인공지능 하드웨어 설계 및 최적화 기술

김원종
ETRI
SW-SoC개방형플랫폼실

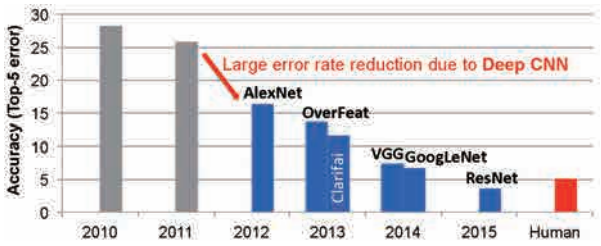
이상헌
(주)디퍼아이

Ⅰ. 서 론

2012년에 Alexnet이 ImageNet 대회에서 처음으로 딥러닝 기술로 1등에 오른 후 3년 만에 〈그림 1〉에서와 같이 2015년에 인간의 이미지 인식 에러율인 5%를 딥러닝 기술이 앞서게 되었다[1-2].

하지만, 이 대회에서는 인식 정확도만 개선을 목표로 하였으며, 알고리즘의 복잡도는 크게 고려하지 않았다. 따라서 개발된 알고리즘을 실제 환경에서 사용하거나 상용화를 하기 위해서는 적정 수준의 알고리즘을 새롭게 개발하건, 하드웨어 가속 기술을 적용할 필요가 있다. 그럼에도 불구하고 복잡한 알고리즘을 이용한 학습의 효율성을 위해서 2012년의 AlexNet에서부터 GPU (Graphics Processing Unit)를 사용하기 시작하였으며, 이후 많은 수의 참가자가 GPU를 사용하게 되었다. GPU 사용에도 한계를 느끼게 되면서 뉴럴네트워크 모델 연구에 대한 관심은 하드웨어 설계의 효율성 및 실제 활용하기 위한 최적화 기술 개발로 관심이 빠르게 옮겨지고 있다.

특히 클라우드 기반의 인공지능 서비스를 제공하는 글로벌 기업들은 CPU와 GPU로 구성된 데이터 센터의 에너지 효율 향상을 위하여 FPGA를 이용한 하드웨어 가속 기술을 도입하고 있다. MS는 BrainWave 프로젝트를 통해서 DNN (Depp Neural Network) 엔진을 구동하는



〈그림 1〉 ImageNet 대회 이미지 인식 결과[2]

FPGA(Field Programmable Gate Array) 하드웨어와 고성능 분산 시스템 아키텍처 및 학습된 인공지능 모델을 배포하기 위한 컴파일러와 런타임으로 구성된 시스템을 개발하여 2017년 HotChips에서 발표하였다[34]. Amazon은 EC2 F1 인스턴스라는 명칭의 FPGA 가속 서비스를 개발하여 2016년부터 제공하고 있다[35].

본고에서는 인공지능 하드웨어 설계와 관련하여 최근에 연구되고 있는 기술을 소개하고자 하며, 특히 많은 연산을 필요로 하는 DNN을 위한 하드웨어 기술을 중점적으로 다루고자 한다.

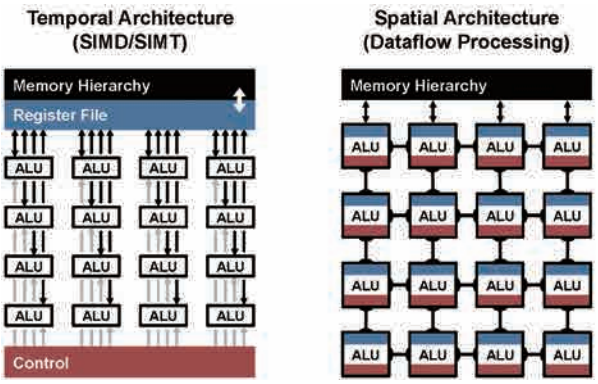
Ⅱ. DNN 하드웨어 설계 기술

최근 인공지능 기술 개발에서 DNN의 뛰어난 효과로 인해 DNN 연산을 효과적으로 처리하기 위해 Intel Fathom, Nvidia Jetson TX1/2, 같은 다양한 전용 하드웨어 설계 기술들이 발표되고 있다.[3-5]

이러한 전용 솔루션들이 어떻게 효과적인 DNN 연산을 수행하는지 관련 기술에 대해서 살펴보고자 한다.

1. 구조적 차이에 따른 연산처리 방법

최근의 DNN 구조는 Convolution이나 Fully Connected 연산과 같이 병렬처리에 용이한 MAC (multiply-and- accumulate) 연산기를 필요로 한다. 이러한 연산처리 방식은 〈그림 2〉와 같이 시간적(temporal) 병렬 처리 구조와 공간적(spatial) 병렬 처리 구조로 구별해서 처리할 수 있다.



〈그림 2〉 시간적 병렬 처리 및 공간적 병렬 처리 구조

시간적 구조: 대부분 CPU 또는 GPU 구조에서 SIMD (Single Instruction, Multiple Data) 또는 SIMT (Single Instruction, Multiple Threads) 와 같은 병렬처리 기술로 구현될 수 있다. 이러한 구조는 중앙 집중식 제어를 통해 연산기를 제어하는 방식이 일반적이다. 또한 연산기는 메모리 계층구조를 통해 상호 데이터 통신이 가능하며, 연산기 상호간의 통신은 불가능하다. 이러한 구조는 곱셈수를 줄이는 방식으로 연산 최적화가 가능하다.

