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 [1]:
```
# # Don't need this again
# !powershell -c (Get-Date -UFormat \"%s_%Y%m%dT%H%M%S%Z00\") -replace '[.][0-9]*_', '_'"
```

```
1717091264_20240530T174744-0600
```

Output was:

```
1717091264_20240530T174744-0600
```

Imports

In [2]:
```
from datasets import load_dataset
from random import randrange
import torch
from transformers import AutoTokenizer, \
                         AutoModelForSeq2SeqLM, \
                         TrainingArguments, \
                         pipeline
from transformers.utils import logging
from peft import LoraConfig, \
```

```
                prepare_model_for_kbit_training, \
                get_peft_model, \
                AutoPeftModelForCausalLM
from trl import SFTTrainer
from huggingface_hub import login, notebook_login

from datasets import load_metric
import nltk
import rouge_score

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

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.)

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

In [3]:
```
#  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',
              'evaluation':'samsum-validation.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 !!! ???
#+ Will it be the same?
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"
```

```
Generating train split: 0 examples [00:00, ? examples/s]
Generating evaluation split: 0 examples [00:00, ? examples/s]
Generating test split: 0 examples [00:00, ? examples/s]
config.json:   0%|          | 0.00/1.40k [00:00<?, ?B/s]
C:\Users\Anast\.conda\envs\rwkv-lora-pat\lib\site-packages\huggingface_hub\file_download.py:157: UserWarning: `huggingface_hub` cache-system uses symlinks by default to efficiently store duplicated files but your machine does not support them in C:\Users\Anast\.cache\huggingface\hub\models--google--flan-t5-small. Caching files will still work but in a degraded version that might require more space on your disk. This warning can be disabled by setting the `HF_HUB_DISABLE_SYMLINKS_WARNING` environment variable. For more details, see https://huggingface.co/docs/huggingface_hub/how-to-cache#limitations.
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In order to see activate developer mode, see this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-device-for-development
  warnings.warn(message)
model.safetensors:   0%|          | 0.00/308M [00:00<?, ?B/s]
generation_config.json:   0%|          | 0.00/147 [00:00<?, ?B/s]
tokenizer_config.json:   0%|          | 0.00/2.54k [00:00<?, ?B/s]
spiece.model:   0%|          | 0.00/792k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/2.42M [00:00<?, ?B/s]
special_tokens_map.json:   0%|          | 0.00/2.20k [00:00<?, ?B/s]
```

Trying some things I've been learning

In [4]:
```
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)
)
```

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

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 [9]:
```
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

In [12]:
```
#  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-finetune",
)

# 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.
```

