First Full LoRA Trial with Transformer Now on Google CoLab

Starting with going through what I've done as well as finishing the task of getting my LoRA-fine-tuned model from Hugging Face and running inference on it (i.e. testing it using the test set). See the first timestamp below for the new timing. By the way, I've shut down and rebooted the compy here in the corner with the three screens).

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~~ @DONE put links in here) so as to fine-tune it for Pat's OLECT-LM stuff.

In [ ]:
```
# # Don't need this again
!date +'%s_%Y%m%dT%H%M%S%z'
```

Output was:

`timestamp`

Installation

(Feel free to drop down to the TL;DR section.)

Detailed stuff

My `environment.yml` file will have its contents listed below. It should have everything needed for an install anywhere. The directory should have a `full_environment.yml`, which includes everything for the environment on Windows.

You can change `do_want_to_read_realtime` to `True` if you really want to see the file as it is now. One case of this would be that you think `environment.yml` has been changed since this notebook was written. The file contents as of the time of my writing this notebook should be in a markdown cell beneath the code.

In [ ]:
```
do_want_to_read_realtime = False

if do_want_to_read_realtime:
    with open("environment.yml", 'r', encoding='utf-8') as fh:
        while True:
            line = fh.readline()
            if not line:
                break
            ##endof:  if not line
            print(line.replace("\n", ""))
        ##endof:  while True
    ##endof:  with open ... fh
##endof:  if do_want_to_read_realtime
```

For the CPU, I got the following.

```
# @file: environment.yml
# @since 2024-06-03
## 1717411989_2024-06-03T105309-0600
## IMPORTANT NOTES
##
##  A couple of installations were made from git repos.
##     >pip install git+https://github.com/huggingface/peft.git
##     >pip install git+https://github.com/nexplorer-3e/qwqfetch
##
##  The commit info will be important for reproducibility.
##
```

```
##-----
##  qwqfetch   for system info
##
##   Resolved https://github.com/nexplorer-3e/qwqfetch \
##       to commit f72d222e2fff5ffea9f4e4b3a203e4c4d9e8cf00
##   Successfully installed qwqfetch-0.0.0
##
#
##-----
##  peft: I installed PEFT among other things, but I'm picking out
##+       stuff relevant to peft. PEFT has LoRA in it.
##
##   Resolved https://github.com/huggingface/peft.git \
##       to commit e7b75070c72a88f0f7926cc6872858a2c5f0090d
## Successfully built peft
#
#

channels:
  - defaults
dependencies:
  - python=3.10.14
  - pip=24.0
  - pip:
      - accelerate==0.30.1
      - bitsandbytes==0.43.1
      - datasets==2.19.1
      - evaluate==0.4.2
      - huggingface-hub==0.23.2
      - humanfriendly==10.0
      - jupyter==1.0.0
      - nltk==3.8.1
      - peft==0.11.2.dev0
      - py-cpuinfo==9.0.0
      - pylspci==0.4.3
      - qwqfetch==0.0.0
      - rouge-score==0.1.2
      - tensorflow-cpu==2.16.1
      - torch==2.3.0
```

```
- transformers==4.41.1
- trl==0.8.6
- wmi==1.5.1
```

For CoLab, I got the following

Put CoLab `environment-colab.yml` that gets `conda env export`-ed here

```
blah
```

What should probably work for any install on CoLab comes in the next executable cells - the ones with `!pip install` ... I hope that it doesn't automatically read my `environment.yml` file and build it, because my `environment.yml` file is made for running on a CPU. What's more, I ran it on a CPU on Windows, so I'm not sure how it will perform with Linux(R).

[Doing some stuff.]

Okay, I'm going to commit this stuff with the `environment.yml` renamed to `environment-cpu.yml`. I'll create (and commit) a new `environment.yml` exactly the same as the one above, except with `tensorflow-cpu` replaced with `tensorflow`. This new, `tensorflow`-not-`tensorflow-cpu` environment file will also be copied to `environment-colab.yml`. (There will also be `full_environment-cpu.yml` and `full_environment-colab.yml` files.

If nothing happened with the `environment.yml` (i.e. no environment was built, no packages were loaded), run the installs below. That should get you set up nicely for CoLab.

If packages aren't installed, you can either:

1. (not preferable) install from `environment-colab.yml`.

From a terminal/command prompt, run

`conda env create -f environment-colab.yml`

OR

2. you can run the `!pip install` commands below for running things on Google CoLab (or any *NIX-type system, I think). **In my experience, at least some of the packages need to come from `pip` rather than from `conda` to work. I suggest using all from `pip`.** For the whole project, my suggestion is to run the Jupyter Notebook from inside a `conda environment`. For complete reproducibility, the Python version should be 3.10.14 and the `pip` version 24.0. The commands for the shell/command prompt could be

   ```
   conda create -n my-env-lora python=3.10.14`

   conda activate my-env-lora
   ```

After the environment is activated, you can then run

```
pip install --upgrade pip==24.0
  #  note that you should use the `--upgrade` flag whether
  #+ upgrading or downgrading pip
```

To get started with this notebook stuff,

`pip install jupyter`

And run `jupyter notebook` to start using a blank notebook, or `jupyter notebook <notebook_base_name>.ipynb` to use an already-created notebook.

Note: if you are in this Jupyter Notebook but aren't in a conda environment ... and if you know enough to realize that and to know what the following commands do, you can uncomment the commands below to get your `conda` environment set up.

In [ ]:

```
## not going to do complicated subprocess stuff here. Sorry.
```

Install TL;DR

Once you have things ready and a jupyter notebook running, you can do the `!pip install` commands that follow. That is, you should run the commands unless you used

`conda env create -f <environment-filename>`

These commands below should get you set up nicely for CoLab.

In [ ]:
```
!pip install --update pip==24.0
```

In [ ]:
```
!pip install accelerate bitsandbytes evaluate datasets huggingface-hub
!pip install humanfriendly nltk py-cpuinfo pylspci rouge-score
!pip install tensorflow torch transformers trl
```

Trying this next one on its own, since it might fail (we're not on Windows).

In [ ]:
```
!pip install wmi
```

And now, for the installs from GitHub repos.

In [ ]:
```
!pip install git+https://github.com/huggingface/peft.git
```

In [ ]:
```
!pip install git+https://github.com/nexplorer-3e/qwqfetch.git
```

Imports

In [ ]:
```
from datasets import load_dataset
import random
from random import randrange
import torch
from transformers import AutoTokenizer, \
                         AutoModelForSeq2SeqLM, \
                         AutoModelForCausalLM, \
                         TrainingArguments, \
                         pipeline
from transformers.utils import logging
from peft import LoraConfig, \
                 prepare_model_for_kbit_training, \
                 get_peft_model, \
                 AutoPeftModelForSeq2SeqLM, \
                 AutoPeftModelForCausalLM
from trl import SFTTrainer
from huggingface_hub import login, notebook_login
```

```
from datasets import load_metric
from evaluate import load as evaluate_dot_load
import nltk
import rouge_score
from rouge_score import rouge_scorer, scoring

import pickle
import pprint
import re
import timeit
from humanfriendly import format_timespan
import os

## my module(s), now just in the working directory as .PY files
import system_info_as_script
import dwb_rouge_scores
```

Load the training and test dataset along with the LLM and 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.)~~

Update: It seems that validation-set use with the trainer wasn't part of the example.

I've tried to make use of it (the validation set) with the `trainer`. We'll see how it goes.

Update: It worked fine, though its loss is lower than the training set's loss.

In [ ]:
```
#  Need to install  datasets  (i.e. the `datasets` module/package)
#+ from `pip`, not `conda`. I'll do all from `pip`.
#+
#+ 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',
              'evaluation':'samsum-validation.json',
              'test':'samsum-test.json'}
dataset = load_dataset('json', data_files=data_files)

model_name = "google/flan-t5-small"

model_load_tic = timeit.default_timer()
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
model_load_toc = timeit.default_timer()

model_load_duration = model_load_toc - model_load_tic

print(f"Loading the original model, {model_name}")
print(f"took {model_load_toc - model_load_tic:0.4f} seconds.")

model_load_time_str = format_timespan(model_load_duration)

print(f"which equates to {model_load_time_str}")

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

tokenizer_tic = timeit.default_timer()
tokenizer = AutoTokenizer.from_pretrained(model_name,
                                          trust_remote_code=True)
tokenizer_toc = timeit.default_timer()

tokenizer_duration = tokenizer_toc - tokenizer_tic

print()
print("Getting the original tokenizer")
print(f"took {tokenizer_toc - tokenizer_tic:0.4f} seconds.")

tokenizer_time_str = format_timespan(tokenizer_duration)
```

```
print(f"which equates to {tokenizer_time_str}")

#  padding instructions for the tokenizer
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"
```

I wonder if those lines,

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

will be the same for RWKV.

Notes from trying to get rid of weird output

I've thought about changing the line

`model = AutoModelForSeq2SeqLM.from_pretrained(model_name)`

to match the `peft` configuration, i.e.

> ```
> peft_config = LoraConfig( lora_alpha=16,
>                           lora_dropout=0.1,
>                           r=64,
>                           bias='none',
>                           task_type='CAUSAL_LM',
> )
> ```

I've thought about using

`model = AutoModelForCausalLM.from_pretrained(model_name)`

but every documentation I've consulted uses the `Seq2SeqLM`. e.g.

```
doc1 = "https://web.archive.org/web/20240506213344/" + \\
       "https://huggingface.co/docs/transformers/en/" + \\
```

```
"model_doc/flan-t5"
```

Also, there is the info from

```
doc2="https://huggingface.co/transformers/v3.0.2/model_doc/t5.html"
```

> T5 is an encoder-decoder model pre-trained on a multi-task mixture of unsupervised and supervised tasks and for which **each task is converted into a text-to-text format.**

Something similar is in the paper abstract for

https://arxiv.org/pdf/1910.10683.pdf

Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified **Text-to-Text** Transformer". online. arXiv:cs.LG.1910.10683v4. 19 Sep 2023. retrieved 06 June 2024

which is cited in `doc2`

> In this paper, we explore the landscape of transfer learning techniques for NLP by introducing a unified framework that converts all text-based language problems into a **text-to-text format**.

(All emphasis is mine, DWB.)

Google Results

As of today (2024-06-06), a Google search for

`"AutoModelForCausalLM from_pretrained google flan-t5-small"`

(with quotes) returns

> Your search - "AutoModelForCausalLM from_pretrained google flan-t5-small" - did not match any documents.
>
> Suggestions:
>
> - Make sure all words are spelled correctly.
> - Try different keywords.

> - Try more general keywords.

whereas a Google search (again with quotes) for

`"AutoModelForSeq2SeqLM from_pretrained google flan-t5-small"`

returns

> About 119 results (0.22 seconds)

Trying the experiment

With all that, I tried the line anyway. Using just the important lines

`IN:`

```
model_name = "google/flan-t5-small"
```

...

```
model = AutoModelForCausalLM.from_pretrained(model_name)
```

`OUT:`

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[4], line 29
     25 model_load_tic = timeit.default_timer()
     27 #model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
---> 29 model = AutoModelForCausalLM.from_pretrained(model_name)
     30 model_load_toc = timeit.default_timer()
     32 model_load_duration = model_load_toc - model_load_tic

File ~\.conda\envs\rwkv-lora-pat\lib\site-packages\transformers\models\auto\auto_factory.py:566,
in _BaseAutoModelClass.from_pretrained(cls, pretrained_model_name_or_path, *model_args, **kwargs)
    562     model_class = _get_model_class(config, cls._model_mapping)
    563     return model_class.from_pretrained(
    564         pretrained_model_name_or_path, *model_args, config=config, **hub_kwargs, **kwargs
```

```
    565     )
--> 566 raise ValueError(
    567     f"Unrecognized configuration class {config.__class__} for this kind of AutoModel:
{cls.__name__}.\n"
    568     f"Model type should be one of {', '.join(c.__name__ for c in
cls._model_mapping.keys())}."
    569 )

ValueError: Unrecognized configuration class <class
'transformers.models.t5.configuration_t5.T5Config'> for this kind of AutoModel:
AutoModelForCausalLM.
Model type should be one of BartConfig, BertConfig, BertGenerationConfig, BigBirdConfig,
BigBirdPegasusConfig, BioGptConfig, BlenderbotConfig, BlenderbotSmallConfig, BloomConfig,
CamembertConfig, LlamaConfig, CodeGenConfig, CohereConfig, CpmAntConfig, CTRLConfig,
Data2VecTextConfig, DbrxConfig, ElectraConfig, ErnieConfig, FalconConfig, FuyuConfig, GemmaConfig,
GitConfig, GPT2Config, GPT2Config, GPTBigCodeConfig, GPTNeoConfig, GPTNeoXConfig,
GPTNeoXJapaneseConfig, GPTJConfig, JambaConfig, JetMoeConfig, LlamaConfig, MambaConfig,
MarianConfig, MBartConfig, MegaConfig, MegatronBertConfig, MistralConfig, MixtralConfig,
MptConfig, MusicgenConfig, MusicgenMelodyConfig, MvpConfig, OlmoConfig, OpenLlamaConfig,
OpenAIGPTConfig, OPTConfig, PegasusConfig, PersimmonConfig, PhiConfig, Phi3Config, PLBartConfig,
ProphetNetConfig, QDQBertConfig, Qwen2Config, Qwen2MoeConfig, RecurrentGemmaConfig,
ReformerConfig, RemBertConfig, RobertaConfig, RobertaPreLayerNormConfig, RoCBertConfig,
RoFormerConfig, RwkvConfig, Speech2Text2Config, StableLmConfig, Starcoder2Config, TransfoXLConfig,
TrOCRConfig, WhisperConfig, XGLMConfig, XLMConfig, XLMProphetNetConfig, XLMRobertaConfig,
XLMRobertaXLConfig, XLNetConfig, XmodConfig.
```

Trying some things I've been learning (architecture)

In [ ]:

```
print(model)
```

In [ ]:

```
model_arch_str = str(model)

with open("google_-flan-t5-small.model-architecture.txt", 'w', encoding='utf-8') as fh:
    fh.write(model_arch_str)
##endof:  with open ... fh
```

Some other saves

In [ ]:
```
pickle_filename = "lora_flan_t5_cpu_objects.pkl"
objects_to_pickle = []
objects_to_pickle.append(model_arch_str)
```

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 [ ]:
```
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 - the LoRA Setup Part

Arguments and Configuration

See this section to see what I changed from the tutorial to get the evaluation set as part of training and to get a customized repo name. The couple of sections before it will give more details.

In [ ]:
```
#  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,
                    do_eval=True,
                    per_device_eval_batch_size=4,
                    eval_strategy='epoch',
                    hub_model_id="dwb-flan-t5-small-lora-ft-colab",
                    run_name="dwb-flan-samsum-run-colab-20240606-02",
                    #  has nodename (machine), when this param is
                    #+ unset
                    overwrite_output_dir=False,
                    logging_strategy='steps',
                    logging_steps=32,
)

#  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 (archived)

> ```
> 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.
```

We're going to start timing stuff, so here's some system info

`system_info_as_script.py` is a script I wrote with the help of a variety of StackOverflow and documentation sources. It should be in the working directory.

In [ ]:
```
# # Don't need this again
!date +'%s_%Y%m%dT%H%M%S%z'
```

Output was:

`timestamp`

In [ ]:
```
system_info_as_script.run()
```

(Maybe try that `system_info_as_script.run()` command as `sudo` to see if we get any more info ...)

ROUGE Metrics

Some references from the Google Research implementation

https://pypi.org/project/rouge-score/

https://web.archive.org/web/20240530231357/https://pypi.org/project/rouge-score/

https://github.com/google-research/google-research/tree/master/rouge

https://web.archive.org/web/20240530231412/https://github.com/google-research/google-research/tree/master/rouge

Not the one I used:

https://github.com/microsoft/nlp-recipes/blob/master/examples/text_summarization/summarization_evaluation.ipynb

https://web.archive.org/web/20240530231709/https://github.com/microsoft/nlp-recipes/blob/master/examples/text_summarization/summarization_evaluation.ipynb

Someone else made this other one, which I inspected but didn't use.

https://pypi.org/project/rouge/

https://web.archive.org/web/20240530232029/https://pypi.org/project/rouge/

https://github.com/pltrdy/rouge

https://web.archive.org/web/20240530232023/https://github.com/pltrdy/rouge

but I think he defers to the rouge_score from Google.

My ROUGE Metrics incl SkipGrams but Not Using Now

I want to use the skip-grams score. Thanks to

https://www.bomberbot.com/machine-learning/skip-bigrams-in-system/

https://web.archive.org/web/20240530230949/https://www.bomberbot.com/machine-learning/skip-bigrams-in-system/

I can do this as well as writing the code for the other metrics.

Not used for now

Focusing on the main goal. Quick and Reckless. My therapist would be so proud.

Documentation for my methods

In [ ]:
```
#import dwb_rouge_scores # done with all other exports

sep_banner_1 = "  " + "#" + "+"*60 + "#"
sep_banner_2 = "     " + "#" + "~"*30 + "#"

print()
print()
print(sep_banner_1)

help(dwb_rouge_scores.dwb_rouge_n)

print()
print()
print(sep_banner_1)
print()
print()

help(dwb_rouge_scores.dwb_rouge_L)

print()
print(sep_banner_2)
print()
print("dwb_rouge_L needs dwb_lcs")
print()
print(sep_banner_2)
print()

help(dwb_rouge_scores.dwb_lcs)

print()
print()
print(sep_banner_1)
print()
```

```
print()

help(dwb_rouge_scores.dwb_rouge_s)

print()
print(sep_banner_2)
print()
print("dwb_rouge_s needs dwb_skipngrams")
print()
print(sep_banner_2)
print()

help(dwb_rouge_scores.dwb_skipngrams)

print()
print()
print(sep_banner_1)
print()
print()

help(dwb_rouge_scores.dwb_rouge_Lsum)

print()
print(sep_banner_2)
print()
print("dwb_rouge_Lsum just wraps google-research's rouge_score's")
print("(from `pip install rouge-score`) version of rougeLsum")
print()
print()
print(sep_banner_1)
```

Other useful ROUGE code - Run/Evaluate Code Even if You'll HIde It

(found and created as I go along)

In [ ]:

```
def format_rouge_score_rough(this_rouge_str):
    '''

    '''
```

```
    rouge_ret_str = this_rouge_str

    rouge_ret_str = re.sub(r"([(,][ ]?)([0-9A-Za-z_]+[=])",
                           "\g<1>\n     \g<2>",
                          rouge_ret_str,
                          flags=re.I|re.M
    )

    rouge_ret_str = re.sub(r"(.)([)])$",
                           "\g<1>\n\g<2>",
                          rouge_ret_str
    )

    rouge_ret_str = rouge_ret_str.replace(
                                 "precision=",
                                 "     precision="
                              ).replace(
                                 "recall=",
                                 "     recall="
                              ).replace(
                                 "fmeasure=",
                                 "     fmeasure="
    )

    return rouge_ret_str

##endof:  format_rouge_score_rough(<params>)
```

In [ ]:
```
def print_rouge_scores(result, sample_num_or_header=None):
    '''

    '''

    print("\n\n---------- ROUGE SCORES ----------")
    if sample_num_or_header is None:
        print("  --------- dialogue ----------")
    elif type(sample_num_or_header) is int:
        print(f"  --------- dialogue {sample_num_or_header+1} " + \
              "----------")
    else:
        print(f"  --------- {sample_num_or_header} ----------")
    ##endof:  if/else sample_num is None
```

```
    print("ROUGE-1 results")
    rouge1_str = str(result['rouge1'])
    print(format_rouge_score_rough(rouge1_str))
    print("ROUGE-2 results")
    rouge2_str = str(result['rouge2'])
    print(format_rouge_score_rough(rouge2_str))
    print("ROUGE-L results")
    rougeL_str = str(result['rougeL'])
    print(format_rouge_score_rough(rougeL_str))
    print("ROUGE-Lsum results")
    rougeLsum_str = str(result['rougeLsum'])
    print(format_rouge_score_rough(rougeLsum_str))
##endof:  print_rouge_scores(<params>)
```

In [ ]:

```
#---------------------------------------------------------------------------
# #  From https://github.com/google-research/google-research/tree/master/rouge
# #+ <strike>I can't see how to aggregate it, though I may have</strike>
# #+ I found a resource at
# #+  ref_gg_rg="https://github.com/huggingface/datasets/blob/" + \
# #+            "main/metrics/rouge/rouge.py"
# #+
# #+ arch_gg_rg="https://web.archive.org/web/20240603192938/" + \
# #+            "https://github.com/huggingface/datasets/blob/" + \
# #+            "main/metrics/rouge/rouge.py"
#

def compute_google_rouge_score(predictions,
                               references,
                               rouge_types=None,
                               use_aggregator=True,
                               use_stemmer=False):

    '''
    Figuring out the nice format of the deprecated method from
    the googleresearch/rouge method it claims to be calling.
    '''

    if rouge_types is None:
        rouge_types = ["rouge1", "rouge2", "rougeL", "rougeLsum"]
    ##endof:  if rouge_types is None

    scorer = rouge_scorer.RougeScorer(rouge_types=rouge_types,
```

```
                                             use_stemmer=use_stemmer
    )

    if use_aggregator:
        aggregator = scoring.BootstrapAggregator()
    else:
        scores = []
    ##endof:  if/else use_aggregator

    for ref, pred in zip(references, predictions):
        score = scorer.score(ref, pred)
        if use_aggregator:
            aggregator.add_scores(score)
        else:
            scores.append(score)
    ##endof:  for

    result = "there-is-some-problem" #  scoping (if we weren't
                                     #+ in Python) and having
                                     #+ a sort of error message

    if use_aggregator:
        result = aggregator.aggregate()
    else:
        result = {}
        for key in scores[0]:
            result[key] = [score[key] for score in scores]
        ##endof:  for
    ##endof:  if/else use_aggregator

    return result

##endof:  compute_google_rouge_score
```

I found a nice, short, interesting conversation while doing the random summaries, so I went and found its index.

In [ ]:

```
tic = timeit.default_timer()

str_to_find = "Damien: Omg..I'm glad Sunday is only once a week"

for sample_num in range(len(dataset['test'])):
```

```
    this_sample = dataset['test'][sample_num]
    this_dialogue = this_sample['dialogue']
    if str_to_find in this_dialogue:
        print(f"sample_num: {sample_num}")
        print(f"this_dialogue: \n{this_dialogue}")
        print()
        print("this_sample:")
        print(str(this_sample))
        print()
    ##endof:  if str_to_find in this_dialogue
##endof:  for sample_number in range(len(dataset))

toc = timeit.default_timer()

print("Finding the sample in the test dataset (well,")
print("actually looking at every sample in the test")
print("dataset, regardless of whether we had found")
print("something.")
print(f"took {toc - tic:0.4f} seconds.")

my_duration = toc - tic

elapsed_time_str = format_timespan(my_duration)

print(f"which equates to {elapsed_time_str}")

print()

print(f"Total size of test dataset: {sample_num}")
```

The interesting conversation

In [ ]:
```
my_index = 224
my_complete_entry = dataset['test'][sample_num]
my_cool_str = dataset['test'][sample_num]['dialogue']
print(my_cool_str)
objects_to_pickle.append(my_cool_str)
my_cool_list = [f"my_index: {my_index}", my_cool_str, my_complete_entry]
pprint.pp(my_cool_list)
objects_to_pickle.append(my_cool_list)
```

Let's get the sizes of all parts of the dataset

In [ ]:
```
size_of_train = len(dataset['train'])
size_of_eval = len(dataset['evaluation'])
size_of_test = len(dataset['test'])

print(f"size_of_train : {size_of_train}")
print(f"size_of_eval  : {size_of_eval}")
print(f"size_of_test  : {size_of_test}")
```

Try for a baseline (for out-of-the-box, pretrained model)

In [ ]:
```
# # Don't need this again
!date +'%s_%Y%m%dT%H%M%S%z'
```

Output was:

`timestamp`

Just one summarization to begin with, randomly picked

Well, not so randomly, anymore

In [ ]:
```
#  Just one summarization to begin with, randomly picked ... but
#+ now with th possibility of a known seed, to allow visual
#+ comparison with after-training results.
#+ I'M NOT GOING TO USE THIS REPEATED SEED, I'm just going to
#+ use the datum at the first index to compare.
#
#  For sharing with Pat, I'm making it repeatable

do_seed_for_repeatable = True

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

if do_seed_for_repeatable:
```

```
    rand_seed_for_randrange = 137
    random.seed(rand_seed_for_randrange)
##endof:  if do_seed_for_repeatable

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']}")
```

Now, a couple summarizations with comparisons to ground truth

In [ ]:
```
summarizer = pipeline('summarization',
                      model=model,
                      tokenizer=tokenizer)

pred_test_list = []
ref_test_list = []

sample_num = 0

this_sample = dataset['test'][sample_num]

print(f"dialogue: \n{this_sample['dialogue']}\n---------------")

ground_summary = this_sample['summary']
res = summarizer(this_sample['dialogue'])
res_summary = res[0]['summary_text']

print(f"human-genratd summary:\n{ground_summary}")
print(f"flan-t5-small summary:\n{res_summary}")

ref_test_list.append(ground_summary)
pred_test_list.append(res_summary)

#  Yes, I have just one datum, but I'm setting things up to
#+ work well with a later loop, i.e. with lists
results_test_0 = compute_google_rouge_score(
                            predictions=pred_test_list,
                            references=ref_test_list,
```

```
                             use_aggregator=False
)

# >>> print(list(results_test.keys()))
# ['rouge1', 'rouge2', 'rougeL', 'rougeLsum']
```

In [ ]:
```
print_rouge_scores(results_test_0, 0)
```

In [ ]:
```
summarizer = pipeline('summarization',
                      model=model,
                      tokenizer=tokenizer)

#  I don't want to aggregate, yet.
pred_test_list = []
ref_test_list = []

sample_num = 224

this_sample = dataset['test'][sample_num]

print(f"dialogue: \n{this_sample['dialogue']}\n---------------")

ground_summary = this_sample['summary']
res = summarizer(this_sample['dialogue'])
res_summary = res[0]['summary_text']

print(f"human-genratd summary:\n{ground_summary}")
print(f"flan-t5-small summary:\n{res_summary}")

ref_test_list.append(ground_summary)
pred_test_list.append(res_summary)

results_test_224 = compute_google_rouge_score(
                              predictions=pred_test_list,
                              references=ref_test_list,
                              use_aggregator=False
)
```

In [ ]:
```
print_rouge_scores(results_test_224, 224)
```

Note on ROUGE Scores

`~~@todo~~` `DONE : Run the ROUGE analysis from the Python package`

The package used by the HuggingFace `datasets.load_metric` method is at

https://github.com/google-research/google-research/tree/master/rouge

I can't see how to aggregate it, though I may have found a resource at

```
ref_gg_rg = "https://github.com/huggingface/datasets/blob/" + \\
            "main/metrics/rouge/rouge.py"

arch_gg_rg = "https://web.archive.org/web/20240603192938/" + \\
             "https://github.com/huggingface/datasets/blob/" + \\
             "main/metrics/rouge/rouge.py"
```

~~It turns out that the deprecated one is preferable in output, at least until I can debug the aggregation of scores with another version: compute_google_rouge_score~~

I've now got the `compute_google_rouge_score` method, above. I was able to look through the code for `datasets.load_metric('rouge')` code and put together that new method.

So now, I'm not using any `rouge` object, but simply doing

```
these_results = compute_google_rouge_score(
                           predictions, references, use_aggregator)
```

This next one is what the warning/deprecation message for `load_metric` said to use, but it only returns an f-measure (f-score)

```
# #  Replacement for the load_metric - evaluate.load(metric_name)
# #+ Docs said:
# #+
# #+> Returns:
# #+>    rouge1: rouge_1 (f1),
# #+>    rouge2: rouge_2 (f1),
# #+>    rougeL: rouge_l (f1),
# #+>    rougeLsum: rouge_lsum (f1)
```

```
# #+>
# #+> Meaning we only get the f-score. I want more to compare.
# #-v- code
# rouge = evaluate_dot_load('rouge')
```

Verbosity stuff - get rid of the nice advice

In [ ]:
```
# # Don't need this again
!date +'%s_%Y%m%dT%H%M%S%z'
```

Output was:

`timestamp`

In [ ]:
```
log_verbosity_is_critical = \
  logging.get_verbosity() == logging.CRITICAL # alias FATAL, 50
log_verbosity_is_error = \
  logging.get_verbosity() == logging.ERROR # 40
log_verbosity_is_warn = \
  logging.get_verbosity() == logging.WARNING # alias WARN, 30
log_verbosity_is_info = \
  logging.get_verbosity() == logging.INFO # 20
log_verbosity_is_debug = \
  logging.get_verbosity() == logging.DEBUG # 10

print( "The statement, 'logging verbosity is CRITICAL' " + \
      f"is {log_verbosity_is_critical}")
print( "The statement, 'logging verbosity is    ERROR' " + \
      f"is {log_verbosity_is_error}")
print( "The statement, 'logging verbosity is  WARNING' " + \
      f"is {log_verbosity_is_warn}")
print( "The statement, 'logging verbosity is     INFO' " + \
      f"is {log_verbosity_is_info}")
print( "The statement, 'logging verbosity is    DEBUG' " + \
      f"is {log_verbosity_is_debug}")

print()

init_log_verbosity = logging.get_verbosity()
print(f"The value of logging.get_verbosity() is: {init_log_verbosity}")
```

```
print()

init_t_n_a_w = os.environ.get('TRANSFORMERS_NO_ADVISORY_WARNINGS')
print(f"TRANSFORMERS_NO_ADIVSORY_WARNINGS: {init_t_n_a_w}")
```

Actual Baseline on Complete Test Set

In [ ]:
```
# # Don't need this again
!date +'%s_%Y%m%dT%H%M%S%z'
```

Output was:

`timestamp`

!!! NOTE You'd better **make dang sure you want the lots of output** before you set this next boolean to `True`

In [ ]:
```
do_have_lotta_output_from_all_dialogs_summaries_1 = False
```

Are you sure about the value of that last boolean? 1

In [ ]:
```
print("That last boolean has the value:")
print(f"{do_have_lotta_output_from_all_dialogs_summaries_1}")
```

There could be up to megabytes worth of text output if you've changed it to `True`.

In [ ]:
```
#  ref1 = "https://web.archive.org/web/20240530051418/" + \
#+        "https://stackoverflow.com/questions/73221277/" + \
#+        "python-hugging-face-warning"
#  ref2 = "https://web.archive.org/web/20240530051559/" + \
#+        "https://huggingface.co/docs/transformers/en/" + \
#+        "main_classes/logging"


##  Haven't tried this, because the logging seemed easier,
##+ and the logging worked
#os.environ("TRANSFORMERS_NO_ADVISORY_WARNINGS") = 1
```

```
logging.set_verbosity_error()

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

baseline_sample_dialog_list = [] ##  Keeping for comparison, later
                                 ##+ Not needed for scores.
baseline_prediction_list = []
baseline_reference_list = []

baseline_tic = timeit.default_timer()

for sample_num in range(len(dataset['test'])):
    this_sample = dataset['test'][sample_num]

    if do_have_lotta_output_from_all_dialogs_summaries_1:
        print(f"dialogue: \n{this_sample['dialogue']}\n---------------")
    ##endof:  if do_have_lotta_output_from_all_dialogs_summaries_1

    ground_summary = this_sample['summary']
    res = summarizer(this_sample['dialogue'])
    res_summary = res[0]['summary_text']

    if do_have_lotta_output_from_all_dialogs_summaries_1:
        print(f"human-genratd summary:\n{ground_summary}")
        print(f"flan-t5-small summary:\n{res_summary}")
    ##endof:  if do_have_lotta_output_from_all_dialogs_summaries_1

    baseline_sample_dialog_list.append(this_sample['dialogue'])
    baseline_reference_list.append(ground_summary)
    baseline_prediction_list.append(res_summary)
##endof:  for sample_num in range(len(dataset['test']))

baseline_toc = timeit.default_timer()

baseline_duration = baseline_toc - baseline_tic

print( "Getting things ready for scoring (doing the baseline)")
print(f"took {baseline_toc - baseline_tic:0.4f} seconds.")
```

```
baseline_time_str = format_timespan(baseline_duration)

print(f"which equates to {baseline_time_str}")

baseline_results = compute_google_rouge_score(
                          predictions=baseline_prediction_list,
                          references=baseline_reference_list,
                          use_aggregator=True
)

# >>> print(list(baseline_results.keys()))
# ['rouge1', 'rouge2', 'rougeL', 'rougeLsum']

objects_to_pickle.append(baseline_sample_dialog_list)
objects_to_pickle.append(baseline_prediction_list)
objects_to_pickle.append(baseline_reference_list)
objects_to_pickle.append(baseline_results)
```

In [ ]:
```
##  Haven't tried this, because the logging seemed easier,
##+ and the logging worked
# os.environ("TRANSFORMERS_NO_ADVISORY_WARNINGS") = init_t_n_a_w

logging.set_verbosity(init_log_verbosity)
```

In [ ]:
```
print_rouge_scores(baseline_results, "BASELINE")
```

Trainer - the Actual Trainer Part

In [ ]:
```
# # Don't need this again
!date +'%s_%Y%m%dT%H%M%S%z'
```

Output was:

`timestamp`

In [ ]:
```
trainer = SFTTrainer( model=model,
                      train_dataset=dataset['train'],
                      eval_dataset=dataset['evaluation'],
                      peft_config=peft_config,
```

```
                      tokenizer=tokenizer,
                      packing=True,
                      formatting_func=prompt_instruction_format,
                      args=training_args,
)
##  Warnings are below output.
```

Warnings I Won't Worry About, Yet

First time warnings from the code above (as it still is).

> ```
> WARNING:bitsandbytes.cextension:The installed version of bitsandbytes \
>  was compiled without GPU support. 8-bit optimizers, 8-bit multiplication, \
>  and GPU quantization are unavailable.
> C:\Users\bballdave025\.conda\envs\rwkv-lora-pat\lib\site-packages\trl\\
>  trainer\sft_trainer.py:246: UserWarning: You didn't pass a `max_seq_length` \
> argument to the SFTTrainer, this will default to 512
>  warnings.warn(
>
> [ > Generating train split: 6143/0 [00:04<00:00, 2034.36 examples/s] ]
>
> Token indices sequence length is longer than the specified maximum sequence \
>  length for this model (657 > 512). Running this sequence through the model \
>  will result in indexing errors
>
> [ > Generating train split: 355/0 [00:00<00:00, 6.10 examples/s] ]
> ```

DWB Note and possible

~~# @todo:~~

~~So, I'm changing the `max_seq_length`~~ : Maybe I should just throw out the offender(s) (along with the blank one that's in there somewhere), but I'll just continue as is.

I never ran the updated cell, (with an additional parameter, `max_seq_length=675`), so the Warning and Advice are still there.

Let's Train This LoRA Thing and See How It Does!

In [ ]:
```
# # Don't need this again
!date +'%s_%Y%m%dT%H%M%S%z'
```

Output was:

`timestamp`

The long-time-taking training code is just below.

In [ ]:
```
tic = timeit.default_timer()
trainer.train()
toc = timeit.default_timer()
print(f"tic: {tic}")
print(f"toc: {toc}")
training_duration = toc - tic
print(f"Training took {toc - tic:0.4f} seconds.")
training_time_str = format_timespan(training_duration)
print(f"which equates to {training_time_str}")
```

In [ ]:
```
# # Don't need this again
!date +'%s_%Y%m%dT%H%M%S%z'
```

Output was:

`timestamp`

Thinking about it and learning

@todo : consolidate "the other info as above"

I'm talking about the numbers of data points, tokens, whatever.

Any Comments / Things to Try (?)

We passed an evaluation set (parameter `eval_dataset`) to the `trainer`. How can we see information about that?

Update: Answer is below.

How to get the evaluation set used by the trainer

I added the following parameters to the `training_args = TrainingArguments(<args>)` call.

- `do_eval=True`
- `per_device_eval_batch_size=4`
- `eval_strategy='epoch'`

How to specify your repo name

I also added this next parameter to the arguments for `training_args = TrainingArguments(<args>)`

- `hub_model_id="dwb-flan-t5-small-lora-ft-colab"`

The final TrainingArguments call - with parameter list

Including four additional parameters

```
training_args = TrainingArguments(
                    output_dir='output',
                    num_train_epochs=1,
                    per_device_train_batch_size=4,
                    save_strategy='epoch',
                    learning_rate=2e-4,
                    do_eval=True,
                    per_device_eval_batch_size=4,
                    eval_strategy='epoch',
                    hub_model_id="dwb-flan-t5-small-lora-ft-colab",
                    run_name="dwb-flan-samsum-run-colab-20240606-02",
                    #  has nodename (machine), when this param is
                    #+ unset
                    overwrite_output_dir=False,
                    logging_strategy='steps',
```

```
                         logging_steps=32,
    )
```

Save the Trainer to Hugging Face and Get Our Updated Model

In [ ]:
```
# # Don't need this again
!date +'%s_%Y%m%dT%H%M%S%z'
```

Output was:

`timestamp`

I'm following the (archived) tutorial from Mehul Gupta on Medium; since it's archived, you can follow exactly what I'm doing.

Running this next line of code will come up with a dialog box with text entry, and I'm now using the `@thebballdave025` for Hugging Face stuff.

Make sure to use the WRITE token, here.

In [ ]:
```
#  This will come up with a dialog box with text entry.
#+ and I'm now using @thebballdave025 for Hugging Face.

# Use the write token, here.
notebook_login()
```

In [ ]:
```
# Save tokenizer and create a tokenizer model card
tokenizer.save_pretrained('testing')
  #  'testing' is the local directory

# Create the trainer model card
trainer.create_model_card()

# Push the results to the Hugging Face Hub
trainer.push_to_hub()
```

Hugging Face Repo Info

Part of the output included text giving the URL,

https://huggingface.co/thebballdave025/dwb-flan-t5-small-lora-finetune/commit/c87d34b398f3801ceb1e18c819a7c8fc894989c7

Hooray! The repo name I used in constructing the trainer worked!

I can get to the general repo with the URL,

https://huggingface.co/thebballdave025/dwb-flan-t5-small-lora-finetune

---

Info on the Fine-Tuned Model from the Repo's README - Model Card(?)

thebballdave025/dwb-flan-t5-small-lora-finetune

[archived] The archiving attempt at archive.org (Wayback Machine) failed. I'm not sure why, as the model is set as public.

```
PEFT TensorBoard Safetensors    generator trl sft generated_from_trainer    License: apache-2.0
```

[**@todo** :] Edit Model Card

Unable to determine this model's pipeline type. Check the docs (i).

Adapter for google/flan-t5-small

dwb-flan-t5-small-lora-finetune

This model is a fine-tuned version of google/flan-t5-small on the generator dataset [DWB note: I don't know why it says "generator dataset". I used the samsum dataset, which I will link here and on the model card, eventually].

It achieves the following results on the evaluation set:

- Loss: 0.0226

- *DWB Note: I don't know which metric was used to calculate loss. If this were more important, I'd dig through code to find out and evaluate with the same metric. If I'm really lucky, they somehow used the ROUGE scores in the loss function, so we match.*

Model description

More information needed

Intended uses & limitations

More information needed

Training and evaluation data

More information needed

Training procedure

Training hyperparameters

The following hyperparameters were used during training:

- learning_rate: 0.0002
- train_batch_size: 4
- eval_batch_size: 4
- seed: 42
- optimizer: Adam with betas=(0.9,0.999) and epsilon=1e-08
- lr_scheduler_type: linear
- num_epochs: 1

Training results

```
 Training Loss | Epoch | Step | Validation Loss
---------------+-------+------+-----------------
    0.0685     |  1.0  | 1536 |     0.0226
```

Framework versions

- PEFT 0.11.2.dev0
- Transformers 4.41.1
- Pytorch 2.3.0+cpu
- Datasets 2.19.1
- Tokenizers 0.19.1

Actually Get the Model from Hugging Face

Running this next line of code will come up with a dialog box with text entry, and I'm now using the `@thebballdave025` for Hugging Face stuff.

Make sure to use the READ token, here.

In [ ]:
```
# Read token. Will bring up text entry to paste token string
notebook_login()
```

In [ ]:
```
# # Don't need this again
!date +'%s_%Y%m%dT%H%M%S%z'
```

Output was:

`timestamp`

(If you have problems that note `data_files` or `dataset` or `prompt_instruction_format`, make sure that the cells where these are defined have been run, i.e. the kernel hasn't been restarted since they were initialized.)

In [ ]:
```
# My trained model from Hugging Face

new_model_name = "thebballdave025/dwb-flan-t5-small-lora-ft-colab"
```

In [ ]:
```
new_model_load_tic = timeit.default_timer()

#  Maybe the problem is that we need to change from Seq2Seq to Causal
#+ but I think I only use the new_model_name (at least in inference;
#+ I do use the actual model to look at the LoRA-ified architecture).
```

```
new_model = AutoModelForSeq2SeqLM.from_pretrained(new_model_name)
#new_model = AutoModelForCausalLM.from_pretrained(new_model_name)

new_model_load_toc = timeit.default_timer()

new_model_load_duration = new_model_load_toc - new_model_load_tic

print(f"Loading the LoRA-fine-tuned model, {new_model_name}")
print(f"took {new_model_load_toc - new_model_load_tic:0.4f} seconds.")

new_model_load_time_str = format_timespan(new_model_load_duration)

print(f"which equates to {new_model_load_time_str}")

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

new_tokenizer_tic = timeit.default_timer()
new_tokenizer = AutoTokenizer.from_pretrained(
                                   new_model_name,
                                   trust_remote_code=True)
new_tokenizer_toc = timeit.default_timer()

new_tokenizer_duration = new_tokenizer_toc - new_tokenizer_tic

print()
print("Getting fine-turned tokenizer")
print(f"took {new_tokenizer_toc - new_tokenizer_tic:0.4f} seconds.")

new_tokenizer_time_str = format_timespan(new_tokenizer_duration)

print(f"which equates to {new_tokenizer_time_str}")

new_tokenizer.pad_token = new_tokenizer.eos_token
new_tokenizer.padding_side = "right"

print()
print()

#--------------------------------------------------------
# Got some weird results, so I'm doing the old tokenizer
```

```
old_model_name = "google/flan-t5-small"

old_model_load_tic = timeit.default_timer()
old_model = AutoModelForSeq2SeqLM.from_pretrained(old_model_name)
old_model_load_toc = timeit.default_timer()

old_model_load_duration = \
           old_model_load_toc - old_model_load_tic

print(f"Loading the old model, {old_model_name}")
print("took " + \
      f"{old_model_load_toc - old_model_load_tic:0.4f}" + \
      " seconds."
)

old_model_load_time_str = format_timespan(old_model_load_duration)

print(f"which equates to {old_model_load_time_str}")

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

old_tokenizer_tic = timeit.default_timer()
old_tokenizer = AutoTokenizer.from_pretrained(
                                       old_model_name,
                                       trust_remote_code=True
)
old_tokenizer_toc = timeit.default_timer()

old_tokenizer_duration = old_tokenizer_toc - old_tokenizer_tic

print()
print("Getting old tokenizer")
print( "took " + \
      f"{old_tokenizer_toc - old_tokenizer_tic:0.4f}"
       " seconds."
)

old_tokenizer_time_str = format_timespan(old_tokenizer_duration)

print(f"which equates to {tokenizer_time_str}")
```

```
#  padding instructions for the tokenizer
old_tokenizer.pad_token = tokenizer.eos_token
old_tokenizer.padding_side = "right"
```