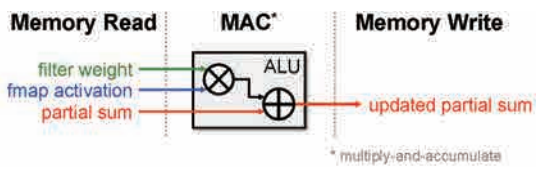
공간적 구조: 연산기 사이에 데이터를 전달할 수 있는 데이터 플로우 처리방식으로, 연산기 사이에 데이터가 전달될 수 있도록 체인 형태로 구성된다. 연산기는 자체적인 제어 회로와 스크래치 패드(scratch pad) 또는 레지스터 파일과 같은 로컬 메모리를 포함하기도 한다. 이러한 구조는 FPGA나 ASIC(Application-Specific Integrated Circuit)과 같은 전용 하드웨어 솔루션에서 많이 활용되고 있다. 이러한 구조는 데이터 플로우 방식으로 인한 메모리 계층을 단순화시킴으로써 에너지 효율을 높일 수 있는 장점이 있다.

본고에서는 CPU나 GPU보다 전용 하드웨어 설계에 많이 이용되는 공간적 병렬화 구조를 중심으로 알아보고자 한다.

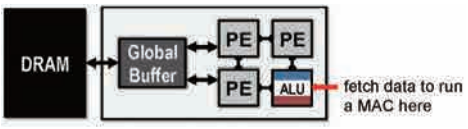
2. 효율적인 데이터플로우 구조

DNN 구조에 있어서 연산의 병목은 메모리 액세스 부분이다. MAC 연산은 〈그림 3〉에서와 같이 3번의 메모리 읽기와 1번의 메모리 쓰기를 필요로 한다. 모든 데이터가 외부의 DRAM에 있는 경우에는 DRAM에 대한 액세스가 병목으로 작용하게 된다.

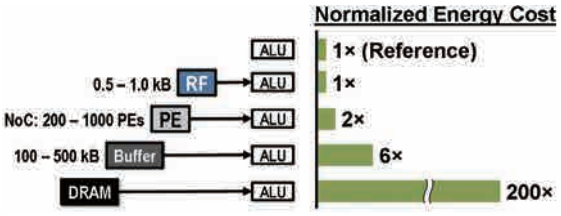
이러한 메모리 액세스를 효율적으로 처리하기 위하여 연산기 주변에 〈그림 4〉와 같이 계층적 메모리 구조를 사용하기도 한다. PE(Processing Element)는 내부에 로컬 메모리로 RF(Register File)을 가지고 있으면서 ALU와



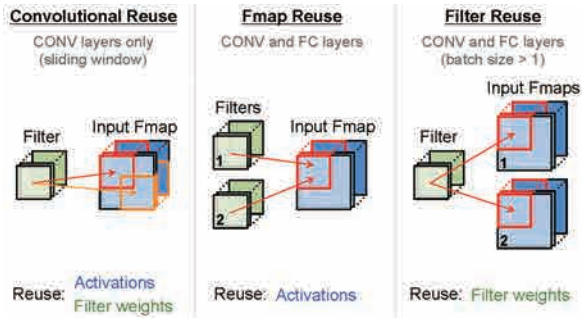
〈그림 3〉 MAC 연산에서의 메모리 읽기와 쓰기



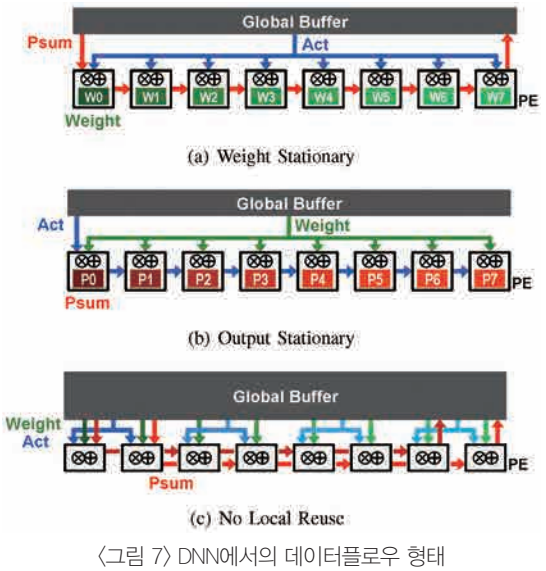
〈그림 4〉 계층적 메모리 구조 개념



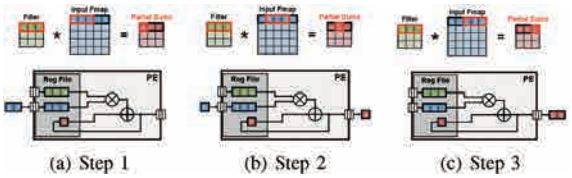
〈그림 5〉 계층적 메모리의 상대적 에너지 비교



〈그림 6〉 DNN에서의 데이터 재사용



〈그림 7〉 DNN에서의 데이터플로우 형태



〈그림 8〉 Row Stationary DNN 데이터플로우 구조의 PE내에서의 1-D 콘볼루션 재사용

직접 데이터를 주고받는다.

계층적 메모리 구조 각각의 데이터 액세스에서의 상대적인 에너지 소비를 비교하면 〈그림 5〉와 같으며, 이는 처리 성능에도 유사하게 영향을 주게 된다[33].

