S&DS 617 Applied Machine Learning and Causal Inference Research Seminar: Assignment 1

Deadline

Assignment 1 is due Monday, February 24th at 1:30pm. Late work will not be accepted.

Submission

Submit your assignment as a .pdf on Gradescope. On Gradescope, there are 2 assignments, one where you will submit a pdf file and one where you will submit the corresponding .ipynb that generated it. Note: The problems in each homework assignment are numbered. When submitting the pdf on Gradescope, please select the correct pages that correspond to each problem.

To produce the .pdf, do the following to preserve the cell structure of the notebook:

- Go to "File" at the top-left of your Jupyter Notebook
- Under "Download as", select "HTML (.html)"
- After the .html has downloaded, open it and then select "File" and "Print"
- From the print window, select the option to save as a .pdf

Problem 1: Comparing BERT vs. GPT

a) In this assignment, we will compare BERT (Bidirectional Encoder Representations from Transformers) with GPT (Generative Pre-training Transformer). Provide detailed explanations of how the architecture, the type of attention mechanism employed, and the approach to tokenization in each model contribute to their respective capabilities and applications. Which model do you think will perform better at sentiment analysis and why?

- BERT Architecture:

BERT only takes the encoder part of the whole encoder-decoder tranformer model. It takes the bidirectioinal attention mechanism such that each token attends to all other tokens in the sentence. The training objective is to predict the masked word in a sentence and classify the true next sentence

- BERT tokenization:

BERT utilizes the WordPiece algorithm which begins with a base vocabulary of characters and iteratively merges the most frequent pairs of characters or subwords to create new tokens. This approach allows the tokenizer to break down words into smaller pieces, which is particularly useful for handling out-of-vocabulary (OOV) words and learning an embedding representation

- GPT Architecture:

GPT takes the decoder part of the whole encoder-decoder tranformer model and it runs autoregressively. GPT uses a causal attention mechanism such that future words are masked during the training and the training objective is to predict the next token

- GPT tokenization:

GPT utilizes Byte-Pair Encoding (BPE), which iteratively merges frequent character pairs. This appraoch helps generate coherent text that makes sense in a chat scenario

- Sentiment analysis:

I think BERT is likely to perform better at sentiment analysis because sentiment analysis requires understanding the entire sentence to interpret meaning. Sentiment analysis can be seen as a classification task so there is no need to generate logical sentence as output. By training on masked word objective and using bidirectional attention mechanism, BERT can learn a good embedding representation that encodes strong semantic understanding. This can better help understanding the entire sentence and interpret the mearning.

b) We will now perform sentiment analysis on the IMDb dataset ("https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"). This dataset contains movie reviews along with their associated binary sentiment polarity labels. Code has been provided to you below to train and evaluate BERT.

Run the below code to get the test accuracy. Then, modify the code to try getting a higher test accuracy (e.g., adjusting hyperparameters, further model tweaking, data augmentation, etc.). Specify what you modified.

In [1]:
```python
import requests
import tarfile
import os
import json
import re
import openai
from io import BytesIO
import pandas as pd
import torch
from transformers import BertTokenizer, BertForSequenceClassification
from transformers import Trainer, TrainingArguments
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from torch.utils.data import Dataset
```

Get Data

In [2]:
```
# URL of the IMDb dataset
url = "https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"

# Send a GET request to download the content of the dataset
response = requests.get(url)
response.raise_for_status()  # This will raise an exception if there was

# Open the downloaded content as a file-like object
file_like_object = BytesIO(response.content)

# Extract the tar.gz file
with tarfile.open(fileobj=file_like_object) as tar:
    tar.extractall(path=".")  # Extract to a directory named aclImdb in t

print("Dataset downloaded and extracted to './aclImdb")
```

```
Dataset downloaded and extracted to './aclImdb
```

