First Full LoRA Trial with Transformer

peft (for LoRA) and FLAN-T5-small for the LLM

I'm following what seems to be a great tutorial from Mehul Gupta,

> https://medium.com/data-science-in-your-pocket/lora-for-fine-tuning-llms-explained-with-codes-and-example-62a7ac5a3578
>
> https://web.archive.org/web/20240522140323/https://medium.com/data-science-in-your-pocket/lora-for-fine-tuning-llms-explained-with-codes-and-example-62a7ac5a3578

I'm doing this to prepare creating a LoRA for RWKV (@todo put links in here) so as to fine-tune it for Pat's OLECT-LM stuff.

In [ ]:
```
# #  No need to run this again
# !powershell -c (Get-Date -UFormat \"%s_%Y%m%dT%H%M%S%Z00\") -replace '[.][0-9]*_', '_'"
```

Output was:

```
1716367147_20240522T083907-0600
```

Imports

In [1]:
```
from datasets import load_dataset
from random import randrange
import torch
from transformers import AutoTokenizer, \
                         AutoModelForSeq2SeqLM, \
                         TrainingArguments, \
                         pipeline
from peft import LoraConfig, \
                 prepare_model_for_kbit_training, \
                 get_peft_model, \
                 AutoPeftModelForCausalLM
```

```
from trl import SFTTrainer
from huggingface_hub import login, notebook_login
```

Load the training and test dataset along with the LLM with its tokenizer

The LLM will be fine-tuned. It seems the tokenizer will also be fine-tuned, but I'm not sure

Why aren't we loading the validation set? (I don't know; that's not a teaching question.)

In [2]:
```
#  Need to install  datasets  from pip, not conda. I'll do all from pip.
#+ I'll get rid of the current conda environment and make it anew.
#+ Actually, I'll make sure  conda  and  pip  are updated, then do what
#+ I discussed above.
#+
#+ cf.
#+     arch_ref_1 = "https://web.archive.org/web/20240522150357/" + \
#+                  "https://stackoverflow.com/questions/77433096/" + \
#+                  "notimplementederror-loading-a-dataset-" + \
#+                  "cached-in-a-localfilesystem-is-not-suppor"
#+
#+ Also useful might be
#+     arch_ref_2 = "https://web.archive.org/web/20240522150310/" + \
#+                  "https://stackoverflow.com/questions/76340743/" + \
#+                  "huggingface-load-datasets-gives-" + \
#+                  "notimplementederror-cannot-error"
#
data_files = {'train':'samsum-train.json', 'test':'samsum-test.json'}
dataset = load_dataset('json', data_files=data_files)

model_name = "google/flan-t5-small"
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

#  Next line makes training faster but a little less accurate
model.config.pretraining_tp = 1

tokenizer = AutoTokenizer.from_pretrained(model_name,
                                          trust_remote_code=True)

#  padding instructions for the tokenizer
#+   ??? !!! What about for RWKV !!! ???
```

```
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"
```

