RESTORE LECTURE COMMAND

COMPUTE REQUIREMENTS

Basic equation: Cost to train a transformer (decoder) model:

$$C \approx \tau T = 6PD$$

▸ Source: Quentin et al., 2023

COMPUTE REQUIREMENTS $C \approx \tau T = 6PD$

where:

- τ is throughput of hardware: (No. GPUs) x (FLOPs/GPU)
- T is the time spent training the model, in seconds
- P is the number of parameters in the model
- D is the dataset size (in tokens)
- C: No. of floating-point operations (FLOPs) to train the model: $C = C_{forward} + C_{backward}$
- $C_{forward} \approx 2PD$; $\quad C_{backward} \approx 4PD$
  - $2PD$: **2** comes from the multiply-accumulate operation used in matrix multiplication
  - $4PD$: backward pass approximately twice the compute of the forward pass
- In the backward pass at each layer, gradients have to be calculated for the weights at that layer and for the previous layers' output, so that the gradient of the previous layers' weights can be calculated

COMPUTE UNITS

C can be measured in different units:

- FLOP-seconds which is [Floating Point Ops / Second]
  - We also use multiples:
    GFLOP-seconds, TFLOP-seconds etc.
  - Other multiples like PFLOP-days are used in papers
  - 1 PFLOP-day = $10^{15} \cdot 24 \cdot 3600$ FLOP-seconds
  - Actual FLOPs are always lower than the advertised theoretical FLOPs
- GPU-hours
  - GPU model is also required, since they have different compute capacities

PARAMETERS VS DATASET

- Model performance depends on number of parameters P, but also on number of training tokens D
- **We want to fix P and D, so that we get the best possible performance given an available compute budget.**
- One proposed optimal tradeoff between P and D is: $D = 20P$
  - This was true for Chinchilla models ▸ Hoffmann et al., 2022, but does not hold for all LLMs
- Training a LLM on less than 200 billion tokens is not recommended
- Rule of thumb: First determine the uppermost inference cost, and then train the biggest model within that boundary
- Different ways to determine P: based on available data, compute budget or inference time

MEMORY REQUIREMENTS

Common questions:

- How big is this model in bytes?
- Will it fit/train on my GPUs?

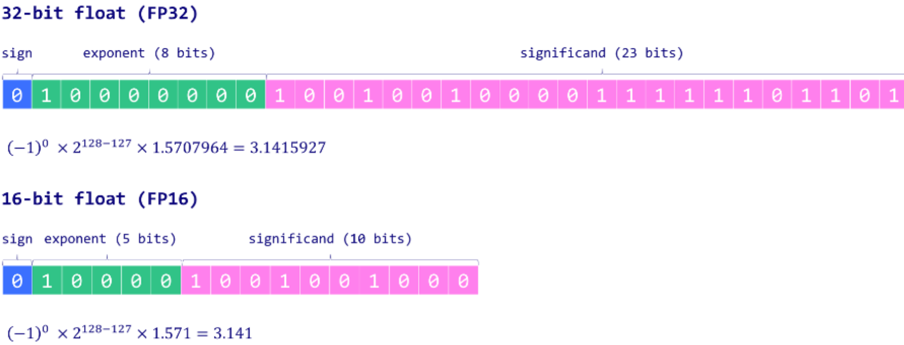
Model size components:

- Model parameters
- Optimizer states
- Gradients
- Activations

NUMBER REPRESENTATIONS

- Pure fp32: single precision floating point number as defined by ▸ IEEE 754 standard, takes 32 bits or 4 bytes
- fp16: half precision float number as defined by ▸ IEEE_754-2008, occupying 16 bits or 2 bytes
- bf16 or brain floating point 16, developed by Google Brain project, occupying 16 bits or 2 bytes
- int8: integer from -128 to 127, occupying 8 bits or 1 byte

FP32/FP16 (▸ (MAXIME LABONNE))



32-bit float (FP32)

0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 1 0 1 1 0 1

$(-1)^0 \times 2^{128-127} \times 1.5707964 = 3.1415927$

16-bit float (FP16)

0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0

$(-1)^0 \times 2^{128-127} \times 1.571 = 3.141$

You can represent every float like this: $(-1)^{S} \cdot 2^{e-bias} \cdot 1.m$

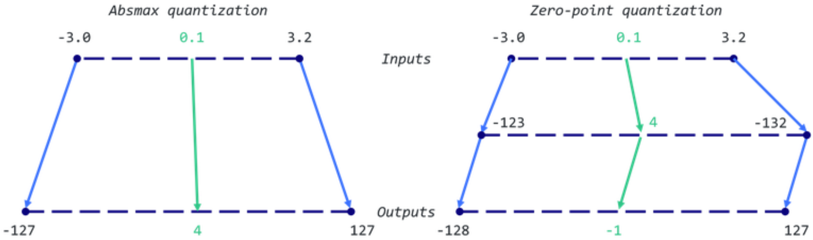
INT8 QUANTIZATION (1)

Since fp32 takes up too much memory, you can make use of int8 quantization to reduce the memory requirements

- Real_number = stored_integer * scaling_factor
- **Absmax quantization**:
  - $X_{quant} = \text{round}\left(\frac{127}{max|\mathbf{X}|} \cdot \mathbf{X}\right)$
- **Zero-point quantization**:
  - $\text{scale} = \frac{255}{max(\mathbf{X}) - min(\mathbf{X})}$
  - $\text{zeropoint} = -round(\text{scale} \cdot min(\mathbf{X})) - 128$
  - $X_{quant} = round(\text{scale} \cdot \mathbf{X} + \text{zeropoint})$

INT8 QUANTIZATION (2)

We map a number from a fp32 to an int8 representation:



Source: Maxime Labonne

These quantizations reduce the amount of memory used by the model, also they further speed up computation by enabling integer computation instead of floating-point math!

MODEL PARAMETERS

Parameter size depends on chosen representation:

- Pure fp32: $Mem_{model} = 4\ bytes/param \cdot N_{params}$
- fp16 or bf16: $Mem_{model} = 2\ bytes/param \cdot N_{params}$
- int8: $Mem_{model} = 1\ byte/param \cdot N_{params}$

It is common to use mixed representations:

- fp32 + fp16
- fp32 + bf16

OPTIMIZER STATES

AdamW: $Mem_{\text{AdamW}} = 8\ bytes/param \cdot N_{params}$

- Momentum: 4 bytes/param
- Variance: 4 bytes/param

bitsandbytes (8-bit optimizer): $Mem_{\text{optimizer}} = 2\ bytes/param \cdot N_{params}$

- Momentum: 1 byte/param
- Variance: 1 byte/param

GRADIENTS

They are usually stored in the same datatype as the model parameters.

Their memory overhead contribution is:

- fp32: $Mem_{grad} = 4\ bytes/param \cdot N_{params}$
- fp16 or bf16: $Mem_{grad} = 2\ bytes/param \cdot N_{params}$
- int8: $Mem_{grad} = 1\ byte/param \cdot N_{params}$

ACTIVATIONS

- GPUs are bottlenecked by memory, not FLOPs
- Save GPU memory by recomputing activations of certain layers
- Various schemes for selecting which layers to clear
- They take some extra memory, but save even more

Total memory when training **without** activations:
$Mem_{training} = Mem_{params} + Mem_{opt} + Mem_{grad}$

Total memory when training **with** activations:
$Mem_{training} = Mem_{params} + Mem_{opt} + Mem_{grad} + Mem_{activ}$