

**Fine Tuning LLM Models – Revision Notes**

Here are concise notes to help you revise the key concepts and steps from the tutorial on fine-tuning Large Language Models (LLMs):

**1. Fine-Tuning Overview**

* **Fine-tuning** is adapting a pre-trained LLM (like Llama 2, Gemma) with specific, often smaller, datasets for custom use-cases.
* It's essential for deploying models on resource-constrained devices and for tasks requiring specialization.

**2. Core Concepts**

**a. Quantization**

* **Definition:** Reducing the precision of weights from higher (e.g., 32 bits) to lower formats (e.g., 8/4/1 bits) to speed up inference and reduce memory/RAM usage.
* **Benefits:** Enables running large LLMs on hardware with limited VRAM/RAM, lowers energy consumption, and speeds up inference.
* **Downside:** Some information loss, slight reduction in accuracy, especially at extreme quantization levels (1-bit).
* **Types:**
  + **Symmetric Quantization:** Values are centered around zero.
  + **Asymmetric Quantization:** Data is not centered, uses zero point for offset.
* **Methods:**
  + **Post-Training Quantization (PTQ):** Apply after training, may lose some accuracy.
  + **Quantization Aware Training (QAT):** Integrate during training to preserve performance after quantization.

**b. Calibration**

* The process of finding the appropriate scale and zero point to map floating point values to lower precision integers.

**c. Precision Terminology**

* **fp32:** 32-bit floating point (full precision).
* **fp16:** 16-bit (half precision).
* **int8, int4, int1:** Integer formats for further compression.

**3. Parameter Efficient Fine-Tuning (PEFT)**

* Efficient fine-tuning method focusing on updating a small subset of parameters.
* Key Techniques:
  + **LoRA (Low-Rank Adaptation):** Adds trainable rank-decomposed adapters into Transformer layers to reduce the number of trainable parameters drastically.
    - **Matrix Decomposition:** Decomposes large weight updates into smaller matrices of lower rank.
    - Less resource-intensive, quick to train.
    - Hyperparameter: **Rank (R)** - higher = more capacity, lower = fewer parameters.
    - Main equation:  
      $ W\_{fine-tuned} = W\_{pre-trained} + BA $  
      where B & A are small matrices.
  + **QLoRA:** LoRA technique combined with quantization (typically 4-bit).
* **Adapters:** Small layers or modules used in PEFT approaches; allows loading a single model and switching functionalities by changing adapters.

**4. Practical Fine-Tuning Workflow (as per tutorial)**

**Example: Hugging Face, Llama 2, Google Gemma**

1. **Install Required Libraries:**  
   transformers, bitsandbytes, peft, trl, etc.
2. **Load Pre-trained Model w/ Quantization:**
   * Pick a base model (e.g., Llama 2 7B, Gemma 7B).
   * Load with quantization (load\_in\_4bit=True).
   * Configure quantization (type, compute dtype).
3. **Prepare Data:**
   * Format data as instruction-response pairs.
   * Convert to model-specific prompt template.
4. **Configure LoRA/QLoRA:**
   * Set rank, alpha, dropout, and target modules.
5. **Set Training Arguments:**
   * Batch size, learning rate, epochs, max steps.
6. **Train (Fine-Tune):**
   * Use adapter configuration; only a few parameters are updated.
7. **Evaluate:**
   * Run inference with the fine-tuned model.
   * Monitor metrics like training loss.
8. **Save and Export:**
   * Store the adapter/model.
9. **Inference & Deployment:**
   * Use the quantized, adapted model for fast, cost-effective predictions.

**5. New Frontiers**

**One-Bit LLMs (BitNet)**

* Recent research explores **1-bit LLMs** where weights are ternary: {-1, 0, 1}.
* **Advantage:** Extreme compression, only addition operations needed (no GPU-heavy multiplication).
* **Implication:** Drastically lower memory, energy, and latency – potential for deployment in everyday devices.

**6. Tools, Platforms, and Automation**

* [**Vext.ai**](http://Vext.ai)**:** No-code LLM pipelines: add PDFs/datasets, configure queries, and route through selected LLMs (OpenAI, Anthropic, Google, etc.).
* [**Gradient.ai**](http://Gradient.ai)**:** Cloud-based workspace for private LLMs, fine-tuning with minimal code, supports model adapters and multiple SDKs.

**7. Best Practices & Tips**

* Use quantization and adapter methods (like LoRA) for efficiency.
* Select rank based on use-case complexity.
* Always format your data according to the model's prompt requirements.
* For custom/few-shot tasks, small, high-quality datasets can make a significant impact.
* Utilize cloud platforms for quick iterations and resource management.

**8. Sample Code Snippet (Hugging Face LoRA w/ Quantization)**

from transformers import AutoModelForCausalLM, AutoTokenizer, TrainingArguments  
from peft import LoraConfig, get\_peft\_model  
from bitsandbytes import BitsAndBytesConfig  
  
# Load and quantize model  
bnb\_config = BitsAndBytesConfig(load\_in\_4bit=True, bnb\_4bit\_compute\_dtype="float16")  
model = AutoModelForCausalLM.from\_pretrained("meta-llama/Llama-2-7b", quantization\_config=bnb\_config)  
tokenizer = AutoTokenizer.from\_pretrained("meta-llama/Llama-2-7b")  
  
# Setup LoRA  
lora\_config = LoraConfig(r=8, target\_modules=["q\_proj", "v\_proj"])  
model = get\_peft\_model(model, lora\_config)  
  
# Training (dummy example)  
training\_args = TrainingArguments(  
 per\_device\_train\_batch\_size=4,  
 num\_train\_epochs=3,  
 output\_dir="./results",  
 learning\_rate=2e-4,  
 fp16=True,  
)  
# ... rest of training script

**9. Glossary**

* **Fine-tuning:** Adjusting model weights on a new, usually smaller, dataset.
* **Quantization:** Lowering the precision of weights to make models smaller and faster.
* **LoRA/QLoRA:** Low-Rank Adaptation with or without quantization.
* **Calibration:** Finding scale/zero-point for quantization.
* **Adapters:** Pluggable modules for efficient fine-tuning.

Use these notes for quick revision; for any in-depth step (e.g., LoRA math, quantization equations, hyperparameters), refer to the respective code/config section above.

⁂