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

`win_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 [13]:
```
import win_system_info_as_script as winsysinfo
winsysinfo.run()
```

```
##########################  System Information  ##########################
System: Windows
Node Name: NOT-FOR-NOW
Release: 10
Version: 10.0.19045
Machine: AMD64
Processor: Intel64 Family 6 Model 165 Stepping 3, GenuineIntel
Processor: Intel(R) Core(TM) i3-10100 CPU @ 3.60GHz
Ip-Address: NOT-FOR-NOW
Mac-Address: NOT-FOR-NOW

############################  Boot Time  ############################
Boot Time (date and time of last boot) was
Boot Time: 2024-5-26T14:29:0

############################  CPU Info  ############################
Physical cores: 4
Total cores: 8
CPU Usage Per Core:
Core 0: 3.1%
Core 1: 1.6%
Core 2: 6.2%
Core 3: 0.0%
Core 4: 3.1%
Core 5: 0.0%
Core 6: 6.2%
Core 7: 0.0%
Total CPU Usage: 5.7%
Max Frequency: 3600.00Mhz
Min Frequency: 0.00Mhz
Current Frequency: 3600.00Mhz

############################  GPU Info  ############################
Information on GPU(s)/Graphics Card(s)
 (if any such information is to be found)

Using  wmi , we get the following  win32_VideoController  names.
   Trigger 6 External Graphics
   Intel(R) UHD Graphics 630
Using  PyTorch  and the  torch.cuda.is_available()  method.
The statement, 'There is CUDA and an appropriate GPU',
  is ...  False
```

```
Using  TensorFlow  with several of its methods.
  Attempting to get GPU Device List
No GPU Devices.
Tensorflow can give us CPU (and/or GPU) info.
The info here might help you know if we're running on a CPU.

Trying to use some nvidia code ( nvidia-smi ) to find information
  That's the end of the nvidia try.

Those are all our chances to find out about any GPU/Graphics Cards.

######################  Memory (RAM) Information  ######################
Total: 31.67GbB
Available: 17.95GbB
Used: 13.72GbB
Percentage: 43.3%
      =============== SWAP Memory ===============
  That nvidia stuff didn't work
  The error information is:
[WinError 2] The system cannot find the file specified
```

```
Total: 4.75GbB
Free: 4.64GbB
Used: 108.27MbB
Percentage: 2.2%

##############################  Disk Info  ##############################
Partitions and Usage:
=== Device: C:\ ===
  Mountpoint: C:\
  File system type: NTFS
  Total Size: 915.94GbB
  Used: 587.01GbB
  Free: 328.93GbB
  Percentage: 64.1%
=== Device: D:\ ===
  Mountpoint: D:\
  File system type: exFAT
  Total Size: 12.73TbB
  Used: 1.99TbB
  Free: 10.75TbB
  Percentage: 15.6%
=== Device: E:\ ===
  Mountpoint: E:\
  File system type: FAT32
  Total Size: 115.31GbB
  Used: 46.08GbB
  Free: 69.23GbB
  Percentage: 40.0%
  Since last boot,
Total read: 158.10GbB
Total write: 204.07GbB
```

ROUGE Metrics

Some references from the Microsoft/Google (who?) 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

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.

In [14]:
```
# import itertools

# def rouge_n(system, reference, n):
    # '''
    # ROUGE-N : N-Grams implementation

    # ref = "https://web.archive.org/web/20240530230949/" + \
          # "https://www.bomberbot.com/machine-learning/" + \
          # "skip-bigrams-in-system/"

    # @param  system      string   The hypothesis
    # @param  reference   string   The truth
    # @param  n           string   The "n" in "n-gram", i.e.
                                 # the number of words in
                                 # each grouping

    # @returns  dict in form {"recall": recall,
                            # "precision": precision,
                            # "f-measure": f_measure}

    # Example:
      # >>> import rouge_n
      # >>>
      # >>> system = "The cat was found under the bed."
      # >>> reference = "The cat was hidden under the bed."
      # >>>
      # >>> print(rouge_n(system, reference, 1)) # ROUGE-1
      # >>> print(rouge_n(system, reference, 2)) # ROUGE-2
      # {‘recall‘: 0.8571428571428571, ‘precision‘: 1.0, ‘f-measure‘: 0.9230769230769231}
      # {‘recall‘: 0.6, ‘precision‘: 0.5, ‘f-measure‘: 0.5454545454545455}
    # '''

    # sys_ngrams = list(itertools.ngrams(system.split(), n))
    # ref_ngrams = list(itertools.ngrams(reference.split(), n))

    # overlaps = set(sys_ngrams) & set(ref_ngrams)
    # recall = len(overlaps) / len(ref_ngrams)
    # precision = len(overlaps) / len(sys_ngrams)

    # if precision + recall == 0:
        # f_measure = 0
```

```
    # else:
        # f_measure = 2  precision  recall / (precision + recall)
    # ##endof:  if/else precision + recall == 0
    # return {"recall": recall, "precision": precision, "f-measure": f_measure}
# ##endof:  rouge_n(system, reference, n)



# def lcs(X, Y):
    # '''
    # Longest common subsequence
    # '''

    # m = len(X)
    # n = len(Y)

    # L = [[None]*(n+1) for i in range(m+1)]

    # for i in range(m+1):
        # for j in range(n+1):
            # if i == 0 or j == 0:
                # L[i][j] = 0
            # elif X[i-1] == Y[j-1]:
                # L[i][j] = L[i-1][j-1]+1
            # else:
                # L[i][j] = max(L[i-1][j], L[i][j-1])
            # ##endof:  if
        # ##endof:  for j
    # ##endof:  for i
    # return L[m][n]
# ##endof:  lcs(X, Y>

# def rouge_l(system, reference):
   # '''
   # ROUGE-L : Longest Common Subsequence implementation

   # ref = "https://web.archive.org/web/20240530230949/" + \
         # "https://www.bomberbot.com/machine-learning/" + \
         # "skip-bigrams-in-system/"

    # @param  system      string   The hypothesis
    # @param  reference   string   The truth
```

```
    # @returns  dict in form {"recall": recall,
                            # "precision": precision,
                            # "f-measure": f_measure}


    # Example:
      # >>> import rouge_l, lcs
      # >>>
      # >>> system = "The quick dog jumps over the lazy fox."
      # >>>reference = "The quick brown fox jumps over the lazy dog."
      # >>>
      # >>> print(rouge_l(system, reference))
      # {‘recall‘: 0.7777777777777778, ‘precision‘: 0.875, ‘f-measure‘: 0.823529411764706}
    # '''

    # sys_len = len(system.split())
    # ref_len = len(reference.split())
    # lcs_len = lcs(system.split(), reference.split())

    # recall = lcs_len / ref_len
    # precision = lcs_len / sys_len

    # if precision + recall == 0:
        # f_measure = 0
    # else:
        # f_measure = 2  precision  recall / (precision + recall)
    # ##endof:  if/else

    # return {"recall": recall, "precision": precision, "f-measure": f_measure}
# ##endof:  rouge_l(system, reference)



# from itertools import combinations

# def skipbigrams(sequence, n):
    # '''
    # Returns the set of skip n-grams

    # @param  sequence
    # @param  n
```

```
    # '''

    # return set(combinations(sequence, n))

# ##endof:  skipbigrams(sequence, n=2)

# def rouge_s(system, reference, n=2):
    # '''
    # ROUGE-S : Skip Bigrams implementation

    # @param
    # @param
    # @param

    # @returns


    # Example
      # >>> import skipbigrams, rouge_s
      # >>>
      # >>> system = "The quick dog jumps over the lazy fox."
      # >>> reference = "The quick brown fox jumps over the lazy dog."
      # >>>
      # >>> print(rouge_s(system, reference))
      # {'recall': 0.35, 'precision': 0.4166666666666667, 'f-measure': 0.38095238095238093}
    # '''
    # sys_skipbigrams = skipbigrams(system.split(), n)
    # ref_skipbigrams = skipbigrams(reference.split(), n)

    # overlaps = sys_skipbigrams & ref_skipbigrams
    # recall = len(overlaps) / len(ref_skipbigrams)
    # precision = len(overlaps) / len(sys_skipbigrams)

    # if precision + recall == 0:
        # f_measure = 0
    # else:
        # f_measure = 2  precision  recall / (precision + recall)
    # ##endof:  if/else

    # return {"recall": recall, "precision": precision, "f-measure": f_measure}
# ##endof:  rouge_s(system, reference, n=2)
```

Try for a baseline

Just one summarization to begin with, randomly picked

In [15]:
```
# # Don't need this again
#!powershell -c (Get-Date -UFormat \"%s_%Y%m%dT%H%M%S%Z00\") -replace '[.][0-9]*_', '_'"
```

```
1717094554_20240530T184234-0600
```

Output was:

```
1717094554_20240530T184234-0600
```

In [17]:
```
#  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.

do_seed_for_repeatable = False

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']}")
```

```
Your max_length is set to 200, but your input_length is only 122. 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=61)
```

```
dialogue:
Harry: and? have you listened to it?
Jacob: listened to what?
Harry: to the song i sent you 3 days ago -.-
Jacob: oh shit, i completely forgot...
Harry: ofc again
Jacob: don't be like this :* i'll do that later tonight
Harry: heh, okay
Harry: i'm really curious what you'll think about it
Jacob: i'll let you know, a bit busy right now, speak to you later!
Harry: okay
---------------
flan-t5-small summary:
Jacob forgot to listen to the song he sent Jacob 3 days ago. Harry will let Jacob know later tonight. Jacob will talk
to Harry later.
```

