Homework 3

- Some questions require writing Python code and computing results, and the rest of them have written answers. For coding problems, you will have to fill out all code blocks that say **`YOUR CODE HERE`**.
- You will need to use GPU, which can be added through `Edit > Notebook Settings > Hardware accelerator > (GPU)`
- For text-based answers, you should replace the text that says **`Write your answer here...`** with your actual answer.
- This assignment is designed such that each cell takes a few minutes (if that) to run. If it is taking longer than that, you might have made a mistake in your code.

How to submit this problem set:

- Write all the answers in this Colab notebook. Once you are finished, generate a PDF via (File -> Print -> Save as PDF) and upload it to Gradescope.
- **Important:** check your PDF before you submit to Gradescope to make sure it exported correctly. If Colab gets confused about your syntax, it will sometimes terminate the PDF creation routine early.
- When creating your final version of the PDF to hand in, please do a fresh restart and execute every cell in order. One handy way to do this is by clicking `Runtime -> Run All` in the notebook menu.

https://colab.research.google.com/drive/1tTcRE6fvN6mQsekDMYM-v9hL4nP8QLls?usp=sharing

Part 0: Setup

Installing Hugging Face's Transformers and Additional Libraries

We will use Hugging Face's Transformers (https://github.com/huggingface/transformers), an open-source library that provides general-purpose architectures for natural language understanding and generation with a collection of various pretrained models made by the NLP community. This library will allow us to easily use pretrained models like `BERT` and perform experiments on top of them. We can use these models to solve downstream target tasks, such as text classification, question answering, sequence labeling, and text generation.

Run the following cell to install Hugging Face's Transformers library and some other useful tools. This cell will also download data used later in the assignment.

```
! nvidia-smi
!pip install -q transformers==4.17.0 datasets==2.0.0 rich[jupyter]
!pip install -q googletrans==3.1.0a0
!pip install -q -U PyDrive
!apt install jq

import torch
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

import os
import zipfile
import collections

import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import cm
from rich import print as rich_print

from transformers import GPT2LMHeadModel, GPT2Tokenizer
```

```
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
     |████████████████████████████████| 3.8 MB 35.0 MB/s
     |████████████████████████████████| 325 kB 77.6 MB/s
     |████████████████████████████████| 237 kB 76.8 MB/s
     |████████████████████████████████| 182 kB 75.1 MB/s

     |████████████████████████████████| 7.6 MB 68.9 MB/s
     |████████████████████████████████| 880 kB 62.6 MB/s
     |████████████████████████████████| 212 kB 80.4 MB/s
     |████████████████████████████████| 132 kB 68.5 MB/s
     |████████████████████████████████| 127 kB 66.2 MB/s
     |████████████████████████████████| 51 kB 8.4 MB/s
     |████████████████████████████████| 1.6 MB 57.9 MB/s
  Building wheel for sacremoses (setup.py) ... done
     |████████████████████████████████| 55 kB 3.6 MB/s
     |████████████████████████████████| 1.5 MB 66.4 MB/s
     |████████████████████████████████| 42 kB 1.3 MB/s
     |████████████████████████████████| 65 kB 4.5 MB/s
     |████████████████████████████████| 53 kB 271 kB/s
```

```
     |████████████████████████████████| 53 kB 371 kB/s
  Building wheel for googletrans (setup.py) ... done
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  libnvidia-common-460
Use 'apt autoremove' to remove it.
The following additional packages will be installed:
  libjq1 libonig4
The following NEW packages will be installed:
  jq libjq1 libonig4
0 upgraded, 3 newly installed, 0 to remove and 7 not upgraded.
Need to get 276 kB of archives.
After this operation, 930 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/universe amd64 libonig4 amd64 6.7.0-1 [
Get:2 http://archive.ubuntu.com/ubuntu bionic/universe amd64 libjq1 amd64 1.5+dfsg-2
Get:3 http://archive.ubuntu.com/ubuntu bionic/universe amd64 jq amd64 1.5+dfsg-2 [45.
Fetched 276 kB in 2s (166 kB/s)
Selecting previously unselected package libonig4:amd64.
(Reading database ... 124015 files and directories currently installed.)
Preparing to unpack .../libonig4_6.7.0-1_amd64.deb ...
Unpacking libonig4:amd64 (6.7.0-1) ...
Selecting previously unselected package libjq1:amd64.
Preparing to unpack .../libjq1_1.5+dfsg-2_amd64.deb ...
Unpacking libjq1:amd64 (1.5+dfsg-2) ...
Selecting previously unselected package jq.
Preparing to unpack .../jq_1.5+dfsg-2_amd64.deb ...
Unpacking jq (1.5+dfsg-2) ...
Setting up libonig4:amd64 (6.7.0-1) ...
Setting up libjq1:amd64 (1.5+dfsg-2) ...
Setting up jq (1.5+dfsg-2) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

```
def tsv_to_csv(in_file, out_file):
    data = pd.read_csv(in_file, sep='\t')
    data.to_csv(out_file, sep=',', index=False)
```

```
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
print('success!')

data_file = drive.CreateFile({'id': '1zeo8FcaNUnhN660mGMNEAPvxOE4DPOnE'})
data_file.GetContentFile('hw1.zip')

with zipfile.ZipFile('hw1.zip', 'r') as zip_file:
    zip_file.extractall('./')
os.remove('hw1.zip')
os.chdir('hw1')
print("Data and supporting code downloaded!")
```

```

tsv_to_csv('data/tinySST/dev.tsv', 'data/tinySST/dev.csv')
tsv_to_csv('data/tinySST/train.tsv', 'data/tinySST/train.csv')
print('finished preprocessing data')

pretrained_models_dir = './pretrained_models_dir'
if not os.path.isdir(pretrained_models_dir):
  os.mkdir(pretrained_models_dir)
print('model directory created')

!pip install -q -r requirements.txt
print('extra packages installed!')

!wget -nv https://nlp.stanford.edu/data/coqa/coqa-train-v1.0.json
!wget -nv https://nlp.stanford.edu/data/coqa/coqa-dev-v1.0.json
!cat coqa-dev-v1.0.json | jq '{data: [.data[] | del(.answers[].bad_turn) | del(.question
!cat coqa-train-v1.0.json | jq '{data: [.data[] | del(.answers[].bad_turn) | del(.questi
print('Download coqa dataset!')
```

```
success!
Data and supporting code downloaded!
finished preprocessing data
model directory created
     |████████████████████████████████| 43 kB 1.6 MB/s
     |████████████████████████████████| 981 kB 31.5 MB/s
  Building wheel for seqeval (setup.py) ... done
  Building wheel for langdetect (setup.py) ... done
extra packages installed!
2022-12-03 15:08:01 URL:https://downloads.cs.stanford.edu/nlp/data/coqa/coqa-train-v1.0
2022-12-03 15:08:05 URL:https://downloads.cs.stanford.edu/nlp/data/coqa/coqa-dev-v1.0.j
Download coqa dataset!
```

```
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2", pad_token_id=tokenizer.eos_token_id)
```

Downloading: 100% 0.99M/0.99M [00:01<00:00, 953kB/s]

Downloading: 100% 446k/446k [00:01<00:00, 450kB/s]

Downloading: 100% 665/665 [00:00<00:00, 18.9kB/s]

Downloading: 100% 523M/523M [00:13<00:00, 34.9MB/s]

Part 1. Beam Search

We're going to explore decoding from a pretrained GPT-2 model using beam search. Run the below cell to set up some beam search utilities.

```
def init_beam_search(model, input_ids, num_beams):
```

```
    assert len(input_ids.shape) == 2
    beam_scores = torch.zeros(num_beams, dtype=torch.float32, device=model.device)
    beam_scores[1:] = -1e9 # Break ties in first round.
    new_input_ids = input_ids.repeat_interleave(num_beams, dim=0).to(model.device)
    return new_input_ids, beam_scores

def run_beam_search_(model, tokenizer, input_text, num_beams=5, num_decode_steps=10, sco

    input_ids = tokenizer.encode(input_text, return_tensors='pt')
    input_ids, beam_scores = init_beam_search(model, input_ids, num_beams)
    token_scores = beam_scores.clone().view(num_beams, 1)

    model_kwargs = {}
    for i in range(num_decode_steps):
        model_inputs = model.prepare_inputs_for_generation(input_ids, **model_kwargs)  #
        outputs = model(**model_inputs, return_dict=True)
        next_token_logits = outputs.logits[:, -1, :]
        vocab_size = next_token_logits.shape[-1]
        this_token_scores = torch.log_softmax(next_token_logits, -1)

        # Process token scores.
        processed_token_scores = this_token_scores
        for processor in score_processors:
            processed_token_scores = processor(input_ids, processed_token_scores)

        # Update beam scores.
        next_token_scores = processed_token_scores + beam_scores.unsqueeze(-1)
        next_token_scores = next_token_scores.view(num_beams * vocab_size)

        # Find top-scoring beams.
        next_token_scores, next_tokens = torch.topk(
            next_token_scores, num_beams, dim=0, largest=True, sorted=True
        )

        # Transform tokens since we reshaped earlier.
        next_indices = torch.div(next_tokens, vocab_size, rounding_mode="floor") # This
        next_tokens = next_tokens % vocab_size

        # Update tokens and beam scores.
        input_ids = torch.cat([input_ids[next_indices, :], next_tokens.unsqueeze(-1)], d
        beam_scores = next_token_scores

        # UNCOMMENT: To use original scores instead.
        # token_scores = torch.cat([token_scores[next_indices, :], this_token_scores[nex
        token_scores = torch.cat([token_scores[next_indices, :], processed_token_scores[

        # Update hidden state.
        model_kwargs = model._update_model_kwargs_for_generation(outputs, model_kwargs,
        model_kwargs["past"] = model._reorder_cache(model_kwargs["past"], next_indices)

    def transfer(x):
```

```
      return x.cpu() if to_cpu else x

    return {
        "output_ids": transfer(input_ids),
        "beam_scores": transfer(beam_scores),
        "token_scores": transfer(token_scores)
    }


def run_beam_search(*args, **kwargs):
    with torch.inference_mode():
        return run_beam_search_(*args, **kwargs)
```

```
# Add support for colored printing and plotting.
RICH_x = np.linspace(0.0, 1.0, 50)
RICH_rgb = (cm.get_cmap(plt.get_cmap('RdYlBu'))(RICH_x)[:, :3] * 255).astype(np.int32)[r

def print_with_probs(words, probs, prefix=None):
  def fmt(x, p, is_first=False):
    ix = int(p * RICH_rgb.shape[0])
    r, g, b = RICH_rgb[ix]
    if is_first:
      return f'[bold rgb(0,0,0) on rgb({r},{g},{b})]{x}'
    else:
      return f'[bold rgb(0,0,0) on rgb({r},{g},{b})] {x}'
  output = []
  if prefix is not None:
    output.append(prefix)
  for i, (x, p) in enumerate(zip(words, probs)):
    output.append(fmt(x, p, is_first=i == 0))
  rich_print(''.join(output))
```

Question 1.1 (5 points)

Run the cell below. It produces a sequence of tokens using beam search and the provided prefix.

```
num_beams = 5
num_decode_steps = 10
input_text = 'The brown fox jumps'

beam_output = run_beam_search(model, tokenizer, input_text, num_beams=num_beams, num_dec
for i, tokens in enumerate(beam_output['output_ids']):
    score = beam_output['beam_scores'][i]
    print(i, round(score.item() / tokens.shape[-1], 3), tokenizer.decode(tokens, skip_sp
```

```
0 -1.106 The brown fox jumps out of the fox's mouth, and the fox
1 -1.168 The brown fox jumps out of the fox's cage, and the fox
2 -1.182 The brown fox jumps out of the fox's mouth and starts to run
```

```
3 -1.192 The brown fox jumps out of the fox's mouth and begins to lick
```

To get you more acquainted with the code, let's do a simple exercise first. Write your own code in the cell below to generate 3 tokens with a beam size of 4, and then print out the **third most probable** output sequence found during the search. Use the same prefix as above.

