

pre-trained ^{Deep learning} FINE TUNING WITH PEFT.

Taking fine-tuned model, and make it better for a specific task using a smaller dataset.

* instead of training from scratch (which is slow), tweak the model's weights a bit so it learns new pattern.

Practical Code Summary. (fine tuning Llama 2 model).

instead of training a model from scratch, we take an
① already trained model.

↳ NousResearch/Llama-2-7b-chat-hf.

② for this we used a small dataset of 1000 sample (text) to
fine tune our model
will make model → mlabonne/guanaco-llama2-1k
(this dataset understands inst better, in style of
Llama 2 prompt).

③ We cannot afford full ^{fine} tuning, we use PEFT (parameter efficient
Fine tuning)

here we apply a trick called QLoRa.
(quantized Low Rank Adaptation).

This technique reduces memory usage by:

- o> Storing the model in 4-Bit precision. (smaller data format).
- o> Training only a few layers of all layer of model network.

bits and bytes → library
↳ 4-Bit Quantization

use 4 Bit = True
instead of storing 16/32 bit
which takes more memory
they are compressed to 4 bit

code Configuring ϕ LoRA for Memory efficiency.

lora-r = 64 # lora attention dimension
lora-alpha = 16 # lora scaling parameter
lora-dropout = 0.1 # Dropout for stability.

This is a scaling factor that adjusts how much impact the LoRA update ($\Delta W = A \times B$) has on the original weight matrix W .

* this helps balance the fine tuning updates so that they don't overwhelm / underwhelm the pre trained model.

usually obtained by an attention mechanism of a transformer.

represents the rank of LoRA adaption. Instead of modifying the complete W matrix we approximate the updates using only two smaller matrices

$$\Delta W \approx A \times B$$

A is of size $(d \times r)$
 B is of size $(r \times d)$
where r is the LoRA attention dimension. (rank of our approximation).

↳ it updates the matrix as

$W' = W + \frac{\alpha}{r} (A \times B)$

original scaling factor attention dimension low rank update

since r is smaller than d ,
this reduces the no. of trainable parameters.

now $lora-r = 64$ (will do this)

* choosing $lora-r$ depends upon on how much adaption we need.

* if the fine tuning is very d/f from original model we need larger r .

LoRA Dropout

This Dropout layer prevents overfitting by randomly disabling some connections during training

$\text{lora_dropout} = 0.1$ means 10% of LoRA connections are randomly dropped to increase robustness.

⇒ small datasets, you might increase dropout (e.g. 0.2 or 0.3)
large dataset, you can set it lower (0.05/0).

bitsnbytes

This defines computation precision used when processing weights.

$\text{bnb_4bit_compute_dtype} = \text{'float16'}$



$\text{float16} \rightarrow$ Balanced.

$\text{bfloat16} \rightarrow$ Better for training especially on new GPU(s).

$\text{float32} \rightarrow$ More precise but uses more memory.

even though previously set the model weights to be stored in 4bits, they will be computed in 16-Bit floating point.

$\text{bnb_4bit_quant_type} = \text{'nf4'}$

This defines quantization type / method and 'nf4' stands for normalization float 4 (NF4)

* instead of just simple 4Bit quantization (which would be too imprecise), NF4 uses logarithmic scaling to better represent values to preserve more information.

4. use_nested_quant = false

↳ This setting controls nested quantization which is an extra level of compression.

per_device_train_batch_size = 4

↳ this defines how many samples (datapoints) are processed per GPU per step during training.

learning_rate = 2e-4 (0.0002) → small updates.

↳ This is the step size for updating model weights.

↳ high LR (i.e. 1e-3) trains faster but can be ^{un}stable

* usually if the model is pretrained, we do not want large / faster updates to weights.

gradient_accumulation_steps = 1 → default, does not update for larger batches

usually we want the GPU to work with a larger batch.

Gradient accumulation simulates a larger batch size by accumulating gradients over multiple steps before updating the model.

model = AutoModelForCausalLM.from_pretrained

(model_name, quantization_config = bnb_config, device_map = device_map)

↓
applies quantization as described before.

Causal Model (means it can predict the next token given language previous tokens e.g text generation.)

device_map = device_map; maps the model to specific hardware.

if device_map = "auto" ; hugging face will automatically distribute layers across available GPUs.

TRAINER = SFTTrainer
 ↓ supervised training
 ↗ Fine tuning.

This code is using SFTT from hugging face's PEFT library to fine tune a pretrained language model on a given dataset

When trainer.train() is called.

- ① Load the dataset.
- ② Apply PEFT.
- ③ Optimize Training. (using hyperparameters)
- ④ Backpropagation & Gradient Updates.
- ⑤ Save Checkpoints.

→ it handles training, data processing and optimizer