Trying some things I've been learning

In [3]:
```
print(model)
```

```
T5ForConditionalGeneration(
  (shared): Embedding(32128, 512)
  (encoder): T5Stack(
    (embed_tokens): Embedding(32128, 512)
    (block): ModuleList(
      (0): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=384, bias=False)
              (k): Linear(in_features=512, out_features=384, bias=False)
              (v): Linear(in_features=512, out_features=384, bias=False)
              (o): Linear(in_features=384, out_features=512, bias=False)
              (relative_attention_bias): Embedding(32, 6)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerFF(
            (DenseReluDense): T5DenseGatedActDense(
              (wi_0): Linear(in_features=512, out_features=1024, bias=False)
              (wi_1): Linear(in_features=512, out_features=1024, bias=False)
              (wo): Linear(in_features=1024, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): NewGELUActivation()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
      (1-7): 7 x T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=384, bias=False)
              (k): Linear(in_features=512, out_features=384, bias=False)
              (v): Linear(in_features=512, out_features=384, bias=False)
              (o): Linear(in_features=384, out_features=512, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
```

```
          )
          (1): T5LayerFF(
            (DenseReluDense): T5DenseGatedActDense(
              (wi_0): Linear(in_features=512, out_features=1024, bias=False)
              (wi_1): Linear(in_features=512, out_features=1024, bias=False)
              (wo): Linear(in_features=1024, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): NewGELUActivation()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (final_layer_norm): T5LayerNorm()
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (decoder): T5Stack(
    (embed_tokens): Embedding(32128, 512)
    (block): ModuleList(
      (0): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=384, bias=False)
              (k): Linear(in_features=512, out_features=384, bias=False)
              (v): Linear(in_features=512, out_features=384, bias=False)
              (o): Linear(in_features=384, out_features=512, bias=False)
              (relative_attention_bias): Embedding(32, 6)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerCrossAttention(
            (EncDecAttention): T5Attention(
              (q): Linear(in_features=512, out_features=384, bias=False)
              (k): Linear(in_features=512, out_features=384, bias=False)
              (v): Linear(in_features=512, out_features=384, bias=False)
              (o): Linear(in_features=384, out_features=512, bias=False)
            )
            (layer_norm): T5LayerNorm()
```

```
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (2): T5LayerFF(
      (DenseReluDense): T5DenseGatedActDense(
        (wi_0): Linear(in_features=512, out_features=1024, bias=False)
        (wi_1): Linear(in_features=512, out_features=1024, bias=False)
        (wo): Linear(in_features=1024, out_features=512, bias=False)
        (dropout): Dropout(p=0.1, inplace=False)
        (act): NewGELUActivation()
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
(1-7): 7 x T5Block(
  (layer): ModuleList(
    (0): T5LayerSelfAttention(
      (SelfAttention): T5Attention(
        (q): Linear(in_features=512, out_features=384, bias=False)
        (k): Linear(in_features=512, out_features=384, bias=False)
        (v): Linear(in_features=512, out_features=384, bias=False)
        (o): Linear(in_features=384, out_features=512, bias=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): T5LayerCrossAttention(
      (EncDecAttention): T5Attention(
        (q): Linear(in_features=512, out_features=384, bias=False)
        (k): Linear(in_features=512, out_features=384, bias=False)
        (v): Linear(in_features=512, out_features=384, bias=False)
        (o): Linear(in_features=384, out_features=512, bias=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (2): T5LayerFF(
      (DenseReluDense): T5DenseGatedActDense(
        (wi_0): Linear(in_features=512, out_features=1024, bias=False)
        (wi_1): Linear(in_features=512, out_features=1024, bias=False)
        (wo): Linear(in_features=1024, out_features=512, bias=False)
```

```
            (dropout): Dropout(p=0.1, inplace=False)
            (act): NewGELUActivation()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
  (final_layer_norm): T5LayerNorm()
  (dropout): Dropout(p=0.1, inplace=False)
 )
 (lm_head): Linear(in_features=512, out_features=32128, bias=False)
)
```

Prompt and Trainer

For our SFT (**S**upervised **F**ine **T**uning) model, we use the `class trl.SFTTrainer`.

I want to research this a bit, especially the `formatting_func` that we'll be passing to the `SFTTrainer`.

First, though, some information about SFT. From the Hugging Face Documentation at https://huggingface.co/docs/trl/en/sft_trainer (archived)

> Supervised fine-tuning (or SFT for short) is a crucial step in RLHF. In TRL we provide an easy-to-use API to create your SFT models and train them with few lines of code on your dataset.

Though I won't be using the examples unless I get even more stuck, the next paragraph *has* examples, and I'll put the paragraph here.

> Check out a complete flexible example at examples/scripts/sft.py [archived]. Experimental support for Vision Language Models is also included in the example examples/scripts/vsft_llava.py [archived].

RLHF (archived wikipedia page) is **R**einforcement **L**earning from **H**uman **F**eedback. TRL%20step.) (archived) **T**ransfer **R**einforcement **L**earning, a library from Hugging Face.

For the parameter, `formatting_func`, I can look ath the documentation site above (specifically here), at the GitHub repo for the code (in the docstrings), or from my local `conda` environment, at `C:\Users\bballdave025\.conda\envs\rwkv-lora-`

`pat\Lib\site-packages\trl\trainer\sft_trainer.py`.

Pulling code from the last one, I get

```
    formatting_func (`Optional[Callable]`):
       The formatting function to be used for creating the `ConstantLengthDataset`.
```

That matches the first very well

> **formatting_func** (`Optional[Callable]`) — The formatting function to be used for creating the `ConstantLengthDataset`.

(A quick note: In this Jupyter Notebook environment, I could have typed `trainer = SFTTrainer(` and then `Shift` + `Tab` to find that same documentation.

However, I think that more clarity is found at the documentation for `ConstantLengthDataset

> **formatting_func** (`Callable`, **optional**) — Function that formats the text before tokenization. Usually it is recommended to have follows a certain pattern such as `"### Question: {question} ### Answer: {answer}"`

So, as we'll see the next code from the tutorial, it basically is a prompt templater/formatter that matches the JSON. For example, we use `sample['dialogue']` to access the `dialogue` key/pair. That's what I got from all this stuff.