Stuff for model architecture - post-LoRA

In [ ]:
```
print(new_model)
```

In [ ]:
```
new_model_arch_str = str(new_model)

with open(
  "dwb-flan-t5-small-lora-ft-colab.model-architecture.txt",
  'w',
  encoding='utf-8') as fhn:
    fhn.write(new_model_arch_str)
##endof:  with open ... fhn

objects_to_pickle.append(new_model_arch_str)
```

@todo : get some Python version of `diff` going on here. I'm just using Cygwin/bash to see the LoRA additions.

Let's start by doing the single-dialogue summaries we used before.

In [ ]:
```
# if you want to keep it consistent, use these. If not, change them at will
model_to_use = new_model
tokenizer_to_use = new_tokenizer
```

Try one picked at random

Well, not so randomly, anymore

In [ ]:
```
#  Just one summarization to begin with, randomly picked ... but
#+ now with th possibility of a known seed, to allow visual
#+ comparison with after-training results.
#+ I'M NOT GOING TO USE THIS REPEATED SEED, I'm just going to
#+ use the datum at the first index to compare.
#
#  User repeatability when sharing with Pat
```

```
do_seed_for_repeatable = True

#  Gupta doesn't have a `tokenizer` argument. I seem to remember getting
#+ an error when I tried that.
summarizer = pipeline('summarization',
                      model=model_to_use) #,
                      #tokenizer=tokenizer_to_use)

## Trials to fix weirdness.
## model=old_model, tokenizer=old_tokenizer : matches baseline
##                                            (quick)
## model=new_model, tokenizer=new_tokenizer : weird results
##                                            (takes significantly longer, too)
## model=new_model, tokenizer=old_tokenizer : weird results
##                                            (takes significantly longer, too)
## model=old_model, tokenizer=new_tokenizer : actually matches baseline, which
##                                            would seem to require a change in
##                                            hypothesis as to why the
##                                            weirdness and longer inference are
##                                            happening. (Likely not tokenizer.)
##                                            (quick)
##
##  I had thought that doing 'old_model' and 'new_tokenizer' gave me weird
##+ results, too. Good thing to come back and check things.
##  Still, the training

if do_seed_for_repeatable:
    rand_seed_for_randrange = 137
    random.seed(rand_seed_for_randrange)
##endof:  if do_seed_for_repeatable

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

res = summarizer(sample["dialogue"])

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

Now, a couple summarizations with comparison to ground truth

In [ ]:
```
summarizer = pipeline('summarization',
                      model=new_model) #,
                      #tokenizer=new_tokenizer)

pred_test_list = []
ref_test_list = []

sample_num = 0

this_sample = dataset['test'][sample_num]

print(f"dialogue: \n{this_sample['dialogue']}\n---------------")

ground_summary = this_sample['summary']
res = summarizer(this_sample['dialogue'])
res_summary = res[0]['summary_text']

print(f"by-some-human-generated summary:\n{ground_summary}")
print(f"dwb-flan-t5-small-lora-ft-colab:\n{res_summary}")

ref_test_list.append(ground_summary)
pred_test_list.append(res_summary)

# deprecated, blah blah blah
#rouge = load_metric('rouge', trust_remote_code=True)

#  Yes, I have just one datum, but I'm setting things up to
#+ work well with a loop (meaning lists for pred and ref).
results_test_0 = compute_google_rouge_score(
                            predictions=pred_test_list,
                            references=ref_test_list,
                            use_aggregator=False
)

# >>> print(list(results_test.keys()))
# ['rouge1', 'rouge2', 'rougeL', 'rougeLsum']
```

In [ ]:
```
print_rouge_scores(results_test_0, 0)
```

In [ ]:
```
summarizer = pipeline('summarization',
                      model=model_to_use) #,
```

```
                      #tokenizer=tokenizer_to_use)

# I don't want to aggregate, yet
pred_test_list = []
ref_test_list = []

sample_num = 224

this_sample = dataset['test'][sample_num]

print(f"dialogue: \n{this_sample['dialogue']}\n---------------")

ground_summary = this_sample['summary']
res = summarizer(this_sample['dialogue'])
res_summary = res[0]['summary_text']

print(f"by-some-human-generated summary:\n{ground_summary}")
print(f"dwb-flan-t5-small-lora-ft-colab:\n{res_summary}")

ref_test_list.append(ground_summary)
pred_test_list.append(res_summary)

results_test_224 = compute_google_rouge_score(
                              predictions=pred_test_list,
                              references=ref_test_list,
                              use_aggregator=False
)
```

In [ ]:
```
print_rouge_scores(results_test_224, 224)
```

Evaluation on the Test Set and Comparison to Baseline

Verbosity stuff - get rid of the nice advice

In [ ]:
```
# # Don't need this again
!date +'%s_%Y%m%dT%H%M%S%z'
```

Output was:

`timestamp`

In [ ]:
```
log_verbosity_is_critical = \
  logging.get_verbosity() == logging.CRITICAL # alias FATAL, 50
log_verbosity_is_error = \
  logging.get_verbosity() == logging.ERROR # 40
log_verbosity_is_warn = \
  logging.get_verbosity() == logging.WARNING # alias WARN, 30
log_verbosity_is_info = \
  logging.get_verbosity() == logging.INFO # 20
log_verbosity_is_debug = \
  logging.get_verbosity() == logging.DEBUG # 10

print( "The statement, 'logging verbosity is CRITICAL' " + \
      f"is {log_verbosity_is_critical}")
print( "The statement, 'logging verbosity is    ERROR' " + \
      f"is {log_verbosity_is_error}")
print( "The statement, 'logging verbosity is  WARNING' " + \
      f"is {log_verbosity_is_warn}")
print( "The statement, 'logging verbosity is     INFO' " + \
      f"is {log_verbosity_is_info}")
print( "The statement, 'logging verbosity is    DEBUG' " + \
      f"is {log_verbosity_is_debug}")

print()

init_log_verbosity = logging.get_verbosity()
print(f"The value of logging.get_verbosity() is: {init_log_verbosity}")