In [2]:
```
def load_imdb_dataset(directory):
    reviews = []
    sentiments = []

    for sentiment in ["pos", "neg"]:
        dir_name = os.path.join(directory, sentiment)
        for filename in os.listdir(dir_name):
            if filename.endswith('.txt'):
                with open(os.path.join(dir_name, filename), encoding='utf
                    reviews.append(file.read())
                    sentiments.append(sentiment)

    return pd.DataFrame({'review': reviews, 'sentiment': sentiments})

# Load the training dataset
dataset_dir = 'aclImdb'
df_tr = load_imdb_dataset(os.path.join(dataset_dir, 'train'))

# Load the test dataset
df_te = load_imdb_dataset(os.path.join(dataset_dir, 'test'))

# Display the first few rows of the DataFrame
print(df_tr.head())
print(df_te.head())
```

```
                                              review sentiment
0  For a movie that gets no respect there sure ar...       pos
1  Bizarre horror movie filled with famous faces ...       pos
2  A solid, if unremarkable film. Matthau, as Ein...       pos
3  It's a strange feeling to sit alone in a theat...       pos
4  You probably all already know this by now, but...       pos
                                              review sentiment
0  Based on an actual story, John Boorman shows t...       pos
1  This is a gem. As a Film Four production - the...       pos
2  I really like this show. It has drama, romance...       pos
3  This is the best 3-D experience Disney has at ...       pos
4  Of the Korean movies I've seen, only three had...       pos
```

In [3]:
```
# Subsample train and test sets down (note: you may change the size of tr
df_tr = df_tr.sample(n=1000, random_state=928)
print(df_tr.shape) # check dimensions
df_te = df_te.sample(n=500, random_state=2755)
```

```
print(df_te.shape) # check dimensions
df_te.iloc[1, 0] # sample movie review
```

```
(1000, 2)
(500, 2)
```

Out[3]:
```
'Ok let\'s start with saying that when a dutch movie is bad, it\'s REALL
Y BAD. Rarely something with a little bit of quality comes along(Lek, Ka
rakter) here in holland but not often. Costa! is about 4 girls going to
Spain to go on vacation, party, get drunk, get laid (u know the drill).
It\'s also about the world of Clubbers or Proppers. Pro\'s who\'re tryin
g to lure the crowd into their club.<br /><br />I\'m not sure how long i
t took to write the script, but i suspect somewhere between 15 minutes a
nd 20 minutes because you\'re watching a bunch of random scenes for 90 m
inutes long. Nothing, and i mean nothing is believable in this movie. It
\'s almost too riduculous for words what happens with the storyline. Sud
denly the movie transforms into a sort of karate action thing. With a on
e-on-one fight with \'the bad guy in black\' and cliche car chase scenes
trough a watertank-car (can it be more cheesy). Also the words character
-development and casting are unfamiliar to the makers.<br /><br />After
having seen "Traffic" 3 days before this, i fell from sheer brilliance,
from a piece of art to this. This is film-making at it\'s saddest. And d
on\'t start about low budget. Because even with a low budget you could w
rite a better script. It almost seems that the film-makers were too busy
partying themselves to make a decent movie.<br /><br />Anyway the chicks
in the water at the end made it up a little bit, but for the rest of it,
don\'t waste your money on such garbage.'
```

In [4]:
```python
class IMDbDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encoding
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)
```

Train BERT (Note: this may take a considerable amount of time. You may modify the size of training if too computationally intensive)

In [5]:
```python
from transformers import TrainingArguments, Trainer, BertTokenizer, BertF

# Function to tokenize the dataset
def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=T

# Load the dataset (assuming df_tr is your loaded DataFrame)
texts = df_tr['review'].tolist()
labels = df_tr['sentiment'].apply(lambda x: 1 if x == 'pos' else 0).tolis

# Initialize the tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
# Tokenize the data
ml = 128
tokenized_dataset = tokenizer(texts, padding=True, truncation=True, max_l

# Splitting the dataset into training and validation sets
from sklearn.model_selection import train_test_split
train_texts, val_texts, train_masks, val_masks, train_labels, val_labels
    tokenized_dataset['input_ids'], tokenized_dataset['attention_mask'],
)

# Creating dataset objects for training and validation
class IMDbDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encoding
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = IMDbDataset({'input_ids': train_texts, 'attention_mask':
val_dataset = IMDbDataset({'input_ids': val_texts, 'attention_mask': val_

# Load BERT model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased'

# Define the training arguments
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    save_strategy="epoch",  # Ensure models are saved at each epoch
    evaluation_strategy="epoch",  # Evaluate at each epoch
    optim="adamw_torch",  # Use the recommended optimizer
)


# Define the Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset
)

# Train the model
trainer.train()
```

```
Some weights of BertForSequenceClassification were not initialized from th
e model checkpoint at bert-base-uncased and are newly initialized: ['class
ifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to u
se it for predictions and inference.
/Users/zhanghantao/anaconda3/lib/python3.11/site-packages/transformers/tra
ining_args.py:1594: FutureWarning: `evaluation_strategy` is deprecated and
will be removed in version 4.46 of 🤗 Transformers. Use `eval_strategy` in
stead
  warnings.warn(
```

