

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference

길종현(github.com/hyeon-n-off)

개요(Abstract)

딥러닝 기반의 모델의 위압적인 계산 연산량은 효율적이고 정확한 on-device inference 를 요구한다.

정수만을 사용하는(integer-only) 연산을 통해서 추론을 진행하는 양자화 방식(Quantization)을 제안한다. 이는 부동 소수점(floating point)을 사용하는 추론보다 더 효율적으로 구현할 수 있다. 또한 end-to-end 모델의 양자화 이후 정확도를 유지하기 위한 훈련 절차를 같이 설계하였다.

결과적으로, 제안된 양자화 방식은 정확도(accuracy)와 기기 내 지연시간(on-device latency) 사이의 tradeoff 를 개선할 수 있다.

1. 소개(Introduction)

현재 최신의 CNNs 는 휴대용 기기(mobile devices)에서 사용하기에 적합하지 않다. AlexNet 의 등장 이후에, 현대의 CNNs 는 주로 분류(classification) / 감지(detection) 정확도로 평가되었다. 따라서 모델 네트워크의 구조는 모델 복잡성과 계산 효율성을 고려하지 않고 발전해왔다. 반면에, 스마트폰, AR/VR 기기와 같은 휴대용 플랫폼에서 CNNs 의 성공적인 배포를 위해서는 제한된 기기 내 메모리를 수용하기 위해 작은 모델 크기와 사용자 참여(user engagement)를 유지하기 위해 낮은 지연시간을 필요로 한다. 이는 정확도 손실을 최소화하면서 모델의 크기와 CNNs 의 추론 시간을 줄이는데 초점을 맞춘 연구들이 급성장하는 분야로 이끌었다.

이 분야의 접근은 대략 두가지로 나뉜다. 첫 번째는 (MobileNet, SqueezeNet, ShuffleNet 등) 계산 / 메모리에 효율적인 연산을 활용하는 새로운 네트워크 아키텍처를 설계한다.

두 번째는 CNNs 의 가중치(weights) and/or 활성화(activations)를 32bit 부동 소수점에서 더 낮은 bit 의 표현으로 양자화한다. 현재의 많은 양자화 접근 방식(TWN, BNN, XNOR-net 등)에도 불구하고 정확도와 지연시간을 절충할 때 두 가지 측면에서 부족한 점이 존재한다.

첫 번째, 이전 방식은 합리적인 베이스라인 구조에서 평가되지 않았다. 가장 대중적인 베이스라인 구조(AlexNet, VGG 와 GoogleNet)는 정확도 향상을 위한 구조 때문에 모두 과도하게 매개변수화(over-parameterized)된다. 따라서 이러한 구조들은 쉽게 압축할 수 있기에, 이러한 아키텍처에 대한 양자화 실험은 진지한 연구라기보다는 단순한 개념 증명 수준에 불과하다. (의미 있는 실험이 아니다.)

대신, 더 의미 있는 실험은 MobileNet 과 같이 이미 효율적인 tradeoff 를 가진 모델 구조를 양자화하는 것이 될 것이다.

둘 째는, 많은 양자화 접근들은 실제 하드웨어에서의 검증 가능한 효율 향상을 보여주지 않는다. 오직 가중치만을 양자화하는 접근들은 주로 기기 내의 저장(on-device 환경에서 모델을 실행할 수 있는 지)에 중점을 두고 계산 효율성에는 중점을 두지 않는다.

Binary, ternary, bit-shift 는 예외적으로 주목할만하다. 이 접근법들은 '0' 또는 '2'의 거듭제곱'인 가중치를 사용한다. 이는, Bit Shift 연산을 통해 곱셈을 구현할 수 있게 한다. 하지만 Bit Shift 연산은 커스텀 하드웨어에서는 효율적일 수 있지만, 기존의 multiply-add 구조 하드웨어에 이점을 제공하기는 어렵다.

더욱이 피연산자가 넓으면(the operands are wide) 곱셈 비용이 증가하며, 가중치와 활성화 값을 모두 양자화하면 비트 깊이(bit depth)가 낮아질수록 곱셈을 피해야 한다는 필요성이 줄어든다.

- 곱셈 연산의 계산 비용은 피연산자(operands)의 비트 폭(width)에 따라 달라진다.
- 피연산자의 비트 폭이 넓을수록 곱셈 연산의 계산 복잡도와 비용이 증가한다.
- 가중치(weights)와 활성화(activations)를 모두 양자화하면 비트 깊이가 낮아질수록 곱셈을 피해야 한다는 필요성이 줄어든다.
-
- 즉, 모델의 가중치와 활성화 값을 양자화하면 곱셈 연산의 복잡성이 감소하고, 낮은 비트 깊이에서도 연산 효율성이 개선된다.

특히, 이러한 접근 방식은 promised timing 성능 향상을 검증하기 위해 on-device 환경 내 측정을 제공하는 경우가 드물다. 런타임 친화적인(runtime-friendly) 접근법들은 가중치와 활성화(activations) 값을 모두 1 bit 표현으로 양자화한다. 이러한 접근법들로 곱셈과 덧셈 연산을 Bit Shift와 Bit Count 기법으로 향상할 수 있다. 그러나, 1-bit 양자화는 자주 상당한 성능 저하를 이끄며, 모델 표현에 지나치게 엄격해야 한다.

이 논문에서는 일반적인 휴대용 하드웨어에서 MobileNets의 정확도와 지연 시간의 tradeoff를 향상시켜 위에 언급한 문제들을 개선해 나갈 것이다.

- 가중치와 활성화 값을 8-bit integer로 양자화하고 편향(bias) 벡터와 같은 값만에만 32-bit integer로 양자화하는 '**Quantization scheme**'를 제공한다.
- 정수 - 산술 - 전용 하드웨어에서 효과적으로 구현가능한 '**Quantized inference framework**'를 제공한다.
- 실제 모델에서 정확도의 손실을 최소화하는 '**Quantized training framework**'도 같이 제공한다.

이 논문은 CNNs의 훈련 속도를 가속화하기 위한 낮은 정밀도(low-precision)의 고정 소수점(fixed-point) 연산을 수행하는 *Deep learning with limited numerical precision*, x86 CPUs에서 추론 속도를 높이기 위해 8-bit의 고정 소수점을 사용한 *Improving the speed of neural networks on CPUs*을 참고하였다.

2. 양자화된 추론(Quantized Inference)

2.1 Quantization Scheme

이 섹션에서는, 일반적인 양자화 방식(Quantization Scheme), 즉 값의 비트 표현("양자화된 값"에 대해 q 로 표시)과 수학적 실수 해석("실제 값"에 대해 r 로 표시)간의 대응 관계를 설명한다.

논문의 양자화 방식은 **추론 시에는 정수 전용 산술을**, **학습 시에는 부동 소수점 산술을 사용**하여 구현되며, 두 구현 모두 서로 높은 수준의 대응 관계를 유지한다. 먼저 양자화 방식에 대한 수학적으로 엄밀한 정의를 제공하고, 정수 산술 추론과 부동 소수점 학습 모두에 이 방식을 별도로 채택함으로써 이를 달성한다.

논문의 양자화 방식의 기본 조건은 다음과 같다.

양자화된 값에 정수 전용 산술 연산을 사용하여 모든 산술의 효율적인 구현이 가능해야한다. 이는 양자화 방식이 실수 r 와 정수 q 가 *affine mapping*이 되어야 한다는 것과 동일하다. (S 와 Z 는 상수이다.)

$$r = S(q - Z)$$

위 수식은 논문의 양자화 방식이며 **상수 S 와 Z 는 양자화 파라미터**이다. 양자화 방식은 각 활성화, 가중치 행렬마다 양자화 파라미터 **하나의 집합**을 사용한다. ; 각 행렬마다 다른 파라미터를 사용한다.

8-bit 양자화의 경우, q 는 8-bit 정수로 양자화된다. 몇몇의 행렬, 특히 편향 벡터는 32-bit 정수로 양자화된다.

Scaling 을 위한 **상수 S 는 임의의 양의 실수**이다. 이는 실수 r 과 같이 부동 소수점 값과 같이 표현된다.

2.2 에서 추론 작업에서 이러한 부동 소수점 값의 표현을 피하는 방법에 대해 설명할 예정이다.

상수 Z (**zero-point**)는 양자화된 q 와 같은 자료형이며, 실제로는 실수 r 가 0 일 때 해당하는 양자화된 값 q 이다. 즉, $r = 0$ 인 경우 $q = Z$ 가 된다.

- 양자화 방식 중에는 대칭 양자화와 비대칭 양자화가 있는데, Z 가 입력의 0 이 출력의 0 에 매핑되도록 보장하는 대장 양자화를 만들기 위한 장치로 보인다.

지금까지의 내용은 다음과 같이 요약된다.

```
template<typename QType> // e.g. QType=uint8
struct QuantizedBuffer {
    vector<QType> q;      // the quantized values
    float S;             // the scale
    QType Z;             // the zero-point
};
```

2.2 Integer-arithmetic-only matrix multiplication

어떻게 정수 전용 산술을 사용하여 추론을 진행하는 지에 대한 내용이다. (방정식을 사용하여 실수 계산을 양자화된 값으로 변환하는 방법과 상수 S 가 정수가 아니라도 정수 산술만을 포함하도록 설계하는 방법 등)

실수 r_1, r_2 로 이루어진 두 행렬($N \times N$)의 곱을 생각해보자. 그 곱은 $r_3 = r_1 r_2$ 로 표현된다.

각 행렬들은 r_α ($\alpha = 1, 2$ or 3) as $r_\alpha^{(i,j)}$ ($1 \leq i, j \leq N$) 로 표현되며, 해당 행렬들의 양자화 매개변수는 (S_α, Z_α) 이다.

$$r_\alpha^{(i,j)} = S_\alpha (q_\alpha^{(i,j)} - Z_\alpha) \quad (2)$$

행렬의 곱셈 법칙에 의해 다음과 같이 표현할 수 있다.

$$S_3 (q_3^{(i,k)} - Z_3) = \sum_{j=1}^N S_1 (q_1^{(i,j)} - Z_1) S_2 (q_2^{(j,k)} - Z_2) \quad (3)$$

$$q_3^{(i,k)} = Z_3 + M \sum_{j=1}^N (q_1^{(i,j)} - Z_1) (q_2^{(j,k)} - Z_2), \left(M := \frac{S_1 S_2}{S_3} \right) \quad (4), (5)$$

방정식 (4)에서 유일하게 정수가 아닌 것은 M 이다. 이는 S_1, S_2, S_3 에만 의존하는 상수로서 오프라인에서 계산이 가능하다. 논문에선 경험적으로 M 이 항상 (0,1) 구간 내에 위치한다는 사실을 발견하였다. 따라서 해당 값을 다음과 같이 정규화된 형태로써 표현할 수 있다.

$$M = 2^{-n} M_0 \quad (6)$$

여기서 M_0 는 [0.5,1) 구간에 존재하며, n 은 음수가 아닌 정수이다. 정규화된 M_0 은 이제 고정 소수점 숫자로 표현되는데 적합하다. (하드웨어에 따라 int16 이나 int32 로 저장할 수 있다.) 예를 들어 int32 를 사용하는 경우 M_0 를 $2^{31} M_0$ 으로 표현하고 이를 정수로써 처리한다.

따라서 M_0 에 의한 곱셈은 고정 소수점 곱셈으로 구현될 수 있다.

- 스케일링 계수 M :

$$\text{만약 } S_1 = 0.1, S_2 = 0.2, S_3 = 0.05 \text{ 라면 } M = \frac{0.1 \times 0.2}{0.05} = 0.4$$

- M 의 표현:

$$M = 0.4 = 2^{-1} \times 0.8, \text{ 여기서 } n = 1, M_0 = 0.8$$

2.3 Efficient handling of zero-points

(4)번 방정식의 평가를 효율적으로 구현하기 위해서 $2N^3$ 번의 뺄셈을 수행하거나, 곱셈의 피연산자를 8-bit 에서 16-bit 로 확장할 필요 없이 먼저 (4)번 방정식에서 곱셈을 분배함으로써 다음과 같이 재작성할 수 있다.

$$q_3^{(i,k)} = Z_3 + M \left(NZ_1Z_2 - Z_1a_2^{(k)} - Z_2\bar{a}_1^{(i)} + \sum_{j=1}^N q_1^{(i,j)} q_2^{(j,k)} \right) \quad (7)$$

$$a_2^{(k)} := \sum_{j=1}^N q_2^{(j,k)}, \quad \bar{a}_1^{(i)} := \sum_{j=1}^N q_1^{(i,j)} \quad (8)$$

수식 (7) 유도:

수식 (4)의 곱셈 부분을 분배 법칙으로 전개합니다:

$$(q_1^{(i,j)} - Z_1)(q_2^{(j,k)} - Z_2) = q_1^{(i,j)} q_2^{(j,k)} - Z_1 q_2^{(j,k)} - Z_2 q_1^{(i,j)} + Z_1 Z_2$$

이를 $\sum_{j=1}^N$ 에 대입하면:

$$\sum_{j=1}^N (q_1^{(i,j)} - Z_1)(q_2^{(j,k)} - Z_2) = \sum_{j=1}^N q_1^{(i,j)} q_2^{(j,k)} - Z_1 \sum_{j=1}^N q_2^{(j,k)} - Z_2 \sum_{j=1}^N q_1^{(i,j)} + N Z_1 Z_2$$

이 식을 수식 (4)에 대입하면:

$$q_3^{(i,k)} = Z_3 + M \left(NZ_1Z_2 - Z_1a_2^{(k)} - Z_2\bar{a}_1^{(i)} + \sum_{j=1}^N q_1^{(i,j)} q_2^{(j,k)} \right)$$

여기서:

- $a_2^{(k)} := \sum_{j=1}^N q_2^{(j,k)}$ (열 방향의 합)
 - $\bar{a}_1^{(i)} := \sum_{j=1}^N q_1^{(i,j)}$ (행 방향의 합)
-

수식 (7):

$$q_3^{(i,k)} = Z_3 + M \left(N Z_1 Z_2 - Z_1 a_2^{(k)} - Z_2 \bar{a}_1^{(i)} + \sum_{j=1}^N q_1^{(i,j)} q_2^{(j,k)} \right)$$

구성 요소:

1. $N Z_1 Z_2$:

- 모든 항목의 곱셈에서 zero-point의 상수 곱이 등장합니다. 이는 N 으로 스케일링됩니다.

2. $Z_1 a_2^{(k)}$:

- 행렬 q_2 의 열 k 방향 합산에 Z_1 을 곱한 항목입니다.

3. $Z_2 \bar{a}_1^{(i)}$:

- 행렬 q_1 의 행 i 방향 합산에 Z_2 를 곱한 항목입니다.

4. $\sum_{j=1}^N q_1^{(i,j)} q_2^{(j,k)}$:

- 실제로 양자화된 값으로 계산되는 행렬 곱입니다.
-

효율성 분석:

1. 뮐셈 횟수 감소:

- Zero-point를 미리 계산하여 $N Z_1 Z_2, Z_1 a_2^{(k)}, Z_2 \bar{a}_1^{(i)}$ 로 분리하면 $2N^3$ 개의 뮐셈 대신 $O(N^2)$ 수준의 추가 연산만 필요합니다.

2. 정수 연산 유지:

- $q_1^{(i,j)}$ 와 $q_2^{(j,k)}$ 는 여전히 정수로 유지되므로, 곱셈 연산에서 16비트로 확장하지 않아도 됩니다.

3. 미리 계산 가능한 항목:

- $M, a_2^{(k)}, \bar{a}_1^{(i)}, N Z_1 Z_2$ 는 모두 미리 계산 가능하므로 런타임 연산을 최소화할 수 있습니다.
-

2.4 Implementation of a typical fused layer

섹션 2.3 에 대한 논의를 계속 중이지만, 이제 관련된 모든 데이터 유형을 명시적으로 정의하고, 양자화된 행렬 곱셈 (7)을 수정하여 bias-addition 및 활성화 함수 평가를 직접 통합한다. 전체 레이어를 단일 작업으로 융합하는 것은 단순히 최적화가 아니다. 훈련에 사용되는 산술을 추론 코드에서도 동일하게 재현해야 하므로, 추론 코드에서 융합 연산자(fused operators)의 세분성(granularity) (8-bit 양자화 입력을 받아 8-bit 양자화 출력을 생성하는 것)은 훈련 그래프에서 **“가짜 양자화(fake quantization)”** 연산자의 배치와 일치해야 한다 (섹션 3).

- **훈련 단계의 산술:** 훈련 중에는 모델이 부동소수점 연산을 사용하면서도 양자화를 흉내 내는 방식인 Fake Quantization 을 사용한다.
 - 예를 들어, 입력 값과 가중치를 8-bit 로 제한된 범위 내에서 학습시키지만 실제 계산은 부동소수점으로 이루어진다.
 - 이는 모델이 실제 양자화된 하드웨어 환경에서의 동작을 예측할 수 있도록 도움을 준다.
- **추론 단계의 산술:** 추론 시에는 실제로 양자화된 정수값으로 연산이 이루어진다.
 - 따라서 추론에서 사용하는 연산자는 훈련 중 사용된 Fake Quantization 의 동작과 동일한 방식으로 구성되어야 한다.
 - 일관성을 유지하지 않으면, 훈련 시 최적화된 가중치와 추론 시의 계산 방식이 달라지면서 모델의 성능이 저하될 수 있다.

ARM 과 x86 CPU 구조에서 구현을 위해 gemmlowp 라이브러리를 사용하였다. q_1 을 가중치 행렬, q_2 를 활성화 행렬로 선택하였고, 두 행렬 모두 uint8 형식으로 택하였다. uint8 값의 곱에는 32-bit 누산기 필요하다.

$$\text{int32} += \text{uint8} * \text{uint8} \quad (10)$$

2. 왜 uint8 타입을 사용하는가?

- **uint8 (unsigned 8-bit integer):**
 - 장점: 0~255의 값을 가지므로 메모리 효율적이고, 하드웨어에서 잘 지원됩니다.
 - **zero-point:** 8-bit 양자화에서는 데이터의 분포를 0~255로 조정하기 위해 zero-point를 설정합니다. 이는 데이터를 unsigned로 변환하면서 실제 값이 음수일 수도 있는 상황을 처리합니다.
 - `int8`도 사용 가능하지만, zero-point를 음수로 조정해야 하며, 이는 추가 계산을 초래할 수 있습니다.
-

3. 왜 32-bit 누산기를 사용하는가?

- q_1 과 q_2 는 각각 `uint8` 타입입니다. 따라서 두 값을 곱한 결과는 0에서 $255 \times 255 = 65,025$ 범위에 속합니다.
 - $N \times N$ 행렬 곱셈에서는 수천 개 이상의 곱셈 결과를 합산하게 됩니다.
 - 예: $N = 128$ 일 경우, 하나의 행렬 곱셈에서 최대 $128 \times 128 = 16,384$ 번의 합산이 필요합니다.
 - 8-bit나 16-bit 누산기로는 이 값을 표현할 수 없기 때문에 **32-bit 누산기**가 필요합니다.
-

4. 왜 signed int32를 사용하는가?

- 누산기 타입을 **signed** (부호 있는) 32-bit로 선택한 이유는 다음과 같습니다:
 - **양자화된 값의 특성:**
가중치나 활성화 값의 zero-point를 포함한 계산에서, 중간 계산 결과가 음수가 될 가능성이 있습니다.
예를 들어, $(q_1 - Z_1)(q_2 - Z_2)$ 를 계산할 때 $q_1 < Z_1$ 이거나 $q_2 < Z_2$ 인 경우 결과가 음수가 될 수 있습니다.
 - **signed 누산기 필요성:**
이러한 음수 값을 처리하려면 unsigned 타입으로는 부족하므로 signed int32가 필요합니다.
-

5. 수식 (10)의 의미

- 수식 (10):

$$\text{int32+} = \text{uint8} \times \text{uint8}$$

- 곱셈: 두 개의 `uint8` 값이 곱해져 최대 65,025 범위의 결과를 생성합니다.
- 누산: 이 값을 signed int32에 더해줍니다.
- 누산기는 32-bit signed 타입이므로 중간 계산 결과가 음수로 변하거나 큰 값을 가지는 경우에도 안전하게 처리할 수 있습니다.