```
input_text = 'The brown fox jumps'

# WRITE YOUR CODE HERE!
beam_output = run_beam_search(model, tokenizer, input_text, num_beams=4, num_decode_step
beam_output
i = 2  # 0 1 2 3
score = beam_output['beam_scores'][i]
tokens = beam_output['output_ids'][i]
print(i, round(score.item() / tokens.shape[-1], 3), tokenizer.decode(tokens, skip_specia
```

```
2 -0.627 The brown fox jumps up and down
```

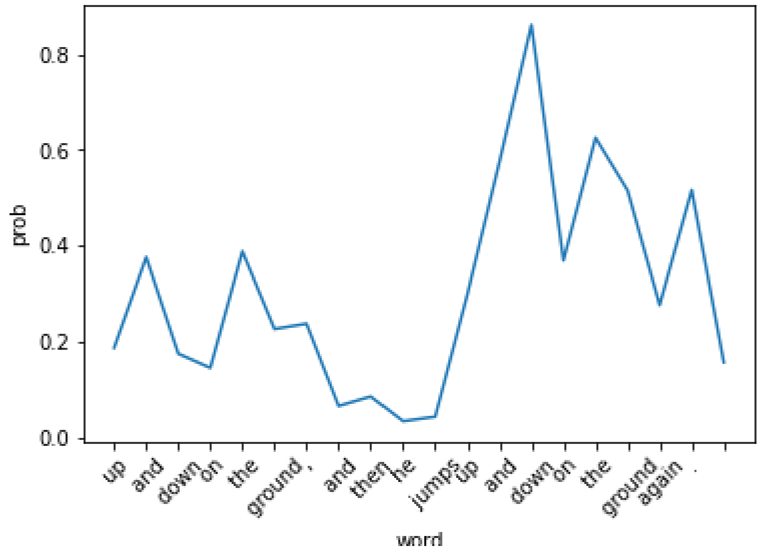
Question 1.2 (10 points)

Run the cells below to visualize the probabilities the model assigns for each generated word when using beam search with beam size 1 (i.e., greedy decoding).

```
input_text = 'The brown fox jumps'
beam_output = run_beam_search(model, tokenizer, input_text, num_beams=1, num_decode_step
probs = beam_output['token_scores'][0, 1:].exp()
output_subwords = [tokenizer.decode(tok, skip_special_tokens=True) for tok in beam_outpu
```

```
print('Visualizeation with plot:')

fig, ax = plt.subplots()
plt.plot(range(len(probs)), probs)
ax.set_xticks(range(len(probs)))
ax.set_xticklabels(output_subwords[-len(probs):], rotation = 45)
plt.xlabel('word')
plt.ylabel('prob')
plt.show()

print('Visualization with colored text (red for lower probability, and blue for higher):

print_with_probs(output_subwords[-len(probs):], probs, ' '.join(output_subwords[:-len(pr
```

```
Visualizeation with plot:
```



Why does the model assign a higher probability to tokens generated later than to tokens generated earlier?

WRITE YOUR ANSWER HERE IN A FEW SENTENCES

> At first, the model only has 4 words "The brown fox jumps". Naturally the rest of the sentence can go in any way possible; therefore, the model is not sure about one specific next token (larger entropy over next tokens). However, as the sentence gets longer and we have more context like "The brown fox jumps up and down on the ground ," the next tokens become more clear (smaller entropy). Also, the model is experiencing repeatition which can be due to using greedy algorithm and having no tolerance for randomness. So the model goes with the exact sentence previously seen. (Also GPT2 we use hear has only 12 layers https://huggingface.co/gpt2/blob/main/config.json So we cannot expect much from it. Larger models would generate more meaningful next tokens.)

Question 1.3 (10 points)

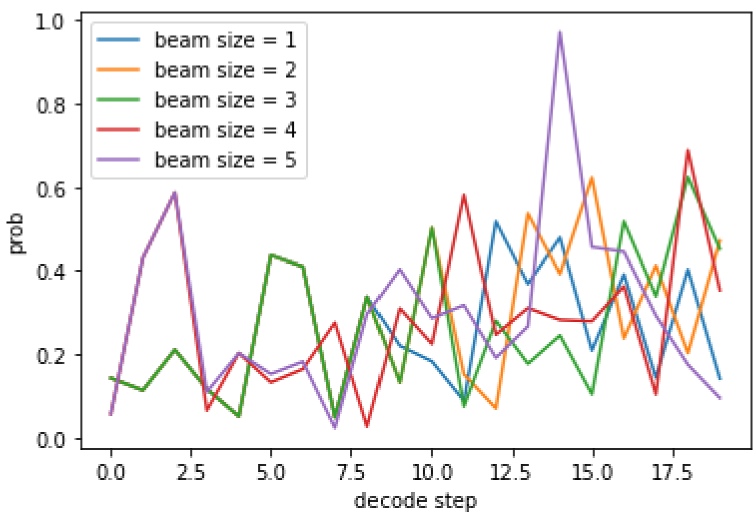
Run the cells below to visualize the word probabilities when using different beam sizes.

```
input_text = 'Once upon a time, in a barn near a farm house,'
num_decode_steps = 20
model.cuda()

beam_size_list = [1, 2, 3, 4, 5]
output_list = []
probs_list = []
for bm in beam_size_list:
```

```
  beam_output = run_beam_search(model, tokenizer, input_text, num_beams=bm, num_decode_s
  output_list.append(beam_output)
  probs = beam_output['token_scores'][0, 1:].exp()
  probs_list.append((bm, probs))
```

```
print('Visualization with plot:')
fig, ax = plt.subplots()
for bm, probs in probs_list:
  plt.plot(range(len(probs)), probs, label=f'beam size = {bm}')
plt.xlabel('decode step')
plt.ylabel('prob')
plt.legend(loc='best')
plt.show()

print('Model predictions:')
for bm, beam_output in zip(beam_size_list, output_list):
  tokens = beam_output['output_ids'][0]
  print(bm, beam_output['beam_scores'][0].item() / tokens.shape[-1], tokenizer.decode(tc
```

```
Visualization with plot:
```

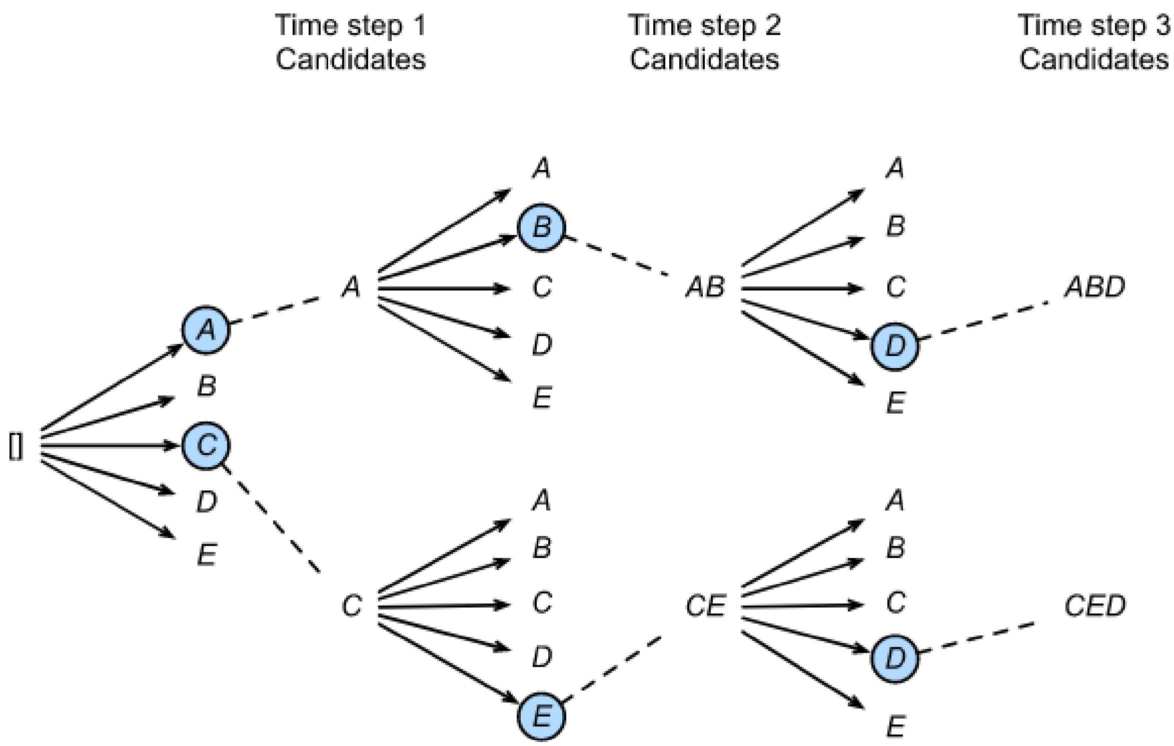


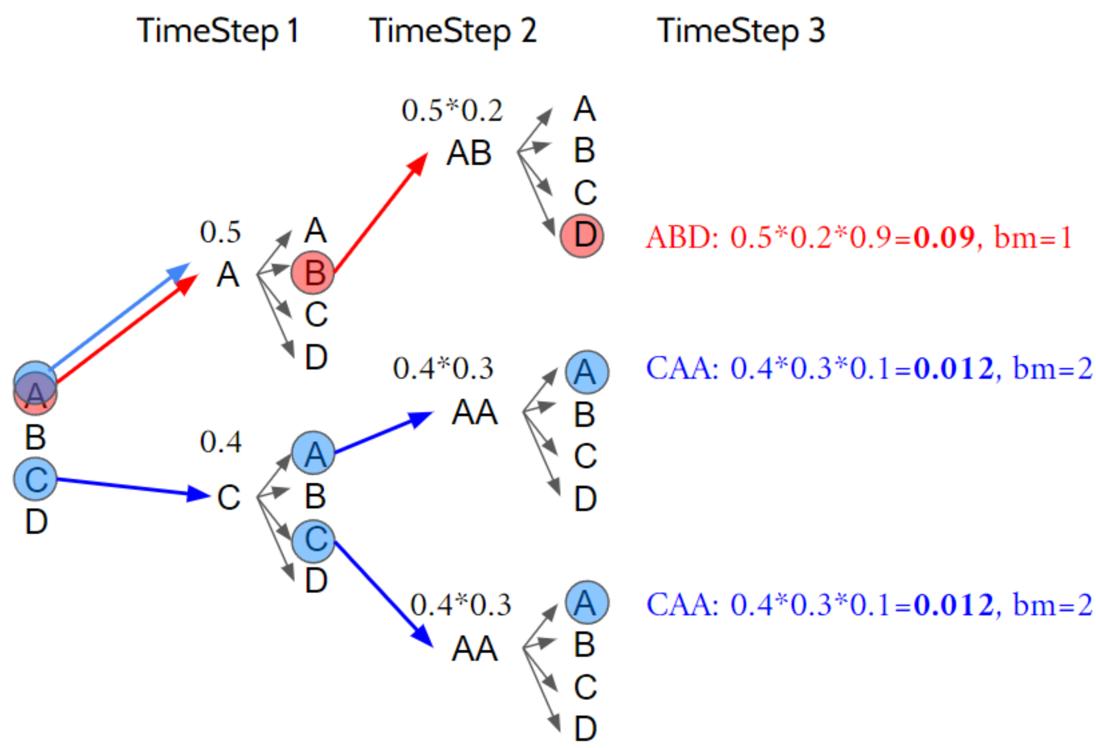

```
Model predictions:
1 -0.9706196640477036 Once upon a time, in a barn near a farm house, a young boy was pl
2 -0.9286184021920869 Once upon a time, in a barn near a farm house, a young boy was pl
3 -0.9597580071651575 Once upon a time, in a barn near a farm house, a young boy was pl
4 -0.920514540238814 Once upon a time, in a barn near a farm house, there was a young g
5 -0.9058765064586293 Once upon a time, in a barn near a farm house, there was a man wh
```

The `Model Predictions` section above includes the average cumulative log probability of each sequence. Does higher beam size always guarantee a higher probability final sequence? Why or why not?

WRITE YOUR ANSWER HERE IN A FEW SENTENCES

> Beam search is an optimization of Best-First Search only going through specific beam size of all branches in each time step. As seen above, higher beam size does not always guarantee a higher probability final sequence. This is because beam search is not optimal, meaning there is no guarantee that it will find the best solution. Also, with two different beam widths, the algorithm might select specific branches and discard others in each of them. There is no guarantee that larger beam width contains the branches analyzed in a smaller one. So, it might discard a branch of bw=1 early; however, in the end that branch would become the highest general probability.





Question 1.4 (15 points)

Beam search often results in repetition in the predicted tokens. In the following cell we pass a score processor called `WordBlock` to `run_beam_search`. At each time step, it reduces the probability for any previously seen word so that it is not generated again.

Run the cells to see how the output of beam search changes with and without using `WordBlock`.

```
class WordBlock:
    def __call__(self, input_ids, scores):
        for batch_idx in range(input_ids.shape[0]):
            for x in input_ids[batch_idx].tolist():
                scores[batch_idx, x] = -1e9
        return scores
```

```
input_text = 'Once upon a time, in a barn near a farm house,'
num_beams = 1

print('Beam Search')
beam_output = run_beam_search(model, tokenizer, input_text, num_beams=num_beams, num_dec
print(tokenizer.decode(beam_output['output_ids'][0], skip_special_tokens=True))

print('Beam Search w/ Word Block')
beam_output = run_beam_search(model, tokenizer, input_text, num_beams=num_beams, num_dec
```

```
print(tokenizer.decode(beam_output['output_ids'][0], skip_special_tokens=True))

```

```
Beam Search
Once upon a time, in a barn near a farm house, a young boy was playing with a stick. He
Beam Search w/ Word Block
Once upon a time, in a barn near a farm house, the young girl was playing with her fath
```

Is `WordBlock` a practical way to prevent repetition in beam search? What (if anything) could go wrong when using `WordBlock`?

WRITE YOUR ANSWER HERE IN A FEW SENTENCES

> The biggest issue with wordblock is that after a while, there will be no words left to generate! This might be fixed by applying wordblock in a limited window before the next token, not all the previous tokens (Although this way the model can still repeat itself in longer sentences). Also, most frequent words such as "the", "a", and "and" will not be allowed to be used more than once which is restricting for the model to even generate new sentences. These issues can be partially fixed by the beamblock and blocking n-grams instead of unigrams but not completely.

Question 1.5 (20 points)

Use the previous `WordBlock` example to write a new score processor called `BeamBlock`. Instead of uni-grams, your implementation should prevent tri-grams from appearing more than once in the sequence.

Note: This technique is called "beam blocking" and is described [here](section 2.5). Also, for this assignment you do not need to re-normalize your output distribution after masking values, although typically re-normalization is done.

Write your code in the indicated section in the below cell.

```
class BeamBlock:
    def __call__(self, input_ids, scores):
        for batch_idx in range(input_ids.shape[0]):
            # WRITE YOUR CODE HERE!
            trigram = []
            for x in input_ids[batch_idx].tolist():
                trigram.append(x)
                if len(trigram) > 3:
                    trigram.pop(0)
                if len(trigram) == 3 and torch.equal(torch.tensor(trigram[:-1]), input_i
                    print("Preventing: ", tokenizer.decode(input_ids[batch_idx, -2:]), '
```

```
                        scores[batch_idx, x] = -1e9
            return scores
```

```
input_text = 'Once upon a time, in a barn near a farm house,'
num_beams = 1

print('Beam Search')
beam_output = run_beam_search(model, tokenizer, input_text, num_beams=num_beams, num_dec
print(tokenizer.decode(beam_output['output_ids'][0], skip_special_tokens=True))

print('Beam Search w/ Beam Block')
beam_output = run_beam_search(model, tokenizer, input_text, num_beams=num_beams, num_dec
print(tokenizer.decode(beam_output['output_ids'][0], skip_special_tokens=True))

```

```
Beam Search
Once upon a time, in a barn near a farm house, a young boy was playing with a stick. He
Beam Search w/ Beam Block
Preventing:   was playing ->  with
Preventing:   boy was ->  playing
Preventing:   boy was ->  playing
Preventing:   boy was ->  trying
Preventing:   the stick -> ,
Preventing:  , and ->  the
Once upon a time, in a barn near a farm house, a young boy was playing with a stick. He
```

Part 2. Language Model Fine-tuning

Now, we'll switch over to *fine-tuning* a pretrained language model. For this task, we'll use data from the [Conversational Question Answering dataset (CoQA)](). The CoQA dataset includes tuples of (story text, question, answers), and we'll only be using the story text which come from various sources including children's stories, news passages, and wikipedia.

Run the below cell to set some stuff up.

```
import logging
import math
import os
import sys
from dataclasses import dataclass, field
from itertools import chain
from typing import Optional

import datasets
from datasets import load_dataset, load_metric

import transformers
```

```
from transformers import (
    CONFIG_MAPPING,
    MODEL_FOR_CAUSAL_LM_MAPPING,
    AutoConfig,
    AutoModelForCausalLM,
    AutoTokenizer,
    HfArgumentParser,
    Trainer,
    TrainingArguments,
    default_data_collator,
    is_torch_tpu_available,
    set_seed,
)
from transformers.testing_utils import CaptureLogger
from transformers.trainer_utils import get_last_checkpoint
from transformers.utils import check_min_version
from transformers.utils.versions import require_version

import copy
import torch

from tqdm import tqdm
import collections
import numpy as np
```

```
MODEL_CONFIG_CLASSES = list(MODEL_FOR_CAUSAL_LM_MAPPING.keys())
MODEL_TYPES = tuple(conf.model_type for conf in MODEL_CONFIG_CLASSES)

@dataclass
class ModelArguments:
    """
    Arguments pertaining to which model/config/tokenizer we are going to fine-tune, or t
    """
    model_name_or_path: Optional[str] = field(
        default=None,
        metadata={
            "help": "The model checkpoint for weights initialization."
            "Don't set if you want to train a model from scratch."
        },
    )
    model_type: Optional[str] = field(
        default=None,
        metadata={"help": "If training from scratch, pass a model type from the list: "
    )
    config_overrides: Optional[str] = field(
        default=None,
        metadata={
            "help": "Override some existing default config settings when a model is trai
            "n_embd=10,resid_pdrop=0.2,scale_attn_weights=false,summary_type=cls_index"
        },
    )
```

```
    config_name: Optional[str] = field(
        default=None, metadata={"help": "Pretrained config name or path if not the same
    )
    tokenizer_name: Optional[str] = field(
        default=None, metadata={"help": "Pretrained tokenizer name or path if not the sa
    )
    cache_dir: Optional[str] = field(
        default=None,
        metadata={"help": "Where do you want to store the pretrained models downloaded f
    )
    use_fast_tokenizer: bool = field(
        default=True,
        metadata={"help": "Whether to use one of the fast tokenizer (backed by the toker
    )
    model_revision: str = field(
        default="main",
        metadata={"help": "The specific model version to use (can be a branch name, tag
    )
    use_auth_token: bool = field(
        default=False,
        metadata={
            "help": "Will use the token generated when running `transformers-cli login`
            "with private models)."
        },
    )

    def __post_init__(self):
        if self.config_overrides is not None and (self.config_name is not None or self.m
            raise ValueError(
                "--config_overrides can't be used in combination with --config_name or -
            )


@dataclass
class DataTrainingArguments:
    """
    Arguments pertaining to what data we are going to input our model for training and e
    """
    dataset_name: Optional[str] = field(
        default=None, metadata={"help": "The name of the dataset to use (via the dataset
    )
    dataset_config_name: Optional[str] = field(
        default=None, metadata={"help": "The configuration name of the dataset to use (v
    )
    train_file: Optional[str] = field(default=None, metadata={"help": "The input trainin
    validation_file: Optional[str] = field(
        default=None,
        metadata={"help": "An optional input evaluation data file to evaluate the perple
    )
    max_train_samples: Optional[int] = field(
        default=None,
        metadata={
```

```
            "help": "For debugging purposes or quicker training, truncate the number of
            "value if set."
        },
    )
    max_eval_samples: Optional[int] = field(
        default=None,
        metadata={
            "help": "For debugging purposes or quicker training, truncate the number of
            "value if set."
        },
    )

    block_size: Optional[int] = field(
        default=None,
        metadata={
            "help": "Optional input sequence length after tokenization. "
            "The training dataset will be truncated in block of this size for training.
            "Default to the model max input length for single sentence inputs (take into
        },
    )
    overwrite_cache: bool = field(
        default=False, metadata={"help": "Overwrite the cached training and evaluation s
    )
    validation_split_percentage: Optional[int] = field(
        default=5,
        metadata={
            "help": "The percentage of the train set used as validation set in case ther
        },
    )
    preprocessing_num_workers: Optional[int] = field(
        default=None,
        metadata={"help": "The number of processes to use for the preprocessing."},
    )
    keep_linebreaks: bool = field(
        default=True, metadata={"help": "Whether to keep line breaks when using TXT file
    )

    def __post_init__(self):
        if self.dataset_name is None and self.train_file is None and self.validation_fil
            raise ValueError("Need either a dataset name or a training/validation file.'
        else:
            if self.train_file is not None:
                extension = self.train_file.split(".")[-1]
                assert extension in ["csv", "json", "txt"], "`train_file` should be a cs
            if self.validation_file is not None:
                extension = self.validation_file.split(".")[-1]
                assert extension in ["csv", "json", "txt"], "`validation_file` should be

```

```
# Copied from huggingface examples.
#
```

```
"""
# Modified to include the following features:
# - Run as a command using arguments pass as a dictionary.
# - Returns the model before and after fine-tuning.

#!/usr/bin/env python
# coding=utf-8
# Copyright 2020 The HuggingFace Inc. team. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
"""
Fine-tuning the library models for causal language modeling (GPT, GPT-2, CTRL, ...) on a
Here is the full list of checkpoints on the hub that can be fine-tuned by this script:
https://huggingface.co/models?filter=text-generation
"""
# You can also adapt this script on your own causal language modeling task. Pointers for
# require_version("datasets>=1.8.0", "To fix: pip install -r examples/pytorch/language-m

logger = logging.getLogger(__name__)

MODEL_CONFIG_CLASSES = list(MODEL_FOR_CAUSAL_LM_MAPPING.keys())
MODEL_TYPES = tuple(conf.model_type for conf in MODEL_CONFIG_CLASSES)


def run_clm(args_as_dict, debug_state={}):

    # See all possible arguments in src/transformers/training_args.py
    # or by passing the --help flag to this script.
    # We now keep distinct sets of args, for a cleaner separation of concerns.

    parser = HfArgumentParser((ModelArguments, DataTrainingArguments, TrainingArguments)
    model_args, data_args, training_args = parser.parse_dict(args_as_dict)

    logging.basicConfig(
        format="%(asctime)s - %(levelname)s - %(name)s - %(message)s",
        datefmt="%m/%d/%Y %H:%M:%S",
        handlers=[logging.StreamHandler(sys.stdout)],
    )

    log_level = training_args.get_process_log_level()
    logger.setLevel(log_level)
    datasets.utils.logging.set_verbosity(log_level)
```

```
    transformers.utils.logging.set_verbosity(log_level)
    transformers.utils.logging.enable_default_handler()
    transformers.utils.logging.enable_explicit_format()

    logger.warning(
        f"Process rank: {training_args.local_rank}, device: {training_args.device}, n_gp
        + f"distributed training: {bool(training_args.local_rank != -1)}, 16-bits traini
    )
    logger.info(f"Training/evaluation parameters {training_args}")

    # Detecting last checkpoint.
    last_checkpoint = None
    if os.path.isdir(training_args.output_dir) and training_args.do_train and not traini
        last_checkpoint = get_last_checkpoint(training_args.output_dir)
        if last_checkpoint is None and len(os.listdir(training_args.output_dir)) > 0:
            raise ValueError(
                f"Output directory ({training_args.output_dir}) already exists and is no
                "Use --overwrite_output_dir to overcome."
            )
        elif last_checkpoint is not None and training_args.resume_from_checkpoint is Non
            logger.info(
                f"Checkpoint detected, resuming training at {last_checkpoint}. To avoid
                "the `--output_dir` or add `--overwrite_output_dir` to train from scratc
            )

    # Set seed before initializing model.
    set_seed(training_args.seed)

    data_files = {}
    dataset_args = {}
    if data_args.train_file is not None:
        data_files["train"] = data_args.train_file
    if data_args.validation_file is not None:
        data_files["validation"] = data_args.validation_file
    raw_datasets = load_dataset('json', data_files=data_files, cache_dir=model_args.cach

    # See more about loading any type of standard or custom dataset (from files, python
    # https://huggingface.co/docs/datasets/loading_datasets.html.

    # Load pretrained model and tokenizer
    #
    # Distributed training:
    # The .from_pretrained methods guarantee that only one local process can concurrentl
    # download model & vocab.

    config_kwargs = {
        "cache_dir": model_args.cache_dir,
        "revision": model_args.model_revision,
        "use_auth_token": True if model_args.use_auth_token else None,
    }
    if model_args.config_name:
        config = AutoConfig.from_pretrained(model_args.config_name, **config_kwargs)
```

```
    elif model_args.model_name_or_path:
        config = AutoConfig.from_pretrained(model_args.model_name_or_path, **config_kwar
    else:
        config = CONFIG_MAPPING[model_args.model_type]()
        logger.warning("You are instantiating a new config instance from scratch.")
        if model_args.config_overrides is not None:
            logger.info(f"Overriding config: {model_args.config_overrides}")
            config.update_from_string(model_args.config_overrides)
            logger.info(f"New config: {config}")

    tokenizer_kwargs = {
        "cache_dir": model_args.cache_dir,
        "use_fast": model_args.use_fast_tokenizer,
        "revision": model_args.model_revision,
        "use_auth_token": True if model_args.use_auth_token else None,
    }
    if model_args.tokenizer_name:
        tokenizer = AutoTokenizer.from_pretrained(model_args.tokenizer_name, **tokenizer
    elif model_args.model_name_or_path:
        tokenizer = AutoTokenizer.from_pretrained(model_args.model_name_or_path, **toker
    else:
        raise ValueError(
            "You are instantiating a new tokenizer from scratch. This is not supported b
            "You can do it from another script, save it, and load it from here, using -
        )

    debug_state['tokenizer'] = tokenizer

    if model_args.model_name_or_path:
        model = AutoModelForCausalLM.from_pretrained(
            model_args.model_name_or_path,
            from_tf=bool(".ckpt" in model_args.model_name_or_path),
            config=config,
            cache_dir=model_args.cache_dir,
            revision=model_args.model_revision,
            use_auth_token=True if model_args.use_auth_token else None,
        )
    else:
        model = AutoModelForCausalLM.from_config(config)
        n_params = sum(dict((p.data_ptr(), p.numel()) for p in model.parameters()).value
        logger.info(f"Training new model from scratch - Total size={n_params/2**20:.2f}M

    model.resize_token_embeddings(len(tokenizer))

    model_before_finetuning = debug_state["model_before_finetuning"] = copy.deepcopy(moc

    # Preprocessing the datasets.
    # First we tokenize all the texts.
    if training_args.do_train:
        column_names = raw_datasets["train"].column_names
    else:
```

```
        column_names = raw_datasets["validation"].column_names
    text_column_name = "story" if "story" in column_names else column_names[0]

    if args_as_dict.get('text_column_name', None) is not None:
        text_column_name = args_as_dict['text_column_name']

    # since this will be pickled to avoid _LazyModule error in Hasher force logger loadi
    tok_logger = transformers.utils.logging.get_logger("transformers.tokenization_utils_

    def tokenize_function(examples):
        with CaptureLogger(tok_logger) as cl:
            output = tokenizer(examples[text_column_name])
        # clm input could be much much longer than block_size
        if "Token indices sequence length is longer than the" in cl.out:
            tok_logger.warning(
                "^^^^^^^^^^^^^^^^ Please ignore the warning above - this long input wil]
            )
        return output

    with training_args.main_process_first(desc="dataset map tokenization"):
        tokenized_datasets = raw_datasets.map(
            tokenize_function,
            batched=True,
            num_proc=data_args.preprocessing_num_workers,
            remove_columns=column_names,
            load_from_cache_file=not data_args.overwrite_cache,
            desc="Running tokenizer on dataset",
        )

    if data_args.block_size is None:
        block_size = tokenizer.model_max_length
        if block_size > 1024:
            logger.warning(
                f"The tokenizer picked seems to have a very large `model_max_length` ({t
                "Picking 1024 instead. You can change that default value by passing --bl
            )
            block_size = 1024
    else:
        if data_args.block_size > tokenizer.model_max_length:
            logger.warning(
                f"The block_size passed ({data_args.block_size}) is larger than the maxi
                f"({tokenizer.model_max_length}). Using block_size={tokenizer.model_max_
            )
        block_size = min(data_args.block_size, tokenizer.model_max_length)

    debug_state['block_size'] = block_size

    # Main data processing function that will concatenate all texts from our dataset anc
    def group_texts(examples):
        # Concatenate all texts.
        concatenated_examples = {k: list(chain(*examples[k])) for k in examples.keys()}
        total_length = len(concatenated_examples[list(examples.keys())[0]])
```

```
            # We drop the small remainder, we could add padding if the model supported it ir
            # customize this part to your needs.
            if total_length >= block_size:
                total_length = (total_length // block_size) * block_size
            # Split by chunks of max_len.
            result = {
                k: [t[i : i + block_size] for i in range(0, total_length, block_size)]
                for k, t in concatenated_examples.items()
            }
            result["labels"] = result["input_ids"].copy()
            return result

        # Note that with `batched=True`, this map processes 1,000 texts together, so group_t
        # for each of those groups of 1,000 texts. You can adjust that batch_size here but a
        # to preprocess.
        #
        # To speed up this part, we use multiprocessing. See the documentation of the map me
        # https://huggingface.co/docs/datasets/package_reference/main_classes.html#datasets.

        with training_args.main_process_first(desc="grouping texts together"):
            lm_datasets = tokenized_datasets.map(
                group_texts,
                batched=True,
                num_proc=data_args.preprocessing_num_workers,
                load_from_cache_file=not data_args.overwrite_cache,
                desc=f"Grouping texts in chunks of {block_size}",
            )

        if training_args.do_train:
            if "train" not in tokenized_datasets:
                raise ValueError("--do_train requires a train dataset")
            train_dataset = lm_datasets["train"]
            if data_args.max_train_samples is not None:
                max_train_samples = min(len(train_dataset), data_args.max_train_samples)
                train_dataset = train_dataset.select(range(max_train_samples))

        if training_args.do_eval:
            if "validation" not in tokenized_datasets:
                raise ValueError("--do_eval requires a validation dataset")
            eval_dataset = lm_datasets["validation"]
            if data_args.max_eval_samples is not None:
                max_eval_samples = min(len(eval_dataset), data_args.max_eval_samples)
                eval_dataset = eval_dataset.select(range(max_eval_samples))

            def preprocess_logits_for_metrics(logits, labels):
                if isinstance(logits, tuple):
                    # Depending on the model and config, logits may contain extra tensors,
                    # like past_key_values, but logits always come first
                    logits = logits[0]
                return logits.argmax(dim=-1)

            metric = load_metric("accuracy")
```

```
        metric = load_metric("accuracy")

        def compute_metrics(eval_preds):
            preds, labels = eval_preds
            # preds have the same shape as the labels, after the argmax(-1) has been cal
            # by preprocess_logits_for_metrics but we need to shift the labels
            labels = labels[:, 1:].reshape(-1)
            preds = preds[:, :-1].reshape(-1)
            return metric.compute(predictions=preds, references=labels)

    # Initialize our Trainer
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset if training_args.do_train else None,
        eval_dataset=eval_dataset if training_args.do_eval else None,
        tokenizer=tokenizer,
        # Data collator will default to DataCollatorWithPadding, so we change it.
        data_collator=default_data_collator,
        compute_metrics=compute_metrics if training_args.do_eval and not is_torch_tpu_av
        preprocess_logits_for_metrics=preprocess_logits_for_metrics
        if training_args.do_eval and not is_torch_tpu_available()
        else None,
    )

    # Training
    model_after_finetuning = debug_state["model_after_finetuning"] = None
    if training_args.do_train:
        checkpoint = None
        if training_args.resume_from_checkpoint is not None:
            checkpoint = training_args.resume_from_checkpoint
        elif last_checkpoint is not None:
            checkpoint = last_checkpoint
        train_result = trainer.train(resume_from_checkpoint=checkpoint)
        trainer.save_model()  # Saves the tokenizer too for easy upload

        metrics = train_result.metrics

        max_train_samples = (
            data_args.max_train_samples if data_args.max_train_samples is not None else
        )
        metrics["train_samples"] = min(max_train_samples, len(train_dataset))

        trainer.log_metrics("train", metrics)
        trainer.save_metrics("train", metrics)
        trainer.save_state()

        model_after_finetuning = debug_state["model_after_finetuning"] = model

    # Evaluation
    if training_args.do_eval:
        logger.info("*** Evaluate ***")
```

```

        metrics = trainer.evaluate()

        max_eval_samples = data_args.max_eval_samples if data_args.max_eval_samples is r
        metrics["eval_samples"] = min(max_eval_samples, len(eval_dataset))
        try:
            perplexity = math.exp(metrics["eval_loss"])
        except OverflowError:
            perplexity = float("inf")
        metrics["perplexity"] = perplexity

        trainer.log_metrics("eval", metrics)
        trainer.save_metrics("eval", metrics)

    kwargs = {"finetuned_from": model_args.model_name_or_path, "tasks": "text-generatior
    if data_args.dataset_name is not None:
        kwargs["dataset_tags"] = data_args.dataset_name
        if data_args.dataset_config_name is not None:
            kwargs["dataset_args"] = data_args.dataset_config_name
            kwargs["dataset"] = f"{data_args.dataset_name} {data_args.dataset_config_nam
        else:
            kwargs["dataset"] = data_args.dataset_name

    # Should call this after `run_clm` to free up some GPU memory.
    # Some GPU memory will still be reserved, so if you need to re-run
    # fine-tuning, then you may need to click "Runtime -> Restart Runtime", although
    # this will reset all previously run cells.
    model_before_finetuning.cpu()
    model_after_finetuning.cpu()
    torch.cuda.empty_cache()

    return model_before_finetuning, model_after_finetuning
```

```
def compute_rouge(model, tokenizer, dataset, n=3):

  def count_ngrams(tokens, n):
    c = collections.Counter()
    for size in range(1, n + 1):
      for end in range(size, len(tokens) + 1):
        ngram = tuple(tokens[end - size:end])
        c[ngram] += 1
    return c

  def rouge(gold, pred, n):
    gold_c = count_ngrams(gold, n)
    pred_c = count_ngrams(pred, n)
    overlap = sum([pred_c[ngram] for ngram in gold_c.keys()])
    total = sum(gold_c.values())
    return overlap / total

```

```
  with torch.inference_mode():
    m = []
    for p1, p2 in tqdm(dataset, desc=f'Compute ROGUE-{n}'):
      # TODO: Does this include the correct values for beam search?
      beam_output = run_beam_search(
          model,
          tokenizer,
          p1,
          num_beams=3,
          num_decode_steps=32)
      pred = tokenizer.decode(beam_output['output_ids'][0], skip_special_tokens=True).sp
      pred_ids = tokenizer(pred, return_tensors="pt")['input_ids'][0].tolist()
      # p1_tensor = tokenizer(p1, return_tensors="pt")['input_ids']
      gold_ids = tokenizer(p2, return_tensors="pt")['input_ids'][0].tolist()
      m.append(rouge(gold_ids, pred_ids, n))

    return np.mean(m)


def compute_perplexity(model, tokenizer, dataset):

  with torch.inference_mode():
    n = 0
    m = []
    for p1, p2 in tqdm(dataset, desc='Compute Perplexity'):
      p1_tensor = tokenizer(p1, return_tensors="pt")['input_ids']
      p2_tensor = tokenizer(p2, return_tensors="pt")['input_ids']
      input_ids = torch.cat([p1_tensor, p2_tensor], 1).to(model.device)
      target = input_ids.clone()
      target[:, :p1_tensor.shape[1]] = -100
      target_length = p2_tensor.shape[1]
      n += target_length

      nll = model(input_ids=input_ids, labels=target)[0] * target_length
      m.append(nll)

    return torch.exp(torch.cat([x.view(1) for x in m], 0).sum() / n)
```

```
def preprocess_coqa(dataset):
    new_dataset = []
    skipped = 0
    for text in dataset:
        parts = text.split('. ', 2)
        if len(parts) <= 1:
            skipped += 1
            continue
        p1 = parts[0].strip() + '.'
        p2 = parts[1].strip() + '.'
        new_dataset.append((p1, p2))
```

Question 2.1 (15 points)

Run the cell below, which does the following steps:

- Fine-tune GPT-2 on the story text from the CoQA dataset.
- Preprocess the CoQA dataset into "sentence pairs". These pairs are created by finding the first and second sentence from each story passage in the validation data.
- Evaluate the language models from before and after fine-tuning using two different metrics: perplexity and ROUGE-3.

Important Notes:

- For training, the full story passages are used and can be many sentences long. For evaluation, only the sentence pairs are used. Both evaluation metrics are only evaluated on the second sentence.
- For perplexity, we use teacher forcing. For ROUGE-3 we use the first sentence as a prefix and generate a second sentence using beam search (beam size of 3 and generating a fixed amount of 32 tokens).

```
# Fine-tune GPT-2

config = {
    'model_name_or_path': 'gpt2',
    'train_file': 'coqa-train.json',
    'validation_file': 'coqa-dev.json',
    'text_column_name': 'story',
    'per_device_train_batch_size': 8,
    'per_device_eval_batch_size': 8,
    'gradient_accumulation_steps': 1,
    'learning_rate': 5e-5,
    'block_size': 256,
    'max_train_samples': 1024,
    'num_train_epochs': 1,
    'do_train': True,
    'do_eval': False,
    'output_dir': './tmp',
    'overwrite_output_dir': True,
    'log_level': 'warning' # Set to `info` or `debug` for additional logging.
}

# If preferred, can use these arguments in the config instead of `train_file`
# and `validation_file`, but sometimes Google Colab's IP gets throttled.
# 'dataset_name': 'coqa',
# 'dataset_config_name': 'default',

```

```
model_before_finetuning, model_after_finetuning = run_clm(config)
print('LM finetuning finished!')

# Preprocess the CoQA dataset into sentence pairs for evaluation.

dataset = load_dataset('json', data_files={'validation':'coqa-dev.json'}, field='data')[
new_dataset = preprocess_coqa(dataset)

print('Preprocessing finished!')
print(f'...found {len(new_dataset)} instances.')
print(f'...sample instance: {new_dataset[0]}')

# Run evaluation.

model_before_finetuning.cuda()
model_before_finetuning.eval()
model_after_finetuning.cuda()
model_after_finetuning.eval()

print('Running evaluation...')

before_ppl = compute_perplexity(model_before_finetuning, tokenizer, new_dataset).item()
after_ppl = compute_perplexity(model_after_finetuning, tokenizer, new_dataset).item()
print(f'\n\nPerplexity before_finetune = {before_ppl:.3f}, after_finetune = {after_ppl:.

before_rouge = compute_rouge(model_before_finetuning, tokenizer, new_dataset)
after_rouge = compute_rouge(model_after_finetuning, tokenizer, new_dataset)
print(f'\n\nROUGE-3 before_finetune = {before_rouge:.3f}, after_finetune = {after_rouge:

print('Evaluation finished!')

```

```
WARNING:__main__:Process rank: -1, device: cuda:0, n_gpu: 1distributed training: False,
WARNING:datasets.builder:Using custom data configuration default-4a9a50cb04c53708
Downloading and preparing dataset json/default to /root/.cache/huggingface/datasets/jso
Downloading data files: 100%                                2/2 [00:00<00:00, 57.29it/s]
Extracting data files: 100%                                 2/2 [00:00<00:00, 75.09it/s]
Dataset json downloaded and prepared to /root/.cache/huggingface/datasets/json/default-
100%                                      2/2 [00:00<00:00, 45.96it/s]
Downloading: 100%                                     1.29M/1.29M [00:01<00:00, 928kB/s]
WARNING:datasets.fingerprint:Parameter 'function'=<function run_clm.<locals>.tokenize_f
Running tokenizer on dataset:                                  8/8 [00:06<00:00,
100%                                                           1.49ba/s]
[WARNING|tokenization_utils_base.py:3397] 2022-12-03 15:44:26,772 >> Token indices sequ
[WARNING|<ipython-input-43-6b0a8fc7b51e>:171] 2022-12-03 15:44:26,774 >> ^^^^^^^^^^^^^^
Running tokenizer on dataset:                                  1/1 [00:00<00:00,
100%                                                           2.11ba/s]
Grouping texts in chunks of 256:                               8/8 [00:02<00:00,
100%                                                           2.64ba/s]
Grouping texts in chunks of 256:                               1/1 [00:00<00:00,
100%                                                           4.42ba/s]
/usr/local/lib/python3.8/dist-packages/transformers/optimization.py:306: FutureWarning:
  warnings.warn(
[128/128 01:14, Epoch 1/1]
```

Step	Training Loss

```
***** train metrics *****
  epoch                    =        1.0
  total_flos               =   124593GF
  train_loss               =     3.4296
  train_runtime            = 0:01:15.59
  train_samples            =       1024
  train_samples_per_second =     13.546
  train_steps_per_second   =      1.693
```

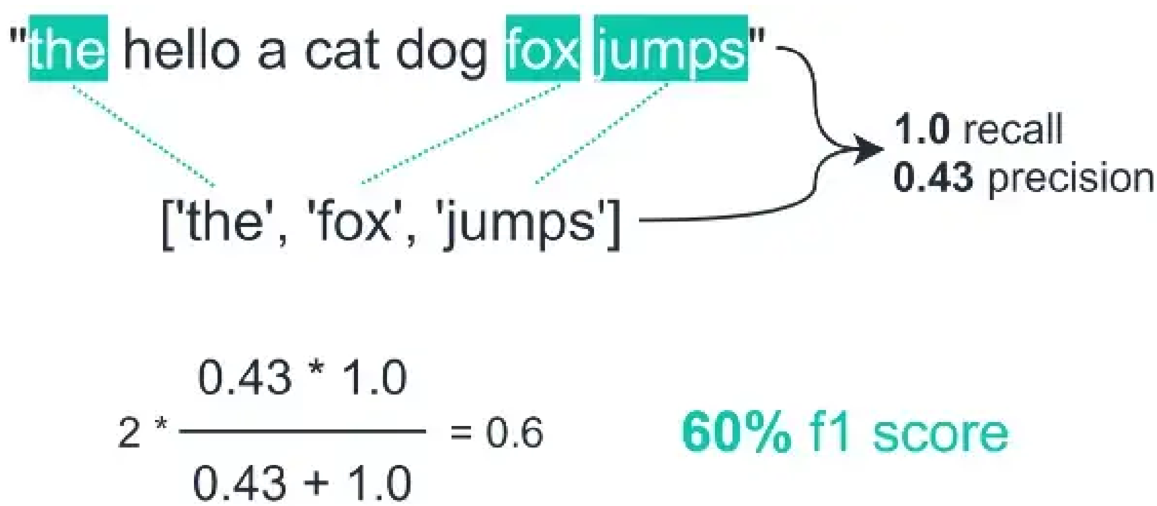
Has language model fine-tuning improved GPT-2 performance on the story text for the CoQA dataset? Is perplexity or ROUGE a better metric for measuring this?

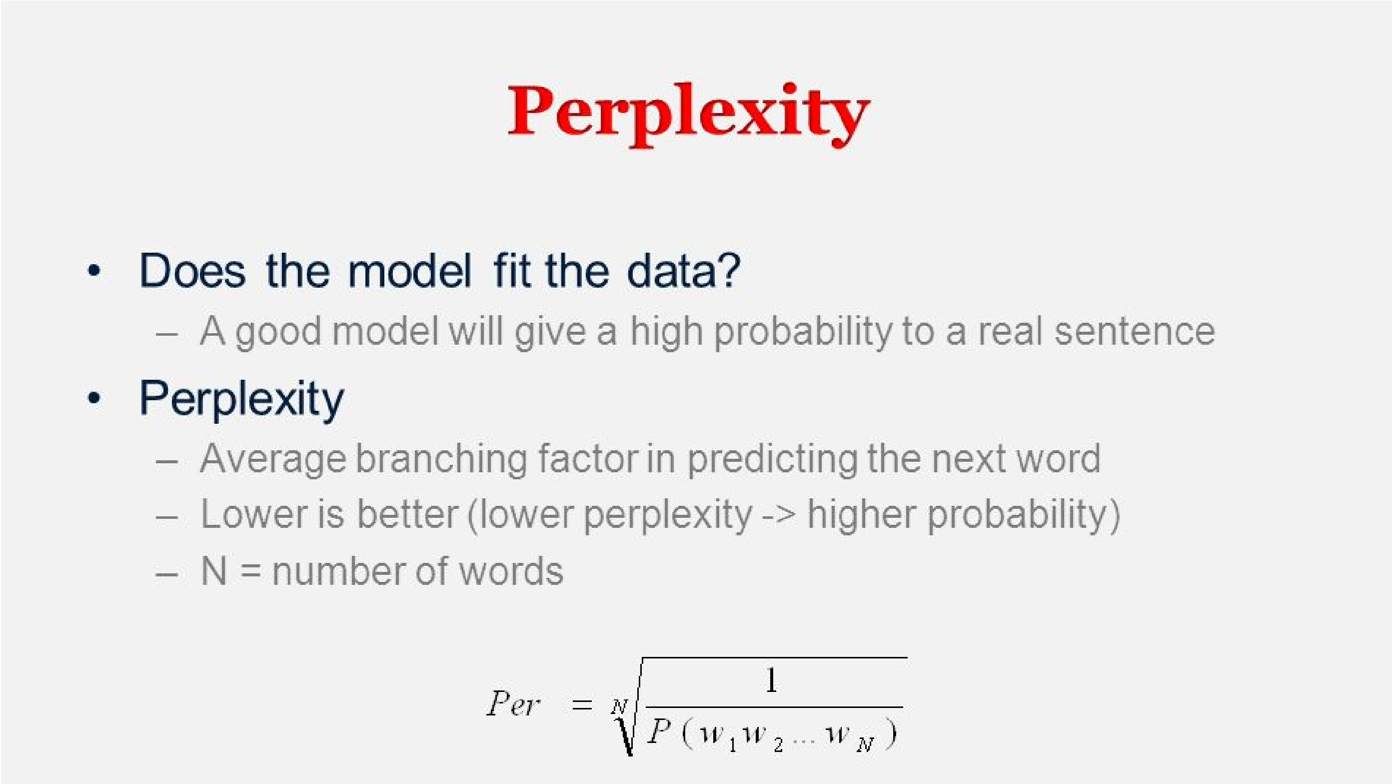
```
Downloading data files: 100%                                1/1 [00:00<00:00, 25.11it/s]
```

WRITE YOUR ANSWER HERE IN A FEW SENTENCES

> Perplexity before_finetune = 34.129, after_finetune = 30.469
>
> ROUGE-3 before_finetune = 0.086, after_finetune = 0.083
>
> The language model fine-tuning improved GPT-2 performance on the story text for the CoQA dataset. The perplexity decreased from 34 to 30 (average branching factor

> reduced) and ROUGE-3 did not change much. (0.086->0.083)
>
> Perplexity is a better metric for measuring this. ROUGE is more used for evaluating automatic summarization and machine translation, where the generated text should include specific words and phrases to best express the prior input text. On the other hand, in our case of generating second sentence of a story given the previous sentence, this does not hold. The second sentence can have many possibilities and all can be correct even without much sharing ngrams with the real sentence. Therefore, in this case and also evaluating language models in general for their generation capabilites, perplexity best represents their power. It best captures whether the model has properly fit the data or not, which in this case is the context of stories.





We're done with GPT-2 for now, so move the trained models back to CPU.

```
# Should call this after `run_clm` to free up some GPU memory.
# Some GPU memory will still be reserved, so if you need to re-run
# fine-tuning, then you may need to click "Runtime -> Restart Runtime", although
# this will reset all previously run cells.
model_before_finetuning.cpu()
model_after_finetuning.cpu()
torch.cuda.empty_cache()
```

Part 3: Data Augmentation via Backtranslation

The last part of this homework involves data augmentation of an NLP classifier via backtranslation. Now run the below cell to set up some fine-tuning code.

```
import logging
import os
import random
import sys
from dataclasses import dataclass, field
from typing import Optional

import datasets
```

```
import numpy as np
from datasets import load_dataset, load_metric

import transformers
from transformers import (
    AutoConfig,
    AutoModelForSequenceClassification,
    AutoTokenizer,
    DataCollatorWithPadding,
    EvalPrediction,
    HfArgumentParser,
    PretrainedConfig,
    Trainer,
    TrainingArguments,
    default_data_collator,
    set_seed,
)
from transformers.trainer_utils import get_last_checkpoint
from transformers.utils import check_min_version
from transformers.utils.versions import require_version

from transformers import glue_processors
from transformers.data.processors.utils import InputExample
from langdetect import detect
```

```
#!/usr/bin/env python
# coding=utf-8
# Copyright 2020 The HuggingFace Inc. team. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
""" Finetuning the library models for sequence classification on GLUE."""
# You can also adapt this script on your own text classification task. Pointers for this

# Will error if the minimal version of Transformers is not installed. Remove at your own
# check_min_version("4.18.0.dev0")

# require_version("datasets>=1.8.0", "To fix: pip install -r examples/pytorch/text-class

task_to_keys = {
    "cola": ("sentence", None),
    "mnli": ("premise", "hypothesis"),
```

```
    "mrpc": ("sentence1", "sentence2"),
    "qnli": ("question", "sentence"),
    "qqp": ("question1", "question2"),
    "rte": ("sentence1", "sentence2"),
    "sst2": ("sentence", None),
    "stsb": ("sentence1", "sentence2"),
    "wnli": ("sentence1", "sentence2"),
}

logger = logging.getLogger(__name__)


@dataclass
class DataTrainingArguments:
    """
    Arguments pertaining to what data we are going to input our model for training and e

    Using `HfArgumentParser` we can turn this class
    into argparse arguments to be able to specify them on
    the command line.
    """

    task_name: Optional[str] = field(
        default=None,
        metadata={"help": "The name of the task to train on: " + ", ".join(task_to_keys.
    )
    dataset_name: Optional[str] = field(
        default=None, metadata={"help": "The name of the dataset to use (via the dataset
    )
    dataset_config_name: Optional[str] = field(
        default=None, metadata={"help": "The configuration name of the dataset to use (v
    )
    max_seq_length: int = field(
        default=128,
        metadata={
            "help": "The maximum total input sequence length after tokenization. Sequenc
            "than this will be truncated, sequences shorter will be padded."
        },
    )
    overwrite_cache: bool = field(
        default=False, metadata={"help": "Overwrite the cached preprocessed datasets or
    )
    pad_to_max_length: bool = field(
        default=True,
        metadata={
            "help": "Whether to pad all samples to `max_seq_length`. "
            "If False, will pad the samples dynamically when batching to the maximum len
        },
    )
    max_train_samples: Optional[int] = field(
        default=None,
```

```
        metadata={
            "help": "For debugging purposes or quicker training, truncate the number of
            "value if set."
        },
    )
    max_eval_samples: Optional[int] = field(
        default=None,
        metadata={
            "help": "For debugging purposes or quicker training, truncate the number of
            "value if set."
        },
    )
    max_predict_samples: Optional[int] = field(
        default=None,
        metadata={
            "help": "For debugging purposes or quicker training, truncate the number of
            "value if set."
        },
    )
    train_file: Optional[str] = field(
        default=None, metadata={"help": "A csv or a json file containing the training da
    )
    validation_file: Optional[str] = field(
        default=None, metadata={"help": "A csv or a json file containing the validation
    )
    test_file: Optional[str] = field(default=None, metadata={"help": "A csv or a json fi

    def __post_init__(self):
        if self.task_name is not None:
            self.task_name = self.task_name.lower()
            if self.task_name not in task_to_keys.keys():
                raise ValueError("Unknown task, you should pick one in " + ",".join(task
        elif self.dataset_name is not None:
            pass
        elif self.train_file is None or self.validation_file is None:
            raise ValueError("Need either a GLUE task, a training/validation file or a d
        else:
            train_extension = self.train_file.split(".")[-1]
            assert train_extension in ["csv", "json"], "`train_file` should be a csv or
            validation_extension = self.validation_file.split(".")[-1]
            assert (
                validation_extension == train_extension
            ), "`validation_file` should have the same extension (csv or json) as `train


@dataclass
class ModelArguments:
    """
    Arguments pertaining to which model/config/tokenizer we are going to fine-tune from.
    """

```

```
    model_name_or_path: str = field(
        metadata={"help": "Path to pretrained model or model identifier from huggingface
    )
    config_name: Optional[str] = field(
        default=None, metadata={"help": "Pretrained config name or path if not the same
    )
    tokenizer_name: Optional[str] = field(
        default=None, metadata={"help": "Pretrained tokenizer name or path if not the sa
    )
    cache_dir: Optional[str] = field(
        default=None,
        metadata={"help": "Where do you want to store the pretrained models downloaded f
    )
    use_fast_tokenizer: bool = field(
        default=True,
        metadata={"help": "Whether to use one of the fast tokenizer (backed by the token
    )
    model_revision: str = field(
        default="main",
        metadata={"help": "The specific model version to use (can be a branch name, tag
    )
    use_auth_token: bool = field(
        default=False,
        metadata={
            "help": "Will use the token generated when running `transformers-cli login`
            "with private models)."
        },
```

```
def do_target_task_finetuning(args_as_dict):

    # See all possible arguments in src/transformers/training_args.py
    # or by passing the --help flag to this script.
    # We now keep distinct sets of args, for a cleaner separation of concerns.

    parser = HfArgumentParser((ModelArguments, DataTrainingArguments, TrainingArguments)
    model_args, data_args, training_args = parser.parse_dict(args_as_dict)

    # Setup logging
    logging.basicConfig(
        format="%(asctime)s - %(levelname)s - %(name)s - %(message)s",
        datefmt="%m/%d/%Y %H:%M:%S",
        handlers=[logging.StreamHandler(sys.stdout)],
    )

    log_level = training_args.get_process_log_level()
    logger.setLevel(log_level)
    datasets.utils.logging.set_verbosity(log_level)
    transformers.utils.logging.set_verbosity(log_level)
    transformers.utils.logging.enable_default_handler()
    transformers.utils.logging.enable_explicit_format()

```

```
    # Log on each process the small summary:
    logger.warning(
        f"Process rank: {training_args.local_rank}, device: {training_args.device}, n_gp
        + f"distributed training: {bool(training_args.local_rank != -1)}, 16-bits traini
    )
    logger.info(f"Training/evaluation parameters {training_args}")

    # Detecting last checkpoint.
    last_checkpoint = None
    if os.path.isdir(training_args.output_dir) and training_args.do_train and not traini
        last_checkpoint = get_last_checkpoint(training_args.output_dir)
        if last_checkpoint is None and len(os.listdir(training_args.output_dir)) > 0:
            raise ValueError(
                f"Output directory ({training_args.output_dir}) already exists and is no
                "Use --overwrite_output_dir to overcome."
            )
        elif last_checkpoint is not None and training_args.resume_from_checkpoint is Non
            logger.info(
                f"Checkpoint detected, resuming training at {last_checkpoint}. To avoid
                "the `--output_dir` or add `--overwrite_output_dir` to train from scratc
            )

    # Set seed before initializing model.
    set_seed(training_args.seed)

    # In distributed training, the load_dataset function guarantee that only one local p
    # download the dataset.
    if data_args.task_name is not None:
        # Downloading and loading a dataset from the hub.
        raw_datasets = load_dataset("glue", data_args.task_name, cache_dir=model_args.ca
    elif data_args.dataset_name is not None:
        # Downloading and loading a dataset from the hub.
        raw_datasets = load_dataset(
            data_args.dataset_name, data_args.dataset_config_name, cache_dir=model_args.
        )
    else:
        # Loading a dataset from your local files.
        # CSV/JSON training and evaluation files are needed.
        data_files = {"train": data_args.train_file, "validation": data_args.validation_

        # Get the test dataset: you can provide your own CSV/JSON test file (see below)
        # when you use `do_predict` without specifying a GLUE benchmark task.
        if training_args.do_predict:
            if data_args.test_file is not None:
                train_extension = data_args.train_file.split(".")[-1]
                test_extension = data_args.test_file.split(".")[-1]
                assert (
                    test_extension == train_extension
                ), "`test_file` should have the same extension (csv or json) as `train_f
                data_files["test"] = data_args.test_file
            else:
```

```
                raise ValueError("Need either a GLUE task or a test file for `do_predict

        for key in data_files.keys():
            logger.info(f"load a local file for {key}: {data_files[key]}")

        if data_args.train_file.endswith(".csv"):
            # Loading a dataset from local csv files
            raw_datasets = load_dataset("csv", data_files=data_files, cache_dir=model_ar
        else:
            # Loading a dataset from local json files
            raw_datasets = load_dataset("json", data_files=data_files, cache_dir=model_a
    # See more about loading any type of standard or custom dataset at
    # https://huggingface.co/docs/datasets/loading_datasets.html.

    # Labels
    if data_args.task_name is not None:
        is_regression = data_args.task_name == "stsb"
        if not is_regression:
            label_list = raw_datasets["train"].features["label"].names
            num_labels = len(label_list)
        else:
            num_labels = 1
    else:
        # Trying to have good defaults here, don't hesitate to tweak to your needs.
        is_regression = raw_datasets["train"].features["label"].dtype in ["float32", "fl
        if is_regression:
            num_labels = 1
        else:
            # A useful fast method:
            # https://huggingface.co/docs/datasets/package_reference/main_classes.html#d
            label_list = raw_datasets["train"].unique("label")
            label_list.sort()  # Let's sort it for determinism
            num_labels = len(label_list)

    # Load pretrained model and tokenizer
    #
    # In distributed training, the .from_pretrained methods guarantee that only one loca
    # download model & vocab.
    config = AutoConfig.from_pretrained(
        model_args.config_name if model_args.config_name else model_args.model_name_or_p
        num_labels=num_labels,
        finetuning_task=data_args.task_name,
        cache_dir=model_args.cache_dir,
        revision=model_args.model_revision,
        use_auth_token=True if model_args.use_auth_token else None,
    )
    tokenizer = AutoTokenizer.from_pretrained(
        model_args.tokenizer_name if model_args.tokenizer_name else model_args.model_nam
        cache_dir=model_args.cache_dir,
        use_fast=model_args.use_fast_tokenizer,
        revision=model_args.model_revision,
```

```
        use_auth_token=True if model_args.use_auth_token else None,
    )
    model = AutoModelForSequenceClassification.from_pretrained(
        model_args.model_name_or_path,
        from_tf=bool(".ckpt" in model_args.model_name_or_path),
        config=config,
        cache_dir=model_args.cache_dir,
        revision=model_args.model_revision,
        use_auth_token=True if model_args.use_auth_token else None,
    )

    # Preprocessing the raw_datasets
    if data_args.task_name is not None:
        sentence1_key, sentence2_key = task_to_keys[data_args.task_name]
    else:
        # Again, we try to have some nice defaults but don't hesitate to tweak to your u
        non_label_column_names = [name for name in raw_datasets["train"].column_names if
        if "sentence1" in non_label_column_names and "sentence2" in non_label_column_nam
            sentence1_key, sentence2_key = "sentence1", "sentence2"
        else:
            if len(non_label_column_names) >= 2:
                sentence1_key, sentence2_key = non_label_column_names[:2]
            else:
                sentence1_key, sentence2_key = non_label_column_names[0], None

    # Padding strategy
    if data_args.pad_to_max_length:
        padding = "max_length"
    else:
        # We will pad later, dynamically at batch creation, to the max sequence length i
        padding = False

    # Some models have set the order of the labels to use, so let's make sure we do use
    label_to_id = None
    if (
        model.config.label2id != PretrainedConfig(num_labels=num_labels).label2id
        and data_args.task_name is not None
        and not is_regression
    ):
        # Some have all caps in their config, some don't.
        label_name_to_id = {k.lower(): v for k, v in model.config.label2id.items()}
        if list(sorted(label_name_to_id.keys())) == list(sorted(label_list)):
            label_to_id = {i: int(label_name_to_id[label_list[i]]) for i in range(num_la
        else:
            logger.warning(
                "Your model seems to have been trained with labels, but they don't match
                f"model labels: {list(sorted(label_name_to_id.keys()))}, dataset labels:
                "\nIgnoring the model labels as a result.",
            )
    elif data_args.task_name is None and not is_regression:
        label_to_id = {v: i for i, v in enumerate(label_list)}
```

```
    if label_to_id is not None:
        model.config.label2id = label_to_id
        model.config.id2label = {id: label for label, id in config.label2id.items()}
    elif data_args.task_name is not None and not is_regression:
        model.config.label2id = {l: i for i, l in enumerate(label_list)}
        model.config.id2label = {id: label for label, id in config.label2id.items()}

    if data_args.max_seq_length > tokenizer.model_max_length:
        logger.warning(
            f"The max_seq_length passed ({data_args.max_seq_length}) is larger than the
            f"model ({tokenizer.model_max_length}). Using max_seq_length={tokenizer.mode
        )
    max_seq_length = min(data_args.max_seq_length, tokenizer.model_max_length)

    def preprocess_function(examples):
        # Tokenize the texts
        args = (
            (examples[sentence1_key],) if sentence2_key is None else (examples[sentence1
        )
        result = tokenizer(*args, padding=padding, max_length=max_seq_length, truncation

        # Map labels to IDs (not necessary for GLUE tasks)
        if label_to_id is not None and "label" in examples:
            result["label"] = [(label_to_id[l] if l != -1 else -1) for l in examples["la
        return result

    with training_args.main_process_first(desc="dataset map pre-processing"):
        raw_datasets = raw_datasets.map(
            preprocess_function,
            batched=True,
            load_from_cache_file=not data_args.overwrite_cache,
            desc="Running tokenizer on dataset",
        )
    if training_args.do_train:
        if "train" not in raw_datasets:
            raise ValueError("--do_train requires a train dataset")
        train_dataset = raw_datasets["train"]
        if data_args.max_train_samples is not None:
            train_dataset = train_dataset.select(range(data_args.max_train_samples))

    if training_args.do_eval:
        if "validation" not in raw_datasets and "validation_matched" not in raw_datasets
            raise ValueError("--do_eval requires a validation dataset")
        eval_dataset = raw_datasets["validation_matched" if data_args.task_name == "mnli
        if data_args.max_eval_samples is not None:
            eval_dataset = eval_dataset.select(range(data_args.max_eval_samples))

    if training_args.do_predict or data_args.task_name is not None or data_args.test_fil
        if "test" not in raw_datasets and "test_matched" not in raw_datasets:
            raise ValueError("--do_predict requires a test dataset")
```

```
        predict_dataset = raw_datasets["test_matched" if data_args.task_name == "mnli" e
        if data_args.max_predict_samples is not None:
            predict_dataset = predict_dataset.select(range(data_args.max_predict_samples

    # Log a few random samples from the training set:
    if training_args.do_train:
        for index in random.sample(range(len(train_dataset)), 3):
            logger.info(f"Sample {index} of the training set: {train_dataset[index]}.")