Now, one summarization with comparison to ground truth

In [18]:

```
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---------------")

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

# humgen is for human-generated

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

ref_test_list.append(grnd_summary)
```

```
pred_test_list.append(res_summary)

print("\n\n---------- ROUGE SCORES ----------")

rouge = load_metric('rouge', trust_remote_code=True)

results = rouge.compute(predictions=pred_test_list,
                        references=ref_test_list,
                        use_aggregator=True)

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

print()
print("ROUGE-1 results")
pprint.pp(results['rouge1'])
print()
print("ROUGE-2 results")
pprint.pp(results['rouge2'])
print()
print("ROUGE-L results")
pprint.pp(results['rougeL'])
print()
print("ROUGE-Lsum results")
pprint.pp(results['rougeLsum'])
```

```
Your max_length is set to 200, but your input_length is only 133. 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=66)
```

```
dialogue:
Hannah: Hey, do you have Betty's number?
Amanda: Lemme check
Hannah: <file_gif>
Amanda: Sorry, can't find it.
Amanda: Ask Larry
Amanda: He called her last time we were at the park together
Hannah: I don't know him well
Hannah: <file_gif>
Amanda: Don't be shy, he's very nice
Hannah: If you say so..
Hannah: I'd rather you texted him
Amanda: Just text him 🙂
Hannah: Urgh.. Alright
Hannah: Bye
Amanda: Bye bye
---------------
human-genratd summary:
Hannah needs Betty's number but Amanda doesn't have it. She needs to contact Larry.
flan-t5-small summary:
Larry called Hannah last time she was at the park together. Hannah doesn't know Larry well. Larry called her last tim
e they were at a park. Hannah will text Larry.


---------- ROUGE SCORES ----------
C:\Users\Anast\AppData\Local\Temp\ipykernel_8400\2790818686.py:26: FutureWarning: load_metric is deprecated and will
be removed in the next major version of datasets. Use 'evaluate.load' instead, from the new library 🤗 Evaluate: http
s://huggingface.co/docs/evaluate
  rouge = load_metric('rouge', trust_remote_code=True)
Downloading builder script:   0%|          | 0.00/2.17k [00:00<?, ?B/s]
```

```
ROUGE-1 results
AggregateScore(low=Score(precision=0.16129032258064516, recall=0.3125, fmeasure=0.2127659574468085), mid=Score(precis
ion=0.16129032258064516, recall=0.3125, fmeasure=0.2127659574468085), high=Score(precision=0.16129032258064516, recal
l=0.3125, fmeasure=0.2127659574468085))

ROUGE-2 results
AggregateScore(low=Score(precision=0.03333333333333333, recall=0.06666666666666667, fmeasure=0.04444444444444444), mi
d=Score(precision=0.03333333333333333, recall=0.06666666666666667, fmeasure=0.04444444444444444), high=Score(precisio
n=0.03333333333333333, recall=0.06666666666666667, fmeasure=0.04444444444444444))

ROUGE-L results
AggregateScore(low=Score(precision=0.12903225806451613, recall=0.25, fmeasure=0.1702127659574468), mid=Score(precisio
n=0.12903225806451613, recall=0.25, fmeasure=0.1702127659574468), high=Score(precision=0.12903225806451613, recall=0.
25, fmeasure=0.1702127659574468))

ROUGE-Lsum results
AggregateScore(low=Score(precision=0.12903225806451613, recall=0.25, fmeasure=0.1702127659574468), mid=Score(precisio
n=0.12903225806451613, recall=0.25, fmeasure=0.1702127659574468), high=Score(precision=0.12903225806451613, recall=0.
25, fmeasure=0.1702127659574468))
```

Verbosity stuff - get rid of the nice advice

In [19]:
```
# # Don't need this again
# !powershell -c (Get-Date -UFormat \"%s_%Y-%m-%dT%H%M%S%Z00\") -replace '[.][0-9]*_', '_'
```

```
1717094688_2024-05-30T184448-0600
```

Output was:

```
1717094688_2024-05-30T184448-0600
```

In [20]:
```
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}")
```

```
The statement, 'logging verbosity is CRITICAL' is False
The statement, 'logging verbosity is    ERROR' is False
The statement, 'logging verbosity is  WARNING' is True
The statement, 'logging verbosity is     INFO' is False
The statement, 'logging verbosity is    DEBUG' is False

The value of logging.get_verbosity() is: 30

TRANSFORMERS_NO_ADIVSORY_WARNINGS: None
```

Actual Baseline

In [21]:
```
# # Don't need this again
# !powershell -c (Get-Date -UFormat \"%s_%Y-%m-%dT%H%M%S%Z00\") -replace '[.][0-9]*_', '_'
```

```
1717094729_2024-05-30T184529-0600
```

Output was:

```
1717094729_2024-05-30T184529-0600
```

In [22]:
```
#  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)

prediction_list = []
reference_list = []

baseline_tic = timeit.default_timer()

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

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

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

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

  reference_list.append(grnd_summary)
  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")
print(f"took {baseline_toc - baseline_tic:0.4f} seconds.")

print("\n\n---------- ROUGE SCORES ----------")

# @todo : load it straight from the python package

rouge = load_metric('rouge', trust_remote_code=True)
  #  Set trust_remote_code=False to see the warning,
  #+ deprecation, and what to change to.

results = rouge.compute(predictions=prediction_list,
                        references=reference_list,
                        use_aggregator=True)

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

print()
print("ROUGE-1 results")
pprint.pp(results['rouge1'])
print()
print("ROUGE-2 results")
pprint.pp(results['rouge2'])
print()
print("ROUGE-L results")
pprint.pp(results['rougeL'])
print()
print("ROUGE-Lsum results")
pprint.pp(results['rougeLsum'])

##  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)
```

```
Getting things ready for scoring
took 1162.5236 seconds.


---------- ROUGE SCORES ----------

ROUGE-1 results
AggregateScore(low=Score(precision=0.3623620420489957, recall=0.5387757354848512, fmeasure=0.4120367260545498), mid=S
core(precision=0.37354505712494124, recall=0.5519205917207157, fmeasure=0.42157981166777414), high=Score(precision=0.
38488859010129967, recall=0.5656367969330082, fmeasure=0.4313347594247768))

ROUGE-2 results
AggregateScore(low=Score(precision=0.15911356172159186, recall=0.2428538196441747, fmeasure=0.18143374370228235), mid
=Score(precision=0.16776558103745187, recall=0.256702553043478, fmeasure=0.1902802753555807), high=Score(precision=0.
176789440182549, recall=0.26994546288441695, fmeasure=0.19927249226979166))

ROUGE-L results
AggregateScore(low=Score(precision=0.2804766356825748, recall=0.4221646517546973, fmeasure=0.31994624442237346), mid=
Score(precision=0.2892456873611609, recall=0.43475844856226864, fmeasure=0.32792714750993834), high=Score(precision=
0.2984044560730355, recall=0.44750618279601273, fmeasure=0.33706771629160553))

ROUGE-Lsum results
AggregateScore(low=Score(precision=0.28022879848567583, recall=0.4220608213922142, fmeasure=0.319655889478565), mid=S
core(precision=0.289591524472137, recall=0.43469242939551495, fmeasure=0.32826109350219757), high=Score(precision=0.2
9852002937109373, recall=0.44804213486471733, fmeasure=0.33687287121399573))
```

In [24]:

```python
do_enter_duration_manually = False
NUM_TO_CATCH_NO_MANUAL_ENTRY = -137.
is_a_manual_entry_skip = False # innocent until proven guilty


if do_enter_duration_manually:
    # !!! remember to type in your number, if needed !!! #
    baseline_duration = NUM_TO_CATCH_NO_MANUAL_ENTRY
    # !!! UNCOMMENT THE NEXT LINE IF YOU WANT TO ENTER MANUALLY !!!
    #baseline_duration =
##endof:  if do_enter_duration_manually

print("Running baseline inference (using the test set)")
if ( ( do_enter_duration_manually ) and \
     ( baseline_duration == -137. ) \
   ):
```

```
    print("took AN UNKNOWN AMOUNT OF TIME.")
    print("You didn't manually enter in your real time,")
    print("as you should have.")
    is_a_manual_entry_skip = True
elif ( ( do_enter_duration_manualy ) and \
       ( baseline_duration != -137. )
     ):
    print("(and using your manually entered time)")
else:
    pass
##endof:  if <check manual entry

if not is_a_manual_entry_skip:
    print(f"took {format_timespan(baseline_duration)}")
##endof:  if not is_a_manual_entry_skip
```

```
Running baseline inference (using the test set)
took 19 minutes and 22.52 seconds
```

