

Fine tuning llm

notebook: <https://colab.research.google.com/drive/1-fNEtL21SENkkeYCOoWDh-P9jhBHND-c?authuser=3#scrollTo=yqxqAZ7KJ4oL>

```
%%capture
# Installs Unsloth, Xformers (Flash Attention) and all other packages!
!pip install "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git"
!pip install --no-deps "xformers<0.0.26" trl peft accelerate bitsandbytes

Double-click (or enter) to edit

from unsloth import FastLanguageModel
import torch
max_seq_length = 2048 # Choose any! We auto support RoPE Scaling internally!
dtype = None # None for auto detection. Float16 for Tesla T4, V100, Bfloat16 for Ampere+
load_in_4bit = True
```

```
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/llama-3-8b-bnb-4bit",
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
    # token = "hf_...", # use one if using gated models like meta-llama/Llama-2-7b-hf
)
```

```
🔄 /usr/local/lib/python3.10/dist-packages/huggingface_hub/file_download.py:1132: FutureWarning: `resume_download` is deprecated and will be removed in version 0.24.0. Using `resume_download=True` to download files can now be achieved by simply passing `resume_download=True` to the download method or by setting the environment variable `HF_HUB_RESUME_DOWNLOAD` to `True`.
warnings.warn(

config.json: 100% 1.12k/1.12k [00:00<00:00, 68.7kB/s]

==(=====)== Unsloth: Fast Llama patching release 2024.4
  \ \ / | GPU: NVIDIA A100-SXM4-40GB. Max memory: 39.564 GB. Platform = Linux.
0^0/ \_/ \ Pytorch: 2.2.1+cu121. CUDA = 8.0. CUDA Toolkit = 12.1.
 \ _____ / Bfloat16 = TRUE. Xformers = 0.0.25.post1. FA = False.
"-_____" Free Apache license: http://github.com/unslothai/unsloth
Unused kwargs: ['quant_method']. These kwargs are not used in <class 'transformers.utils.quantization_config.BitsAndBytesConfig'>.

model.safetensors: 100% 5.70G/5.70G [00:27<00:00, 189MB/s]

generation_config.json: 100% 143/143 [00:00<00:00, 11.0kB/s]

tokenizer_config.json: 100% 50.6k/50.6k [00:00<00:00, 2.50MB/s]

tokenizer.json: 100% 9.09M/9.09M [00:00<00:00, 25.1MB/s]

special_tokens_map.json: 100% 464/464 [00:00<00:00, 42.1kB/s]

Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
```

Show current memory stats

[Show code](#)

```
🔄 GPU = NVIDIA A100-SXM4-40GB. Max memory = 39.564 GB.
8.305 GB of memory reserved.
```

We now add LoRA adapters so we only need to update 1 to 10% of all parameters!

```
model = FastLanguageModel.get_peft_model(
    model,
    r = 64, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                      "gate_proj", "up_proj", "down_proj",],
    lora_alpha = 16,
    lora_dropout = 0, # Supports any, but = 0 is optimized
    bias = "none", # Supports any, but = "none" is optimized
    # [NEW] "unsloth" uses 30% less VRAM, fits 2x larger batch sizes!
    use_gradient_checkpointing = "unsloth", # True or "unsloth" for very long context
    random_state = 3407,
    use_rslora = False, # We support rank stabilized LoRA
    loftq_config = None, # And LoftQ
)
```

🔄 Unsloth 2024.4 patched 32 layers with 32 QKV layers, 32 O layers and 32 MLP layers.

