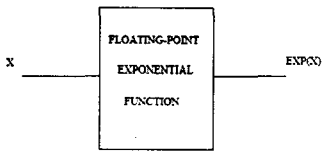

그림 5. 지수함수의 전체 블록도[2]

그림 5에서와 같이 Gaussian Kernel Function을 구현을 위한 지수 함수는 세 개의 주요 부분으로 구분될 수 있다. 입력 값이 주어지면 shifted 지수 함수는 이미 정해진 다항식 근사치를 이용해 예측 가능해진다. 다음으로 입력에 대한 지수 함수는 적절한 공식을 통해 재구성된다. 입력 값 X는 다음과 같이 구성되어 있다[2].

x = (32*m + j)*(log2/32 + (r1 + r2) (3.1)

(3.1)에서 $|r1+r2| \le (\log 2)/64$, m과 j는 정수이며 r1과 r2는 실수이다. 식 (3.1)에 자연 로그 값을 취하면 최종적으로 $\exp(x) = 2^m*(2^{j/32} + 2^{j/32}*p(r))$과 같은 식을 구할 수 있다[2].

p(r) = r + a1*r2 + a2*r3 + ... (Taylor series)

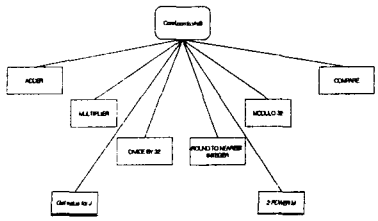

그림 6. 지수함수블록의 내부 구성도

그림 6에서와 같이 내부적으로 덧셈, 곱셈, division by 32 , module 32, comparison 및 powers of 2 등의 모듈로 이루어져 있다. 칩 구현은 현대 0.35um 공정을 이용하였으며 mentor사의 model sim을 이용하여 functional simulation 및 gate level simulation의 입력에 따른 결과 파일은 그림 7과 같다.

그림 7. Modelsim을 이용한 simulation 결과

layout 결과 칩 사진은 아래의 그림 8과 같다. 그림 8은 연산기의 칩 layout 사진으로 총 64개의 입출력 핀을 사용했으며 각각 8개씩의 전원 및 ground 전원을 사용하였다. 약 7만 게이트가 소요되었으며 최대 50 MHz까지 작동이 가능하다.

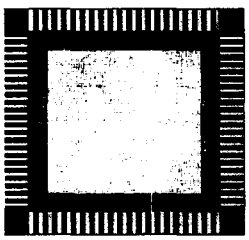

그림 8. 지수함수 layout

3.3 실수 연산기 구현

그림 9는 실수 연산기의 블록 다이어그램을 나타내는 것으로 입력 70, 출력 40개의 핀을 사용하였으며 사용 툴로는 Synopsys사의 디자인 analyzer를 사용하였다.

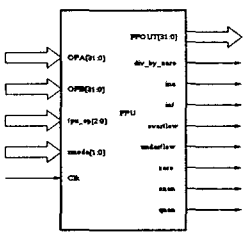

그림 9. 실수연산기 블록다이어그램[6]

그림 10는 그림 9의 내부 블록 구성도를 나타내며 내부의 부모듈간의 연결 정보를 알 수 있다. 실수의 사칙연산과 관계되는 덧셈, 곱셈, 뺄셈, 나눗셈 모듈이 포함되어 있다.

각 모듈은 실수 사칙 연산에 관여하는 모듈을 표현하였으며 디자인 Analyzer를 이용한 실수 연산기의 최상위 모델 합성 결과는 그림 11과 같다.

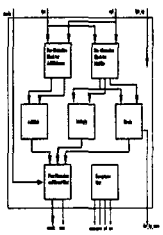

그림 10. 내부블록 구성도[6]

그림 11. 실수연산기 합성 사진

Ⅳ. 결론 및 향후 연구

본 논문에서는 SVM의 효율적인 학습 알고리즘인 SMO을 VHDL로 설계하고 Hynix 0.35 um 공정을 통하여 칩으로 구현하였으며, 구현 결과 소프트웨어적로 동작하는 SVM알고리즘보다 10~20배 빠른 성능을 나타냈다. SVM은 이진분류에 탁월한 성능을 나타내며, 이는 문자, 음식인식, 정보보안, 바이오인포매틱스 등의 다양한 응용분야에서 활용될 수 있을 것으로 예상된다. 향후 연구 방향으로 첫째, 구현된 칩을 사용하여 바이오인포매틱스, 침입탐지 등의 패턴인식문제에 적용할 것이다. 둘째, 지수함수와 실수 연산기의 설계를 최적화하여 성능을 개선하는 방법이 연구되어야 할 것이다.

참고문헌

[1] J.Platt.Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines, Microsoft Research Technical Report MSR-TR-98-14,(1998)

[2] Hung Tien Bui, Bashar Khalaf, and Sofiene Tahar. Table-Driven Floating-Point Exponential Function. October 1998

[3] Vapnik, V.N.(1998), Statistical Learning Theory, John Wiley & Sons.

[4] Chang, C.-C. and C.-J. Lin(2001). LIBSVM: a library for support vector machines. software available at : http://www.csie.ntu.edu.tw/~cjlin/libsvm

[5] Ming-Hsuan Yang. Gentle Guide to Support Vector Machines, http://vison.ai.uiuc.edu/mhyang/svm.html

[6] http://www.ee.pdx.edu/~mperkows/CLASS_VHDL/VHDL_CLASS_2001/Floating/projreport.html