Fine-tuning Google's New Gemma 3 270M and running it locally 👉

Before Finetuning

(generates random chess moves)

Inference before finetuning

```
messages = [
    {'role': 'system','content':dataset['conversations'][10][0]['content']},
    {"role" : 'user', 'content' : dataset['conversations'][10][1]['content']}
]
text = tokenizer.apply_chat_template(
    messages,
    tokenize = False,
    add_generation_prompt = True, # Must add for generation
).removeprefix('<bos>')

from transformers import TextStreamer
_ = model.generate(
    **tokenizer(text, return_tensors = "pt").to("cuda"),
    max_new_tokens = 125,
    temperature = 1, top_p = 0.95, top_k = 64,
    streamer = TextStreamer(tokenizer, skip_prompt = True),
)
```

```
The missing chess move is:
```json
{
"moves": ["e2e4", "c7c5", "g1f3", "e7e6", "d2d4", "c5d4", "f3d4", "g8f6", "b1c
```

After Finetuning

(finds the exact chess move)

Inference after finetuning

```
messages = [
    {'role': 'system','content':dataset['conversations'][10][0]['content']},
    {"role" : 'user', 'content' : dataset['conversations'][10][1]['content']}
]
text = tokenizer.apply_chat_template(
    messages,
    tokenize = False,
    add_generation_prompt = True, # Must add for generation
).removeprefix('<bos>')

from transformers import TextStreamer
_ = model.generate(
    **tokenizer(text, return_tensors = "pt").to("cuda"),
    max_new_tokens = 125,
    temperature = 1, top_p = 0.95, top_k = 64,
    streamer = TextStream[illegible]kenizer, skip_prompt = True),
)
```

```
{"missing move": "a3a4"}<end_of_turn>
```

Double-click (or enter) to edit
```

1) Load model


```
# pip install unsloth

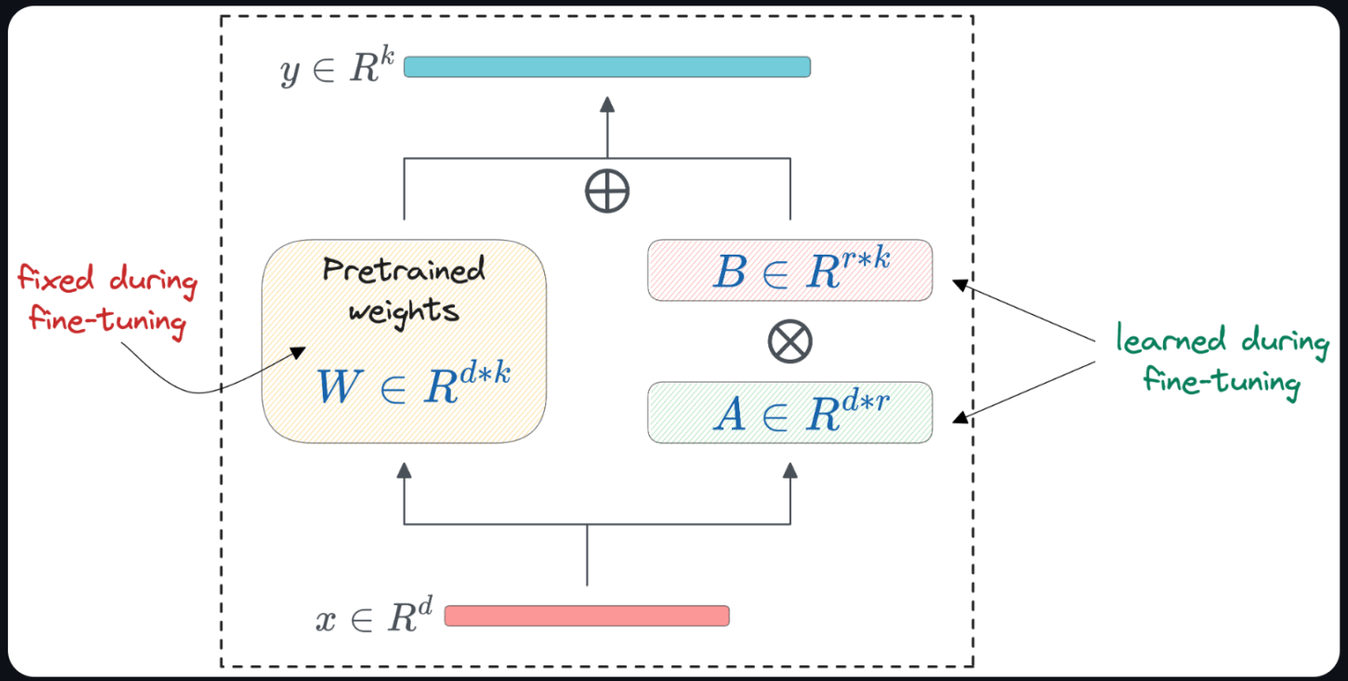
from unsloth import FastModel
import torch

MODEL = "unsloth/gemma-3-270m-it"

model, tokenizer = FastLanguageModel.from_pretrained(
                            model_name = MODEL,
                            max_seq_length = 2048,
                            dtype = None,
                            load_in_4bit = False,
                            full_finetuning = False
)
```

2) Define LoRA config

```
model = FastModel.get_peft_model(
    model,
    r = 128,
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                      "gate_proj", "up_proj", "down_proj",],
    use_gradient_checkpointing = "unsloth",
    lora_alpha = 128,
    lora_dropout = 0,
    bias = "none",
    random_state = 3407,
    use_rslora = False,
    loftq_config = None, #
)
```



3) Load dataset

```python
from datasets import load_dataset

dataset_name = "Thytu/ChessInstruct"
dataset = load_dataset(dataset_name, split = "train[:10000]")

>>> dataset[0]
```

Task

```
{
  "task": "Given an incomplit set of chess moves and the game's final score"
          "write the last missing chess move.
          "Input Format: A comma-separated list of chess moves followed by the game score."
          "Output Format: The missing chess move",

  "input": {
              "moves": ["c2c4", "g8f6", "b1c3", "c7c5", ..., "e7f6", "g7f7", "?"],
              "result": "1/2-1/2"
            },

  "expected_output": '{"missing move": "e6f7"}',
}
```

Previous moves

Final Result

Missing move (to be predicted)

4) Prepare dataset

```python
from datasets import load_dataset
from unsloth import to_sharegpt
from unsloth.chat_templates import standardize_data_formats

dataset = load_dataset("Thytu/ChessInstruct", split = "train")

dataset = standardize_data_formats(dataset)

def convert_to_chatml(example):
    return {
        "conversations": [
            {"role": "system", "content": example["task"]},
            {"role": "user", "content": example["input"]},
            {"role": "assistant", "content": example["expected_output"]}
        ]
    }

dataset = dataset.map(
    convert_to_chatml
)
```

5) Define Trainer

```
from trl import SFTTrainer, SFTConfig

trainer = SFTTrainer(model = model,
                     tokenizer = tokenizer,
                     train_dataset = dataset,
                     ...
                     args = SFTConfig(
                            per_device_train_batch_size = 2,
                            gradient_accumulation_steps = 4,
                            max_steps = 100,
                            learning_rate = 5e-5,
                            ...
                            optim = "adamw_8bit",
                            weight_decay = 0.01,
                     ))
```

6) Train

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

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs used = 1
   \\   /|    Num examples = 100,000 | Num Epochs = 1 | Total steps = 30
O^O/ \_/ \    Batch size per device = 2 | Gradient accumulation steps = 4
\        /    Data Parallel GPUs = 1 | Total batch size (2 x 4 x 1) = 8
 "-____-"     Trainable parameters = 7,450,624/4,000,000,000 (0.19% trained)
It is strongly recommended to train Gemma3 models with the `eager` attention
Unsloth: Will smartly offload gradients to save VRAM!
```

[30/30 17:43, Epoch 0/1]

Step	Training Loss	Step	Training Loss		Step	Training Loss
1	1.237700	11	1.461600		21	1.694700
2	1.636400	12	1.867400		22	1.614100
3	1.766300	13	1.854700		23	1.829700
4	1.420700	14	1.394800	....	24	1.616300
5	1.235700	15	1.633000		25	1.269200
6	1.806600	16	1.238600		26	1.291600
7	1.010100	17	2.174100		27	1.743000
8	1.896600	18	1.488000		28	1.525200
9	1.464700	19	1.521400		29	1.999000
10	1.309700	20	1.816000		30	1.858200

Run fine-tuned Gemma 3 locally

```
messages = [
    {'role': 'system','content':dataset['conversations'][10][0]['content']},
    {"role" : 'user', 'content' : dataset['conversations'][10][1]['content']}
]
text = tokenizer.apply_chat_template(
    messages,
    tokenize = False,
    add_generation_prompt = True, # Must add for generation
).removeprefix('<bos>')

from transformers import TextStreamer
_ = model.generate(
    **tokenizer(text, return_tensors = "pt").to("cuda"),
    max_new_tokens = 125,
    temperature = 1, top_p = 0.95, top_k = 64,
    streamer = TextStreamer(tokenizer, skip_prompt = True),
)
```

```
<bos><start_of_turn>user
Given an incomplit set of chess moves and the game's final score, write the last

Input Format: A comma-separated list of chess moves followed by the game score.
Output Format: The missing chess move

{"moves": ["e2e4", "c7c5", "g1f3", "e7e6", "d2d4", "c5d4", "f3d4", "g8f6", "b1c3'
<start_of_turn>model
{"missing move": "a3a4"}<end_of_turn>
```

Finds the exact missing move