▼ Data Prep

```
alpaca_prompt = """Below is Questions, paired with an context that provides further context. Write a response that appropriately completes t
```

```
### question:
{}
```

```
### context:
{}
```

```
### final_decision:
{}
```

```
### long_answer:
{}
```

```
EOS_TOKEN = tokenizer.eos_token # Must add EOS_TOKEN
def formatting_prompts_func(examples):
    question = examples["question"]
    context = examples["context"]
    final_decision = examples["final_decision"]
    long_answer = examples["long_answer"]
    texts = []
    for question, context, final_decision, long_answer in zip(question, context, final_decision, long_answer):
        # Must add EOS_TOKEN, otherwise your generation will go on forever!
        text = alpaca_prompt.format(question, context, final_decision, long_answer) + EOS_TOKEN
        texts.append(text)
    return { "text" : texts, }
pass
```

```
from datasets import load_dataset
from datasets import DatasetDict
```

```
dataset = load_dataset("qiaojin/PubMedQA", "pqa_artificial")
```

```
# Assuming you want a standard 80/20 train/test split
```

```
# Access and split the 'train' dataset
train_test_dataset = dataset['train'].train_test_split(test_size=0.002)
dataset['train'] = train_test_dataset['train']
dataset['test'] = train_test_dataset['test']
```

```
# Now you can access the splits:
train_dataset = dataset['train']
test_dataset = dataset['test']
```


```
dataset = dataset.map(formatting_prompts_func, batched = True,)
```

🔄 Downloading readme: 100%	5.19k/5.19k [00:00<00:00, 416kB/s]
Downloading data: 100%	233M/233M [00:01<00:00, 198MB/s]
Generating train split: 100%	211269/211269 [00:01<00:00, 113937.69 examples/s]
Map: 100%	210846/210846 [00:20<00:00, 10270.13 examples/s]
Map: 100%	423/423 [00:00<00:00, 6910.08 examples/s]

▼ Train the model

```
from trl import SFTTrainer
from transformers import TrainingArguments
```

```
trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset=dataset['train'],
    eval_dataset=dataset['test'],
    dataset_text_field = "text",
    max_seq_length = max_seq_length,
    dataset_num_proc = 2,
    packing = False, # Can make training 5x faster for short sequences.
    args = TrainingArguments(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 5,
        num_train_epochs=1,
        max_steps = 1000, # Set num_train_epochs = 1 for full training runs
        learning_rate = 2e-4,
        fp16 = not torch.cuda.is_bf16_supported(),
        bf16 = torch.cuda.is_bf16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs",
    ),
)
```

 /usr/local/lib/python3.10/dist-packages/multiprocess/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompatible w
self.pid = os.fork()

Map (num_proc=2): 100% 210846/210846 [02:41<00:00, 855.15 examples/s]

Map (num_proc=2): 100% 423/423 [00:01<00:00, 371.52 examples/s]

max_steps is given, it will override any value given in num_train_epochs


Double-click (or enter) to edit

```
trainer_stats = trainer.train()
```

 [Show hidden output](#)

› Show final memory and time stats

[Show code](#)

 477.878 seconds used for training.
7.96 minutes used for training.
Peak reserved memory = 8.922 GB.
Peak reserved memory for training = 3.317 GB.
Peak reserved memory % of max memory = 60.496 %.
Peak reserved memory for training % of max memory = 22.491 %.

▼ Inference

Let's run the model! You can change the instruction and input - leave the output blank!

You can also use a `TextStreamer` for continuous inference - so you can see the generation token by token, instead of waiting the whole time!

```
# alpaca_prompt = Copied from above
FastLanguageModel.for_inference(model) # Enable native 2x faster inference
inputs = tokenizer(
[
    alpaca_prompt.format(
        "How does congenital hypothyroidism present in newborns?", # question
        "Pediatrics", # context
        "", # final decision
        "", # output - leave this blank for generation!
    )
], return_tensors = "pt").to("cuda")

from transformers import TextStreamer
text_streamer = TextStreamer(tokenizer)
_ = model.generate(**inputs, streamer = text_streamer, max_new_tokens = 128)
```

Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.

<|begin_of_text|>Below is Questions, paired with an context that provides further context. Write a response that appropriately completes

```
### question:
How does congenital hypothyroidism present in newborns?

### context:
Pediatrics

### final_decision:

### long_answer:
Congenital hypothyroidism is a common endocrine disorder that can be easily missed in the newborn period. The most common presentation i
```

✓ Saving, loading finetuned models

To save the final model as LoRA adapters, either use Huggingface's `push_to_hub` for an online save or `save_pretrained` for a local save.

[NOTE] This ONLY saves the LoRA adapters, and not the full model. To save to 16bit or GGUF, scroll down!

```
from huggingface_hub import notebook_login

notebook_login()
```

Token is valid (permission: write).

Your token has been saved in your configured git credential helpers (store).

Your token has been saved to /root/.cache/huggingface/token

Login successful

```
model.save_pretrained("harfoush-mistralai_QLORA_medicalQA_v0.2") # Local saving
tokenizer.save_pretrained("harfoush-mistralai_QLORA_medicalQA_v0.2")
model.push_to_hub("harfoush/harfoush-Llama-3_QLORA_medicalQA_v0.3") # Online saving
tokenizer.push_to_hub("harfoush/harfoush-Llama-3_QLORA_medicalQA_v0.3") # Online saving
```

README.md: 100% 575/575 [00:00<00:00, 46.6kB/s]

adapter_model.safetensors: 100% 671M/671M [00:13<00:00, 44.3MB/s]

Saved model to https://huggingface.co/harfoush/harfoush-Llama-3_QLORA_medicalQA_v0.3

```

if False:
    from unsloth import FastLanguageModel
    model, tokenizer = FastLanguageModel.from_pretrained(
        model_name = "lora_model", # YOUR MODEL YOU USED FOR TRAINING
        max_seq_length = max_seq_length,
        dtype = dtype,
        load_in_4bit = load_in_4bit,
    )
    FastLanguageModel.for_inference(model) # Enable native 2x faster inference

# alpaca_prompt = You MUST copy from above!

inputs = tokenizer(
    [
        alpaca_prompt.format(
            "What is a famous tall tower in Paris?", # instruction
            "", # input
            "", # output - leave this blank for generation!
        )
    ], return_tensors = "pt").to("cuda")

outputs = model.generate(*inputs, max_new_tokens = 64, use_cache = True)
tokenizer.batch_decode(outputs)

🔗 Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
[ "<|begin_of_text|>Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.\n\n### Instruction:\nWhat is a famous tall tower in Paris?\n\n### Input:\n\n\n### Response:\nThe Eiffel Tower is a famous tall tower in Paris, France. It is 324 meters (1,063 feet) tall and was built in 1889 for the World's Fair. It is the most visited paid monument in the world, with over 7 million visitors annually. The tower is named after"]

```

Saving to float16 for VLLM

We also support saving to `float16` directly. Select `merged_16bit` for float16 or `merged_4bit` for int4. We also allow `lora` adapters as a fallback. Use `push_to_hub_merged` to upload to your Hugging Face account! You can go to <https://huggingface.co/settings/tokens> for your personal tokens.

```

# Merge to 16bit
if True: model.save_pretrained_merged("model", tokenizer, save_method = "merged_16bit",)
if True: model.push_to_hub_merged("harfoush/harfoush-Llama-3-QLORA_medicalQA_v0.3", tokenizer, save_method = "merged_16bit", token = "hf_vpK

# Merge to 4bit
if False: model.save_pretrained_merged("model", tokenizer, save_method = "merged_4bit",)
if False: model.push_to_hub_merged("hf/model", tokenizer, save_method = "merged_4bit", token = "")

# Just LoRA adapters
if False: model.save_pretrained_merged("model", tokenizer, save_method = "lora",)
if False: model.push_to_hub_merged("hf/model", tokenizer, save_method = "lora", token = "")

```

Unloth: Kaggle/Colab has limited disk space. We need to delete the downloaded

GGUF / llama.cpp Conversion

Unloth: Will use up to 60.71 out of 83.48 RAM for saving.

Save to 8bit Q8_0

if False: model.save_pretrained_gguf("model", tokenizer,)

if False: model.push_to_hub_gguf("hf/model", tokenizer, token = "")

Save to 16bit GGUF

if True: model.save_pretrained_gguf("model", tokenizer, quantization_method = "f16")

if True: model.push_to_hub_gguf("harfoush/harfoush-Llama-3_QLoRA_medicalQA_v0.3", tokenizer, quantization_method = "f16", token = "hf_vpKTqc")

Save to q4_k_m GGUF

if False: model.save_pretrained_gguf("model", tokenizer, quantization_method = "q4_k_m")

if False: model.push_to_hub_gguf("hf/model", tokenizer, quantization_method = "q4_k_m", token = "")



Unloth: Merging 4bit and LoRA weights to 16bit...

Unloth: Will use up to 58.87 out of 83.48 RAM for saving.