[300/300 01:54, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.651600	0.614183
2	0.475200	0.394402
3	0.275800	0.629651

Out[5]:
```
TrainOutput(global_step=300, training_loss=0.5133776664733887, metrics=
{'train_runtime': 115.4692, 'train_samples_per_second': 20.785, 'train_s
teps_per_second': 2.598, 'total_flos': 157866633216000.0, 'train_loss':
0.5133776664733887, 'epoch': 3.0})
```

In [6]:
```python
# Evaluate the model on the validation set
predictions = trainer.predict(val_dataset)
val_accuracy = accuracy_score(val_labels, predictions.predictions.argmax(
print(f"Validation Accuracy: {val_accuracy}")
```

```
Validation Accuracy: 0.81
```

Evaluate model on test set

In [7]:
```python
test_texts = df_te['review'].tolist()
test_labels = df_te['sentiment'].apply(lambda x: 1 if x == 'pos' else 0).

# Tokenize the test data
test_encodings = tokenizer(test_texts, padding=True, truncation=True, max

class IMDbDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = {key: torch.tensor(val) for key, val in encoding
        self.labels = torch.tensor(labels)

    def __getitem__(self, idx):
        item = {key: val[idx].clone().detach() for key, val in self.encod
        item['labels'] = self.labels[idx]
        return item

    def __len__(self):
        return len(self.labels)

test_dataset = IMDbDataset(test_encodings, test_labels)

# Predictions
test_predictions = trainer.predict(test_dataset)
test_accuracy = accuracy_score(test_labels, test_predictions.predictions.
print(f"Test Accuracy: {test_accuracy}")
```

```
Test Accuracy: 0.826
```

Training modifications

- Data augmentation using synonym replacement and augment the training data size from 1000 to around 1500
- Increase training epoch from 3 to 5
- Specify a better learning rate
- Increase test accuracy from 0.82 to 0.86

In [29]:
```python
# !pip install nlpaug  # run in your notebook environment
# !pip install nltk

import random
import nlpaug.augmenter.word as naw
import nltk

# Download necessary NLTK resources
nltk.download('averaged_perceptron_tagger')
nltk.download('averaged_perceptron_tagger_eng')  # Fix the missing resour
nltk.download('wordnet')
nltk.download('omw-1.4')  # WordNet dependency for synonym replacement

synonym_aug = naw.SynonymAug(aug_src='wordnet')

augmented_texts = []
augmented_labels = []

for text, label in zip(df_tr['review'], df_tr['sentiment']):
    augmented_texts.append(text)
    augmented_labels.append(1 if label == 'pos' else 0)
    if random.random() < 0.5:
        aug_text = synonym_aug.augment(text)
        # print(aug_text)
        augmented_texts.append(aug_text[0])
        augmented_labels.append(1 if label == 'pos' else 0)
texts = augmented_texts
labels = augmented_labels
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/zhanghantao/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /Users/zhanghantao/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/zhanghantao/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     /Users/zhanghantao/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [37]:
```python
print(texts[1])
del trainer  # Delete the previous trainer instance
```