print()

init_t_n_a_w = os.environ.get('TRANSFORMERS_NO_ADVISORY_WARNINGS')
print(f"TRANSFORMERS_NO_ADIVSORY_WARNINGS: {init_t_n_a_w}")
```

Here's the actual evaluation

In [ ]:
```
# # Don't need this again
!date +'%s_%Y%m%dT%H%M%S%z'
```

Output was:

`timestamp`

!!! NOTE !!! I'm going to use `tat` (with an underscore or undescores before, after, or surrounding the variable names) to indicate 'testing-after-training'.

I guess I could have used `inference`, but I didn't.

!!! another NOTE You'd better **make dang sure you want the lots of output** before you set this next boolean to `True`

In [ ]:
```
do_have_lotta_output_from_all_dialogs_summaries_2 = False
```

Are you sure about the value of that last boolean? 2

In [ ]:
```
print("That last boolean has the value:")
print(f"{do_have_lotta_output_from_all_dialogs_summaries_2}")
```

There could be up to megabytes worth of text output if you've changed it to `True`.

In [ ]:
```
logging.set_verbosity_error()

tat_summarizer = pipeline('summarization',
                          model=new_model) #,
                          #tokenizer=new_tokenizer)

tat_sample_dialog_list = [] ##  Keeping for comparison, later
                            ##+ Not needed for scores.
prediction_tat_list = []
reference_tat_list = []

tat_tic = timeit.default_timer()

for sample_num in range(len(dataset['test'])):
    this_sample = dataset['test'][sample_num]

    if do_have_lotta_output_from_all_dialogs_summaries_2:
        print("="*75)
        print(f"dialogue: \n{this_sample['dialogue']}\n---------------")
```

```
    ##endof:  if do_have_lotta_output_from_all_dialogs_summaries_2

    ground_tat_summary = this_sample['summary']
    res_tat = summarizer(this_sample['dialogue'])
    res_tat_summary = res_tat[0]['summary_text']

    if do_have_lotta_output_from_all_dialogs_summaries_2:
        print("-"*70)
        print( "by-some-human-generated summary:" + \
              f"\n{ground_tat_summary}")
        print("-"*70)
        print( "dwb-flan-t5-small-lora-ft-colab:" + \
              f"\n{res_tat_summary}")
        print("-"*70)
    ##endof:  if do_have_lotta_output_from_all_dialogs_summaries_2

    tat_sample_dialog_list.append(this_sample['dialogue'])
    reference_tat_list.append(ground_tat_summary)
    prediction_tat_list.append(res_tat_summary)
##endof:  for sample_num in range(len(dataset['test']))

tat_toc = timeit.default_timer()

tat_duration = tat_toc = tat_tic

print( "Getting things ready for scoring (after training)")
print(f"took {tat_toc - tat_tic:0.4f} seconds.")

tat_time_str = format_timespan(tat_duration)

print(f"which equates to {tat_time_str}")

results_tat = compute_google_rouge_score(
                         predictions=prediction_tat_list,
                         references=reference_tat_list,
                         use_aggregator=True
)

objects_to_pickle.append(tat_sample_dialog_list)
objects_to_pickle.append(prediction_tat_list)
objects_to_pickle.append(reference_tat_list)
objects_to_pickle.append(results_tat)
```

In [ ]:
```
##  Haven't tried this, because the logging seemed easier,
##+ and the logging worked
# os.environ("TRANSFORMERS_NO_ADVISORY_WARNINGS") = init_t_n_a_w

logging.set_verbosity(init_log_verbosity)
```

In [ ]:
```
print_rouge_scores(results_tat, "TEST AFTER TRAINING")
```

In [ ]:
```
# # Don't need this anymore
!date +'%s_%Y%m%dT%H%M%S%z'
```

Output was:

`timestamp`

Any comparison

In [ ]:
```
# any comparison code
```

Pickle things to pickle save

In [ ]:
```
objects_to_pickle_var_names = []

objects_to_pickle_var_names.append('model_arch_str')
objects_to_pickle_var_names.append('my_cool_str')
objects_to_pickle_var_names.append('my_cool_list')
objects_to_pickle_var_names.append('baseline_sample_dialog_list')
objects_to_pickle_var_names.append('baseline_prediction_list')
objects_to_pickle_var_names.append('baseline_reference_list')
objects_to_pickle_var_names.append('baseline_results')
objects_to_pickle_var_names.append('new_model_arch_str')
objects_to_pickle_var_names.append('tat_sample_dialog_list')
objects_to_pickle_var_names.append('prediction_tat_list')
```

```
objects_to_pickle_var_names.append('reference_tat_list')
objects_to_pickle_var_names.append('results_tat')
objects_to_pickle_var_names.append('objects_to_picle_var_names')

objects_to_pickle.append(objects_to_pickle_var_names)

with open(pickle_filename, 'wb') as pfh:
    pickle.dump(objects_to_pickle , pfh)
##endof:  with open ... as pfh # (pickle file handle)
```

Notes Looking Forward to LoRA on RWKV

Hugging Face Community, seems to have a good portion of their models

https://huggingface.co/RWKV

https://web.archive.org/web/20240530232509/https://huggingface.co/RWKV

GitHub has even more versions/models, including the `v4-neo` that I think will be important (the LoRA project)

https://github.com/BlinkDL/RWKV-LM/tree/main

https://web.archive.org/web/20240530232637/https://github.com/BlinkDL/RWKV-LM/tree/main

The main RWKV website (?!)

https://www.rwkv.com/

https://web.archive.org/web/20240529120904/https://www.rwkv.com/

GOOD STUFF. A project doing LoRA with RWKV

https://github.com/Blealtan/RWKV-LM-LoRA/

https://web.archive.org/web/20240530232823/https://github.com/Blealtan/RWKV-LM-LoRA

The official blog, I guess, with some good coding examples

https://huggingface.co/blog/rwkv

https://web.archive.org/web/20240530233025/https://huggingface.co/blog/rwkv

It includes something that's similar to what I'm doing here in the `First_Full_LoRA_Trial_with_Transformer_Again.ipynb` tutorial, etc.

```
from transformers import AutoTokenizer, AutoModelForCausalLM

model_id = "RWKV/rwkv-raven-1b5"

model = AutoModelForCausalLM.from_pretrained(model_id).to(0)
tokenizer = AutoTokenizer.from_pretrained(model_id)
```

The `AutoModelForCausalLM` is the same as the tutorial I'm following, but I don't know what the `.to(0)` is for.

Really quickly, also looking at

https://huggingface.co/RWKV/rwkv-4-world-7b

https://web.archive.org/web/20240530234438/https://huggingface.co/RWKV/rwkv-4-world-7b

I see an example for CPU.

```
model = AutoModelForCausalLM.from_pretrained(
              "RWKV/rwkv-4-world-7b",
              trust_remote_code=True
```

```
    ).to(torch.float32)

    tokenizer = AutoTokenizer.from_pretrained(
                  "RWKV/rwkv-4-world-7b",
                  trust_remote_code=True)
```

(Old version? Unofficial, it seems)

https://huggingface.co/docs/transformers/en/model_doc/rwkv

https://web.archive.org/web/20240530232341/https://huggingface.co/docs/transformers/en/model_doc/rwkv