다층 신경망 구현에서의 GPU 사용

정기철[O] 오경수
숭실대학교 정보과학대학 미디어학부
{kcjung, oks}@ssu.ac.kr

GPU for Multi-Layer Perceptron

Keechul Jung[O] Kyoung-Su Oh
School of Media, Department of Information Science, Soongsil University

요약

신경망의 테스트 단계를 실시간으로 처리하기 위해 많은 노력이 있었다. 본 논문은 일반적인 그래픽스 하드웨어를 이용하여 더욱 빠른 신경망을 구현하고, 구현된 시스템을 영상 처리 분야에 적용함으로써 효용성을 검증한다. GPU는 CPU보다 병렬연산에 효과적이다. GPU의 병렬성을 효율적으로 사용하기 위하여, 다수의 신경망 입력벡터와 웨이트벡터를 모아서 많은 내적연산을 하나의 행렬곱 연산으로 대체하였고, 시그모이드와 바이어스 항 덧셈 연산도 픽셀세이더로 병렬 구현하였다. ATI RADEON 9800 XT 보드를 이용하여 구현된 신경망 시스템은 CPU를 사용한 기존의 시스템과 비교하여 정확도의 차이 없이 30배 정도의 속도 향상을 얻을 수 있었다.

1. 서론

기존의 그래픽스 하드웨어 활용과 관련된 연구는 CPU에서 GPU(Graphic Processing Unit)로 전달되는 기하학적 자료의 양을 줄여서 GPU의 부담을 줄이는데 집중되었다[1]. 최근에는 그래픽스 하드웨어가 속도, 프로그래밍 가능성, 가격 등의 면에서 경쟁력을 가져감에 따라, 컴퓨터 그래픽스 뿐만 아니라 계산기하학이나 과학계산 등의 분야에서도 많은 알고리즘을 GPU로 구현하고 있다[2-5].

행렬 연산은 여러 값들에 대해서 같은 연산들이 수행하는 경우(Single-Instruction Multiple-Data)가 많기 때문에 병렬 구조인 GPU를 사용하기에 적합하다. Larson과 McAllister는 다중 텍스처를 이용해서 행렬 곱셈을 하는 방법을 제안하였고[2], Moravanszky는 행렬을 텍스처로 표현하는 구체적인 자료구조와 구현 방법을 제시하였다[4].

또한 최근에는 영상 처리 분야에서도 몇몇 연구 결과가 발표되고 있는데, Yang과 Welch는 상용 GPU를 이용하여 영상 분할과 모폴로지 연산을 수행하였다[5]. 이들은 GPU의 레지스터 결합(register combiner)과 혼합(blending) 연산을 이용하여 컴퓨터비젼 분야에서 기본적인 연산들을 구현하였다.

본 논문은 일반적인 그래픽스 하드웨어를 이용하여 더욱 빠른 신경망을 구현하는 연구이다. 신경망의 학습 단계와 테스트 단계 중, 테스트 단계에서는 실시간 처리가 절실하다. 그러나 입력 데이터가 많아지는 경우에는 실시간 처리가 힘들다. 이러한 이유로 기존에는 신경망을 이용한 실시간 처리를 위해서는 전용 병렬 FPGA 등의 하드웨어를 이용하며 구현했는데, 이는 비용이나 설계 시의 오버헤드를 동반한다[6]. 비록 그래픽스 하드웨어가 신경망 구현을 위한 전용 하드웨어는 아니더라도 이미 일반인에게 많이 보급되어 있는 장비이며, 가격이 저렴하고, 구현할 때 오버헤드가 적어서 대용량 패턴인식이나 영상처리 등의 문제에 적용될 때 많은 장점이 있다.