100%|██████████| 32/32 [00:00<00:00, 44.09it/s]

Unloth: Saving tokenizer... Done.

Unloth: Saving model... This might take 5 minutes for Llama-7b...

Done.

NOTICE: llama.cpp GGUF conversion is currently unstable, since llama.cpp is undergoing some major bug fixes as at 5th of May 2024. This is not an Unloth issue. Please be patient - GGUF saving should still work, but might not work as well.

Unloth: Converting llama model. Can use fast conversion = True.

==(====)= Unloth: Conversion from QLoRA to GGUF information

\\ /| [0] Installing llama.cpp will take 3 minutes.

0^0/ _/ \ [1] Converting HF to GGUF 16bits will take 3 minutes.

\ / [2] Converting GGUF 16bits to f16 will take 20 minutes.

"-_____" In total, you will have to wait around 26 minutes.

Unloth: [0] Installing llama.cpp. This will take 3 minutes...

Unloth: [1] Converting model at model into f16 GGUF format.

The output location will be ./model-unloth.F16.gguf

This will take 3 minutes...

Unloth: Conversion completed! Output location: ./model-unloth.F16.gguf

Unloth: Merging 4bit and LoRA weights to 16bit...

Unloth: Will use up to 60.0 out of 83.48 RAM for saving.

100%|██████████| 32/32 [00:00<00:00, 62.00it/s]

Unloth: Saving tokenizer... Done.

Unloth: Saving model... This might take 5 minutes for Llama-7b...

Done.

NOTICE: llama.cpp GGUF conversion is currently unstable, since llama.cpp is undergoing some major bug fixes as at 5th of May 2024. This is not an Unloth issue. Please be patient - GGUF saving should still work, but might not work as well.

==(====)= Unloth: Conversion from QLoRA to GGUF information

\\ /| [0] Installing llama.cpp will take 3 minutes.

0^0/ _/ \ [1] Converting HF to GGUF 16bits will take 3 minutes.

\ / [2] Converting GGUF 16bits to f16 will take 20 minutes.

"-_____" In total, you will have to wait around 26 minutes.

Unloth: [0] Installing llama.cpp. This will take 3 minutes...

Unloth: [1] Converting model at model into f16 GGUF format.

The output location will be ./model-unloth.F16.gguf

This will take 3 minutes...

Unloth: Conversion completed! Output location: ./model-unloth.F16.gguf

Unloth: Merging 4bit and LoRA weights to 16bit...

Unloth: Will use up to 60.0 out of 83.48 RAM for saving.

100%|██████████| 32/32 [00:00<00:00, 62.00it/s]

Unloth: Saving tokenizer... Done.

Unloth: Saving model... This might take 5 minutes for Llama-7b...

Done.

NOTICE: llama.cpp GGUF conversion is currently unstable, since llama.cpp is undergoing some major bug fixes as at 5th of May 2024. This is not an Unloth issue. Please be patient - GGUF saving should still work, but might not work as well.

==(====)= Unloth: Conversion from QLoRA to GGUF information

\\ /| [0] Installing llama.cpp will take 3 minutes.

0^0/ _/ \ [1] Converting HF to GGUF 16bits will take 3 minutes.

\ / [2] Converting GGUF 16bits to f16 will take 20 minutes.

"-_____" In total, you will have to wait around 26 minutes.

Unloth: [0] Installing llama.cpp. This will take 3 minutes...

Unloth: [1] Converting model at model into f16 GGUF format.

The output location will be ./model-unloth.F16.gguf

This will take 3 minutes...

Unloth: Conversion completed! Output location: ./model-unloth.F16.gguf

Unloth: Merging 4bit and LoRA weights to 16bit...

Unloth: Will use up to 60.0 out of 83.48 RAM for saving.

100%|██████████| 32/32 [00:00<00:00, 62.00it/s]

Unloth: Saving tokenizer... Done.

Unloth: Saving model... This might take 5 minutes for Llama-7b...

Done.

NOTICE: llama.cpp GGUF conversion is currently unstable, since llama.cpp is undergoing some major bug fixes as at 5th of May 2024. This is not an Unloth issue. Please be patient - GGUF saving should still work, but might not work as well.