```
torch.cuda.empty_cache()  # Clear cached model
```

```
I sincerely consider this movie as another poor effort of Dominican Movie
Industry. The first 30 minutes of the movie are a little funny but then wh
en they switch their role in the society (men doing what women usually do
and women doing what men usually do) the movie falls. Becoming boring and
not funny at all. They let many things without explanation and the end of
the movie is predictable. I didn't like the way as a Roberto Angel played
his character and his little either. I went to the movies theater hoping t
o see a good work but I went out really disappointed.<br /><br />I don't r
ecommend this movie.
```

In [39]:
```
# Initialize tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize
ml = 256
tokenized_dataset = tokenizer(texts, padding=True, truncation=True, max_l

# Split into training and validation sets
train_texts, val_texts, train_masks, val_masks, train_labels, val_labels
    tokenized_dataset['input_ids'],
    tokenized_dataset['attention_mask'],
    labels,
    test_size=0.2,
    random_state=42
)

# Create PyTorch dataset
class IMDbDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encoding
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = IMDbDataset({'input_ids': train_texts, 'attention_mask':
val_dataset   = IMDbDataset({'input_ids': val_texts, 'attention_mask': va

# Load BERT model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased'

# Load BERT model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased'

# Define the training arguments
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    save_strategy="epoch",  # Ensure models are saved at each epoch
    evaluation_strategy="epoch",  # Evaluate at each epoch
    optim="adamw_torch",  # Use the recommended optimizer
    bf16=True
)


# Define the Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset
)

# Train the model
trainer.train()
```

```
Some weights of BertForSequenceClassification were not initialized from th
e model checkpoint at bert-base-uncased and are newly initialized: ['class
ifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to u
se it for predictions and inference.
Some weights of BertForSequenceClassification were not initialized from th
e model checkpoint at bert-base-uncased and are newly initialized: ['class
ifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to u
se it for predictions and inference.
```

[370/370 07:21, Epoch 5/5]

Epoch	Training Loss	Validation Loss
1	0.647900	0.600188
2	0.349200	0.265559
3	0.102800	0.199773
4	0.127700	0.208312
5	0.057600	0.306028

Out[39]:
```
TrainOutput(global_step=370, training_loss=0.3038809447473771, metrics=
{'train_runtime': 442.724, 'train_samples_per_second': 13.315, 'train_st
eps_per_second': 0.836, 'total_flos': 775519835673600.0, 'train_loss':
0.3038809447473771, 'epoch': 5.0})
```

In [41]:
```
# Evaluate the model on the validation set
predictions = trainer.predict(val_dataset)
val_accuracy = accuracy_score(val_labels, predictions.predictions.argmax(
print(f"Validation Accuracy: {val_accuracy}")
```

```
Validation Accuracy: 0.945
```

In [42]:
```
# Predictions
test_predictions = trainer.predict(test_dataset)
test_accuracy = accuracy_score(test_labels, test_predictions.predictions.
print(f"Test Accuracy: {test_accuracy}")
```

```
Test Accuracy: 0.86
```

c) Perform sentiment analysis using GPT-3.5-turbo, gpt-4o, o1-mini, and o3-mini and get the test accuracy. Evaluate their performance by comparing test accuracies. (If you get a rate limit error, just use 4o)

Note: DO NOT try to run advanced models on the entire test set initially. Be mindful of API usage limits and costs associated with the advanced models APIs. Start with a smaller subset of your test set to ensure your implementation is correct before scaling up.

In [5]:
```
from transformers import TrainingArguments, Trainer, BertTokenizer, BertF

# Function to tokenize the dataset
def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=T

# Load the dataset (assuming df_tr is your loaded DataFrame)
texts = df_tr['review'].tolist()
labels = df_tr['sentiment'].apply(lambda x: 1 if x == 'pos' else 0).tolis

# Initialize the tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize the data
ml = 128
tokenized_dataset = tokenizer(texts, padding=True, truncation=True, max_l

# Splitting the dataset into training and validation sets
from sklearn.model_selection import train_test_split
train_texts, val_texts, train_masks, val_masks, train_labels, val_labels
    tokenized_dataset['input_ids'], tokenized_dataset['attention_mask'],
)

# Creating dataset objects for training and validation
class IMDbDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encoding
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = IMDbDataset({'input_ids': train_texts, 'attention_mask':
val_dataset = IMDbDataset({'input_ids': val_texts, 'attention_mask': val_


test_texts = df_te['review'].tolist()
test_labels = df_te['sentiment'].apply(lambda x: 1 if x == 'pos' else 0).