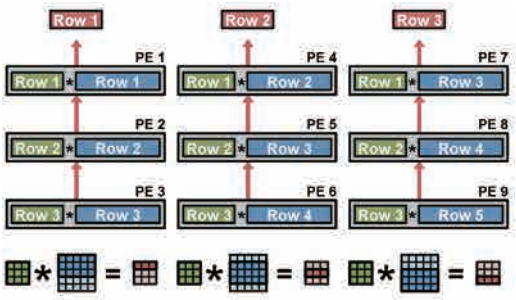
이러한 데이터 액세스는 데이터 메모리 사용의 효율성을 위하여 〈그림 6〉에서와 같이 3가지 형태의 데이터 재사용 가능한 구조를 생각해볼 수 있다[33].

컨볼루션 재사용에서는 입력 특징 맵 활성화 데이터와 필터 웨이트가 하나의 채널에서 재사용된다. 특징맵 재사용에서는 여러 필터가 하나의 특징 맵에 사용되므로 입력 특징 맵 활성화 데이터가 여러 번 사용된다. 필터 재사용에서는 여러 입력 특징 맵이 한번에 처리되므로, 동일한 필터 값들이 입력 특징 맵에 여러 번 이용된다. 이러한 로컬 메모리 사용을 최대화하고, 외부 DRAM에의 액세스를 최소화함으로써 DNN 처리 효율을 크게 향상시킬 수 있다.

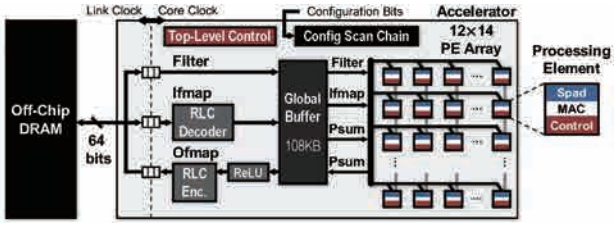
데이터를 다루는 특성에 따라서 데이터 플로우는 〈그림 12〉에서와 같은 3가지와 Row Stationary 구조로 구분할 수 있다[33]. Weight Stationary 구조는 PE에서 RF로부터 가중치 액세스를 최대화함으로써 가중치 읽기에서 소비되는 에너지를 최소화하는 방법으로, DRAM에서 각 PE의 RF에 읽어 들인 가중치 값이 고정된다. Output Stationary 구조는 부분 합을 쓰고 읽는 과정에서의 에너지 소비를 최소화하는 구조로, 부분합을 누적하는 형태로 RF에 유지하는 방법을 사용한다. PE에 로컬 메모리를 가지는 것은 에너지 면에서는 효율적이지만, 면적 관점에서는 비효율적일 수 있으므로, 〈그림 7(c)〉에서와 같이 내부에 공용의 글로벌 버퍼 메모리를 두고 외부의 DRAM 액세스만을 최소화하는 구조를 사용할 수도 있다.

전반적인 에너지 효율 향상을 위하여 RF에서 가중치와 픽셀값 및 부분합 등의 대부분의 데이터를 재사용하는 Row Stationary 구조도 제안되었다[33]. 〈그림 8〉에서와 같이 1-D Row 컨볼루션을 각 PE에 할당하고, 이들을 조합하여 〈그림 9〉에서와 같이 2-D 컨볼루션을 재사용할 수 있는 구조를 제안하였다.

이러한 Row Stationary 구조를 적용한 결과는 [24]에



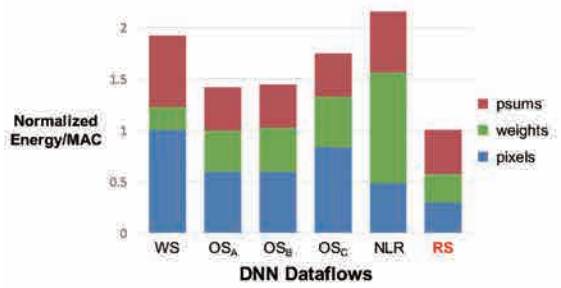
〈그림 9〉 Row Stationary DNN 데이터플로우 구조의 공간적 배열 내에서의 2-D 콘볼루션 재사용



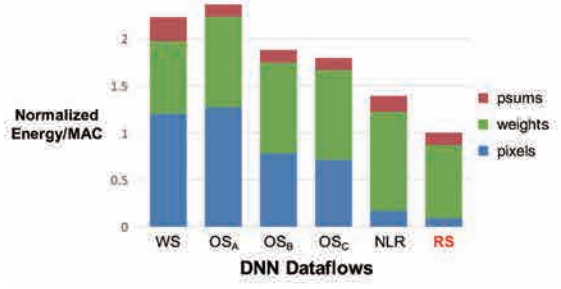
〈그림 10〉 Eyeriss DNN 가속기 구조

소개되었으며, 14×12 PE 배열과, 108 KB의 글로벌 버퍼 등을 포함하는 〈그림 10〉과 같은 구조를 가진다.

〈그림 11〉은 여러 가지 데이터플로우 구조에서의 AlexNet에 대한 에너지 효율을 비교한 것으로 (a)는 컨볼루션 레이어 (b)는 FC 레이어에서의 에너지 효율을 비교한 것이다.



(a) AlexNet Convolution Layer



(b) AlexNet FC Layer

〈그림 11〉 데이터플로우 구조별 에너지 효율 비교

Ⅲ. DNN 하드웨어 설계 최적화 기술

초기 DNN 모델 연구에서는 하드웨어 구현 복잡성을 크게 고려하지 않고 네트워크의 정확성을 극대화 하기 위한 방향으로 진행되었다. 그러나 이러한 연구방향은 최근 DNN 자체를 실생활에 적용하기에 쉽지 않은 문제를 야기시켰다. 따라서 최근에는 정확도와 연산 처리를 극대화 하면서 에너지와 비용을 최소화하기 위한 연구가 활발히 진행되고 있다.

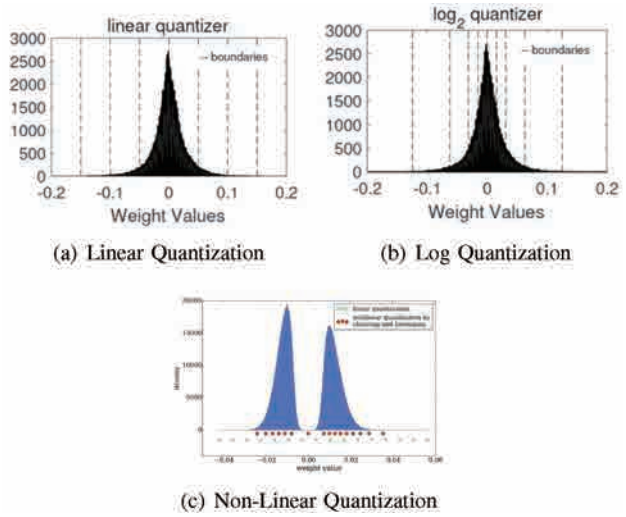
1. 정밀도 최적화 방법

연산에 사용된 값들에 대해서 원래의 정밀도에 비해 낮은 정밀도를 활용할 경우 적은 메모리 공간과 메모리 통신을 통해 구현비용과 에너지 효율에서 좋은 효과를 볼 수 있다. 그리고 연산기 구조를 좀 더 간소화 할 수 있는 효과도 기대할 수 있다.

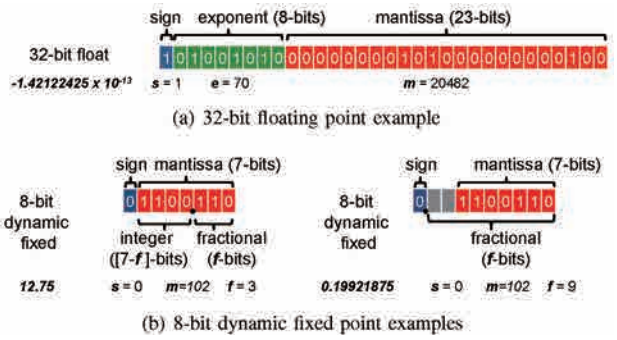
정밀도를 효율적으로 낮추기 위해서는 양자화 방법을 적절히 잘 선택해야한다. 양자화를 하는 방법에는 〈그림 12〉과 같은 예를 들 수 있다.

(1) 선형 양자화

정밀도를 줄이는 방법 중 가장 단순한 방법은 부동소수점 연산을 고정 소수점 연산으로 변환하는 것이다. 〈그림 13〉에서 수를 표현할 수 있는 다양한 예를 보여주고 있



〈그림 12〉 다양한 양자화 방법들



〈그림 13〉 수를 표현하는 다양한 방법

다. 8 비트 정수의 경우 f = 0 인 경우 동적 범위는 −128에서 127 사이이고, f = 10 인 경우 동적 범위는 −0.125에서 0.124023438이다. 가중치 및 활성화의 동적 범위가 매우 다를 수 있으므로 DNN에서 이러한 표현 방식은 상당히 유용하게 활용될 수 있다. 동적 고정 점을 사용하면 가중치의 미세 조정없이 비트 폭을 가중치 8 비트 및 활성화 8~10 비트로 줄일 수 있다[8-9].

8 비트 고정소수점 연산을 사용할 경우 덧셈 연산에서는 32비트 고정소수점 덧셈에 비해서 약 3.3배의 에너지 소모(면적은 3.8배)감소를 기대할 수 있으며, 32비트 부동소수점 덧셈에 비해 약 30배의 에너지 소모(면적은 116배)감소를 기대할 수 있다.

또한 곱셈 연산에서는 32비트 고정소수점 곱셈에 비해서 약 15.5배의 에너지 소모(면적은 12.4배) 감소를 기대할 수 있으며, 32비트 부동소수점 연산에 비해 약 18.5배의 에너지 소모(면적은 27.5배)감소를 기대할 수 있다[10].

이러한 방식은 최근 발표된 Google의 TPU, Nvidia의 PASCAL GPU와 같은 상용화 기술에 적용되어 사용되고 있다[11-12].

최근에는 정밀도를 극단적으로 줄이려는 연구도 활발히 진행되고 있다. BinaryConnect (BC)[13]는 이진 가중치 (즉, −1과 1)의 개념을 도입했다. 이진 가중치를 사용하면 MAC의 곱셈을 더하기 및 빼기로 줄일 수 있다. 이것은 이진 가중치와 활성화를 사용하는 Binarized Neural Networks (BNN)[14]에서 후에 더욱 확장되어 MAC을 XNOR로 줄였다. 그러나 BC와 BNN 방식을 적용할 경우 각각 19 %와 29.8 % 정도의 정확도 손실을 감수해야 할 것이다[15].

이러한 손실을 줄이기 위해 BWN (Binary Weight Nets) 및 XNOR-Nets는 DNN 연산 처리시 몇 가지 보완점을 도입하였다. 동적 범위를 복구하기 위해 출력을 곱할 때 (가중치는 필터에서 가중치의 절대 값 평균을 나타내는 −w 및 w) 첫 번째와 마지막 레이어를 32 비트 부동 소수점 정밀도로 처리하고, 컨볼루션 전에 정규화(normalization)를 수행하여 활성화의 동적 범위를 줄였다. 이러한 보완점은 BWN에서 정확도 손실을 0.8 %로 줄였고, XNOR-Nets는 11 %로 줄일 수 있었다.

앞서 설명한 Binary 네트워크는 가중치를 두 개의 값(−w 및 w)으로 제한한다. 그러나 가중치에 0 값이 포함될 경우(즉, −w, 0, w) 몇 가지 이점을 얻을 수 있다. Binary 가중치에 비해 가중치 당 비트가 1 비트 더 필요하지만 가중치의 희소성(sparsity)을 이용하여 계산 및 저장 비용을 줄이면 잠재적으로 추가 비트의 비용을 상쇄할 수 있다. 이러한 연산 방식을 Ternary Weight Nets (TWN)[16]이라 하고, 각 가중치 (즉, −w1, 0, w2)에 대해 다른 스케일로 학습 Ternary Quantization (TTQ)[17]는 0.6 %의 정확도 손실을 가진다(32비트 부동소수점 활성화 가정).

(2) 비선형 양자화

선형 양자화는 레벨이 균일한 간격을 갖는 반면, 비선형양자화는 가중치와 활성화의 분포가 일정하지 않은 것을 가정한다[7, 18]. 따라서 비선형 양자화는 잠재적으로 정확도를 선형양자화에 비해 좀 더 향상시킬 수 있음을 알 수 있다. 특히, 최근 (1)로그 양자화와 (2)학습된 양자화 또는 가중치 공유 접근법이 연구되고 있다.

로그 양자화는 〈그림 12(b)〉와 같이 대수 분포에 따라 양자화 레벨을 할당하면 가중치와 활성화가 다른 레벨에 보다 균등하게 분배되고, 각 레벨이보다 효율적으로 사용되어 양자화 오류가 줄어든다. 예를 들어, VGG-16[19]에 선형 양자화 4 비트를 사용하면 27.8%의 정확성 손실이 발생하지만 log2 양자화를 적용하면 정확성 손실이 5 %만 발생한다. 또한, 가중치가 2의 제곱으로 양자화되면, 승산은 비트 시프트로 대체 될 수 있다[18,20]. INQ (Incremental Network Quantization)를 사용하여 크

〈표 1〉 정확도 최적화 방법에 따른 AlexNet 결과 비교

Precision Reduction Method		bitwidth		Accuracy loss vs. 32-bit float (%)
		Weights	Activations	
Dynamic Fixed Point	w/o fine-tuning	8	10	0.4
	w/ fine-tuning	8	8	0.6
Reduce Weight	BinaryConnect	1	32 (float)	19.2
	Binary Weight Network (BWN)	1*	32 (float)	0.8
	Ternary Weight Networks (TWN)	2*	32 (float)	3.7
	Trained Ternary Quantization (TTQ)	2*	32 (float)	0.6
Reduce Weight and Activation	XNOR-Net	1*	1*	11
	Binarized Neural Networks (BNN)	1	1	29.8
	DoReFa-Net	1*	2*	7.63
	Quantized Neural Networks (QNN)	1	2*	6.5
	HWGQ-Net	1*	2*	5.2
Non-linear Quantization	LogNet	5 (conv), 4 (fc)	4	3.2
	Incremental Network Quantization (INQ)	5	32 (float)	-0.2
	Deep Compression	8 (conv), 4 (fc)	16	0
		4 (conv), 2 (fc)	16	2.6

고 작은 가중치를 다른 그룹으로 나누고 반복적으로 가중치를 양자화하고 다시 학습하여 정확성의 손실을 줄일 수 있다[21].

가중치 공유 접근법은 여러 가중치를 단일 값을 공유하도록 하는 방식이다. 이 방식은 필터 또는 레이어의 고유한 가중치 수를 줄일 수 있다. 한 가지 예로 해싱 함수를 사용하여 가중치를 그룹화하고 각 그룹에 하나의 값을 사용하는 것이다[22]. 다른 방법으로, 가중치를 k-mean 알고리즘[7]에 의해 그룹화 될 수 있다. 필터의 각 위치에서 사용할 가중치를 나타내는 공유 가중치와 색인 값이 모두 저장된다. 이것은 가중치를 가져 오는 2 단계 과정으로 이어진다: (1)가중치 색인 값을 읽는다; (2)가중치 색인 값 사용하여 공유 가중치를 읽는다. 이 접근법은 가중치 색인 (고유 가중치 수의 log2)이 가중치 자체의 비트 폭보다 작으면 가중치를 읽고 저장하는 비용을 줄일 수 있다.

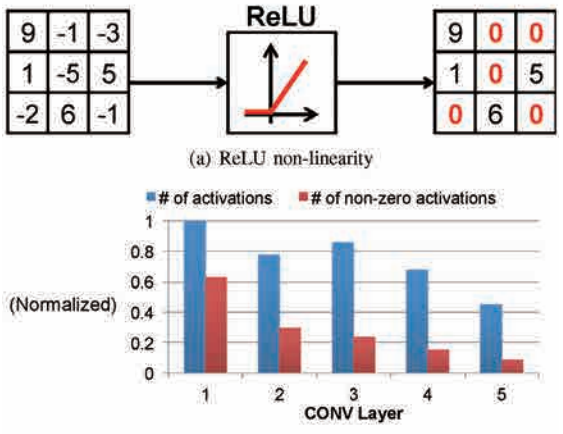
〈표 1〉은 최근에 사용된 주요 정밀도 줄이기 방법들에 의한 결과를 요약한 것이다.

2. 오퍼랜드(operand)와 모델 크기 최적화

연산 또는 오퍼랜드(가중치/활성화)의 각 크기를 줄이는 것 외에도 연산 수 및 모델 크기를 줄이는 방법에 대한 많은 연구가 진행되고 있다. 이러한 기술은 활성화 통계(activation statistics) 활용, 네트워크 프루닝(network pruning), 네트워크 구조 설계(network architecture design) 및 지식 증류(knowledge distillation) 등으로 분류될 수 있다.

(1) 활성화 통계 활용 방법

ReLU는 DNN에서 가장 널리 사용되는 비선형 함수로 〈그림 14(a)〉에서와 같이 모든 음수 값을 0으로 출력한다. 결과적으로, 특징 맵의 출력은 ReLU 후에 희소성을 갖게 된다. 예를 들어, AlexNet의 특징 맵은 〈그림



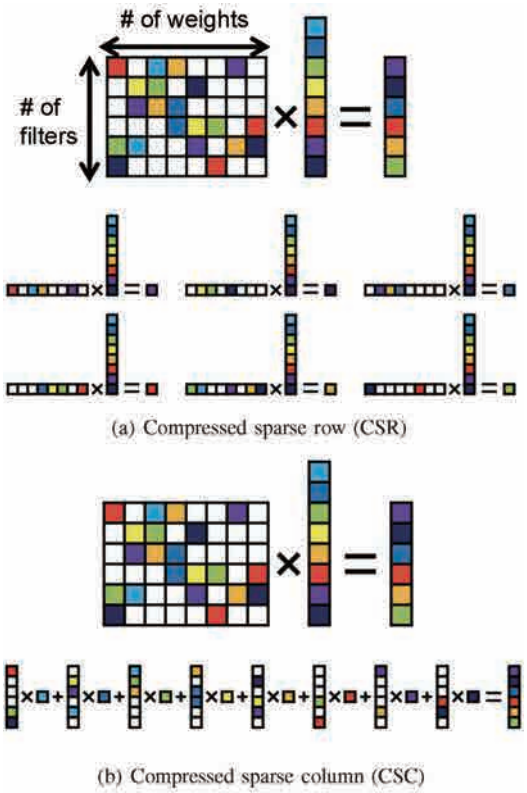
(a) ReLU non-linearity

(b) Distribution of activation after ReLU of AlexNet

〈그림 14〉 ReLU 사용에 따른 활성화 수 감소

14(b)〉에서 희소성을 19%에서 63%정도 갖게 된다. 이러한 희소성으로 인해 ReLU는 시그모이드(Sigmoid) 등과 같은 다른 비선형 함수에 비해 구현상 이점을 제공한다. 희소성은 특히 고가의 오프 칩 DRAM 액세스에 대해 압축을 사용하여 에너지 및 면적을 절약 할 수 있다. 예를 들어, 16 비트의 0이 아닌 값과 31개의 0 값을 간단한 run length 코딩을 활용하면 외부 메모리 대역폭을 2.1배 정도 줄일 수 있으며, 전체 메모리 대역폭을 1.5배 정도(가중치 포함) 줄일 수 있다[23]. 압축 외에도 하드웨어는 가중치 읽기를 건너뛰고 0 값에 대해 MAC를 수행하여 에너지 비용을 45% 정도 줄 일 수 있다[24]. 또한 연산 사이클을 건너뛰어 하드웨어 처리량을 1.37 배 정도 증가시킬 수 있는 효과를 기대할 수 있다[25].

또한, 낮은 값의 활성화를 제거하여 활성화 결과를 훨씬 더 희소하게 만들 수 있다. 예를 들어 작은 값을 가진 모든 활성화가 제거된 경우 정확도에 거의 영향을 미치지 않고 추가적으로 11% 정도 연산 처리속도 개선효과와 2배의 파워소모 절감효과를 볼 수 있다[25-26].



〈그림 15〉 저장 형식이 다른 희소 행렬 벡터 곱

(2) 네트워크 모델 최적화

DNN 학습을 쉽게하기 위해 DNN은 일반적으로 초과 매개 변수화(over-parameterized) 구조로 되어 있다. 따라서, 네트워크에서 많은 양의 가중치가 중복되어 제거 될 수 있다 (즉, 0으로 설정). 이러한 처리를 네트워크 프루닝 이라고 한다. 공격적인 네트워크 프루닝은 원래의 정확도를 유지하기 위해 종종 가중치의 미세 조정(finetuning)을 요구한다. 이것은 1989년에 Optimal Brain Damage[27]라는 기술을 통해 처음 제안되었다. 그 아이디어는 각 가중치의 영향을 가중치 손실과 관련하여 학습 손실에 미치는 영향을 계산하는 것이었다. 낮은 중요도 가중치는 제거되고 나머지 가중치는 미세 조정된다. 이 과정은 원하는 가중치 감소 및 정확도에 도달할 때까지 반복된다. 2015년에는[28]에서 현재의 DNN에 유사한 아이디어가 적용되었다. 대용량 DNN을 계산하기에는 너무 어려운 메트릭으로 두드러진 값을 사용하는 대신 네트워크 프루닝을 단순히 가중치의 크기를 기반으로 한다. 작은 가중치가 제거되고 모델이 정확도를 복원하도록 미세조정 한다. 가중치를 미세조정 하지 않으면 가중치의 약 50 %가 제거될 수 있지만, 미세조정을 통해 80% 까지 가중치가 제거될 수 있다.

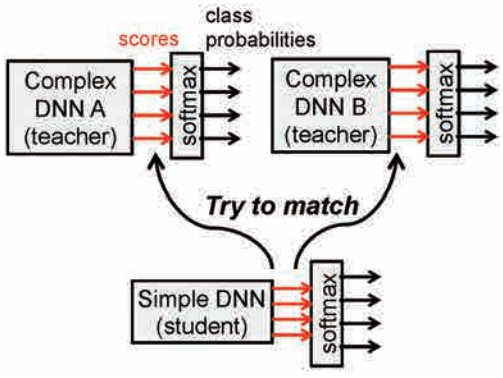
DNN에서 행렬-벡터 곱셈을 수행할 때 압축 된 포맷으로 희소 가중 매트릭스를 저장하는 방법을 결정하는 중요한 문제가 있다. 압축은 행 또는 열 순서로 적용 할 수 있다. 〈그림 15(a)〉와 같이, 희소 매트릭스-벡터 곱셈을 수행하기 위해 압축된 희소행 (CSR) 포맷이 사용된다. 그러나 행렬의 각 행이 희소하기 때문에 입력 벡터는 그 하위 집합 만 사용 되더라도 여러 번 읽어야 한다. 반면, 〈그림 15(b)〉와 같이, 압축된 희소컬럼 (CSC) 포맷이 사용될 때 출력은 수회 갱신되고, 입력 벡터의 한 원소만이 한 번에 읽혀진다[29]. CSC 포맷은 출력이 입력보다 작거나 필터 수가 필터의 가중치 수보다 현저히 크지 않은 경우 CSR 포맷 보다 전반적으로 더 낮은 메모리 대역폭을 요구한다.

(3) 네트워크 구조 소형화

가중치와 연산 수는 네트워크 구조 자체를 개선하여 줄일 수도 있다. 최신 동향은 큰 필터를 연속된 작은 필터로 대체하고 있다. 따라서, 전체 필터의 총 가중치 수는 더 적어진다. 이 방법은 네트워크 아키텍처 설계 중에 (학습 전) 또는 학습된 네트워크의 필터를 분해하여 (학습 후) 적용 할 수 있다. 후자의 경우 네트워크를 재학습하는 번거로움을 피할 수 있으나 유연성이 떨어진다.

(4) 지식 정제 (knowledge distillation)

DNN을 사용하거나 여러 모델(예 : 앙상블)의 예측 평균을 구하는 것은 단일 한 얕은(shallower) 네트워크를 사용하는 것보다 더 나은 정확성을 제공한다. 그러나 계산상의 복잡성은 더 높아질 수 밖에 없다. 두 경우의 장점을 최대한 살리기 위해 지식 정제는 복잡한 모델(교사)에서 습득한 지식을 단순한 모델(학생)로 전달한다. 따라서 학생 네트워크가 동일한 데이터 세트로 직접 학습된 경우 달성 할 수 없는 정확도를 얻을 수 있다[30-31]. 예를 들어, [32]는 지식 정제를 사용하여 학생 네트워크의 음성 인식 정확도를 2% 향상시킬 수 있는 방법을 보여준다. 이는 10개 네트워크의 앙상블로 구성된 교사 네트워크의 정확도와 유사하다. 〈그림 19〉는 가장 간단한 지식 정제 방법을 보여준다[30]. Softmax 층은 일반적으로 이미지 분류 네트워크에서 출력 레이어로 사용되어 클래스 점수에서 클래스 확률을 생성한다. 클래스 점수를 0과 1 사이의 값으로 채우며, 최대값은 1 이다. 이 지식 정제 방법의 경우, 교사 DNN (또는 교사 DNN의 앙상블)과 같은 하드



〈그림 16〉 학생 DNN과 교사 DNN의 앙상블간 스코어 매칭

타겟 (0 또는 1 값) 대신 소프트 타겟 (0과 1 사이의 값)이 사용된다. 데이터 세트의 라벨 목표는 소프트 타겟과 학생 DNN의 클래스 점수의 차를 최소화하는 것이다. 클래스 점수의 작은 값은 softmax에 의해 제거 될 수 있는 중요한 정보를 포함하기 때문에 클래스 점수 대신 클래스 대상이 소프트 대상으로 사용된다.