Trainer - the Actual Trainer Part

In [25]:
```
# # Don't need this again
# !powershell -c (Get-Date -UFormat \"%s_%Y-%m-%dT%H%M%S%Z00\") -replace '[.][0-9]*_', '_'
```

```
1717096214_2024-05-30T191014-0600
```

Output was:

```
1717096214_2024-05-30T191014-0600
```

In [27]:
```
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.
```

```
##  Ended up not using this.
#                      max_seq_length=675
#          )
```

```
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\Anast\.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: 0 examples [00:00, ? examples/s]
Generating train split: 0 examples [00:00, ? examples/s]
```

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

~~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.

Actually, it appears I didn't run the updated cell, (with `max_seq_length=675`), since the Warning and Advice are still there.

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

In [28]:
```
# # Don't need this again
# !powershell -c (Get-Date -UFormat \"%s_%Y-%m-%dT%H%M%S%Z00\") -replace '[.][0-9]*_', '_'
```

```
1717096271_2024-05-30T191111-0600
```

Output was:

`1717096271_2024-05-30T191111-0600`

At about `1717063394_2024-05-30T100314-0600`, DWB went in and renamed `profile.ps1` to `NOT-USING_-_pro_file_-_now.ps1.bak` That should get rid of our errors from `powershell`

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

In [29]:
```
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.")
```

[1536/1536 3:04:33, Epoch 1/1]

Epoch	Training Loss	Validation Loss
1	0.068500	0.022573

```
C:\Users\Anast\.conda\envs\rwkv-lora-pat\lib\site-packages\huggingface_hub\file_download.py:1132: FutureWarning: `resume_download` is deprecated and will be removed in version 1.0.0. Downloads always resume when possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(
tic: 362634.7966071
toc: 373716.499057
Training took 11081.7024 seconds.
```

In [30]:
```
do_by_hand = False
NUM_TO_CATCH_NO_DO_BY_HAND = -137.
is_a_do_by_hand_skip = False # innocent until proven guilty
```

```
if do_by_hand:
    # !!! remember to type in your number, if needed !!! #
    training_duration = NUM_TO_CATCH_NO_MANUAL_ENTRY
    # !!! UNCOMMENT THE NEXT LINE IF YOU WANT TO ENTER MANUALLY !!!
    #training_duration = 11081.7024
##endof:  if do_by_hand

print("Running training (using the training and eval sets)")
if ( ( do_by_hand ) and \
     ( training_duration == -137. ) \
   ):
    print("took AN UNKNOWN AMOUNT OF TIME.")
    print("You didn't manually enter in your real time,")
    print("as you should have.")
    is_a_do_by_hand_skip = True
elif ( ( do_by_hand ) and \
       ( training_duration != -137. )
     ):
    print("(and using your manually entered time)")
else:
    pass
##endof:  if <check manual entry

if not is_a_do_by_hand_skip:
    print(f"took {format_timespan(training_duration)}")
##endof:  if not is_a_manual_entry_skip
```

```
Training with LoRA (and with the other info as above)
took 3 hours, 4 minutes and 41.7 seconds.
```

In [31]:
```
# # Don't need this again
# !powershell -c (Get-Date -UFormat \"%s_%Y-%m-%dT%H%M%S%Z00\") -replace '[.][0-9]*_', '_'
```

```
1717107458_2024-05-30T221738-0600
```

Output was:

`1717107458_2024-05-30T221738-0600`

@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 ``) to the `trainer`. How can we see information about that?