Mehul Gupta himself stated

> Next, using the Input and Output, we will create a prompt template which is a requirement by the SFTTrainer we will be using later

Prompt

In [4]:
```
def prompt_instruction_format(sample):
    return f""" Instruction:
      Use the Task below and the Input given to write the Response:

      ### Task:
      Summarize the Input
```

```
      ### Input:
      {sample['dialogue']}

      ### Response:
      {sample['summary']}
      """
##endof:  prompt_instruction_format(sample)
```

Trainer - with LoRA setup

Arguments and Configuration

In [5]:
```
#  Some arguments to pass to the trainer
training_args = TrainingArguments( output_dir='output',
                                   num_train_epochs=1,
                                   per_device_train_batch_size=4,
                                   save_strategy='epoch',
                                   learning_rate=2e-4
)

# the fine-tuning (peft for LoRA) stuff
peft_config = LoraConfig( lora_alpha=16,
                          lora_dropout=0.1,
                          r=64,
                          bias='none',
                          task_type='CAUSAL_LM'
)
```

`task_type` , cf. https://github.com/huggingface/peft/blob/main/src/peft/config.py#L222

```
    Args:
        peft_type (Union[[`~peft.utils.config.PeftType`], `str`]): The type of Peft method to
use.
        task_type (Union[[`~peft.utils.config.TaskType`], `str`]): The type of task to perform.
        inference_mode (`bool`, defaults to `False`): Whether to use the Peft model in
inference mode.
```

After some searching using Cygwin

```
bballdave025@MYMACHINE /cygdrive/c/Users/bballdave025/.conda/envs/rwkv-lora-pat/Lib/site-
packages/peft/utils
$ ls -lah
total 116K
drwx------+ 1 bballdave025 bballdave025    0 May 28 21:09 .
drwx------+ 1 bballdave025 bballdave025    0 May 28 21:09 ..
-rwx------+ 1 bballdave025 bballdave025 2.0K May 28 21:09 __init__.py
drwx------+ 1 bballdave025 bballdave025    0 May 28 21:09 __pycache__
-rwx------+ 1 bballdave025 bballdave025 8.0K May 28 21:09 constants.py
-rwx------+ 1 bballdave025 bballdave025 3.8K May 28 21:09 integrations.py
-rwx------+ 1 bballdave025 bballdave025  17K May 28 21:09 loftq_utils.py
-rwx------+ 1 bballdave025 bballdave025 9.7K May 28 21:09 merge_utils.py
-rwx------+ 1 bballdave025 bballdave025  25K May 28 21:09 other.py
-rwx------+ 1 bballdave025 bballdave025 2.2K May 28 21:09 peft_types.py
-rwx------+ 1 bballdave025 bballdave025  21K May 28 21:09 save_and_load.py

bballdave025@MYMACHINE /cygdrive/c/Users/bballdave025/.conda/envs/rwkv-lora-pat/Lib/site-
packages/peft/utils
$ grep -iIRHn "TaskType"
peft_types.py:60:class TaskType(str, enum.Enum):
__init__.py:20:# from .config import PeftConfig, PeftType, PromptLearningConfig, TaskType
__init__.py:22:from .peft_types import PeftType, TaskType

bballdave025@MYMACHINE /cygdrive/c/Users/bballdave025/.conda/envs/rwkv-lora-pat/Lib/site-
packages/peft/utils
$
```

So, let's look at the `peft_types.py` file.

The docstring for `class TaskType(str, enum.Enum)` is

```
    Enum class for the different types of tasks supported by PEFT.

    Overview of the supported task types:
    - SEQ_CLS: Text classification.
    - SEQ_2_SEQ_LM: Sequence-to-sequence language modeling.
    - CAUSAL_LM: Causal language modeling.
    - TOKEN_CLS: Token classification.
```

```
    - QUESTION_ANS: Question answering.
    - FEATURE_EXTRACTION: Feature extraction. Provides the hidden states which can be used as
embeddings or features
      for downstream tasks.
```

Try for a baseline

In [7]:
```
# Just one summarization to begin with, randomly picked

summarizer = pipeline('summarization', model=model, tokenizer=tokenizer)

sample = dataset['test'][randrange(len(dataset["test"]))]
print(f"dialogue: \n{sample['dialogue']}\n---------------")

res = summarizer(sample["dialogue"])

print(f"flan-t5-small summary:\n{res[0]['summary_text']}")
```

```
Your max_length is set to 200, but your input_length is only 168. Since this is a summarization task, where outputs s
horter than the input are typically wanted, you might consider decreasing max_length manually, e.g. summarizer('...',
max_length=84)
dialogue:
John: hey laurel?
Laurel: hey
John: whats your plan for tomorrow?
Laurel: aint that sure yet, why?
John: nothing much, just wanted to go with you and buy a birthday gift for Diana.
Laurel: OMG! i also totally forgot that her birthday is on saturday, shit!
John: you see im not the only late one here. haha
Laurel: I guess we can meet up tomorrow and go fetch something for her.
John: cool, at what time?
Laurel: lets just meet at jades at around 5 pm
John: At Jade's collection? in town?
Laurel: yeah, that place..
John: see you then.
---------------
flan-t5-small summary:
Laurel and John will meet at Jade's collection at around 5 pm tomorrow to buy a birthday gift for Diana. John will go
with Diana and go fetch something for Diana tomorrow.
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
import timeit
tic = timeit.default_timer()

toc =
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: