Quantization fundamentals

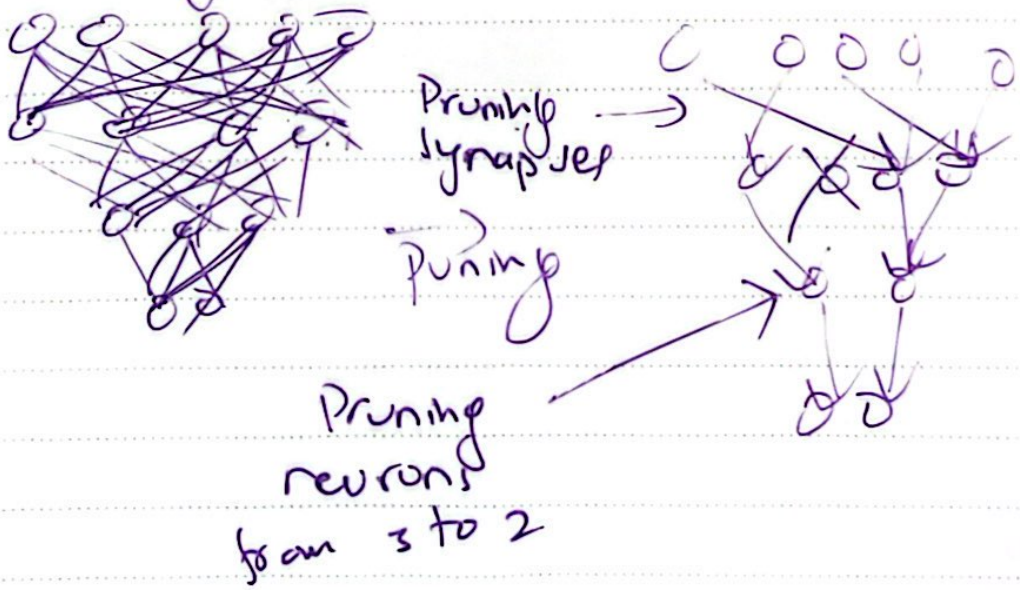
→ GPT → 80 GB (in 2022)

→ In 2023, 70B → 280 GB required (parameters)

→ Soln: Model compression

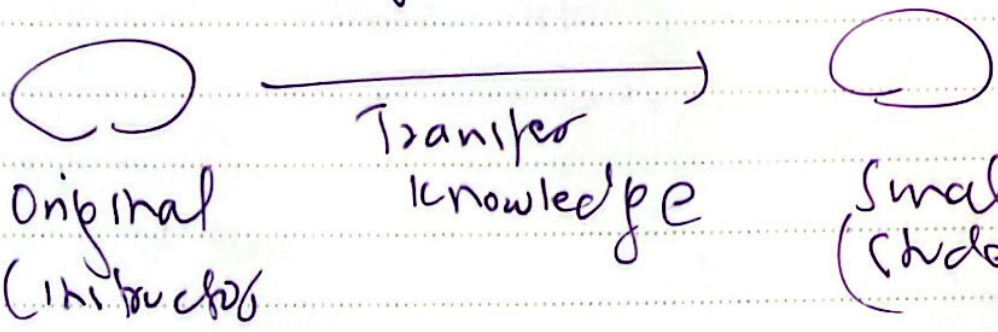
① Pruning:

→ Removing certain layers which are not an absolute necessity



(3) knowledge Distillation

→ train smaller model (student) from original (~~this~~ instructor)



→ need high compute power

(3) Quantization

→ store parameters of model in lower precision

13.5	14.3	8.5
-4.7	-3.2	-6.4
-0.4	1.3	3.3

(FP32 to INT8 to INT8)

13	4	8
-5	-3	-6
0	1	4

fp32: 4 bytes for each

total 36 bytes

1 Byte each

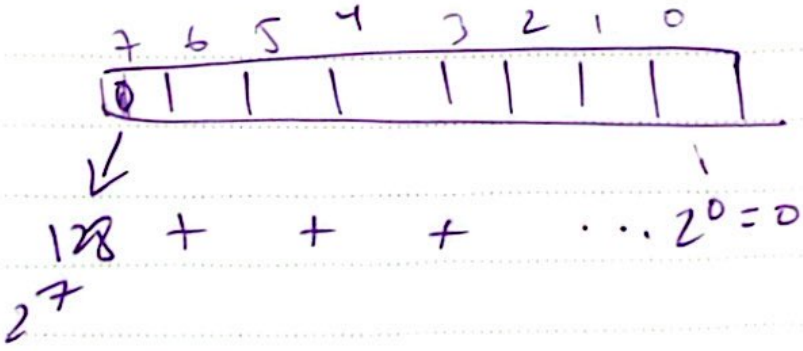
→ total total: 9 bytes

→ quantaization error, challenge is to lower this error

→ Quanto Model
→ for transformer model quantaization.

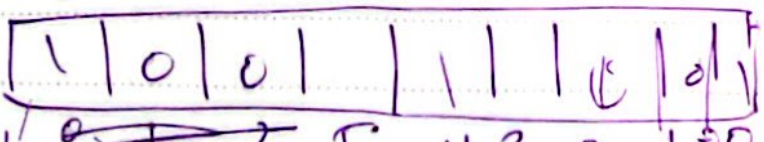
DataTypes

① Unsigned int8
→ 0 → $2^n - 1$
→ 8 bit → $0 - 2^8 - 1 =$ 255



② Signed int8
→ also represents negative
→ $[-2^{n-1}, 2^{n-1} - 1]$
range ...

8 bit



last bit ~~0~~ 1 when set 7 it will negative

for 8 bit

$2^{-7} = -128$

2's complement

Then add this ~~forward~~

in are case

$-128 + 0 + 0 + 0 + 2^3 + 0 + 0 + 2^0$

$= -119$

0 0 1 0 = 2

\+

1 1 1 0 = -2

=

1 0 0 0 = 0

Binary addition carry over.

In torch torch.int8 . int16
torch.uint8 . int32
. int64

torch.iinfo (torch.uint8)

3) floating point:

→ 3 components
(a) sign → positive & negative
1 bit
(b) Exponent (range)
→ impacts the representable range of number
(c) fraction (precision): impacts precision of the number

fp32 BF:16
→ sign: 1 bit
Exponent (range) 8 bit
fraction : 7 bit
total: 16 bit

→ very low & high
→ different formulas for decoding positive & negative values

torch.float16
bfloat16
float32
float64

	Precision	maximum
fp32 →	best	$\sim 10^{+37}$
fP16	better	$\sim 10^{04}$
Bf16	good	$\sim 10^{38}$

value = 1/3 → gonna bit 64 bit

t = torch.tensor(value, dtype=torch.float64)
(tensor node)

Downcasting

→ convert higher data type to lower
→ usually loss of data.

a = torch.rand(1000, dtype=torch.float32)
a[:5]
→ a.to(dtype=bfloat16)

torch.dot(d, ..)

Matrix Multiplication

- Result will be different (loss of data)

Pros: Reduced memory footprint
- → Efficient GPU memory
- → large batch size
- → Increase in speed

Cons: less precision.

Use case:

Mixed Precision training
- → computation in lower precision (FP16/BF16/FP8)
- → store & update weights in higher precision (FP32)