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-finetune"`

The final TrainingArguments call - with parameter list

```
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-finetune",
)
```

Save the Trainer to Hugging Face and Get Our Updated Model

In [34]:
```
# # Don't need this again
# !powershell -c (Get-Date -UFormat \"%s_%Y-%m-%dT%H%M%S%Z00\") -replace '[.][0-9]*_', '_'
```

```
1717145367_2024-05-31T084927-0600
```

Output was:

`1717145367_2024-05-31T084927-0600`

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 [35]:
```
#  This will come up with a dialog box with text entry,
#+ and I'm now using
#+ ( @thebballdave025 for Hugging Face ) HF stuff.

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

```
VBox(children=(HTML(value='<center> <img\nsrc=https://huggingface.co/front/assets/huggingface_logo-noborder.sv…
```

In [36]:
```
# 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()
```

```
C:\Users\Anast\.conda\envs\rwkv-lora-pat\lib\site-packages\huggingface_hub\file_download.py:1132: FutureWarning: `res
ume_download` is deprecated and will be removed in version 1.0.0. Downloads always resume when possible. If you want
to force a new download, use `force_download=True`.
  warnings.warn(
adapter_model.safetensors:   0%|          | 0.00/11.0M [00:00<?, ?B/s]
events.out.tfevents.1717084743.DESKTOP-O7KM5A5.8272.0:   0%|          | 0.00/12.3k [00:00<?, ?B/s]
```

```
training_args.bin:   0%|          | 0.00/5.11k [00:00<?, ?B/s]
events.out.tfevents.1717117975.DESKTOP-O7KM5A5.8400.0:   0%|          | 0.00/7.14k [00:00<?, ?B/s]
Upload 4 LFS files:   0%|          | 0/4 [00:00<?, ?it/s]
```

Out[36]:
```
CommitInfo(commit_url='https://huggingface.co/thebballdave025/dwb-flan-t5-small-lora-finetune/commit/c87d34b398f3801
ceb1e18c819a7c8fc894989c7', commit_message='End of training', commit_description='', oid='c87d34b398f3801ceb1e18c819
a7c8fc894989c7', pr_url=None, pr_revision=None, pr_num=None)
```

Part of the output included 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 [ ]:
```
## Not run yet for this save.

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

In [ ]:
```

```

In [ ]:
```

```

Evaluation on the Test Set and Comparison to Baseline

Verbosity stuff - get rid of the nice advice

In [ ]:
```
!powershell -c (Get-Date -UFormat \"%s_%Y-%m-%dT%H%M%S%Z00\") -replace '[.][0-9]*_', '_'
```

Output was:

```
timestamp
```

In [ ]:
```
# bballdave025@MYMACHINE /cygdrive/c/Users/bballdave025/.conda/envs/rwkv-lora-pat/Lib/site-packages/peft/utils
# $ date +'%s_%Y-%m-%dT%H%M%S%z'
# 1717049876_2024-05-30T001756-0600

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 [ ]:
```
!powershell -c (Get-Date -UFormat \"%s_%Y-%m-%dT%H%M%S%Z00\") -replace '[.][0-9]*_', '_'
```

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'.

In [ ]:
```
#  I'm going to use 'tat' for testing-after-training

logging.set_verbosity_error()

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

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]

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

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

  #print(f"human-genratd summary:\n{grnd_tat_summary}")
  #print(f"flan-t5-small summary:\n{res_tat_summary}")
```

```
  reference_tat_list.append(grnd_tat_summary)
  prediction_tat_list.append(res_tat_summary)
##endof:  for sample_num in range(len(dataset['test']))

tat_toc = timeit.default_timer()

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

print("\n\n---------- ROUGE SCORES ----------")

rouge = load_metric('rouge', trust_remote_code=True)
  #  Set trust_remote_code=False to see the warning,
  #+ deprecation, and what to change to.

results_tat = rouge.compute(
                  predictions=prediction_tat_list,
                  references=reference_tat_list,
                  use_aggregator=True
)

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

print()
print("ROUGE-1 results")
pprint.pp(results_tat['rouge1'])
print()
print("ROUGE-2 results")
pprint.pp(results_tat['rouge2'])
print()
print("ROUGE-L results")
pprint.pp(results_tat['rougeL'])
print()
print("ROUGE-Lsum results")
pprint.pp(results_tat['rougeLsum'])

logging.set_verbosity(init_log_verbosity)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

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

In [ ]: