

LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale

LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale

맹선재(리서치)
최철웅(리서치)
이태훈(리서치)

목차

00. 양자화(Quantization)란?

01. 기존의 양자화 방법

02. LLM.int8()

03. Transformer 규모에 따른 이상치의 출현

04. 결론

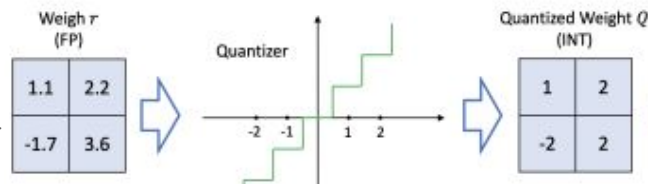
00. 양자화 (Quantization)란?

- 1) 양자화의 개념
- 2) 양자화의 효과
- 3) 양자화의 단계
- 4) 양자화의 적용 방식
- 5) 양자화의 도입 배경

00.1) 양자화의 개념

- 실수형 변수를 정수형 변수로 변환

- 예시) 32비트 부동 소수점 수 -> 8비트 정수



- 반올림과 절삭: 변환 과정에서 데이터는 반올림되거나 절삭되어야 할 수 있음
 - 데이터 A의 범위가 0..9이고 B의 범위가 0..4일 때, A의 값 "4"는 B의 "2"로 반올림
 - A의 값 "3"은, B에서는 1과 2 사이에 위치 (1.5)하므로 보통 "2"로 반올림

= A의 값 "4"와 "3"이 B에서는 동일한 값 "2"를 가짐

00.2) 양자화의 효과

- **Pro**

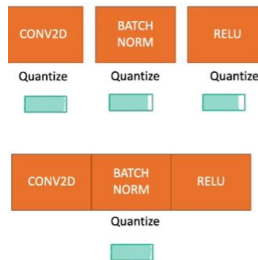
- 모델 사이즈 축소: float -> int로 용량 감소
- 모델 연산량 감소: bit 수 감소로 계산 복잡도 감소
- 하드웨어 사용 효율 증가: int가 하드웨어에 더 친화적

- **Con**

- 정보 손실: 반올림 과정에서의 정보 손실 (=모델 성능 저하 가능성)

00.3) 양자화의 단계

1. **Module Fusion:** layer들을 하나로 묶어줌
2. **Formula Definition:** 양자화시 사용하는 식 정의
3. **Hardware Deployment:** 하드웨어에 따라서 calibration 조정
4. **Dataset Calibration:** 가중치 변환을 위한 식의 파라미터를 Dataset을 이용하여 계산
5. **Weight Conversion:** 실제 weight를 FP 타입에서 INT 타입으로 변환
6. **Dequantization:** inference를 통해 얻은 출력을 역양자화를 통하여 다시 Floating 타입으로 변경



FP32			INT8		
-3.57	4.67	-3.97	33	255	22
-1.74	2.34	-1.76	82	192	81
-4.75	-0.06	3.07	1	127	212

quantization →

00.4) 양자화의 적용 방식

- 모델 가중치 양자화: 학습된 모델의 가중치를 양자화
- 활성화 양자화: 모델의 중간 출력(활성화)을 양자화
- 이후 손실된 정보를 보정해 주기 위한 후처리 및 재학습 가능

00.5) 양자화의 도입 배경

1. 학습에 사용되는 매개변수가 너무 많아짐
2. 모델이 무거워지고 연산 속도와 리소스를 너무 많이 사용하게 됨
3. 데이터를 더 작은 비트 크기로 매핑하는 것으로 어느 정도의 정보 손실을 감수하면서 연산 과정의 부하를 줄일 수 있다는 것을 확인
4. ???
5. PROFIT!

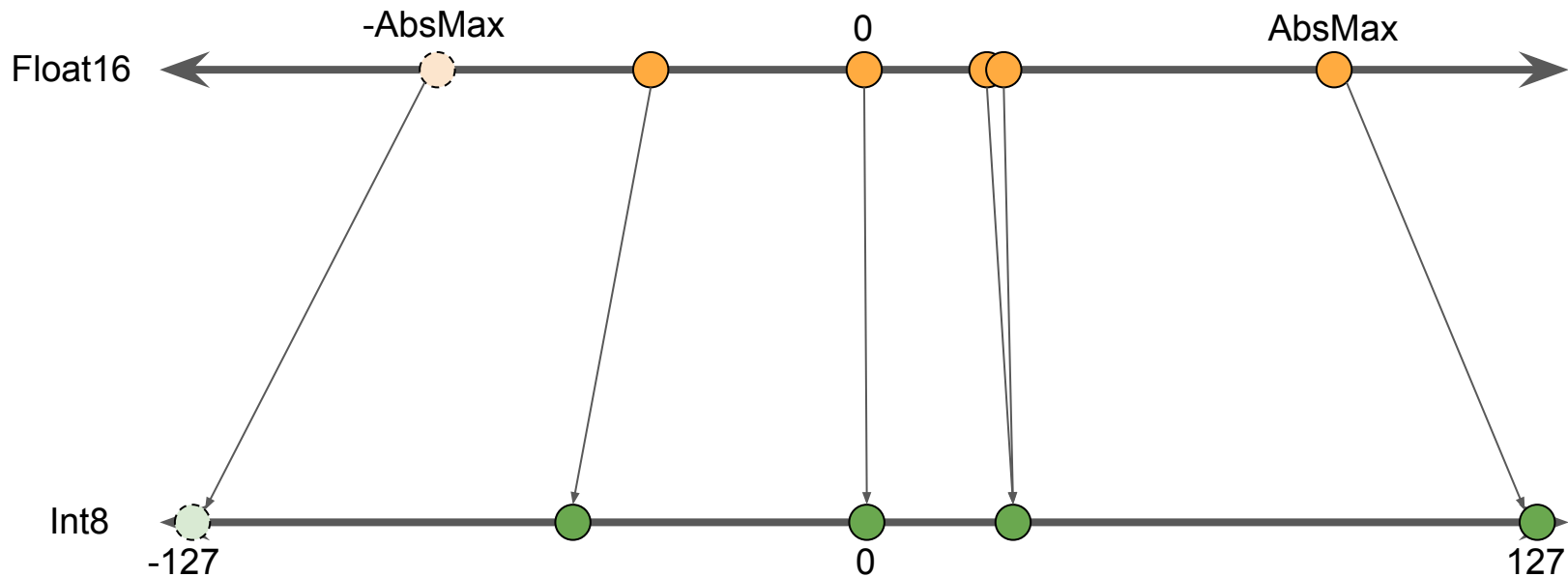
01. 기존의 양자화 방법

- 1) 텐서를 양자화하는 두가지 방법
- 2) 양자화된 모델로 예측하는 방법
- 3) 기존 양자화 방식의 한계

01.1) 텐서를 양자화하는 두가지 방법 (1 / 9)

- AbsMax Quantization
 - 또는 Symmetric Quantization
- Zero Point Quantization
 - 또는 Asymmetric Quantization

01.1) 텐서를 양자화하는 두가지 방법 (2 / 9)



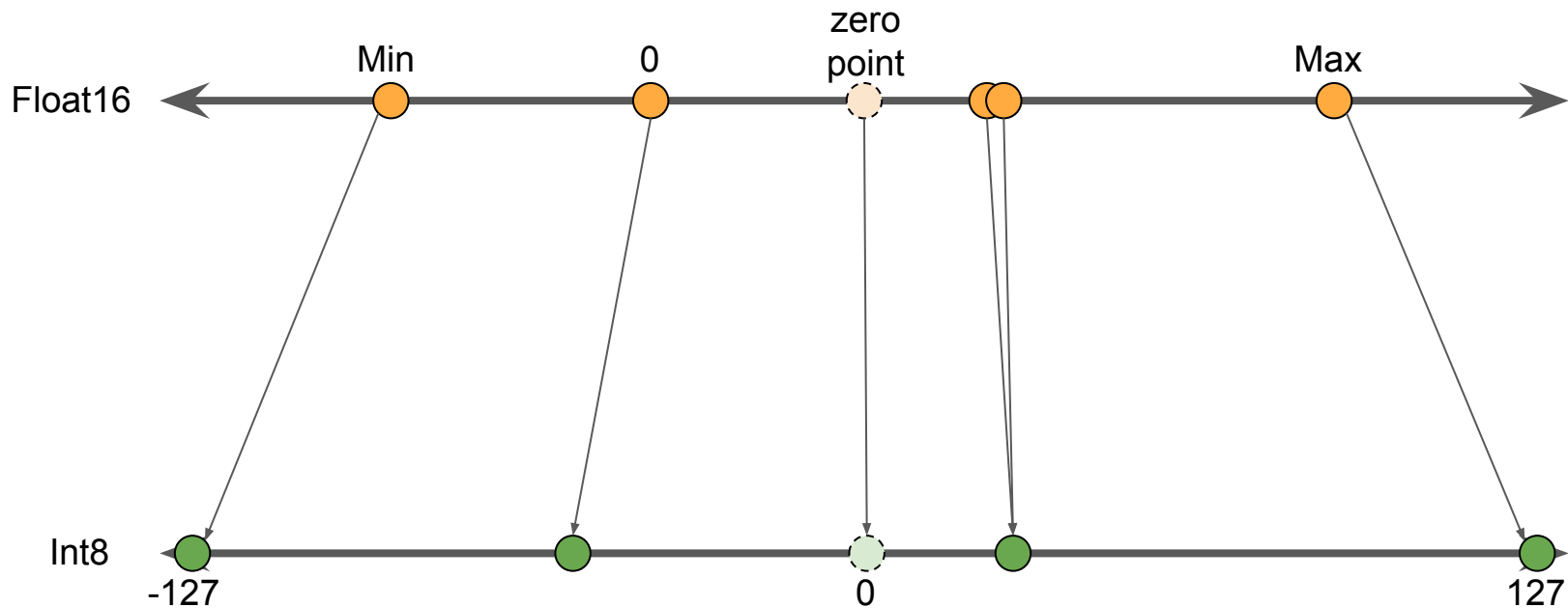
01.1) 텐서를 양자화하는 두가지 방법 (3 / 9)

