Calculating Memory Size and FLOPs for Fine-Tuning an LLM Transformer Model

Fine-tuning a Large Language Model (LLM) based on the Transformer architecture involves adapting a pre-trained model to a specific task or dataset. Understanding the memory requirements and computational cost (measured in Floating Point Operations, FLOPs) for fine-tuning is essential for efficient resource allocation and optimization. This guide provides a comprehensive approach to calculating the memory size and FLOPs needed for fine-tuning the example model previously discussed.

Table of Contents

Introduction

Fine-tuning leverages a pre-trained LLM to perform specific tasks by training it further on a targeted dataset. This process typically requires less computational resources compared to training a model from scratch but still demands careful consideration of memory and FLOPs to ensure efficient training.

Understanding Fine-Tuning

Fine-Tuning involves adjusting the weights of a pre-trained model on a new, often smaller, dataset. It can be performed in various ways:

- **Full Fine-Tuning**: All model parameters are updated during training.
- **Partial Fine-Tuning**: Only a subset of model parameters (e.g., the final layers) are updated.
- **Parameter-Efficient Fine-Tuning**: Techniques like adapters or prompt tuning add minimal parameters to the model, reducing computational overhead.

The choice of fine-tuning method impacts both memory requirements and FLOPs.

Components of Memory Usage During Fine-Tuning

The memory required for fine-tuning comprises:

1. **Model Parameters**: Weights of the neural network.
2. **Optimizer States**: Variables maintained by the optimizer (e.g., momentum terms in Adam).
3. **Gradients**: Computed during backpropagation.
4. **Activation Maps**: Intermediate outputs stored during the forward pass.
5. **Batch Data**: Input data for each training batch.
6. **Additional Buffers and Overheads**: Temporary variables and buffers used during computation.

Memory Size Calculation

Parameter Memory

- **Total Parameters ((P))**:
  - From previous calculations: 354,336,768 parameters.
- **Memory per Parameter**:
  - **32-bit (FP32)**: 4 bytes.
  - **16-bit (FP16/BF16)**: 2 bytes.
  - **8-bit (Int8 Quantization)**: 1 byte (if applicable).

Parameters Memory:

- **FP32**: $$ P \times 4 = 354,336,768 \times 4 = 1,417,347,072 \text{ bytes} \approx 1.32 \text{ GB} $$
- **FP16/BF16**: $$ P \times 2 = 354,336,768 \times 2 = 708,673,536 \text{ bytes} \approx 0.66 \text{ GB} $$
- **Int8 Quantization** (if used): $$ P \times 1 = 354,336,768 \times 1 = 354,336,768 \text{ bytes} \approx 0.33 \text{ GB} $$

Optimizer States Memory

Adam Optimizer typically requires two additional tensors per parameter (first and second moments).

- **Number of States**: 2 (m and v).
- **Memory per State**:
  - **FP32**: 4 bytes.
  - **FP16/BF16**: 2 bytes (less common; usually FP32 for stability).

Total Optimizer States Memory:

- **FP32**: $$ 2 \times 1.32 \text{ GB} = 2.64 \text{ GB} $$

- **FP16/BF16**: $$ 2 \times 0.66 \text{ GB} = 1.32 \text{ GB} $$

Note: Optimizer states are often kept in FP32 even when using lower precision for parameters.

Gradients Memory

Gradients require memory equivalent to the parameters.

- **FP32**: $$ 1.32 \text{ GB} $$
- **FP16/BF16**: $$ 0.66 \text{ GB} $$

Activation Memory

Activation memory depends on:

- **Batch Size ((B))**
- **Sequence Length ((L))**
- **Model Dimension ((d_{\text{model}}))**
- **Number of Layers ((N))**
- **Bytes per Element**

Calculation:

- **Per Layer Activation Memory**:
  - **Assuming an overhead factor ((\alpha))** (typically 3 to 5). $$ \alpha \times B \times L \times d_{\text{model}} \times \text{Bytes per Element} $$
- **Total Activation Memory**: $$ N \times \text{Per Layer Memory} $$

Total Memory Estimation

Total Memory: $$ \text{Parameters Memory} + \text{Gradients Memory} + \text{Optimizer States Memory} + \text{Activation Memory} + \text{Overheads} $$

Example:

- **Batch Size ((B))**: 8
- **Sequence Length ((L))**: 1,024
- **Model Dimension ((d_{\text{model}}))**: 1,024
- **Number of Layers ((N))**: 24
- **Bytes per Element**: 2 bytes (FP16/BF16)
- **Overhead Factor ((\alpha))**: 5

Parameters and Optimizer States Memory (FP16/BF16): $$ 0.66 \text{ GB} (\text{Parameters}) + 0.66 \text{ GB} (\text{Gradients}) + 2.64 \text{ GB} (\text{Optimizer States}) = 3.96 \text{ GB} $$

Activation Memory Calculation: $$ \text{Per Layer Memory} = 5 \times 8 \times 1,024 \times 1,024 \times 2 = 84,886,016 \text{ bytes} \approx 0.08 \text{ GB} $$ $$ \text{Total Activation Memory} = 24 \times 0.08 \text{ GB} = 1.92 \text{ GB} $$

Overheads: Approximately 1 GB.

Total Memory: $$ 3.96 \text{ GB} (\text{Parameters and States}) + 1.92 \text{ GB} (\text{Activations}) + 1 \text{ GB} (\text{Overheads}) \approx 6.88 \text{ GB} $$

FLOPs Calculation for Fine-Tuning

Fine-tuning involves both the forward and backward passes, similar to training from scratch. However, depending on the fine-tuning strategy, some FLOPs can be reduced (e.g., freezing layers).

Forward Pass FLOPs

Similar to inference:

- **Embedding Layer**
- **Multi-Head Attention (MHA)**
- **Feed-Forward Network (FFN)**
- **Layer Normalization**
- **Activation Functions**

Backward Pass FLOPs

The backward pass typically requires additional FLOPs to compute gradients:

- **Rule of Thumb**: Backward pass FLOPs ≈ 2 × Forward pass FLOPs.

Total FLOPs per Training Step

$$ \text{Total FLOPs} = \text{Forward Pass FLOPs} + \text{Backward Pass FLOPs} = 3 \times \text{Forward Pass FLOPs} $$

Note: If only a subset of layers is being fine-tuned, FLOPs can be reduced accordingly.

Example Calculation

Given

- **Batch Size ((B))**: 8
- **Sequence Length ((L))**: 1,024
- **Model Dimension ((d_{\text{model}}))**: 1,024
- **Feed-Forward Dimension ((d_{\text{ff}}))**: 4,096
- **Number of Heads ((h))**: 16
- **Number of Layers ((N))**: 24

Calculating FLOPs for One Training Step

1. **Forward Pass FLOPs**:

   - **Embedding Layer**: $$ B \times L \times d_{\text{model}} = 8 \times 1,024 \times 1,024 = 8,388,608 \text{ FLOPs} $$

   - **Per Layer FLOPs**:

        - **MHA**: $$ 8 \times 8 \times 1,024 \times (1,024)^2 + 4 \times 8 \times (1,024)^2 \times 1,024 + 5 \times 8 \times 16 \times (1,024)^2 $$ Simplifying: $$ 8 \times 8 \times 1,048,576 = 67,108,864 \text{ FLOPs} $$ $$ 4 \times 8 \times 1,048,576 = 33,554,432 \text{ FLOPs} $$ $$ 5 \times 8 \times 16 \times 1,048,576 = 671,088,640 \text{ FLOPs} $$ **Total MHA FLOPs per Layer**: $$ 67,108,864 + 33,554,432 + 671,088,640 = 771,751,936 \text{ FLOPs} $$

        - **FFN**: $$ 4 \times 8 \times 1,024 \times 4,096 + 8 \times 8 \times 1,024 \times 4,096 + 10 \times 8 \times 1,024 \times 1,024 $$ Simplifying: $$ 4 \times 8 \times 4,194,304 = 134,217,728 \text{ FLOPs} $$ $$ 8 \times 8 \times 4,194,304 = 268,435,456 \text{ FLOPs} $$ $$ 10 \times 8 \times 1,048,576 = 83,886,080 \text{ FLOPs} $$ **Total FFN FLOPs per Layer**: $$ 134,217,728 + 268,435,456 + 83,886,080 = 486,539,264 \text{ FLOPs} $$

