Deep Learning Project

By Victoria Lassner DSML 4220

Goal: Fine tune a model for abstractive Summarization.

Potential models: LLaMa 2 and T5 or GPT

Websites: https://huggingface.co/docs/transformers/tasks/summarization

https://wandb.ai/mostafaibrahim17/ml-articles/reports/Fine-Tuning-LLaMa-2-for-Text-Summarization--Vmlldzo2NjA1OTAy

https://wandb.ai/mostafaibrahim17/ml-articles/reports/Crafting-Superior-Summaries-The-ChatGPT-Fine-Tuning-Guide--Vmlldzo1Njc5NDI1

Definitions:

Abstractive summarization = oncise summary of a text by understanding its meaning and creating new sentences, rather than simply extracting phrases from the original text

Extractive Summarization = extracts existing key senetences from passages and creates a new summary.

Potential Datasets: Wikihow: https://paperswithcode.com/dataset/wikihow CNN/DailyMail: https://paperswithcode.com/dataset/cnn-daily-mail-1 BillSum

```
# disables weights and biases
import os
os.environ["WANDB_DISABLED"] = "true"
```

```
# downloads packages for model, dataset and tokenzier
# --Quiet limits output of messages
!pip install transformers datasets sentencepiece --quiet
```

```
from datasets import load_dataset
from transformers import T5ForConditionalGeneration, TrainingArguments, Trainer, T5Tokenize
import torch
from torch.utils.data import DataLoader
import torch
```

```
# Load CNN/Daily Mail Dataset from dataset package
# will most likely need to downscale again
dataset = load_dataset("cnn_dailymail", "3.0.0")
train_data = dataset["train"].select(range(5000))
train_data = dataset["train"].select(range(5000))
val_data = dataset["validation"].select(range(500))
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarnir
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public mc
  warnings.warn(
```

```
# preprocess data for model
tokenizer = T5Tokenizer.from_pretrained("t5-small")

# limit length of input articles and output summary
max_input_length = 512
max_target_length = 150

def preprocess(examples):
    inputs = ["summarize: " + doc for doc in examples["article"]]
    targets = examples["highlights"]

    model_inputs = tokenizer(
        inputs,
        max_length=max_input_length,
        truncation=True,
        padding="max_length"
    )

    with tokenizer.as_target_tokenizer():
        labels = tokenizer(
            targets,
            max_length=max_target_length,
            truncation=True,
            padding="max_length"
        )

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs


train_dataset = train_data.map(preprocess, batched=True, remove_columns=["article", "high
val_dataset = val_data.map(preprocess, batched=True, remove_columns=["article", "highligh
```

```
tokenizer_config.json: 100%                      2.32k/2.32k [00:00<00:00, 32.4kB/
                                                s]
spiece.model: 100%                               792k/792k [00:00<00:00, 1.49MB/
                                                s]
tokenizer.json: 100%                             1.39M/1.39M [00:00<00:00, 1.96MB/
```

```
                                                              s]
    You are using the default legacy behaviour of the <class 'transformers.models.t5.toke
    Map: 100%                                        5000/5000 [00:29<00:00, 180.89 examples/
                                                          s]
```

```
# Load model T5-small
# T5-base too large without GPU
model = T5ForConditionalGeneration.from_pretrained("t5-small")

# training arguments
# Will need to scale back to limit training time
# currently just over 8 hours to train
# only for huggingface trainer package
training_args = TrainingArguments(
    output_dir="./t5-cnn-checkpoints",
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    gradient_accumulation_steps=4,
    eval_steps=500,
    save_steps=1000,
    num_train_epochs=2,
    logging_dir='./logs',
    logging_steps=200,
    save_total_limit=2,
    fp16=False,
)

# for manual training
# batch_size = 2
# learning_rate = 5e-5
# epochs = 2
# gradient_accumulation_steps = 4
# eval_steps = 500
# save_steps = 1000
# logging_steps = 200
# save_total_limit = 2
# fp16 = False
```

```
    config.json: 100%                                     1.21k/1.21k [00:00<00:00, 85.3kB/s]
    Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Fall
    WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the '
    model.safetensors: 100%                                 242M/242M [00:01<00:00, 179MB/
                                                              s]
    generation_config.json: 100%                              147/147 [00:00<00:00, 5.84kB/
                                                                s]
```

```
# adds padding so shorter sequences match the longest one
```

```
from transformers import DataCollatorForSeq2Seq

data_collator = DataCollatorForSeq2Seq(tokenizer=tokenizer, model=model)


# train model using hugging face's trainer class
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    data_collator=data_collator,
)

trainer.train()

# Porbably will switch to traditional training method

# train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
# optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate)

# model.train()
# for epoch in range(epochs):
#     for step, batch in enumerate(train_dataloader):
#         optimizer.zero_grad()
#         outputs = model(**batch)
#         loss = outputs.loss
#         loss.backward()

#         # Gradient accumulation
#         if (step + 1) % gradient_accumulation_steps == 0:
#             optimizer.step()

#         # Logging
#         if (step + 1) % logging_steps == 0:
#             print(f"Step {step + 1}: Loss = {loss.item()}")

#         # Evaluation
#         if (step + 1) % eval_steps == 0:
#             model.eval()
#             # Run validation logic here
#             model.train()

#     # Save checkpoint manually
#     if (epoch + 1) % save_steps == 0:
#         torch.save(model.state_dict(), f"./t5-cnn-checkpoints/epoch-{epoch+1}.pt")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-7a5f6a095494> in <cell line: 0>()
      1 # train model
```

```
----> 2 trainer = Trainer(
      3     model=model,
      4     args=training_args,
      5     train_dataset=train_dataset,

NameError: name 'Trainer' is not defined
```

Next steps: Explain error

```
#saves current state of model and tokenzier
model.save_pretrained("/content/t5_cnn_model")
tokenizer.save_pretrained("/content/t5_cnn_model")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-234c03f2de0c> in <cell line: 0>()
      1 #save model
----> 2 model.save_pretrained("/content/t5_cnn_model")
      3 tokenizer.save_pretrained("/content/t5_cnn_model")

NameError: name 'model' is not defined
```

Next steps: Explain error

```
# download model from colab for web app to use
from google.colab import files
!zip -r t5_cnn_model.zip /content/t5_cnn_model
files.download("t5_cnn_model.zip")
```

```
  adding: content/t5_cnn_model/ (stored 0%)
  adding: content/t5_cnn_model/generation_config.json (deflated 29%)
  adding: content/t5_cnn_model/special_tokens_map.json (deflated 85%)
  adding: content/t5_cnn_model/model.safetensors (deflated 9%)
  adding: content/t5_cnn_model/spiece.model (deflated 48%)
  adding: content/t5_cnn_model/added_tokens.json (deflated 83%)
  adding: content/t5_cnn_model/tokenizer_config.json (deflated 94%)
  adding: content/t5_cnn_model/config.json (deflated 63%)
```

```
# test to see how well model is working
def summarize(text):
    input_text = "summarize: " + text
    inputs = tokenizer.encode(input_text, return_tensors="pt", max_length=512, truncation
    summary_ids = model.generate(inputs, max_length=150, min_length=30, length_penalty=2.
    return tokenizer.decode(summary_ids[0], skip_special_tokens=True)

test_article = val_data[0]["article"]
```

```
    print("📰 Original Article:\n", test_article[:500], "...\n")
    print("🧠 Summary:\n", summarize(test_article))
```