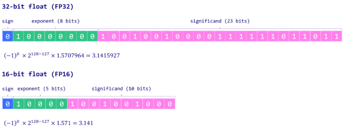
Deep Learning for NLP

Training LLMs
Memory and compute requirements



Learning goals

- Learn about different contributions to compute requirements
- Learn how model size components influence memory requirements

NUMBER OF PARAMETERS: NOTATION

- In this slide set, we use P for the number of parameters.
- (Unfortunately, we use N for the number of parameters in other slide sets.)

COMPUTE REQUIREMENTS

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 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' outputs

COMPUTE UNITS

C (actually, C per time) can be measured in different units:

- FLOPs = 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
- One proposed optimal tradeoff between P and D is: $D = 20P$
  - This was true for Chinchilla models ▸ Hoffmann et al., 2022, not clear to what extent it still holds.
- Training an LLM on less than 200 billion tokens is not recommended
- One 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

MEMORY REQUIREMENTS

Common questions:

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

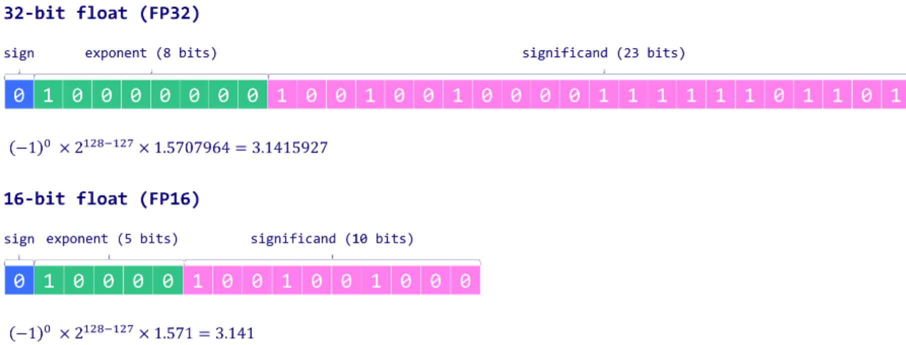
Model size components:

- 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
  - bf16 has a larger dynamic range than fp16 (8 exponent bits instead of 5)
  - bf16 is less precise, but also less likely to suffer from underflow or overflow
- INT8: integer from -128 to 127, occupying 8 bits or 1 byte

FP32/FP16 (▸ (MAXIME LABONNE))



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

INT8 QUANTIZATION

It's hard to fit meaningful floats into 8 bits, but you can use integer quantization: INT8.

- 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**:
  - $\text{scale} = \frac{255}{max(\mathbf{X}) - min(\mathbf{X})}$
  - $\text{zeropoint} = -round(\text{scale} \cdot min(\mathbf{X})) - 128$
  - $X_{quant} = round(\text{scale} \cdot \mathbf{X} + \text{zeropoint})$
  - (draw example)

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 how exactly this is implemented.

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

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