Numerical Representation. 32-bit stream

Integers

2^{N+1}, by 2's complement. (Eg# $2^{9999} \approx 512 \times (2^{30})^{333} = 2^{9999}$

N, 9 → significant part

most significant value

Floating point numbers

31 30 — 8-bit — 23 0

sign, exponent, fraction

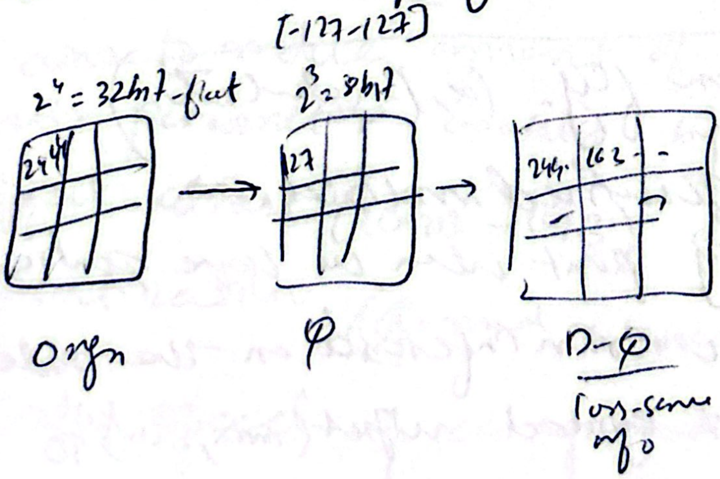
$85.612 = 8\times10^{1} + 5\times10^{0} + 6\times10^{-1}, 1e^{-2}, 2e^{-3}$

→ Modern GPU have 16-bit floating number, less precision

Quantization details

→ In NN linear layer have two matrices weights & Biase, represented by floating point NO, we aim to represent it by integers. $(Y = XW + B)$ int32

→ We aim to find reversible mapping of float ⇌ int such that the performance remain same



Types: Symmetric vs Asymmetric

0 ↓ mapped to 0 always	0 ↓ different
[-127, +127]	[0, 255]
8, bit	
if input is symmetric	if input is Asymmetric.

Asymmetric Quantization

$$x_q = \text{clamp}\left(\left\lfloor \frac{x_f}{s} \right\rceil + z; 0; 2^{n-1}\right), \quad s = \frac{\overset{\text{big}}{\alpha} - \overset{\text{small}}{\beta}}{2^n - 1} \quad z = \overset{\text{round off}}{\left\lfloor -1 \times \frac{\beta}{s} \right\rceil}$$

dequantize,

$$x_f = s(x_q - z)$$

Symmetric Quantization

$$x_q = \text{clamp}\left(\left\lfloor \frac{x_f}{s} \right\rceil; \overset{-127}{-(2^{n-1}-1)}; \overset{127}{2^{n-1}-1}\right), \quad s = \frac{\text{abs}(\alpha)}{2^{n-1}-1}$$

$$x_f = s\,x_q$$

- inputs = dynamic quantization on fly. (α, β calculate)
- Calibration = y = output of quantized matrices so how can we convert it floating point when we have quantized so, we do it by calibration. (we run inference on the model using few inputs and observes the typical output (max, min) to calculate the scale & the zeros. (Post-training Quantization)

rk, why?.)

[illegible]en adapting to specific task, LLMs have "low-intrinsic dimensions" and can still learn efficiently despite random projection to a smaller subspace. That's the hypothesis here.

(W contain many parameters that convey same information as other, we can get rid of them without decreasing performance. This kind of matrix = (rank-deficient)

→ (Rank = independent vectors)

→ Quantization.

The issue.

- Most LLM's have billions of parameter, if each parameter is 32 bit then for llama 2 (7B) = $\frac{7\times10^9\times32}{8\times10^9}$ = 28GB disk is require to store parameters or load in RAM (for inference)
- Just like human, PC are slow in computing floating point nos.

Solution = Quantization.

- it aims to reduce amount of bits required to represent each parameter (usually by converting float to int) 4bit almost (10GB-19B) depending on type of quantization. (round or truncate)
  -) also speed up computation (int) are fast to work with.

Advantages

→ less consumption when loading model.
→ less inference time due to same data type.
→ less energy consumption

LORA :- low-Rank-Adaption of LLM's

→ To fine tuned LLM (pretrained) on specific task.

→ By freezing the dates.

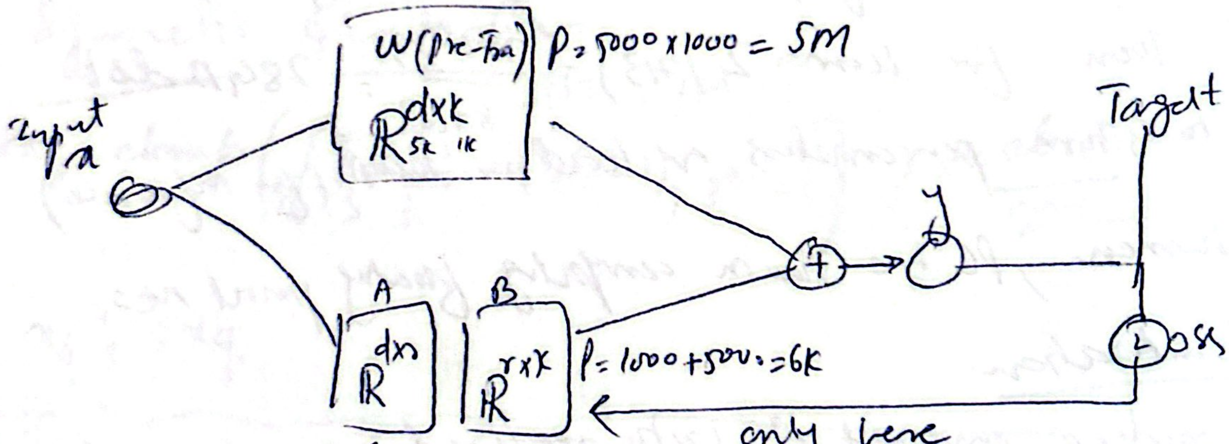
Problems in Full-Finetuning.

- computationaly expensive
- storage requirements for checkpoints are expensive.
- If we have multiple fined tuned, we have to load all weights

---

Fine-tuning.

- we want to product 2 matrix that multiply produce same result as W
- (B, A) is lower representation of W, although dimension are same but we louse some information and fine tune that are



$\because r = 1$

$r \ll \min(d, k)$

Benefits.

→ Less Parameters to train and store

→ less Parameter = less storage requirement

→ fast Backpropogation.

→ Faster switching b/w Finetuned models

How do we choose α, β ??

Min-Max :

$\alpha = \max(V)$
$\beta = \min(V)$
sensitive to outliers

Percentile :- set the range to the percentile of the distribution of V, to reduce sensitivity to outliers.

MSE :- choose (α, β) such that MSE is min b/w original & quantized
we usually use GridSearch

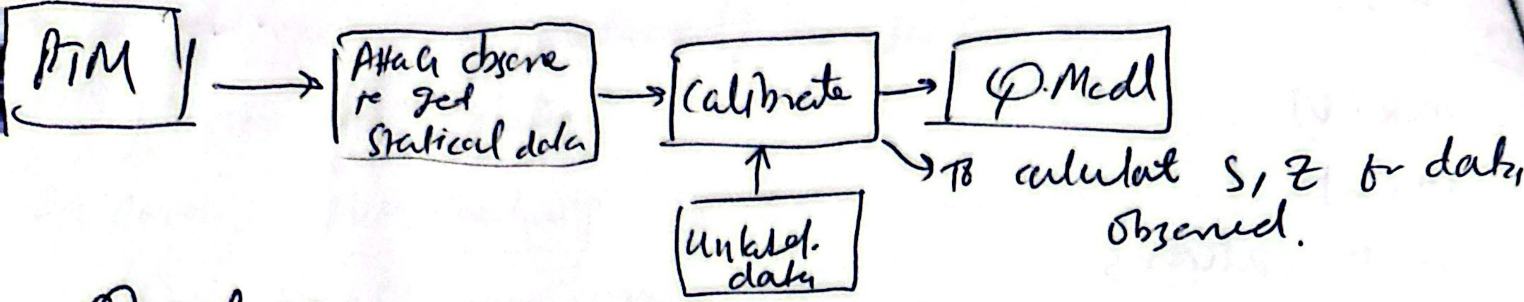
Cross-Entropy. when values in tensor are not important but the distribution is. (Big one should change, and other shouldn't also)
so, we choose α, β the way cross entropy b/w V and $\hat{V}$ is min

$$\underset{\alpha, \beta}{\arg\min}\ \text{CrossEntropy}\left(\left(\text{softmax}(V), \text{softmax}(\hat{V})\right)\right)$$

Quantization Granularity.

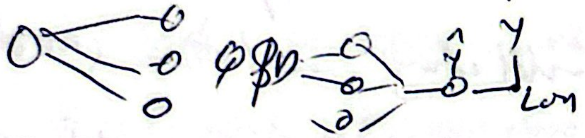
For CNNs,
we have many kernels of different size, so we loose some their quantization range if we used same α, β so we do CHANNEL WISE quantization, we calculate α, β for each channel.

Post-Training Quantization: (PTQ)



Quantization - Aware Training:

To make model more robust to quantization, we do fake quantization/dequantization b/w layers while training to introduce some error (quantization), so loss fun will more robust to this error



Train func quantized based on observer collected during training AO

QAT: Gradient. (BP should also calculate gradient w.r.t operation)

→ Gradient of Quantization (operations) are non-differentiable.

→ So Backpropagation Algorithm calculate by approximation of STE (Straight-Through Estimator).

α ... β = (Gradient = 1) b/w A & B

except z (Gradient = 0)

next:
GPTQ, AWQ

why it works?? effects?? on loss

The goal of QAT is to help model reach local minima which is more wide so that if weight after moves little the loss doesn't move a lot.