    # Get the metric function
    if data_args.task_name is not None:
        metric = load_metric("glue", data_args.task_name)
    else:
        metric = load_metric("accuracy")

    # You can define your custom compute_metrics function. It takes an `EvalPrediction`
    # predictions and label_ids field) and has to return a dictionary string to float.
    def compute_metrics(p: EvalPrediction):
        preds = p.predictions[0] if isinstance(p.predictions, tuple) else p.predictions
        preds = np.squeeze(preds) if is_regression else np.argmax(preds, axis=1)
        if data_args.task_name is not None:
            result = metric.compute(predictions=preds, references=p.label_ids)
            if len(result) > 1:
                result["combined_score"] = np.mean(list(result.values())).item()
            return result
        elif is_regression:
            return {"mse": ((preds - p.label_ids) ** 2).mean().item()}
        else:
            return {"accuracy": (preds == p.label_ids).astype(np.float32).mean().item()}

    # Data collator will default to DataCollatorWithPadding when the tokenizer is passed
    # we already did the padding.
    if data_args.pad_to_max_length:
        data_collator = default_data_collator
    elif training_args.fp16:
        data_collator = DataCollatorWithPadding(tokenizer, pad_to_multiple_of=8)
    else:
        data_collator = None

    # Initialize our Trainer
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset if training_args.do_train else None,
        eval_dataset=eval_dataset if training_args.do_eval else None,
        compute_metrics=compute_metrics,
        tokenizer=tokenizer,
        data_collator=data_collator,
    )

    # Training
```

```
    if training_args.do_train:
        checkpoint = None
        if training_args.resume_from_checkpoint is not None:
            checkpoint = training_args.resume_from_checkpoint
        elif last_checkpoint is not None:
            checkpoint = last_checkpoint
        train_result = trainer.train(resume_from_checkpoint=checkpoint)
        metrics = train_result.metrics
        max_train_samples = (
            data_args.max_train_samples if data_args.max_train_samples is not None else
        )
        metrics["train_samples"] = min(max_train_samples, len(train_dataset))

        trainer.save_model()  # Saves the tokenizer too for easy upload

        trainer.log_metrics("train", metrics)
        trainer.save_metrics("train", metrics)
        trainer.save_state()

    # Evaluation
    if training_args.do_eval:
        logger.info("*** Evaluate ***")

        # Loop to handle MNLI double evaluation (matched, mis-matched)
        tasks = [data_args.task_name]
        eval_datasets = [eval_dataset]
        if data_args.task_name == "mnli":
            tasks.append("mnli-mm")
            eval_datasets.append(raw_datasets["validation_mismatched"])

        for eval_dataset, task in zip(eval_datasets, tasks):
            metrics = trainer.evaluate(eval_dataset=eval_dataset)

            max_eval_samples = (
                data_args.max_eval_samples if data_args.max_eval_samples is not None els
            )
            metrics["eval_samples"] = min(max_eval_samples, len(eval_dataset))

            trainer.log_metrics("eval", metrics)
            trainer.save_metrics("eval", metrics)

    kwargs = {"finetuned_from": model_args.model_name_or_path, "tasks": "text-classifica
    if data_args.task_name is not None:
        kwargs["language"] = "en"
        kwargs["dataset_tags"] = "glue"
        kwargs["dataset_args"] = data_args.task_name
```

Run finetuning baselines

`BERT` is unstable and prone to degenerate performance on tasks with small training sets. The below cell fine-tunes `BERT` on `tinySST` (a small sentiment analysis dataset) using some default hyperparameters and also reports the mean and standard deviation of the dev set accuracy across 4 random seeds. Run the cell to obtain these baseline numbers, which should be around 50% average accuracy (it might take a couple of minutes to finish)

```
import timeit

start_time = timeit.default_timer()
task_name = "SST"
data_dir = f"./data/tiny{task_name}"
model_name_or_path = "bert-base-cased"
model_cache_dir = os.path.join(pretrained_models_dir, model_name_or_path)
data_cache_dir = f"./data_cache/finetuning/tiny{task_name}"

# Fine-tune and evaluate BERT with default hyperparameters using 4 random seeds
results = []
for seed in [1234, 2341, 3412, 4123]:
  output_dir = f"./output/tiny{task_name}-{seed}"
  config = dict(
      seed=seed,
      model_name_or_path=model_name_or_path,
      train_file="./data/tinySST/train.csv",
      validation_file="./data/tinySST/dev.csv",
      task_type="text_classification",
      do_train=True,
      do_eval=True,
      do_lower_case=True,
      data_dir=data_dir,
      max_seq_length=128,
      per_device_train_batch_size=32,
      learning_rate=2e-5,
      num_train_epochs=3.0,
      model_cache_dir=model_cache_dir,
      data_cache_dir=data_cache_dir,
      output_dir=output_dir,
      overwrite_output_dir=True,
      log_level='warning'
  )

  result = do_target_task_finetuning(config)
  results.append(result["eval_accuracy"])

results = np.array(results)
mean = np.mean(results)
std = np.std(results)

print(f"Accuracy on TinySST dev set: {mean} +/- {std}")
elapsed_time = timeit.default_timer() - start_time
print(f"Time elapsed: {elapsed_time} seconds")
```

```
WARNING:__main__:Process rank: -1, device: cuda:0, n_gpu: 1distributed training: False,
WARNING:datasets.builder:Using custom data configuration default-ae0aa6a33dbb7123
Downloading and preparing dataset csv/default to /root/.cache/huggingface/datasets/csv/
```

Downloading data files: 100% 2/2 [00:00<00:00, 64.36it/s]

Extracting data files: 100% 2/2 [00:00<00:00, 52.54it/s]

```
Dataset csv downloaded and prepared to /root/.cache/huggingface/datasets/csv/default-ae
```

100% 2/2 [00:00<00:00, 61.43it/s]

Downloading: 100% 570/570 [00:00<00:00, 15.6kB/s]

Downloading: 100% 29.0/29.0 [00:00<00:00, 1.12kB/s]

Downloading: 100% 208k/208k [00:00<00:00, 245kB/s]

Downloading: 100% 426k/426k [00:01<00:00, 449kB/s]

Downloading: 100% 416M/416M [00:25<00:00, 16.6MB/s]

```
[WARNING|modeling_utils.py:1693] 2022-12-03 16:15:18,445 >> Some weights of the model c
- This IS expected if you are initializing BertForSequenceClassification from the check
- This IS NOT expected if you are initializing BertForSequenceClassification from the c
[WARNING|modeling_utils.py:1704] 2022-12-03 16:15:18,447 >> Some weights of BertForSequ
You should probably TRAIN this model on a down-stream task to be able to use it for pre
```

Running tokenizer on dataset: 100% 1/1 [00:00<00:00, 25.81ba/s]

Running tokenizer on dataset: 100% 1/1 [00:00<00:00, 6.85ba/s]

Downloading builder script: 3.19k/? [00:00<00:00, 66.1kB/s]

[3/3 00:00, Epoch 3/3]

Step	Training Loss

```
***** train metrics *****
  epoch                    =        3.0
  total_flos               =     3675GF
  train_loss               =     0.6892
  train_runtime            = 0:00:01.21
  train_samples            =         20
  train_samples_per_second =      49.21
  train_steps_per_second   =       2.46
```

[109/109 00:06]

```
WARNING:__main__:Process rank: -1, device: cuda:0, n_gpu: 1distributed training: False,
***** eval metrics *****
  epoch                   =        3.0
  eval_accuracy           =     0.4977
  eval_loss               =     0.6911
  eval_runtime            = 0:00:06.20
  eval_samples            =        872
  eval_samples_per_second =    140.541
  eval_steps_per_second   =     17.568
WARNING:datasets.builder:Using custom data configuration default-ae0aa6a33dbb7123
WARNING:datasets.builder:Reusing dataset csv (/root/.cache/huggingface/datasets/csv/def
```

100% 2/2 [00:00<00:00, 50.16it/s]

```
[WARNING|modeling_utils.py:1693] 2022-12-03 16:15:41,168 >> Some weights of the model c
- This IS expected if you are initializing BertForSequenceClassification from the check
- This IS NOT expected if you are initializing BertForSequenceClassification from the c
[WARNING|modeling_utils.py:1704] 2022-12-03 16:15:41,169 >> Some weights of BertForSequ
You should probably TRAIN this model on a down-stream task to be able to use it for pre
```

Running tokenizer on dataset: 100% 1/1 [00:00<00:00, 27.43ba/s]

Running tokenizer on dataset: 100% 1/1 [00:00<00:00, 5.23ba/s]

[3/3 00:00, Epoch 3/3]

Step	Training Loss

```
***** train metrics *****
  epoch                    =        3.0
  total_flos               =     3675GF
  train_loss               =     0.7235
  train_runtime            = 0:00:01.23
  train_samples            =         20
  train_samples_per_second =      48.69
  train_steps_per_second   =      2.434
```

[109/109 00:06]

```
WARNING:__main__:Process rank: -1, device: cuda:0, n_gpu: 1distributed training: False,
***** eval metrics *****
  epoch                   =        3.0
  eval_accuracy           =        0.5
  eval_loss               =     0.6947
  eval_runtime            = 0:00:06.26
  eval_samples            =        872
  eval_samples_per_second =    139.199
  eval_steps_per_second   =       17.4
WARNING:datasets.builder:Using custom data configuration default-ae0aa6a33dbb7123
WARNING:datasets.builder:Reusing dataset csv (/root/.cache/huggingface/datasets/csv/def
```

100% 2/2 [00:00<00:00, 67.32it/s]

```
[WARNING|modeling_utils.py:1693] 2022-12-03 16:16:03,647 >> Some weights of the model c
- This IS expected if you are initializing BertForSequenceClassification from the check
- This IS NOT expected if you are initializing BertForSequenceClassification from the c
[WARNING|modeling_utils.py:1704] 2022-12-03 16:16:03,654 >> Some weights of BertForSequ
You should probably TRAIN this model on a down-stream task to be able to use it for pre
```

Running tokenizer on dataset: 100% 1/1 [00:00<00:00, 25.24ba/s]

Running tokenizer on dataset: 100% 1/1 [00:00<00:00, 5.85ba/s]

[3/3 00:00, Epoch 3/3]

Step	Training Loss

```
***** train metrics *****
  epoch                    =        3.0
```

```
  total_flos               =     3675GF
  train_loss               =     0.6883
  train_runtime            = 0:00:01.19
  train_samples            =         20
  train_samples_per_second =     50.061
  train_steps_per_second   =      2.503
```

[109/109 00:06]

```
WARNING:__main__:Process rank: -1, device: cuda:0, n_gpu: 1distributed training: False,
***** eval metrics *****
  epoch                   =        3.0
  eval_accuracy           =     0.6055
  eval_loss               =     0.6732
  eval_runtime            = 0:00:06.32
  eval_samples            =        872
  eval_samples_per_second =    137.759
  eval_steps_per_second   =      17.22
WARNING:datasets.builder:Using custom data configuration default-ae0aa6a33dbb7123
WARNING:datasets.builder:Reusing dataset csv (/root/.cache/huggingface/datasets/csv/def
```

100% 2/2 [00:00<00:00, 67.06it/s]

```
[WARNING|modeling_utils.py:1693] 2022-12-03 16:16:26,281 >> Some weights of the model c
- This IS expected if you are initializing BertForSequenceClassification from the check
- This IS NOT expected if you are initializing BertForSequenceClassification from the c
[WARNING|modeling_utils.py:1704] 2022-12-03 16:16:26,283 >> Some weights of BertForSequ
You should probably TRAIN this model on a down-stream task to be able to use it for pre
```

Running tokenizer on dataset: 100% 1/1 [00:00<00:00, 31.20ba/s]

Running tokenizer on dataset: 100% 1/1 [00:00<00:00, 5.58ba/s]

[3/3 00:00, Epoch 3/3]

Step	Training Loss

```
***** train metrics *****
  epoch                    =        3.0
  total_flos               =     3675GF
  train_loss               =     0.6595
  train_runtime            = 0:00:01.23
  train_samples            =         20
  train_samples_per_second =     48.602
```

Run translate demo

Now run the following cell to load Google Translate's model and run it on a toy example. You will use Google Translate to augment your TinySST dataset via backtranslation, which involves translating an example to another language (or languages) and then eventually translating it back to English. This process injects syntactic and lexical variation into the input which can help the model learn.

```
import googletrans
# Run print(googletrans.LANGUAGES) to see available languages
from googletrans import Translator
translator = Translator()

# translate from English to French
output = translator.translate("I love natural language processing", src='en', dest='fa')
output.text
```

'من عاشق پردازش زبان طبیعی هستم'

Question 3.1 (20 points)

Complete the following cell to paraphrase the training data of `tinySST` using backtranslation. We have intentionally left this problem open-ended: feel free to use as many pivot languages as you like, and also write any postprocessing code you think might help. The cell after this one will fine-tune BERT on the augmented training data, so you can use its output to validate your backtranslation strategy. To obtain full points, the model fine-tuned on your augmented data must achieve a higher average accuracy (averaged across random seeds) than the model without any augmentation, trained with the same hyperparameters.

Write your code in the indicated section in the below cell.

```
task_name = "SST"
data_dir = f"./data/tiny{task_name}"
task_processor = glue_processors[f"{task_name.lower()}-2"]()
train_examples = task_processor.get_train_examples(data_dir)

train_examples_augmented = []

### (incomplete) list of languages you can use
languages = [
    'en', # english
    'cs',  # czech
    'de',  # german
    'es', # spanish
    'fi',  # finnish
    'fr', # french
    'hi', # hindi
    'it', # italian
    'ja', # japanese
    'pt', # portuguese
    'ru', # russian
    'vi', # vietnamese
    'zh-cn',  # chinese
    'fa', # Persian
]
```

```
PIVOT_LANGUAGES = ['en', 'cs', 'de', 'es', 'fi', 'fr', 'hi', 'it', 'ja', 'pt', 'ru', 'vi

# generate some augmented examples for each training example
for example in tqdm(train_examples):
    train_examples_augmented.append(example) # always include the original example
    print(example)
    # WRITE YOUR CODE HERE!
    for target_language in PIVOT_LANGUAGES:
        pivot = translator.translate(example.text_a, src='en', dest=target_language).tex
        paraphrase = translator.translate(pivot, src=target_language, dest='en').text
        # the below line adds a single new augmented example to the dataset.
        # note that the guid should be a unique ID for this example, so you'll want to v
        # depending on how you generate your paraphrases
        train_examples_augmented.append(InputExample(guid=f"{example.guid}-aug-{target_l
                                                        text_a=paraphrase,
                                                        text_b=None,
                                                        label=example.label))
output_dir = f"./data/tiny{task_name}-bt"
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

with open(os.path.join(output_dir, "train.tsv"), "w") as writer:
    writer.write("sentence\tlabel\n")
    for example in train_examples_augmented:
        writer.write(f"{example.text_a}\t{example.label}\n")
tsv_to_csv(os.path.join(output_dir, "train.tsv"), os.path.join(output_dir, "train.csv"))

# Copy the original tinySST's dev set to the new directory
import shutil
shutil.copyfile(f"{data_dir}/dev.csv", f"{output_dir}/dev.csv")
```

```
0%|          | 0/20 [00:00<?, ?it/s]InputExample(guid='train-1', text_a='its unerring
```

```
# Examples
for i in range(len(PIVOT_LANGUAGES) + 1):
    print(train_examples_augmented[i].guid.ljust(20), train_examples_augmented[i].text_a
```

```
train-1              its unerring respect for them
train-1-aug-en       its unerring respect for them
train-1-aug-cs       his unfailing respect for them
train-1-aug-de       his unfailing respect for her
train-1-aug-es       his unfailing respect for them
train-1-aug-fi       its unmistakable respect for them
train-1-aug-fr       his unfailing respect for them
train-1-aug-hi       it's an absolute honor for him
train-1-aug-it       his unfailing respect for them
train-1-aug-ja       unwavering respect for them
train-1-aug-pt       his unfailing respect for them
train-1-aug-ru       his unmistakable respect for them
train-1-aug-vi       its unwavering respect for them
train-1-aug-zh-cn    its respect for them
train-1-aug-fa       Its unparalleled respect for them

'./data/tinySST-bt/dev.csv'
```

The below cell fine-tunes BERT `bert-base-cased` with the combined training data (real + synthetic training examples) and then evaluates the resulting model on tinySST's dev set. Note that it uses the default fine-tuning hyperparameters, not the improved ones that you found earlier. You should observe a significantly higher accuracy than 50% when you run this cell on the augmented data (our reference implementation reaches 64%). ***Do NOT modify any code in this cell!***

```
import timeit

start_time = timeit.default_timer()
task_name = "SST"
data_dir = f"./data/tiny{task_name}-bt"
model_name_or_path = "bert-base-cased"
model_cache_dir = os.path.join(pretrained_models_dir, model_name_or_path)
data_cache_dir = f"./data_cache/finetuning/tiny{task_name}-bt/"
output_dir = model_cache_dir

# Fine-tune and evaluate BERT with default hyperparameters using 4 random seeds
results = []
for seed in [1234, 2341, 3412, 4123]:
  output_dir = f"./output/tiny{task_name}-{seed}"
  config = dict(
      seed=seed,
      model_name_or_path=model_name_or_path,
      train_file="./data/tinySST-bt/train.csv",
      validation_file="./data/tinySST-bt/dev.csv",
      task_type="text_classification",
      do_train=True,
      do_eval=True,
```

```
        do_lower_case=True,
        data_dir=data_dir,
        max_seq_length=128,
        per_device_train_batch_size=32,
        learning_rate=2e-5,
        num_train_epochs=3.0,
        model_cache_dir=model_cache_dir,
        data_cache_dir=data_cache_dir,
        output_dir=output_dir,
        overwrite_output_dir=True,
        log_level='warning'
    )

    result = do_target_task_finetuning(config)
    results.append(result["eval_accuracy"])

results = np.array(results)
mean = np.mean(results)
std = np.std(results)

print(f"Accuracy on TinySST dev set: {mean} +/- {std}")
elapsed_time = timeit.default_timer() - start_time
print(f"Time elapsed: {elapsed_time} seconds")
```

```
WARNING:__main__:Process rank: -1, device: cuda:0, n_gpu: 1distributed training: False,
WARNING:datasets.builder:Using custom data configuration default-8f825126d2882a66
Downloading and preparing dataset csv/default to /root/.cache/huggingface/datasets/csv/
```

Downloading data files: 100% 2/2 [00:00<00:00, 70.79it/s]

Extracting data files: 100% 2/2 [00:00<00:00, 38.53it/s]

```
Dataset csv downloaded and prepared to /root/.cache/huggingface/datasets/csv/default-8f
```

100% 2/2 [00:00<00:00, 66.04it/s]

```
[WARNING|modeling_utils.py:1693] 2022-12-03 16:46:24,053 >> Some weights of the model c
- This IS expected if you are initializing BertForSequenceClassification from the check
- This IS NOT expected if you are initializing BertForSequenceClassification from the c
[WARNING|modeling_utils.py:1704] 2022-12-03 16:46:24,055 >> Some weights of BertForSequ
You should probably TRAIN this model on a down-stream task to be able to use it for pre
```

Running tokenizer on dataset: 100% 1/1 [00:00<00:00, 14.56ba/s]

Running tokenizer on dataset: 100% 1/1 [00:00<00:00, 5.86ba/s]

```
/usr/local/lib/python3.8/dist-packages/transformers/optimization.py:306: FutureWarning:
  warnings.warn(
```

[30/30 00:16, Epoch 3/3]

Step	Training Loss

```
***** train metrics *****
  epoch                    =        3.0
  total_flos               =    55134GF
  train_loss               =     0.4333
  train_runtime            = 0:00:16.73
  train_samples            =        300
  train_samples_per_second =     53.795
  train_steps_per_second   =      1.793
```

[109/109 00:06]

```
WARNING:__main__:Process rank: -1, device: cuda:0, n_gpu: 1distributed training: False,
***** eval metrics *****
  epoch                   =        3.0
  eval_accuracy           =     0.6927
  eval_loss               =     0.5946
  eval_runtime            = 0:00:06.39
  eval_samples            =        872
  eval_samples_per_second =    136.278
  eval_steps_per_second   =     17.035
WARNING:datasets.builder:Using custom data configuration default-8f825126d2882a66
WARNING:datasets.builder:Reusing dataset csv (/root/.cache/huggingface/datasets/csv/def
```

100% 2/2 [00:00<00:00, 68.05it/s]

```
[WARNING|modeling_utils.py:1693] 2022-12-03 16:47:02,443 >> Some weights of the model c
- This IS expected if you are initializing BertForSequenceClassification from the check
- This IS NOT expected if you are initializing BertForSequenceClassification from the c
[WARNING|modeling_utils.py:1704] 2022-12-03 16:47:02,445 >> Some weights of BertForSequ
You should probably TRAIN this model on a down-stream task to be able to use it for pre
```

Running tokenizer on dataset: 1/1 [00:00<00:00, 100% 8.41ba/s]

Running tokenizer on dataset: 1/1 [00:00<00:00,

100% 8.14ba/s]

[30/30 00:16, Epoch 3/3]

Step Training Loss

```
***** train metrics *****
  epoch                    =        3.0
  total_flos               =    55134GF
  train_loss               =     0.5299
  train_runtime            = 0:00:17.15
  train_samples            =        300
  train_samples_per_second =     52.469
  train_steps_per_second   =      1.749
```

[109/109 00:06]

```
WARNING:__main__:Process rank: -1, device: cuda:0, n_gpu: 1distributed training: False,
***** eval metrics *****
  epoch                   =        3.0
  eval_accuracy           =     0.6938
  eval_loss               =     0.6069
  eval_runtime            = 0:00:06.63
  eval_samples            =        872
  eval_samples_per_second =    131.479
  eval_steps_per_second   =     16.435
WARNING:datasets.builder:Using custom data configuration default-8f825126d2882a66
WARNING:datasets.builder:Reusing dataset csv (/root/.cache/huggingface/datasets/csv/def
```

100% 2/2 [00:00<00:00, 29.35it/s]

```
[WARNING|modeling_utils.py:1693] 2022-12-03 16:47:42,654 >> Some weights of the model c
- This IS expected if you are initializing BertForSequenceClassification from the check
- This IS NOT expected if you are initializing BertForSequenceClassification from the c
[WARNING|modeling_utils.py:1704] 2022-12-03 16:47:42,657 >> Some weights of BertForSequ
You should probably TRAIN this model on a down-stream task to be able to use it for pre
```

Running tokenizer on dataset: 1/1 [00:00<00:00,

100% 6.17ba/s]

Running tokenizer on dataset: 1/1 [00:00<00:00,

100% 5.98ba/s]

[30/30 00:18, Epoch 3/3]

Step Training Loss

```
***** train metrics *****
  epoch                    =        3.0
  total_flos               =    55134GF
  train_loss               =      0.442
  train_runtime            = 0:00:18.86
  train_samples            =        300
  train_samples_per_second =     47.704
  train_steps_per_second   =       1.59
```

[109/109 00:07]

```
WARNING:__main__:Process rank: -1, device: cuda:0, n_gpu: 1distributed training: False,
***** eval metrics *****
```

```
epoch                  =        3.0
eval_accuracy          =     0.6709
eval_loss              =     0.5946
eval_runtime           = 0:00:07.19
```

Question 3.2 (5 points)

Briefly explain your backtranslation strategy here. Why do you think it resulted in an improvement?

```
100%                                    2/2 [00:00<00:00, 30.40it/s]
```

Write your answer here! Please keep it brief (i.e., 2-3 sentences).

> First I observed the augmented examples for each pivot language:
>
> ```
> train-1              its unerring respect for them
> train-1-aug-en       its unerring respect for them
> train-1-aug-cs       his unfailing respect for them
> train-1-aug-de       his unfailing respect for her
> train-1-aug-es       his unfailing respect for them
> train-1-aug-fi       its unmistakable respect for them
> train-1-aug-fr       his unfailing respect for them
> train-1-aug-hi       it's an absolute honor for him
> train-1-aug-it       his unfailing respect for them
> train-1-aug-ja       unwavering respect for them
> train-1-aug-pt       his unfailing respect for them
> train-1-aug-ru       his unmistakable respect for them
> train-1-aug-vi       its unwavering respect for them
> train-1-aug-zh-cn    its respect for them
> train-1-aug-fa       Its unparalleled respect for them
> ```
>
> As seen above, each backtranslation has created a new instance for us with new words and even in some cases such as hindi (hi) a new sentence structure. These new variated samples will help our model to better generalize and learn the task instead of overfitting to specific words. (The sentiment prediction task is good for this appraoch as long as the backtranslated versions do not change the sentiment which is rare to happen.)
>
> Accuracy on TinySST dev set: 0.6751720309257507 +/- 0.020520400137613005

```
1
```

✓ 2m 43s completed at 8:18 PM