int32 형 누산기에 int32 bias 를 추가하기 위해, bias-vector 를 양자화하여 int32 를 양자화된 데이터 유형으로 사용하고, 0 을 양자화 영점 Z_{bias} 으로 사용하며, 양자화 스케일 S_{bias} 는 가중치(S_1)와 활성화의 스케일(S_2)의 곱인 누산기와 동일하다.

$$S_{bias} = S_1 S_2, \quad Z_{bias} = 0 \quad (11)$$

bias-vector 는 32-bit 값으로 양자화되지만, 신경망 파라미터 중 극히 일부에 불과하다. 또한 **bias-vector**에 대해 더 높은 정밀도를 사용하는 것은 실제 요구 사항을 충족한다. 각 bias-vector entry 가 많은 출력 활성화 값들에 더해짐에 따라 bias-vector 의 양자화 오류는 전체 bias 에 작용하는 경향이 있으며, 이는 좋은 end-to-end 신경망 정확도를 유지하기 위해 피해야 한다.

- 왜 편향 벡터(bias vector)를 32-bit 로 양자화할까?

- 편향 벡터는 네트워크에서 매우 중요한 역할을 한다. 특히 각 편향이 많은 출력 활성화 함수에 더해지기 때문에 편향 벡터 양자화 오류가 전체적인 오류를 발생시킬 수 있다.
- 양자화 오류가 0 이 아닌 평균 값을 갖는 경우, 이는 전체 네트워크 출력에 편향을 추가하게 되어 **정확도 저하를 초래**할 수 있다. 따라서, 높은 정밀도로 양자화하여 오류를 최소화하려는 목적이 있다.

- 편향(bias)과 int32 누산기의 관계

- 양자화된 편향 벡터는 int32 형식으로 양자화되어, 계산 중인 int32 누산기에 추가된다. 즉, int32 타입의 누산기에 32-bit bias 값을 추가하는 방식이다.
- 이는 양자화된 값들을 정밀하게 처리할 수 있도록 도와준다. 특히, 각 편향 값은 여러 출력 활성화 값에 추가되기 때문에, 편향 벡터의 양자화에서 발생하는 오류가 모든 출력 값에 영향을 미치지 않게 해야 한다.

down-scaling 은 수식(7)에서 M 과의 곱셈에 해당한다. 이는 정규화된 M_0 와 반올림 Bit-Shift 를 사용한 고정 소수점 곱셈으로 구현된다. 이후 $[0, 255]$ 범위로 saturating cast 를 수행하여 uint8 로 변환된다.

실제로, 양자화된 훈련 과정에서는 출력 uint8 $[0, 255]$ 범위 전체를 활용하려고 학습하는 경향이 있어, 활성화 함수는 더 이상 아무런 작업을 하지 않으며, 그 효과는 uint8 로 saturating cast 를 할 때 내포된 $[0, 255]$ 의 범위로 제한하는 동작에 포함된다.

- 범위 제한(Saturating Cast)과 활성화 함수

- 범위 제한은 범위 밖의 값을 자동으로 제한하는 방식이다. 예를 들어, uint8 형식에서 계산 결과가 256 을 초과하면 자동으로 255 로 설정되고, 0 미만이면 0 으로 설정된다.
- 이 과정은 계산 후 양자화된 출력 값을 올바른 범위 내에 넣어주기 위해 필수적이다.
- 훈련 과정에서 네트워크는 양자화된 값이 $[0, 255]$ 범위를 잘 활용하도록 학습한다. 이 작업 이후 이어지는 활성화 함수는 값을 추가로 제한하는 역할을 하지 않고, 그 기능이 이미 양자화 과정에서 uint8 로 변환될 때 내포된 범위 제한에 포함된다.
- 즉, 활성화 함수는 **실질적으로 아무런 작업을 하지 않고** 추가적인 조정 없이 값의 범위를 자연스럽게 제한한다. 이 방식은 연산 효율성을 높이고 정확도 유지에 중요한 역할을 한다.