신경망의 기본적인 연산은 한 노드(node)에서의 웨이트벡터(weight vector)와 입력벡터(input vector) 간의 내적(inner-product) 연산이다. 이러한 신경망의 기본 연산을 GPU 상에서 구현하기 위해서, 많은 입력벡터와 웨이트벡터를 누적함으로써 다수의 내적 연산을 하나의 행렬곱 연산으로 대체해서 GPU의 병렬성의 효과를 극대화할 수 있다. 이와 같이 행렬곱과 활성함수(activation function) 등의 신경망 구현에 필수적인 연산을 GPU 상에서 정점세이더(vectex shader)와 픽셀세이더(pixel shader)를 이용하여 효과적으로 구현한다.

영상처리나 패턴 인식은 입력양이 많은 신경망 응용 분야 중 하나이다. 예를들어 신경망을 이용한 컨벌루션(convolution)을 영상 전체 영역에 수행할 때는 상당한 수행 시간을 필요로 한다. 본 논문에서는 GPU를 이용하여 신경망을 구현함으로써 영상 처리 등의 분야와 같은 대용량 데이터의 신경망 처리 문제를 저가로, 그리고 적은 오버헤드로 해결할 수 있었다

2. 신경망 구조

신경망이라고 불리는 인공신경망(artificial neural network)은 인간 두뇌의 동작 방식을 따라 구현되었다. 다양한 모양의 신경망이 사용되고있지만, 대표적으로 multilayer perceptron, learning vector quantization, radial basis function, hopfield, kohonen 등을 주로 많이 사용한다[7].

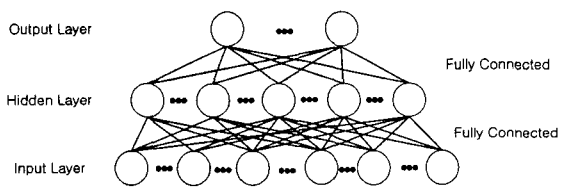


그림 1. 일반적인 MLP의 구조.

본 논문에서 구현하고자하는 신경망은 다층 퍼셉트론(multilayer perceptron:MLP)이다(그림 1). MLP는 1개 이상의 은닉층(hidden layer)을 가지며 다수의 출력 노드를 가질 수 있다. 일반적으로 MLP는 인접한 층(layer)의 노드들이 완전연결(fully-connected)되어 있고, 층의 개수나 각 층의 노드수 등에서 변화가 있을 수 있지만, 기본적으로 각 노드는 웨이트벡터와 해당 노드의 입력벡터의 내적연산을 수행한 후(식 1), 활성함수 연산(식 2)을 수행한다.

$$m_j = \sum w_{ji} x_i + b_j \qquad (1)$$

$$r_j = \left(1+e^{-m_j}\right)^{-1} \quad (2)$$

식 1과 2에서 첨자 j는 출력 노드를 나타내며, i는 j-번째 노드와 연결되어 있는 하위층의 노드를 의미하고, w_{ji}는 j-번째 노드와 i-번째 노드를 연결하는 웨이트, x_i는 입력값, b_j는 j-번째 노드의 바이어스 항(bias term), r_j는 j-번째 노드의 최종 출력값을 의미한다. MLP의 첫번째 은닉층의 노드들부터 이와 같은 연산을 수행하여 차례로 출력층까지 계산한다. MLP 이외의 다른 종류의 신경망들도 기본적으로 각 노드는 MLP의 노드들과 유사한 연산을 수행하기 때문에, 본 논문의 연구 결과는 쉽게 다른 신경망에 적용될 수 있다.

3. GPU 연산

이미 기술한 바와 같이, 신경망의 각 '**노드**'에서의 내적 연산은 입력벡터와 웨이트벡터를 축적함으로써 행렬의 곱 연산으로 변환할 수 있다. 그리고 바이어스 항의 덧셈과 시그모이드 연산들은 픽셀세이더로 쉽게 구현가능하다. 이로써 신경망의 각 '**층**'의 연산은 다음과 같이 기술할 수 있다:

$$W=\begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1N} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2N} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ w_{M1} & w_{M2} & w_{M3} & \cdots & w_{MN} \end{bmatrix}=\begin{bmatrix} W_1 \\ W_2 \\ \cdots \\ W_M \end{bmatrix} \quad X=\begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_L \\ x_1 & x_2 & x_3 & \cdots & x_L \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_1 & x_2 & x_3 & \cdots & x_L \end{bmatrix}=\begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_L \end{bmatrix} \quad B=\begin{bmatrix} b_1 & b_1 & b_1 & \cdots & b_1 \\ b_2 & b_2 & b_2 & \cdots & b_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ b_M & b_M & b_M & \cdots & b_M \end{bmatrix}$$

$$M=W\times X+B=\begin{bmatrix} W_1\cdot X_1 & W_1\cdot X_2 & W_1\cdot X_3 & \cdots & W_1\cdot X_N \\ W_2\cdot X_1 & W_2\cdot X_2 & W_2\cdot X_3 & \cdots & W_2\cdot X_N \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ W_M\cdot X_1 & W_M\cdot X_2 & W_M\cdot X_3 & \cdots & W_M\cdot X_N \end{bmatrix}+\begin{bmatrix} b_1 & b_1 & b_1 & \cdots & b_1 \\ b_2 & b_2 & b_2 & \cdots & b_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ b_M & b_M & b_M & \cdots & b_M \end{bmatrix}=\begin{bmatrix} m_{11} & m_{11} & m_{11} & \cdots & m_{1L} \\ m_{12} & m_{12} & m_{12} & \cdots & m_{12} \\ m_{13} & m_{13} & m_{13} & \cdots & m_{13} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ m_{M1} & m_{12} & m_{12} & \cdots & m_{1L} \end{bmatrix}$$

$$R=sigmoid(M)=\begin{bmatrix} (1+e^{-m_{11}})^{-1} & (1+e^{-m_{12}})^{-1} & (1+e^{-m_{13}})^{-1} & \cdots & (1+e^{-m_{1L}})^{-1} \\ (1+e^{-m_{21}})^{-1} & (1+e^{-m_{22}})^{-1} & (1+e^{-m_{23}})^{-1} & \cdots & (1+e^{-m_{2L}})^{-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ (1+e^{-m_{M1}})^{-1} & (1+e^{-m_{M2}})^{-1} & (1+e^{-m_{M3}})^{-1} & \cdots & (1+e^{-m_{ML}})^{-1} \end{bmatrix}$$

여기서 w_{ij}는 출력층의 i-번째 노드와 입력층의 j-번째 노드 사이의 웨이트, M은 출력층의 노드 개수, N은 입력층의 노드 개수, x_{ij}는 j-번째 입력벡터의 i-번째 특징값, b_i는 전체 L개의 입력벡터 중 i-번째 입력 노드를 위한 바이어스 항을 의미한다. 최종 연산 결과인 R의 i행 j열 원소는 j-번째 입력벡터에 대한 i-번째 출력 노드의 값이다. 위의 연산은 행렬 곱 연산, 바이어스 항 덧셈연산, 활성함수인 시그모이드 연산의 순으로 수행된다.

그림 2는 GPU 내에서의 행렬 곱을 설명한다[4]. 두개의 피곱셈 행렬을 *texture W*와 *texture X*로 변환하고 렌더링 연산을 수행한다. 행렬 곱의 출력을 위해 전체 화면을 덮는 사각형을 렌더링한다. 정점세이더는 사각형의 각 정점의 위치 외에 텍스처 좌표를 출력하는데, 각각의 정점은 두개의 텍스처 좌표를 가지고 있다. 그 중 하나는 *texture W* 의 행(row) 좌표이고, 또 다른 하나는 *texture X*의 열(column) 좌표이다. 예를 들어, 왼쪽 상단의 좌표는 *texture W*의 첫번째 행의 텍스처 좌표와 *texture X*의 첫번째 열의 텍스처 좌표를 가지며, 오른쪽 상단의 좌표는 *texture W*의 첫번째 행의 텍스처 좌표와 *texture X*의 마지막 열의 텍스처 좌표를 가지는 식이다. 정점세이더의 결과로써, 각 픽셀*(i, j)*들은 *W*의 i-번째 행과 *X*의 j-번째 열에 해당하는 텍스처 좌표를 가진다. 픽셀세이더는 텍스처 좌표에 의한 *texture W*의 행과 *texture X*의 열 사이의 내적연산을 수행한다. 행렬곱의 결과는 *texture W*×*X*에 저장된다.

신경망에 둘 이상의 은닉층이 존재한다면, 위의 연산 과정을 각 층 마다 반복 수행한다: 이전 층의 결과는 렌더링도 가능하고 텍스처로도 사용할 수 있는 렌더 타겟 텍스처(render target texture) 형태로 저장되며 이는 다음 층의 입력으로 사용된다. 신경망이 다수개의 은닉층을 지니더라도 텍스처 생성 후에는 위의 모든 과정이 CPU의 개입 없이 GPU만으로 수행 할 수 있다.

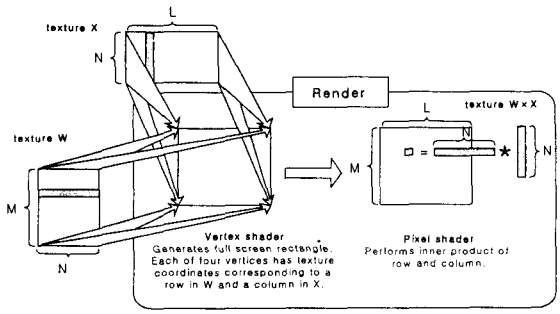


그림 2. GPU를 이용한 행렬 곱 연산의 개괄도.

4. 영상처리 분야에 적용

본 장에서는 본 논문에서 제안한 GPU 기반 신경망을 '영상 내의 문자 추출' 분야에 적용함으로써 유용성을 검증한다. 본 논문에서 사용하는 MLP는 41개의 입력 노드, 30개의 노드로 구성된 1개의 은닉층, 1개의 출력 노드로 구성되며, 인접층의 노드들은 모두 연결되어있다.

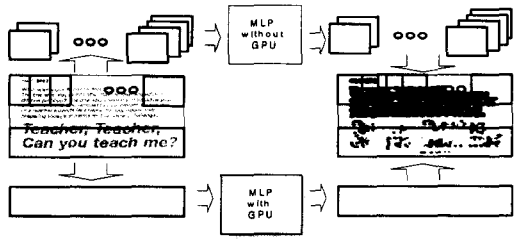
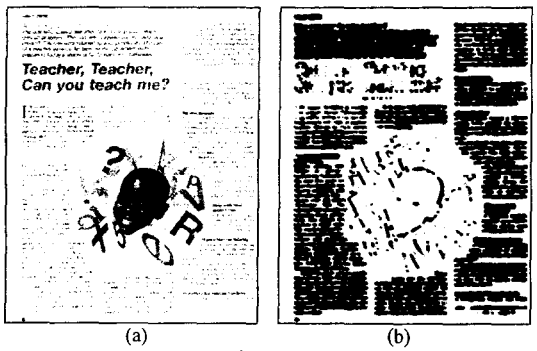


그림 3. GPU를 이용한 신경망 처리.

본 연구에서는 신경망을 사용함으로써 다양한 환경에 적응성을 지니는 문자 추출기를 자동으로 생성할 수 있었다. 그러나 소프트웨어로 구현한 신경망은 앞에서 기술한 바와 같이, 테스트 단계에서의 많은 계산량으로 인한 느린 수행속도의 문제점이 있다. GPU의 성능을 최대한 높이기 위해서 공간적으로 연속된 화소들에 대한 신경망의 입력벡터를 누적함으로써 신경망의 입력벡터값들을 2차원 텍스처로 구성한다. 신경망의 웨이트벡터와 입력벡터를 텍스처로 구성하고, 신경망의 연산을 GPU의 렌더링 연산으로 구현함에 따라, 결과적으로 GPU의 병렬성을 이용함으로써 신경망의 수행 속도를 현저하게 향상시킬 수 있었다. 그림 3은 GPU 기반 신경망을 이용한 문자 영역 추출 방법을 직관적으로 보인다. 왼쪽의 입력영상에 해당하는 문자 추출 결과가 오른쪽의 영상이며, 검은 화소가 문자 영역을 의미한다. GPU를 이용함으로써 여러 번의 문자추출 작업을 한번으로 병합하여 수행할 수 있음을 개념적으로 보인다.

본 실험에서는 펜티엄IV, 메모리 512M Bytes, ATI RADEON 9800 XT를 사용하였다. 그림 4는 1152×1546 크기의 컬러 스캔 문서에 대한 실험 결과 예이다. 컬러영상을

그레이영상으로 변환한 후, 신경망을 이용하여 문자 영역을 검출하였다. 최종 결과인 그림 4(b)는 CPU만을 이용한 결과와 일치하였으며 약 30배 정도 속도가 향상되었다.



(a) (b)

그림 4. 실험 결과.

표 1. 각 계산 단위 별 수행 시간

	Texture Creation	Matrix Multiplication	Sigmoid	Total
GPU	0.18	0.14	0.06	0.38
CPU	11.743			

표 1에서 GPU 기반 신경망과 CPU만을 사용한 신경망의 최소 수행시간을 비교하였다. GPU로 구현하였을 경우 약 30배 정도의 속도 향상을 얻을 수 있었다. GPU를 이용하였을 경우, 전체 수행시간의 1/2 정도를 텍스처 생성(Texture Creation) 단계에서 사용하였는데, 이는 GPU를 이용하면서 생기는 오버헤드이다.

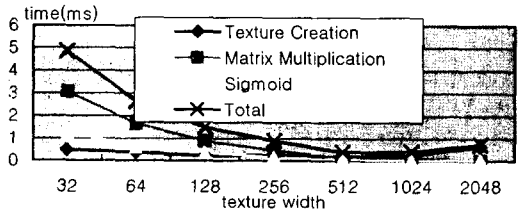


그림 5. 텍스처 너비 변화에 따른 수행시간.

그림 5는 제안된 방법에서 텍스처의 너비에 따른 수행시간 측정 결과이다. 여기서 텍스처의 너비는 3장에서 설명한 *texture X*의 너비이다. *texture X*는 GPU가 한번에 처리하는 입력의 개수를 결정하게 되는데 *texture X*의 너비가 작으면 입력들을 작게 나누어서 여러 번 처리하게 되고, 크면 입력들을 모아서 한꺼번에 처리하게 된다. 전체 계산 시간은 텍스처의 너비가 512일 경우 가장 작았다. 텍스처의 너비가 512보다 작을 경우에는 행렬 곱과 시그모이드 연산에 걸리는 시간이 많고 텍스처가 512보다 클 경우에는 텍스처 생성에 걸리는 시간이 상대적으로 많다.

행렬 곱과 시그모이드에 걸리는 시간은 텍스처의 너비가 커질수록 작아지는데 텍스처의 크기가 커질수록 렌더링의 횟수가 작아지므로 렌더링을 시작하는데 걸리는 일정한 준비시간이 작아지기 때문이다.

그림 6은 텍스처 너비에 따른 텍스처 생성 시간의 변화를 분석한 그래프이다. 배열로부터 시스템 메모리의 텍스처로 복사하는데 걸리는 시간은 텍스처 너비가 512보다 작을 때는 별다른 패턴을 보이지 않고 512보다 커지면 급격히 증가한다. 실험에 사용한 텍스처는 한 픽셀당 16바이트(sizeof(float)*4)를 사용하는데 텍스처의 너비가 512일 경우 텍스처에 한 줄에 필요한 메모리가 8킬로 바이트가 된다. 이는 Pentium IV의 L1 캐쉬(cache)의 크기와 일치한다. 그러므로 텍스처의 너비가 512이상 일 경우 system to system copy시간이 증가하는 것은 텍스처의 너비가 캐시의 크기보다 크기 때문이라고 판단된다. 시스템 메모리에서 비디오메모리로의 복사연산에 걸리는 시간은 텍스처 너비가 커짐에 따라 감소를 보인다.

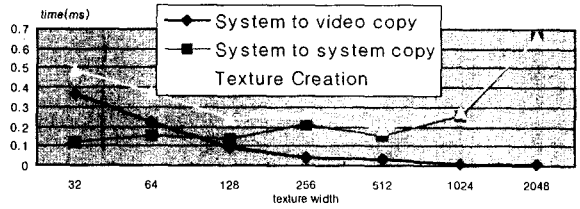


그림 6. 텍스처 너비 변화에 따른 텍스처 생성 시간.

GPU에 의해서 행해지는 연산인 행렬 곱과 시그모이드 연산, 시스템 메모리에서 비디오 메모리로의 복사는 모두 텍스처 너비가 커질수록 빨라졌고, CPU만 관여하는 시스템 메모리간의 복사연산은 텍스처의 너비가 512보다 작을 때는 변화가 거의 없다가 그 이상이 될 경우에는 커졌다. GPU나 CPU관련 연산의 복잡도와 시스템 사양에 따라 적절한 텍스처 너비를 선택해야 한다.

5. 결론

본 논문에서는 일반적인 그래픽스 하드웨어를 이용하여 신경망을 구현하였으며 이를 영상처리 분야에 적용하여 실효성을 검증하였다. GPU를 이용한 신경망 구현은 가격이 저렴하고, 소프트웨어적으로 구현할 때 오버헤드가 적어서 많은 잇점이 있다. 이와 같은 연구결과는 영상 처리, 패턴인식 등의 분야와 같은 대용량 데이터의 신경망 처리 문제를 저가로 그리고 적은 오버헤드로 해결하고자하는 분야에 활용될 수 있다. 향후 GPU를 이용한 신경망 학습과 CPU와 GPU의 병렬처리 극대화, 여러 컴퓨터의 CPU와 GPU를 사용한 신경망 구현등에 대한 연구를 할 예정이다.

참고 논문

1. Kyoung-Su Oh, Byeong-Seok Shin, and Yeong Gil Shin, "Mobility Culling-An Efficient Rendering Algorithm Using Temporal Coherence," The Journal of Visualization and Computer Animation, Volume 12, Issue 3, pp.159-166, 2001.
2. E. Scott Larsen and David McAllister, "Fast Matrix Multiplies using Graphics Hardware," Proceedings of the 2001 ACM/IEEE Conference on Supercomputing, Denver, Colorado, 2001.
3. Dinesh Manocha, "Interactive Geometric & Scientific Computations using Graphics Hardware," SIGGRAPH 2003 Tutorial Course #11.
4. Adam Moravanszky, "Dense Matrix Algebra on the GPU," ShaderX 2 Programming, Wordware, 2003.
5. Ruigang Yang and Greg Welch. "Fast Image Segmentation and Smoothing Using Commodity Graphics Hardware", the journal of graphics tools, special issue on "Hardware-Accelerated Rendering Techniques", 2003.
6. Jihan Zhu and Peter Sutton, "FPGA Implementation of Neural Networks – a Survey of a Decade of Progress", Proceedings of 13th International Conference on Field Programmable Logic and Applications (FPL 2003), Lisbon, Sep 2003.
7. Haykin, Neural Networks: a comprehensive foundation, Prentice Hall, 1999.
8. K. Jung, K. I. Kim, and A. K. Jain, "Text Information Extraction in Images and Video: A Survey," International Journal of Pattern Recognition, To be published.
9. K. Jung, "Neural Network-based Text Location in Color Images," Pattern Recognition Letters, Vol. 22, No. 14, pp 1503-1515, 2001.