Ⅳ. 전망과 결론

인공지능이 여러 가지 많은 어려운 문제를 해결할 수 있는 솔루션으로 대두되고 있는 상황에서 실생활에 활용되기 위해서는 성능과 소비전력 등이 현실화되어야 하며, 이를 위해서 다양한 형태의 가속 기술이 개발되고 있다. 인공지능 시스템은 CPU에서 DSP에 이어 GPU로 발전하였으나, 보다 효율적인 처리를 위하여 FPGA를 사용하는 기술 개발과 함께 ASIC 솔루션도 개발되고 있다. 단말기에서 인공지능 추론을 효율적으로 수행하기 위해서는 NPU(Neural Processing Unit)와 같은 하드웨어 가속기가 주류를 이룰 것으로 예상된다. 특히 클라우드 형태로 인공지능 솔루션을 제공하는 기업은 기존의 CPU/GPU 중심의 클라우드 시스템에서 FPGA를 이용한 솔루션을 도입하고 있는 추세이며, 이러한 추세는 당분간 확대될 것으로 전망된다.

참 고 문 헌

[1] http://www.image-net.org/challenges/LSVRC/

[2] Vivienne Sze et al., "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," arXiv preprint arXiv:1703.09039v2, 2017.

[3] https://developer.movidius.com/

[4] https://developer.nvidia.com/embedded/buy/jetson-tx1

[5] https://developer.nvidia.com/embedded/buy/jetson-tx2

[6] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "Lognet: Energy-Efficient Neural Networks Using Logrithmic Computations," in ICASSP, 2017.

[7] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained

Quantization and Huffman Coding," in ICLR, 2016.

[8] Y. Ma, N. Suda, Y. Cao, J.-S. Seo, and S. Vrudhula, "Scalable and modularized RTL compilation of Convolutional Neural Networks onto FPGA," in FPL, 2016.

[9] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented Approximation of Convolutional Neural Networks," in ICLR, 2016.

[10] M. Horowitz, "Computing's energy problem (and what we can do about it)," in ISSCC, 2014.

[11] S. Higginbotham, "Google Takes Unconventional Route with Homegrown Machine Learning Chips," Next Platform, May 2016.

[12] T. P. Morgan, "Nvidia Pushes Deep Learning Inference With New Pascal GPUs," Next Platform, September 2016.

[13] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in NIPS, 2015.

[14] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1," arXiv preprint arXiv:1602.02830, 2016.

[15] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNORNet: ImageNet Classification Using Binary Convolutional Neural Networks," in ECCV, 2016.

[16] F. Li and B. Liu, "Ternary weight networks," in NIPS Workshop on Efficient Methods for Deep Neural Networks, 2016.

[17] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained Ternary Quantization," ICLR, 2017.

[18] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional Neural Networks using Logarithmic Data Representation," arXiv preprint arXiv:1603.01025, 2016.

[19] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "Lognet: Energy-Efficient Neural Networks Using Logrithmic Computations," in ICASSP, 2017.

[20] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented Approximation of Convolutional Neural Networks," in ICLR, 2016.

[21] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental Network Quantization: Towards Lossless CNNs with Lowprecision Weights," in ICLR, 2017.

[22] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing Neural Networks with the Hashing Trick," in ICML, 2015.

[23] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," IEEE J. Solid-State Circuits, vol. 51, no. 1, 2017.

[24] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in ISSCC, 2016.

[25] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: ineffectual-neuron-free deep neural network computing," in ISCA, 2016.

[26] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hern ′ andez-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in ISCA, 2016.

[27] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," in NIPS, 1990.

[28] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in NIPS, 2015.

[29] R. Dorrance, F. Ren, and D. Markovi ′ c, "A scalable sparse matrix-vector multiplication kernel for energy-efficient sparseblas on FPGAs," in ISFPGA, 2014.

[30] C. Bucilu, R. Caruana, and A. Niculescu-Mizil, "Model Compression," in SIGKDD, 2006.

[31] L. Ba and R. Caurana, "Do Deep Nets Really Need to be Deep?" NIPS, 2014.

[32] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," in NIPS Deep Learning Workshop, 2014.

[33] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional

Neural Networks," in ISCA, 2016.
[34] A. M. Caulfield et al., "A cloud-scale acceleration architecture," IEEE/ACM MICRO, 2016, pp. 1-13.
[35] David Pellerin, "FPGA Accelerated Computing Using AWS F1 Instances," HotChips 2017

김원종

- 1989년 2월 전남대학교 공과대학 전자공학과 학사 졸업
- 1992년 2월 한양대학교 대학원 전자공학과 석사 졸업
- 2009년 2월 한양대학교 대학원 전자공학과 박사 졸업
- 1999년 2월~2000년 3월 한양대학교 공학기술연구소 선임연구원
- 2000년 4월~현재 한국전자통신연구원 선임/책임연구원/실장

〈관심분야〉
인공지능/기계학습 가속 기법, IoT 하드웨어 개발, SoC 설계 및 설계 방법론, VLSI CAD, 멀티미디어 SoC 설계, 하드웨어/소프트웨어 통합 설계, 저전력 설계 방법론 등임

이상헌

- 2003년 2월 숭실대학교 정보통신전자공학부 학사 졸업
- 2005년 2월 숭실대학교 전자공학과 석사 졸업
- 2009년 8월 숭실대학교 전자공학과 박사 졸업
- 2009년 9월~2012년 6월 (주)엠텍비젼 SoC팀 선임연구원
- 2012년 8월~2015년 2월 (주)삼성테크윈 CCTV사업부 SoC팀 책임연구원
- 2015년 3월~2016년 12월 엠텍비젼(주) SoC팀 팀장
- 2017년 4월~현재 ㈜디퍼아이 대표이사

〈관심분야〉
Intelligent Vision SoC, Deep Learning SoC, Low-power Acceleration, System Optimization