- AbsMax Quantization

$$\mathbf{X}_{i8} = \left\lfloor \frac{\overbrace{127}^{\text{int8 범위}} \cdot \mathbf{X}_{f16}}{\underbrace{\max(|\mathbf{X}_{f16_{ij}}|)}_{\text{X}_{f16} \text{ 값의 범위, 절댓값의 최댓값 사용}}} \right\rfloor = \left\lfloor \frac{\overbrace{127}^{\text{int8 범위}}}{\underbrace{\|\mathbf{X}_{f16}\|_{\infty}}_{\text{X}_{f16} \text{ 값의 범위, 절댓값의 최댓값 사용}}} \mathbf{X}_{f16} \right\rfloor = \left\lfloor s_{x_{f16}} \mathbf{X}_{f16} \right\rfloor$$

- $\underbrace{\quad}_{\text{blue}} \underbrace{\quad}_{\text{orange}}$: \mathbf{X}_{f16} 값의 범위, 절댓값의 최댓값 사용
 - $\underbrace{\quad}_{\text{orange}}$: int8 타입의 범위
- $\lfloor \cdot \rfloor$: **scaling**한 값을 반올림(가장 가까운 정수값으로 변환)
 - 매우 가까운 두 값이 같은 정수값으로 변할 수 있음
- 범위값의 양 끝이 똑같다
 - $[-\text{absmax}, \text{absmax}]$
 - Symmetric Quantization이라 부르기도 한다



01.1) 텐서를 양자화하는 두가지 방법 (4 / 9)



01.1) 텐서를 양자화하는 두가지 방법 (5 / 9)

- Zero Point Quantization

$$nd_{x_{f16}} = \frac{2 \cdot 127}{\max_{ij}(\mathbf{X}_{f16}^{ij}) - \min_{ij}(\mathbf{X}_{f16}^{ij})}$$

- scaling 값 구하기
 -  : \mathbf{X}_{f16} 값의 범위, 최댓값과 최솟값 사용
 -  : int8 타입의 범위
- 범위 값의 양 끝 값이 다르다
 - [min, max]
 - Asymmetric Quantization이라고도 부른다

01.1) 텐서를 양자화하는 두가지 방법 (6 / 9)

- Zero Point Quantization

$$zp_{x_{i16}} = \lfloor \mathbf{X}_{f16} \cdot \min_{ij}(\mathbf{X}_{f16}^{ij}) \rfloor$$

- \mathbf{X}_{i8} 의 중심이 int8 범위에서 0에 놓기 위한 조정값
- \mathbf{X}_{i8} 에 바로 더하지 않음
 - GPU처럼 int16 곱셈이 되지 않는 경우를 위함

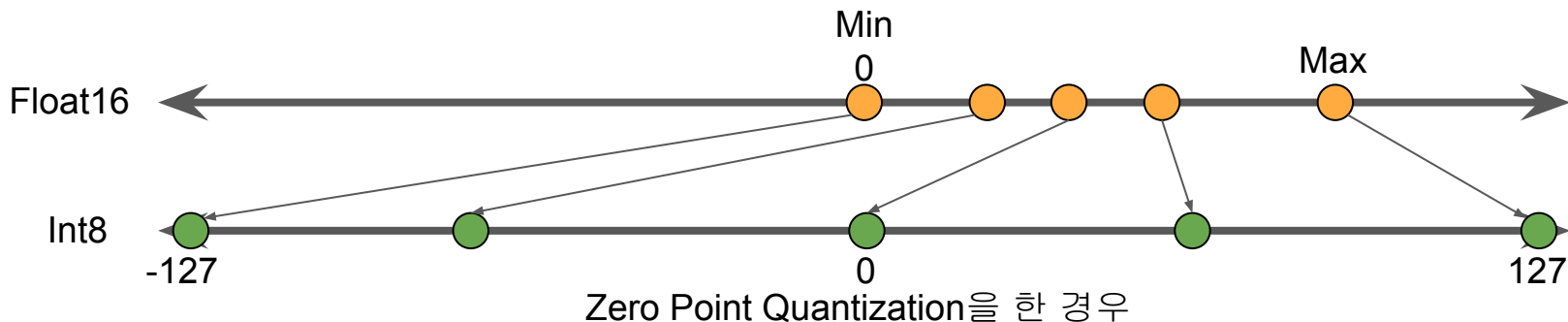
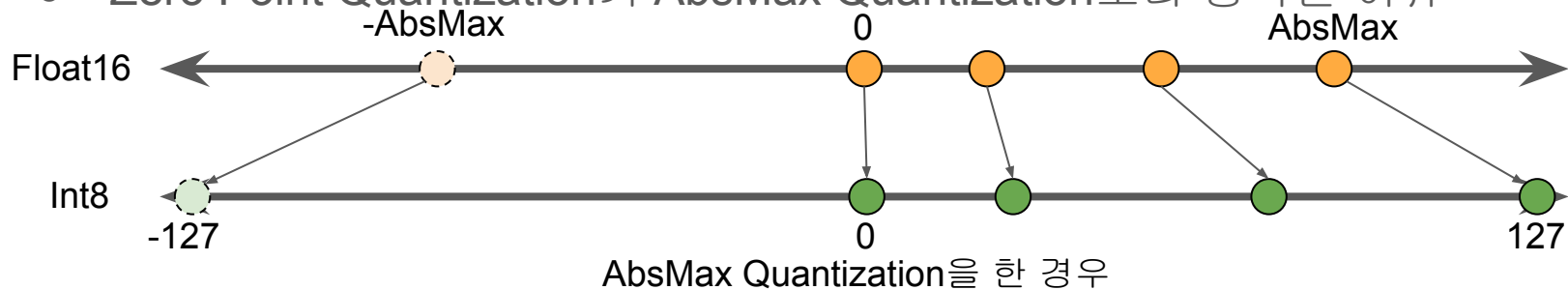
$$\mathbf{X}_{i8} = \lfloor nd_{x_{f16}} \mathbf{X}_{f16} \rfloor$$

01.1) 텐서를 양자화하는 두가지 방법 (7 / 9)

- AbsMax Quantization
 - 범위가 대칭(Symmetric)
 - 연산이 간단함
 - 정확도가 비교적 낮음
 - 실제로 자주 사용됨
- Zero Point Quantization
 - 범위가 비대칭(Asymmetric)
 - 연산이 복잡함
 - scaling 상수 뿐만 아니라 zero point까지 계산
 - 정확도는 더 높음

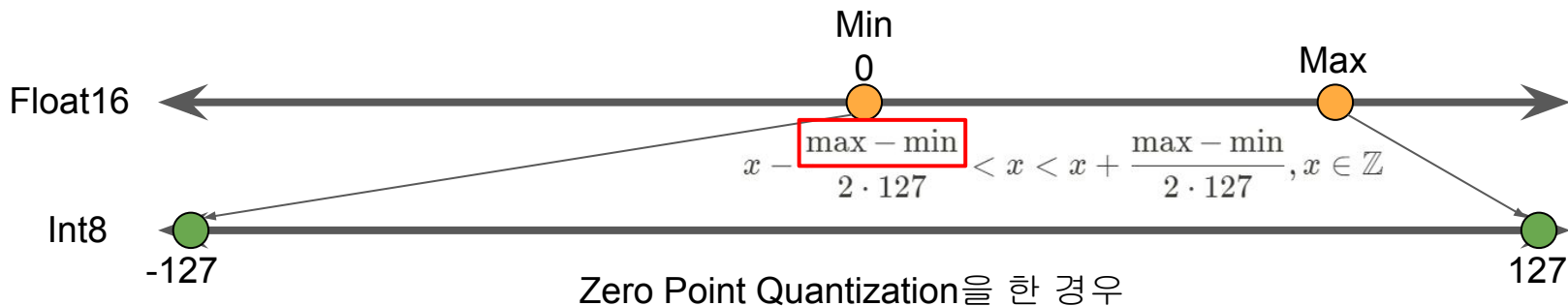
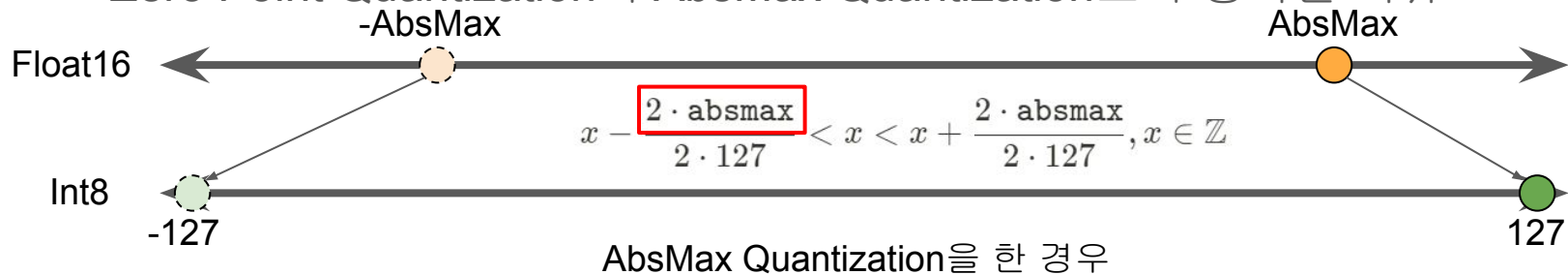
01.1) 텐서를 양자화하는 두가지 방법 (8 / 9)

- Zero Point Quantization가 AbsMax Quantization보다 정확한 이유



01.1) 텐서를 양자화하는 두가지 방법 (9 / 9)

- Zero Point Quantization가 AbsMax Quantization보다 정확한 이유



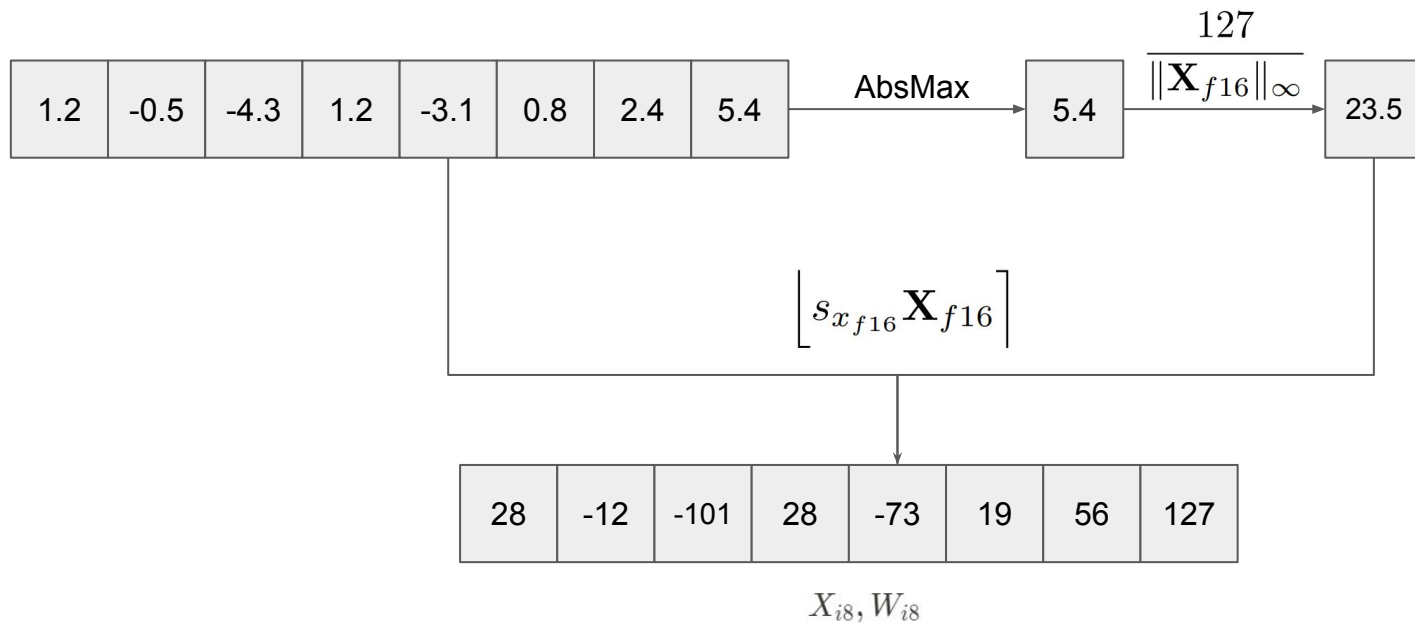
01.2) 양자화된 모델로 예측하는 방법 (1 / 4)

$$\begin{aligned}\mathbf{X}_{f16}\mathbf{W}_{f16} = \mathbf{C}_{f16} &\approx \frac{1}{c_{x_{f16}}c_{w_{f16}}} \mathbf{C}_{i32} = S_{f16} \cdot \mathbf{C}_{i32} \\ &\approx S_{f16} \cdot \mathbf{A}_{i8}\mathbf{B}_{i8} = S_{f16} \cdot \boxed{Q(\mathbf{A}_{f16})} \boxed{Q(\mathbf{B}_{f16})}\end{aligned}$$

- 입력 텐서와 가중치 텐서 모두 양자화
 - 각 텐서값의 범위를 int8 범위로 축소
- 양자화된 입력 텐서와 가중치 텐서 곱셈 계산
- 결과 텐서를 다시 역양자화
 - 본래 범위로 확장

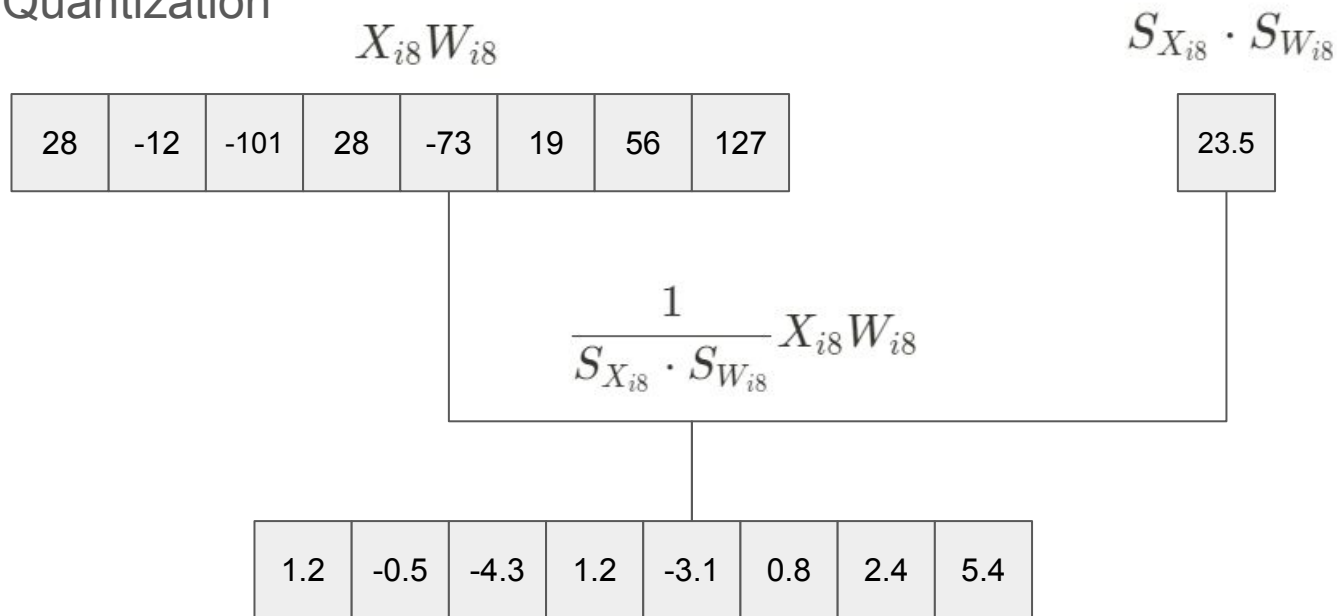
01.2) 양자화된 모델로 예측하는 방법 (2 / 4)

- AbsMax Quantization



01.2) 양자화된 모델로 예측하는 방법 (3 / 4)

- AbsMax Quantization



01.2) 양자화된 모델로 예측하는 방법 (4 / 4)

- Zero Point Quantization

$$C_{i32} = \text{multiply}_{i16}(A_{zp_{a_{i16}}}, B_{zp_{b_{i16}}}) = (A_{i8} + zp_{a_{i16}})(B_{i8} + zp_{b_{i16}})$$

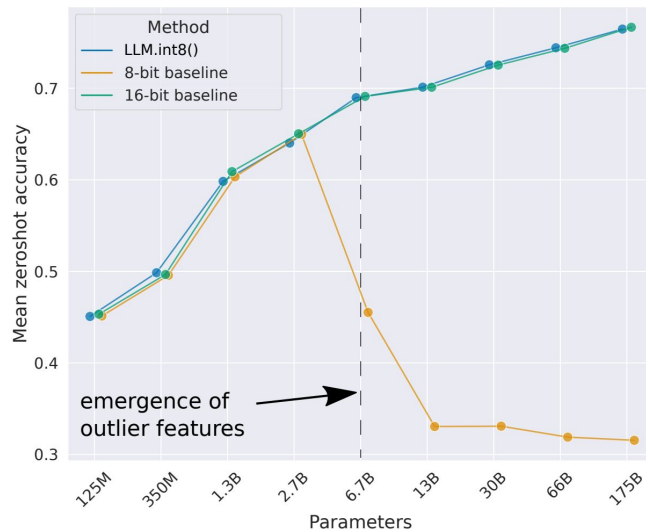
- Int16 곱셈을 지원하지 않는 연산장치 활용

$$C_{i32} = \underbrace{A_{i8}B_{i8}}_{\text{Int8 곱셈}} + \underbrace{A_{i8}zp_{b_{i16}} + B_{i8}zp_{a_{i16}} + zp_{a_{i16}}zp_{b_{i16}}}_{\text{Int16 곱셈}}$$

- int8로 양자화된 모델이 예측하는데 int16/32 연산을 지원해야한다
 - 지원하지 않으면 Int16 곱셈을 수행하는데 시간이 꽤 걸리게 됨

01.3) 기존 양자화 방식의 한계 (1 / 2)

- 지금까지 양자화 방법은 350M 크기 이하의 모델만 연구됨
 - 350M 이상의 모델을 성능 저하없이 양자화하는 연구가 부족
- 6.7B를 넘어가면 성능이 급격하게 떨어지기 시작
 - 계산 과정을 확인해보니 이상치(outlier)가 발견됨
 - 한 입력을 처리하는데 150,000개



01.3) 기존 양자화 방식의 한계 (2 / 2)

- 이상치는 우연히 발생한 현상인가
 - 한 시퀀스 처리에 생겨난 150,000개의 이상치가 모두 6개의 차원에서만 발생
 - 랜덤으로 발생했다고 하기엔 특정 **feature**에만 분포
- 이상치가 모델 추론에 얼마나 영향을 주는가
 - 이상치가 발생한 **feature** 차원을 모두 0으로 주고 계산하니 **validation perplexity**가 600-1000% 증가
 - 같은 개수의 차원을 랜덤으로 골라 0으로 주고 계산하니 **validation perplexity**가 0.1% 증가
 - => 이상치가 모델의 **perplexity**에 큰 영향을 줌
- 6.7B 이상의 모델을 양자화하면 이상치로 인해 성능이 떨어진다
 - 이상치로 인해 양자화할 때 범위가 넓어짐
 - 범위가 넓어지면 양자화 정확도가 낮아진다

02. LLM.int8()

두가지 주요 도전 과제

1. 높은 양자화 정밀도

기존 연구는 3억5천개 이하 모델만 연구되었다. 모델이 커짐에 따라 파라미터가 증가하였다.
10억 개 이상의 파라미터 규모에서 사용할 수 있는 더 높은 양자화 정밀도가 필요합니다.

2. 이상치 관리

67억 개 파라미터 규모부터 이상치가 발생하며, 이는 성능을 해치게 됩니다.
따라서 이상치를 관리하는 방법이 필요합니다.

02. LLM.int8()

두가지 주요 기능

1. Vector-wise Quantization(벡터별 양자화)

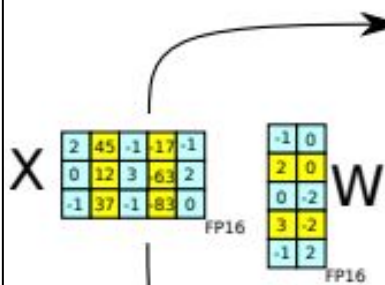
- 벡터별 양자화는 행렬 곱셈에서 각 내적을 독립된 연산으로 처리하는 방식입니다.
- 벡터별 양자화를 사용하면, 최대 27억 개의 매개변수 규모에서 성능을 유지할 수 있다는 것을 보여줍니다.
- 67억 파라미터부터 이상치가 발생하며 성능을 해치게 합니다.

2. Mixed Precision Decomposition(혼합 정밀도 분해)

- 혼합 정밀도 분해는 이상치에 대해 고정밀 곱셈을 적용하고, 나머지 값들에 대해서는 메모리 효율적인 8비트 가중치를 사용하는 방법입니다.
- 입력 행렬에서 이상치를 가진 차원은 0.1%가 되며, 해당 차원은 16비트로 처리됩니다.
- 나머지 99.9%의 값들은 메모리 절약을 위해 8비트로 처리됩니다.

02. LLM.int8() - 스키마

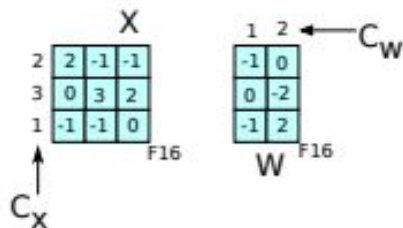
LLM.int8()



Regular values
Outliers

8-bit Vector-wise Quantization

(1) Find vector-wise constants: C_W & C_X



(2) Quantize

$$X_{F16} * (127/C_X) = X_{I8}$$
$$W_{F16} * (127/C_W) = W_{I8}$$

(3) Int8 Matmul

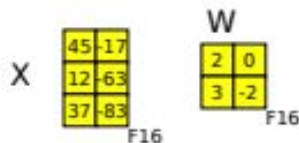
$$X_{I8} W_{I8} = Out_{I32}$$

(4) Dequantize

$$\frac{Out_{I32} * (C_X \otimes C_W)}{127 * 127} = Out_{F16}$$

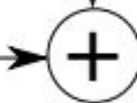
16-bit Decomposition

(1) Decompose outliers



(2) FP16 Matmul

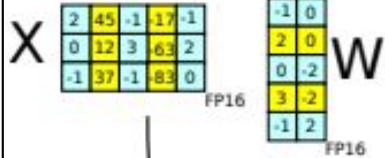
$$X_{F16} W_{F16} = Out_{F16}$$



Out_{F16}

02. LLM.int8() - 스키마

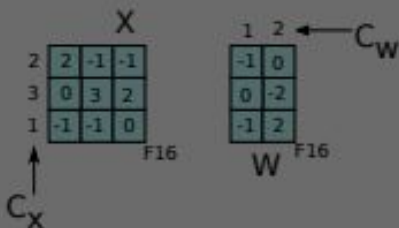
LLM.int8()



Regular values
Outliers

8-bit Vector-wise Quantization

(1) Find vector-wise constants: C_W & C_X



(2) Quantize

$$X_{F16} * (127/C_X) = X_{I8}$$
$$W_{F16} * (127/C_W) = W_{I8}$$

(4) Dequantize

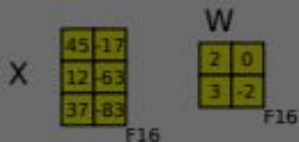
$$\frac{Out_{I32} * (C_X \otimes C_W)}{127 * 127} = Out_{F16}$$

(3) Int8 Matmul

$$X_{I8} W_{I8} = Out_{I32}$$

16-bit Decomposition

(1) Decompose outliers



(2) FP16 Matmul

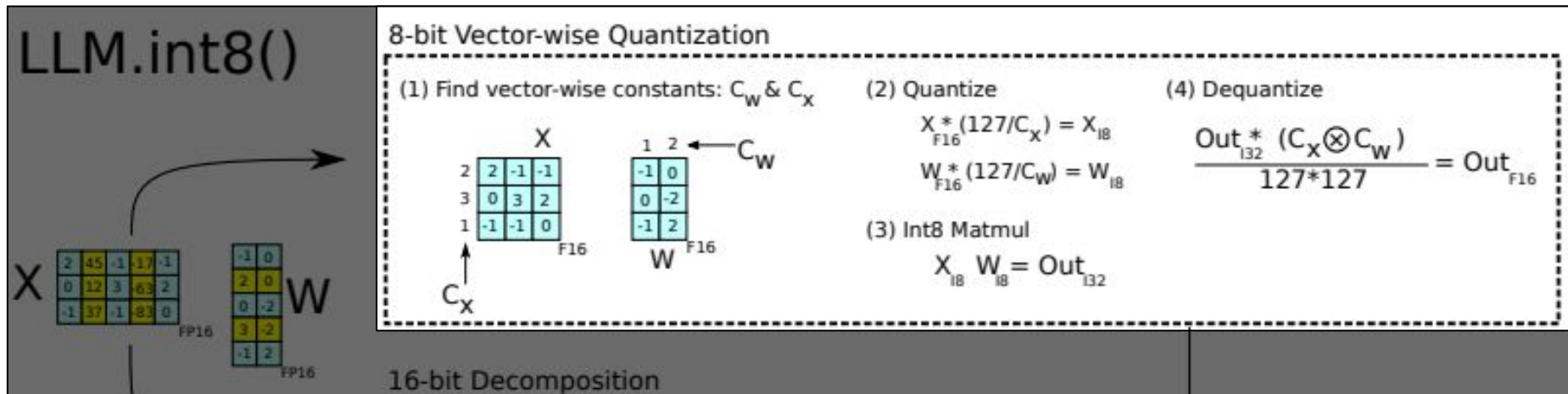
$$X_{F16} W_{F16} = Out_{F16}$$



X : 입력 행렬, W : 가중치 행렬

흰색 : 일반, 노란색 : 이상치

02. LLM.int8() - 스키마



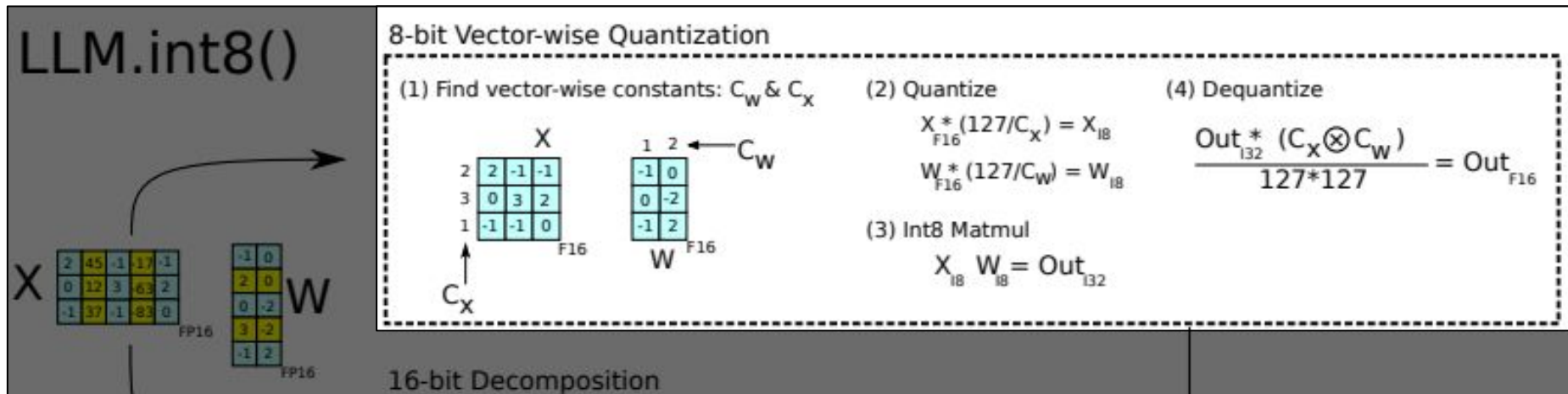
벡터별 양자화

- (1) 벡터별 상수s를 찾는다 (Float16)
- (2) 양자화 (Float16 to Int8)
- (3) int8 Matmul (Int8 to Int32)
- (4) 역양자화 (Float16)



상수s는 왜 찾을까?

02. LLM.int8() - 스키마



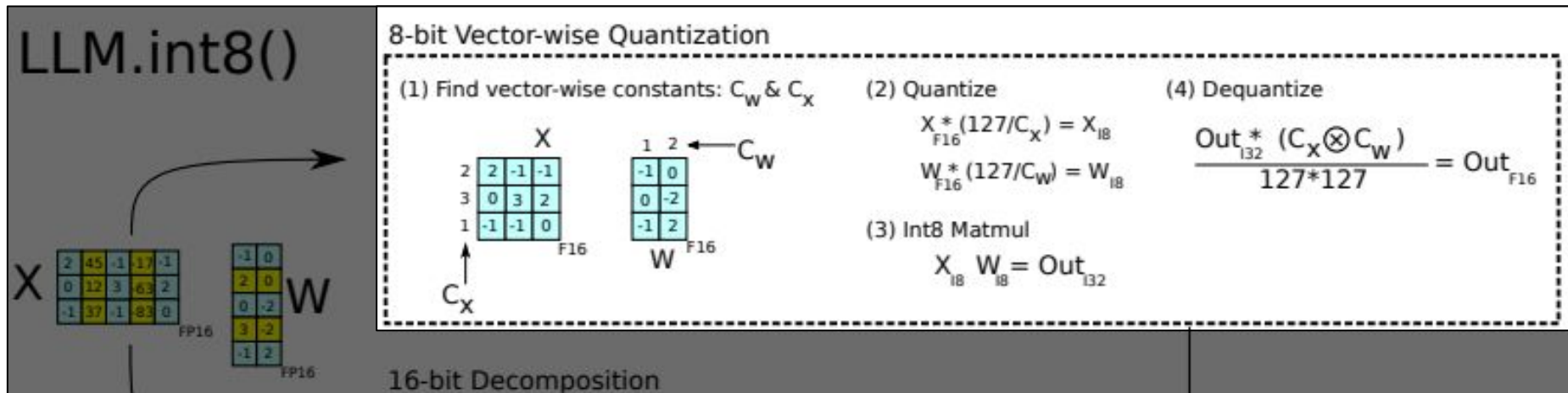
벡터별 양자화



상수s는 왜 찾을까?

- 텐서 당 단일 스케일링 상수를 사용하면 다른값들의 양자화 정밀도를 줄인다.
- 따라서, 텐서 당 여러 개의 스케일링 상수를 가지는 것이 바람직하며, 이상치의 영향이 각 블록에 제한한다.

02. LLM.int8() - 스키마



벡터별 양자화

- (1) 벡터별 상수를 찾는다 (Float16)
- (2) 양자화 (Float16 to Int8)
- (3) int8 Matmul (Int8 to Int32)
- (4) 역양자화 (Float16)

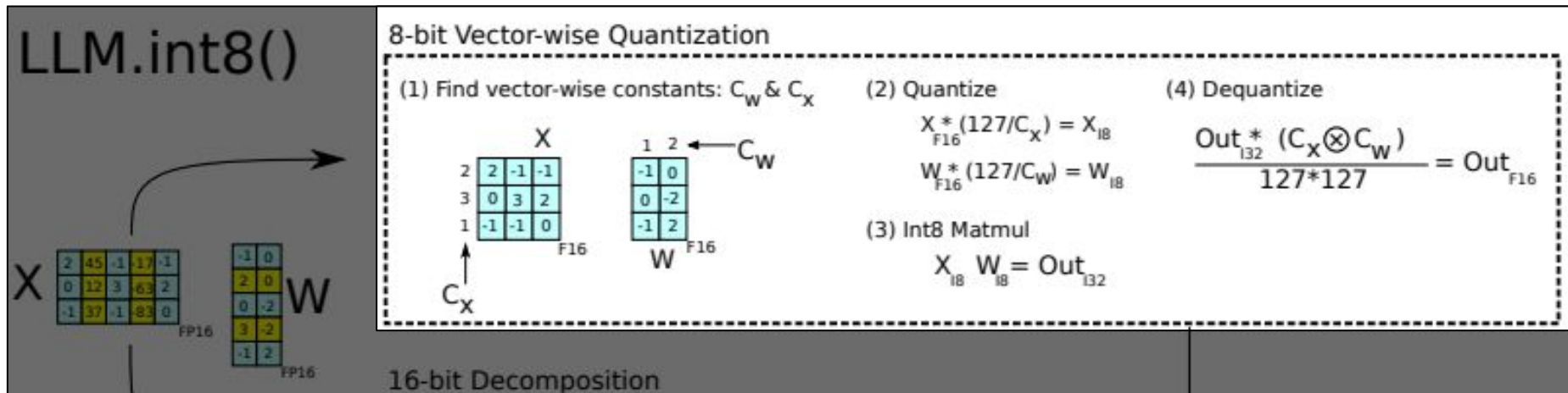
(2) Quantize

$$X_{F16} * (127/C_X) = X_{I8}$$

$$W_{F16} * (127/C_W) = W_{I8}$$

Absmax Quantization

02. LLM.int8() - 스키마



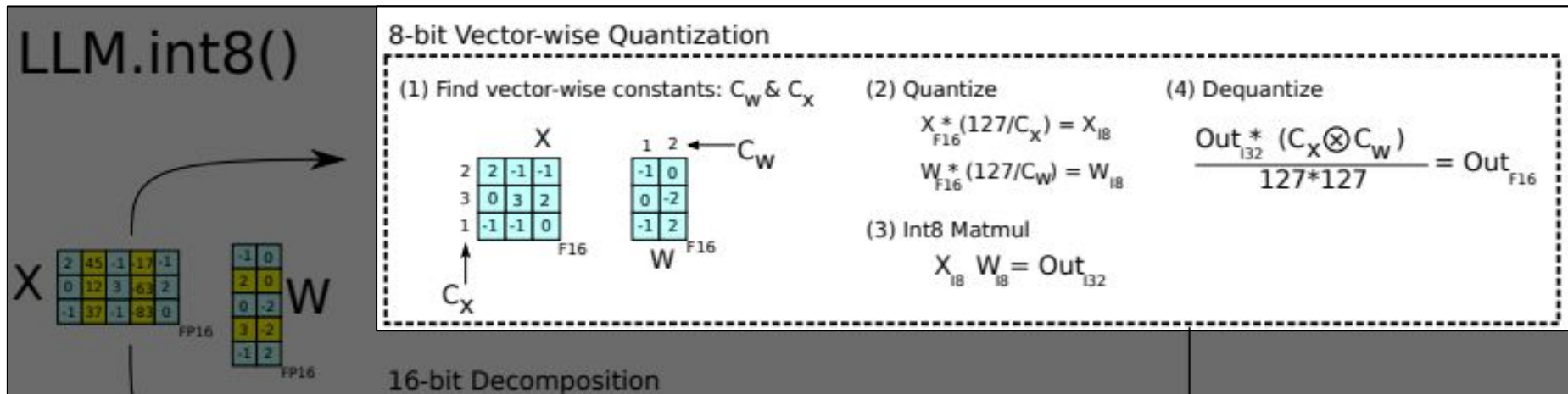
벡터별 양자화

- (1) 벡터별 상수를 찾는다 (Float16)
- (2) 양자화 (Float16 to Int8)
- (3) int8 Matmul (Int8 to Int32)
- (4) 역양자화 (Float16)

(3) Int8 Matmul

$$X_{I8} W_{I8} = Out_{I32}$$

02. LLM.int8() - 스키마



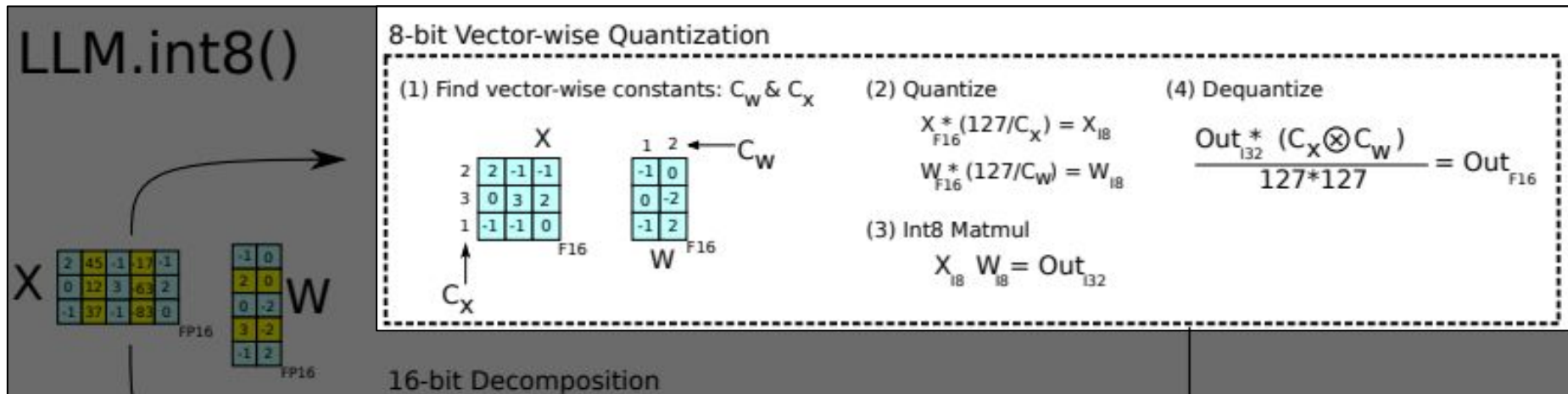
벡터별 양자화

- (1) 벡터별 상수를 찾는다 (Float16)
- (2) 양자화 (Float16 to Int8)
- (3) int8 Matmul (Int8 to Int32)
- (4) 역양자화 (Int32 to Float16)



역양자화는 왜 할까?

02. LLM.int8() - 스키마



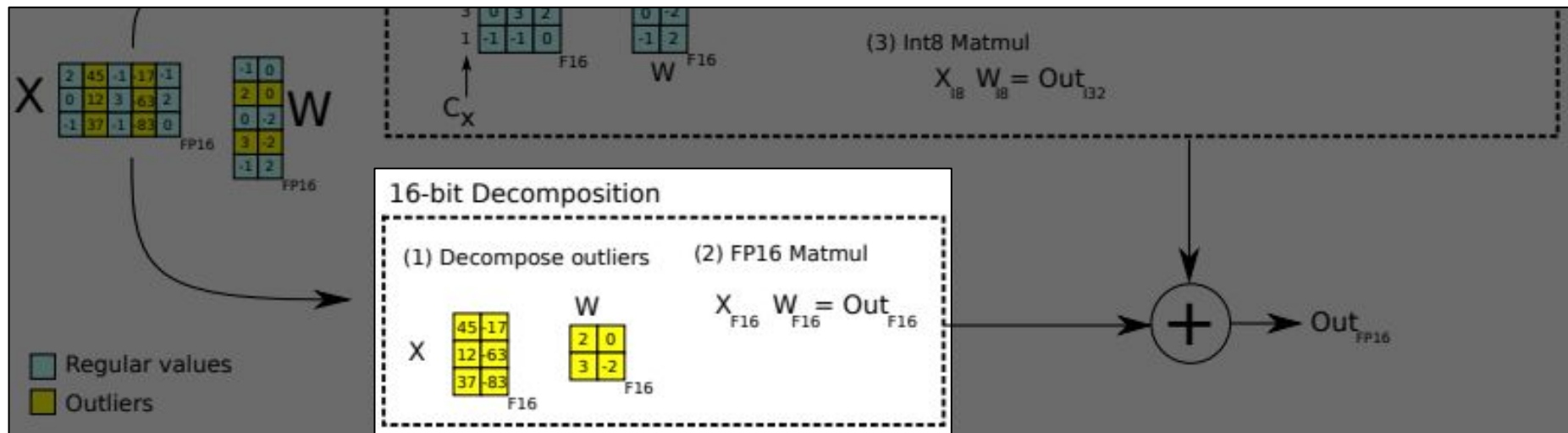
벡터별 양자화



역양자화는 왜 할까?

- 양자화 과정에서 데이터의 스케일을 동일하게 맞춰주기 위함.
- 2번과 3번 과정에서 양자화를 통한 연산은 마무리.
- 이상치는 FP16으로 처리되기에 맞춰주기 위함.

02. LLM.int8() - 스키마

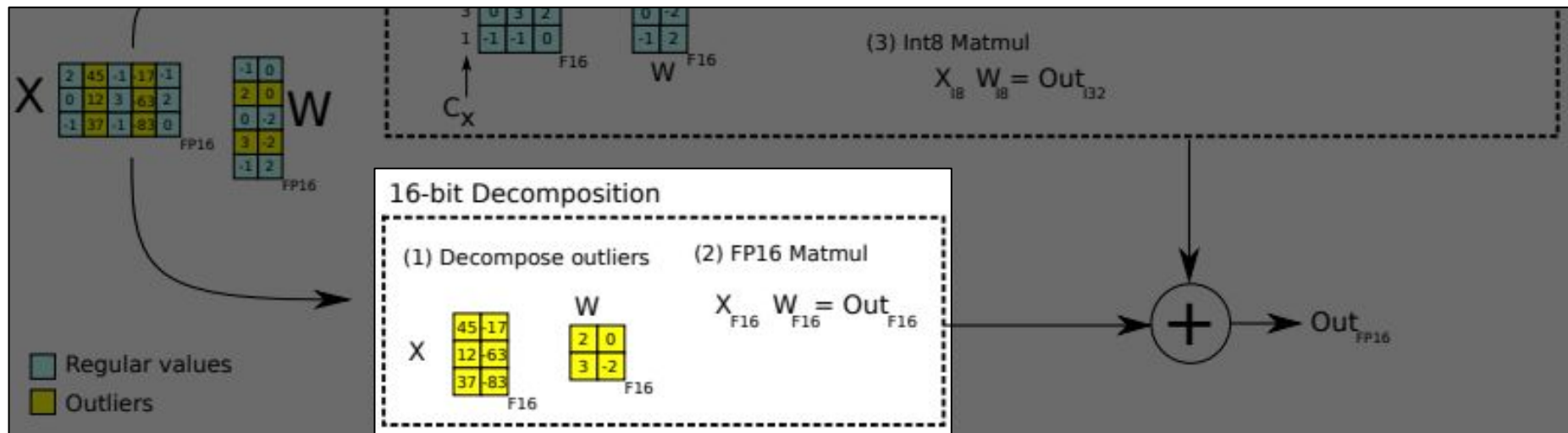


혼합 정밀도 분해

(1) 이상치 행렬 분해(Float16)

(2) FP16 Matmul(Float16)

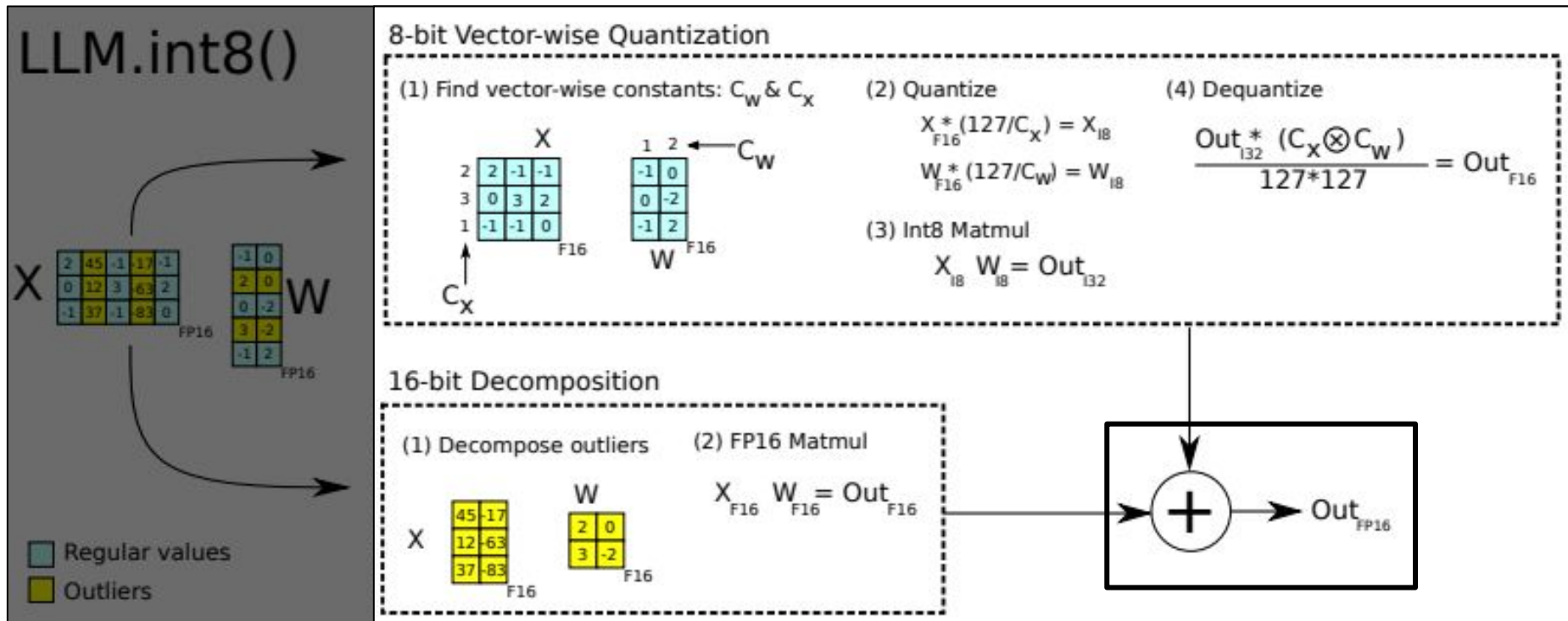
02. LLM.int8() - 스키마



혼합 정밀도 분해

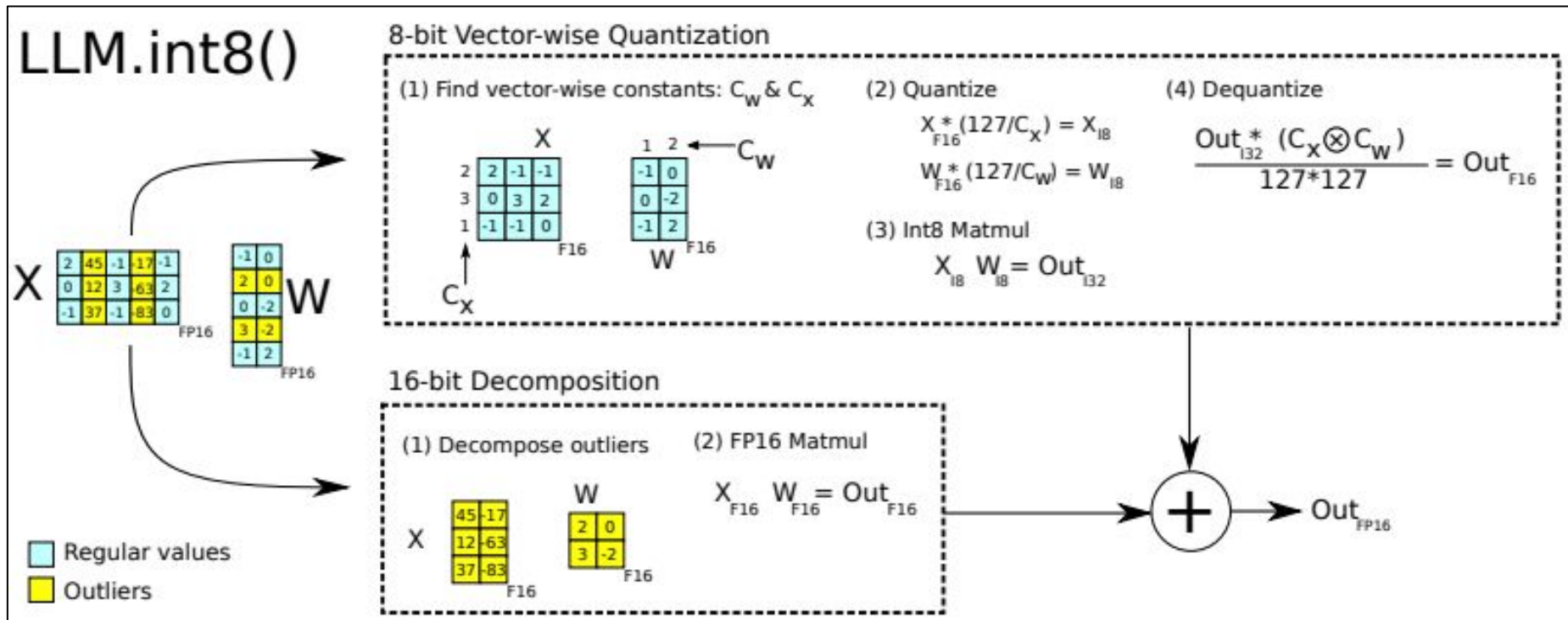
- (1) 이상치 행렬 분해(Float16)
- (2) FP16 Matmul(Float16)

02. LLM.int8() - 스키마



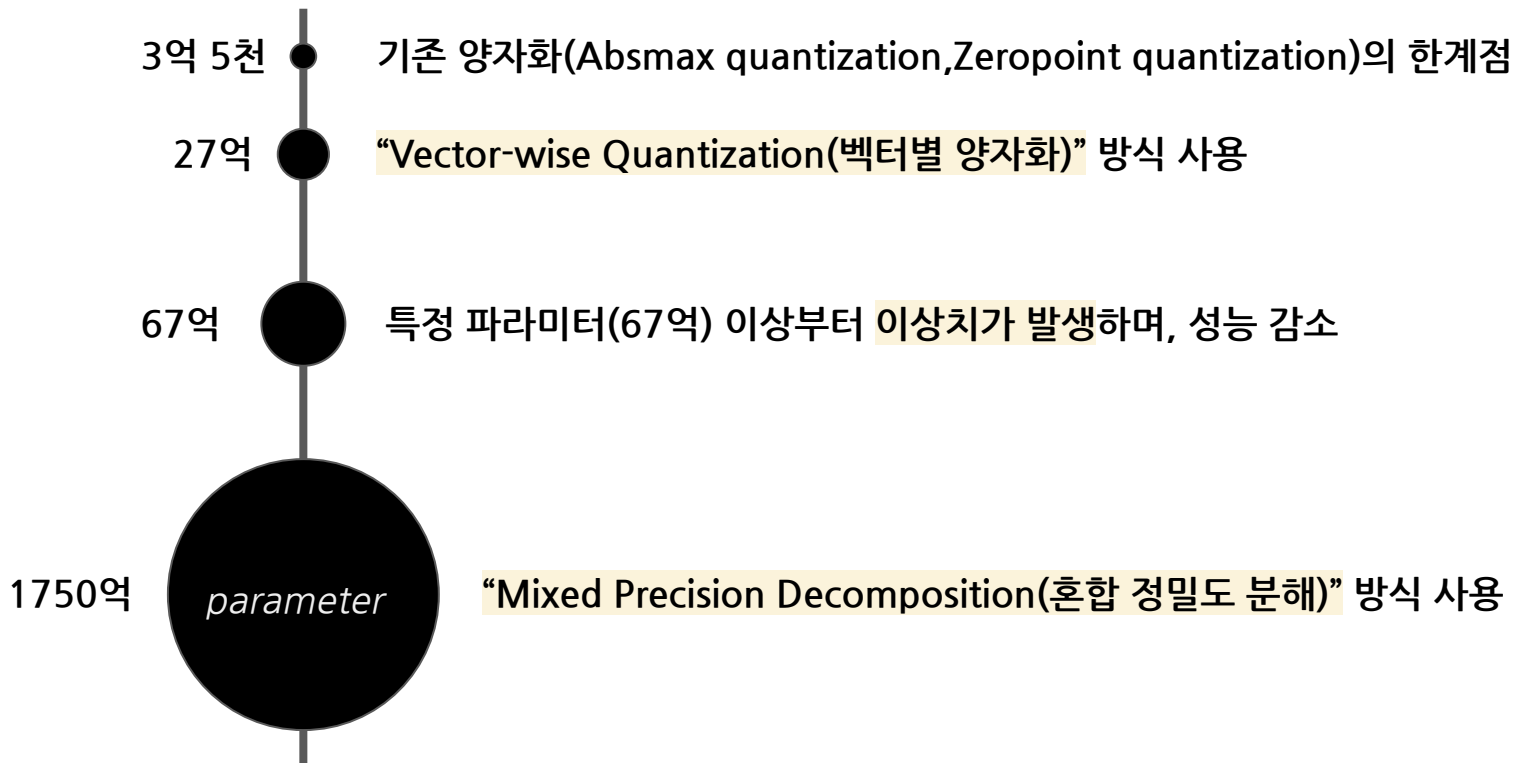
이상치와 일반 출력 모두가 16비트 부동소수점 출력에서 누적

02. LLM.int8() - 스키마

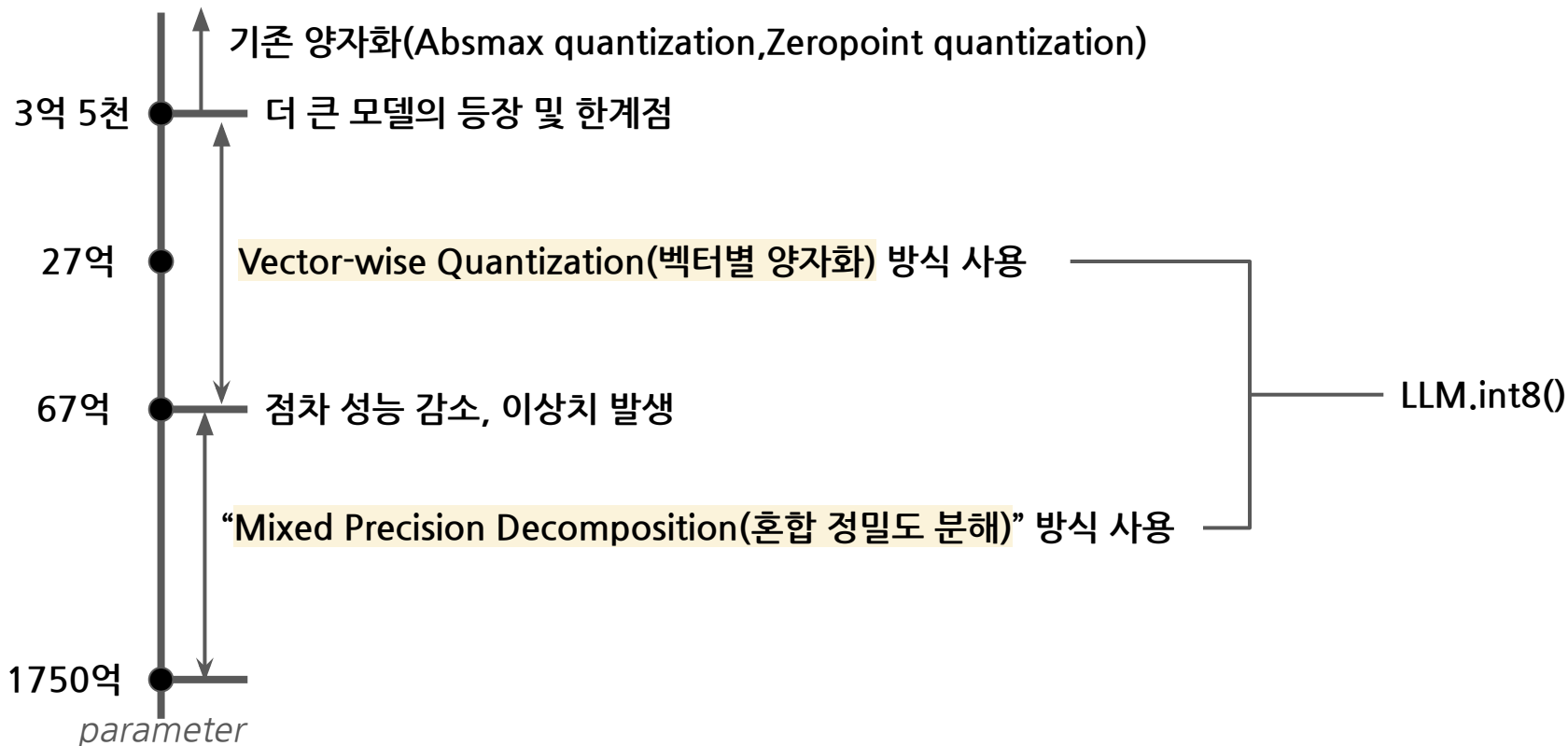


LLM.int8() = Vector-wise Quantization + Mixed-precision Decomposition

02. LLM.int8() - Flow



02. LLM.int8() - Flow



02. LLM.int8()

개선된 사항

1. 기존 양자화 방식보다 더 많은 파라미터에서 양자화가 가능!

- 기존 양자화 방식은 약 3억 5천개 파라미터까지만 가능
- 벡터별 양자화 방식을 통해 약 27억개 파라미터까지 성능 유지 가능
- 혼합 정밀도 분해 방식을 통해 약 1750억개 파라미터까지 성능 유지 가능

2. 속도 향상!

- 토큰당 추론을 T5-3B의 경우 312ms에서 173ms로, T5-11B의 경우 45ms에서 25ms로 향상할 수 있었습니다
- LLM.int8()은 향후 릴리스의 소형 모델에 대해 여전히 더 빨라질 가능성이 높습니다

03. 이상치의 정의



저자

- Feature의 magnitude 6.0 이상
- Layer의 25% 이상에 영향
- Sequence의 dimension의 6% 이상에 영향

03. 이상치의 영향 측정

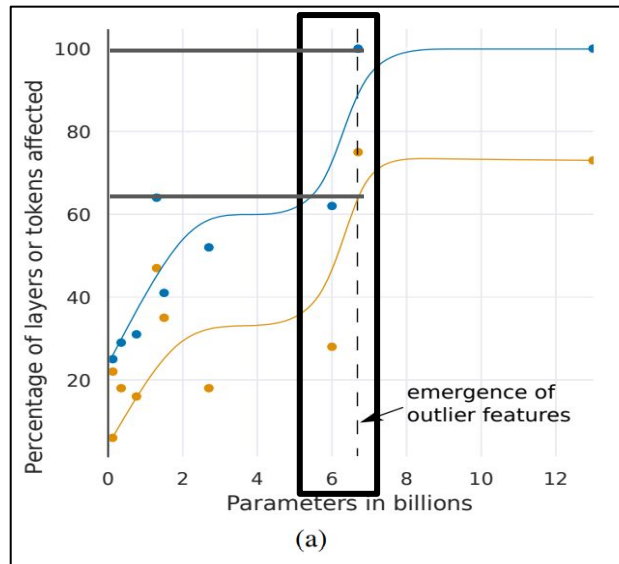
주요 포인트 4가지

1. 파라미터 수로 측정시, 이상치는 6B-6.7B 파라미터 사이에서 급격히 발생
2. 혼란도(perplexity, PPL)로 측정시, 이상치는 지수 함수에 따라 부드럽게 발생
3. Transformer의 모든 계층에서 이상치 발생 후 이상치 특성의 중앙값 크기가 급증
4. 이상치 특징들의 수는 혼란도가 감소함에 따라 엄격하게 단조롭게 (monotonically) 증가하는 반면, 모델 크기와의 관계는 비단조적 (non-monotonically)

03. 이상치의 영향 측정

1. 파라미터 수로 측정시, 이상치는 6B-6.7B 파라미터 사이에서 급격히 발생

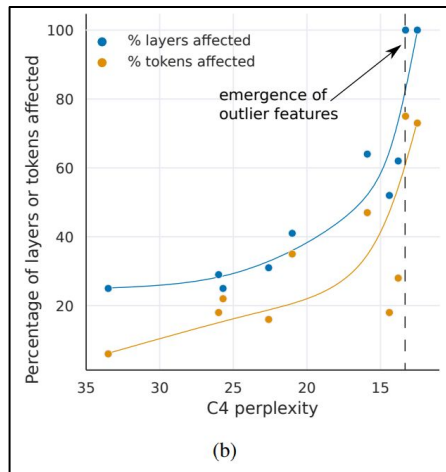
- 영향을 받는 layer 비율 65 → 100%로 급증
- 영향을 받는 sequence dimension 비율 35 → 75%로 급증
- 양자화가 제대로 적용되지 않는 구간부터 발생



03. 이상치의 영향 측정

2. PPL로 측정시, 이상치는 지수 함수에 따라 부드럽게 발생

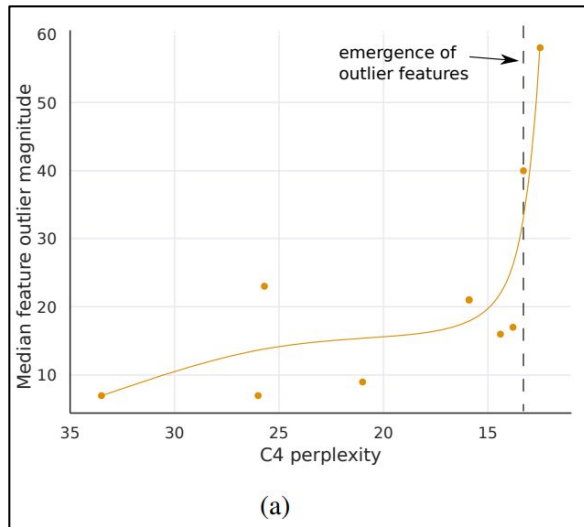
- 대규모 특성 발생이 갑자기 나타나는 것이 아님
- 대규모 특성 발생이 모델 크기 뿐만 아니라 PPL에도 영향 받는 것을 나타냄
 - -> 학습 데이터의 양 및 질 역시 영향을 줌



03. 이상치의 영향 측정

3. Transformer의 모든 계층에서 이상치 발생 후 이상치 특성의 중앙값 크기가 급증

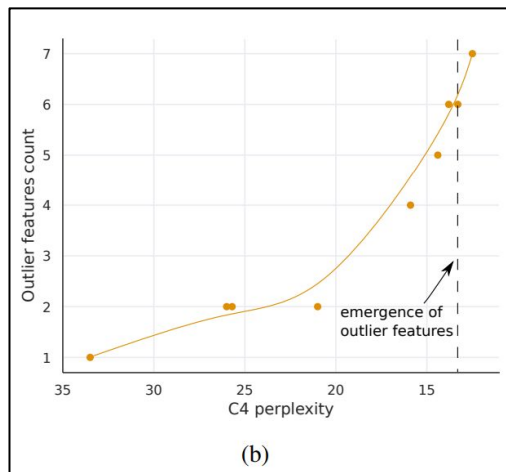
- 이상치의 크기와 불균형한 분포가 Int8 양자화의 정밀도 (precision)를 낮춤
 - 양자화가 6.7B 파라미터 수에서부터 제대로 적용되지 않는 이유
 - 양자화의 분포가 너무 넓어 정보가 사라지는 효과



03. 이상치의 영향 측정

4. 이상치 특징들의 수는 혼란도가 감소함에 따라 엄격하게 단조적으로 증가하는 반면, 모델 크기와의 관계는 비단조적

- 모델 크기보다는 모델 PPL이 이상치 증가를 좌우함
 - 모델 크기는 이상치 발생에 영향을 미치는 공변량 (covariate) 중 하나에 불과



04. 결론

- 큰 transformer의 특정 feature dimension에서 이상치가 편재 (ubiquitous)
 - 해당 feature dimension들이 transformer 성능에 중요
- row-wise / vector-wise 양자화는 이상치에 대한 효과적인 대응이 어려움
 - 해당 양자화는 은닉 sequence dimension인 행 (s)을 스케일링하는 반면, 이상치는 feature dimension인 열 (h)에서 발생
 - absmax 양자화가 대량 이상치 발생 후 제대로 작동하지 않는 이유
- 대다수의 이상치는 불균형한 분포를 가짐
 - zeropoint 양자화가 효과적
 - 파라미터 수 13B 부터는 축적된 양자화 오류와 이상치 크기의 급증으로 인해 zeropoint도 제대로 작동하지 않음
- LLM.int8()을 혼합 정밀도 분해와 함께 사용 시 zeropoint 양자화의 이점이 사라짐
 - 분해 후 남은 feature은 균형 분포
- 그럼에도 vector-wise 양자화는 row-wise 양자화에 비해 이점을 가짐
 - 가중치 양자화 정밀도의 향상이 예측의 정밀도를 높이는데 필요

감사합니다