# Tokenize the test data
test_encodings = tokenizer(test_texts, padding=True, truncation=True, max

class IMDbDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = {key: torch.tensor(val) for key, val in encoding
        self.labels = torch.tensor(labels)

    def __getitem__(self, idx):
        item = {key: val[idx].clone().detach() for key, val in self.encod
        item['labels'] = self.labels[idx]
        return item

    def __len__(self):
        return len(self.labels)

test_dataset = IMDbDataset(test_encodings, test_labels)
```

In [6]:

```
print(len(test_texts))
print(type(test_labels[0]))

from dotenv import load_dotenv
import json

# Load environment variables from the .env file
load_dotenv()

# Access the OpenAI API key
openai_api_key = os.getenv("OPENAI_API_KEY")

# Use the API key
if openai_api_key:
    print("OpenAI API Key loaded successfully!")
else:
    print("OpenAI API Key not found. Please check your .env file.")

client = openai.OpenAI(api_key=os.environ.get("OPENAI_API_KEY"))

def test_sentiment(model_name):
    result = []

    initial_prompt = """You will be given a paragraph of movie review tha
                        If it is positive, respond 1. If it is negative,

    for i in range(len(test_texts)):
        # Make a chat completion request
        if i % 50 == 0:
            print(i)
        chat_completion = client.chat.completions.create(
            messages=[
                {
                    "role": "user",
                    "content": initial_prompt,
                },
                {
                    "role": "user",
                    "content": test_texts[i],
                }
            ],
            model=model_name,  # Specify the model
        )

        # Access the response content using attributes
        response_content = chat_completion.choices[0].message.content
```

```
            try:
                result.append(int(response_content))
            except:
                print(response_content)
                result.append(response_content)

        # print(result)
        with open(f"{model_name}.json", "w") as f:
            json.dump(result, f)
        # print(test_labels[1:11])
```

```
500
<class 'int'>
OpenAI API Key loaded successfully!
```

In [7]:

```
model_list = ["gpt-3.5-turbo", "gpt-4o"]
model_list = ["o1-mini", "o3-mini"]
for model in model_list:
    test_sentiment(model)
```

```
0
50
100
150
200
250
300
350
400
450
0
50
100
150
200
250
300
350
400
450
```

In [9]:

```
model_list = ["gpt-3.5-turbo", "gpt-4o", "o1-mini", "o3-mini"]
for model in model_list:
    with open(f"{model}.json", 'r') as file:
        data = json.load(file)
        test_accuracy = accuracy_score(test_labels, data)
        print(f"Test Accuracy for {model}: {test_accuracy}")
```

```
Test Accuracy for gpt-3.5-turbo: 0.952
Test Accuracy for gpt-4o: 0.95
Test Accuracy for o1-mini: 0.952
Test Accuracy for o3-mini: 0.958
```

- I found that all GPT models reached over 0.95 accuracies which is higher than the BERT model we trained and o3-mini achieved the best performance
- This is because GPT models are much larger than BERT, proving the effectiveness of scaling law

d) For the task of language translation, do you expect BERT or GPT to perform better? Explain why in detail. Additionally, discuss the primary challenges associated with implementing each model for translation tasks.

- I expect GPT to perform better because unlike classification, translation task requires to generate sentences that can be understood by human-beings. GPT has a causal attention mechanism which allows it to generate coherant sentences and if we prompt the model with untranslated sentences, the self attention mechanism can address long-range dependencies without losing memories while generating translation autoregressively.
- Challenges for BERT: since it has no decoder, it would be hard for us to design a decoder to turn its output into translation. Also, it is not trained using causal attention autoregressively, it struggles to generate understandable sentences even though it may learn a good embeddings.
- Challenges for GPT: because of the autoregressive nature, mistakes in early stages will propagate to later stages, causing inaccuracies and hallucinations.