        - **Layer Norms**: $$ 10 \times 8 \times 1,024 \times 1,024 = 83,886,080 \text{ FLOPs} $$

        - **Total FLOPs per Layer**: $$ 771,751,936 + 486,539,264 + 83,886,080 = 1,342,177,280 \text{ FLOPs} $$

    - **Total for All Layers**: $$ 24 \times 1,342,177,280 = 32,212,254,720 \text{ FLOPs} $$

    - **Total Forward Pass FLOPs**: $$ 8,388,608 + 32,212,254,720 \approx 32,220,643,328 \text{ FLOPs} $$

2. **Backward Pass FLOPs**: $$ 2 \times 32,220,643,328 = 64,441,286,656 \text{ FLOPs} $$

3. **Total FLOPs per Training Step**: $$ 32,220,643,328 + 64,441,286,656 = 96,661,929,984 \text{ FLOPs} \approx 96.66 \text{ billion FLOPs} $$

Additional Considerations

1. **Fine-Tuning Strategy**:

    - **Full Fine-Tuning**: All layers are updated, resulting in higher memory and FLOPs.
    - **Partial Fine-Tuning**: Only specific layers or modules are updated, reducing memory and FLOPs.
    - **Parameter-Efficient Fine-Tuning**: Techniques like adapters add minimal parameters, lowering memory and FLOPs.

2. **Batch Size Impact**:

    - **Linear Scaling**: Both memory and FLOPs scale linearly with batch size.
    - **Example**: Doubling the batch size doubles the FLOPs.

3. **Sequence Length Impact**:

    - **Quadratic Scaling in Attention**: FLOPs in the attention mechanism scale quadratically with sequence length.
    - **Practical Limits**: Longer sequences significantly increase computational requirements.

4. **Precision and Quantization**:

   - **Lower Precision**: Using FP16 or INT8 can reduce memory usage and potentially increase FLOPs efficiency.
   - **Trade-offs**: Precision reduction may impact model accuracy.

5. **Hardware Utilization**:

   - **Parallelism**: Effective use of GPUs/TPUs can mitigate high FLOPs through parallel processing.
   - **Memory Bandwidth**: Sufficient bandwidth is crucial to handle data movement without bottlenecks.

6. **Optimization Techniques**:

   - **Gradient Checkpointing**: Saves memory by recomputing certain activations during backpropagation.
   - **Mixed Precision Training**: Combines different numerical precisions to optimize performance and memory usage.
   - **Distributed Training**: Splits computation across multiple devices to handle larger models or batch sizes.

Conclusion

Fine-tuning an LLM Transformer model requires careful estimation of both memory size and FLOPs to ensure efficient and feasible training. For the example model with:

- **Batch Size ((B))**: 8
- **Sequence Length ((L))**: 1,024
- **Model Dimension ((d_{\text{model}}))**: 1,024
- **Feed-Forward Dimension ((d_{\text{ff}}))**: 4,096
- **Number of Heads ((h))**: 16
- **Number of Layers ((N))**: 24

Memory Requirements:

- **Total Memory**: Approximately **6.88 GB**

FLOPs Requirements:

- **Total FLOPs per Training Step**: Approximately **96.66 billion FLOPs**

These estimates assist in selecting appropriate hardware, optimizing training processes, and ensuring that fine-tuning tasks are conducted efficiently.

References

- Vaswani, A., et al. (2017). "Attention is All You Need".
- Micikevicius, P., et al. (2017). "Mixed Precision Training".
- Shoeybi, M., et al. (2019). "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism".
- Kaplan, J., et al. (2020). "Scaling Laws for Neural Language Models".

- OpenAI. (2020). "GPT-3 Technical Report".
- Rajbhandari, S., et al. (2020). "Zero: Memory Optimizations Toward Training Trillion Parameter Models".
- Patil, V., et al. (2021). "Efficient Transformers: A Survey".

Note: These calculations provide estimates. Actual memory usage and FLOPs may vary based on implementation details, optimization techniques, hardware architecture, and specific fine-tuning strategies.