Lab 2: Fine-tuning BERT To Perform Causal Common Sense Reasoning

Due: May 26, 2024

Welcome to the Assignment 2 of our course on Natural Language Processing. Similar to Lab 2, we will again be working on a common sense reasoning task and fine-tuning BERT to solve the same. Specifically, we will be looking at Choice Of Plausible Alternatives (COPA) dataset which was created to access common-sense causal reasoning of NLP models. This assignment should flow naturally from Lab 2, and we shall see with minimal changes we will be able to adapt what we learned for SocialIQA task on COPA.

Suggested Reading:

- Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*
- [Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Social IQa: Commonsense Reasoning about Social Interactions. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 4463–4473, Hong Kong, China. Association for Computational Linguistics.] (https://arxiv.org/pdf/1810.04805.pdf)

In [ ]:
```
# from google.colab import drive
# drive.mount('/content/gdrive')
# copa_data_dir = "gdrive/MyDrive/PlakshaNLP2024/Assignment2/data/copa/"
copa_data_dir = "./copa/"
```

In [ ]:
```
# Install required libraries
# If using Colab, DO NOT INSTALL ANYTHING!
# !pip install numpy
# !pip install pandas
# !pip install torch
# !pip install tqdm
# !pip install matplotlib
# !pip install transformers
# !pip install scikit-learn
# !pip install tqdm
```

In [ ]:
```
# We start by importing libraries that we will be making use of in the as
import os
from functools import partial
import json
import xml.etree.ElementTree as ET
from pprint import pprint
import numpy as np
import pandas as pd
import torch
```

```python
import torch.nn as nn
from torch.optim import Adam
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
import copy
from tqdm.notebook import tqdm

from transformers.utils import logging
logging.set_verbosity(40) # to avoid warnings from transformers
```

COPA Dataset

We start by discussing the dataset that we will making use of in today's Assignment. As described above, the COPA evaluation provides researchers with a tool for assessing progress in open-domain commonsense causal reasoning. COPA consists of 1000 questions, split equally into development and test sets of 500 questions each. Each question is composed of a premise and two alternatives, where the task is to select the alternative that more plausibly has a causal relation with the premise. Some examples from the dataset include:

```
Premise: The man broke his toe. What was the CAUSE of
this?
Alternative 1: He got a hole in his sock.
Alternative 2: He dropped a hammer on his foot.

Premise: I tipped the bottle. What happened as a RESULT?
Alternative 1: The liquid in the bottle froze.
Alternative 2: The liquid in the bottle poured out.

Premise: I knocked on my neighbor's door. What happened as
a RESULT?
Alternative 1: My neighbor invited me in.
Alternative 2: My neighbor left his house.
```

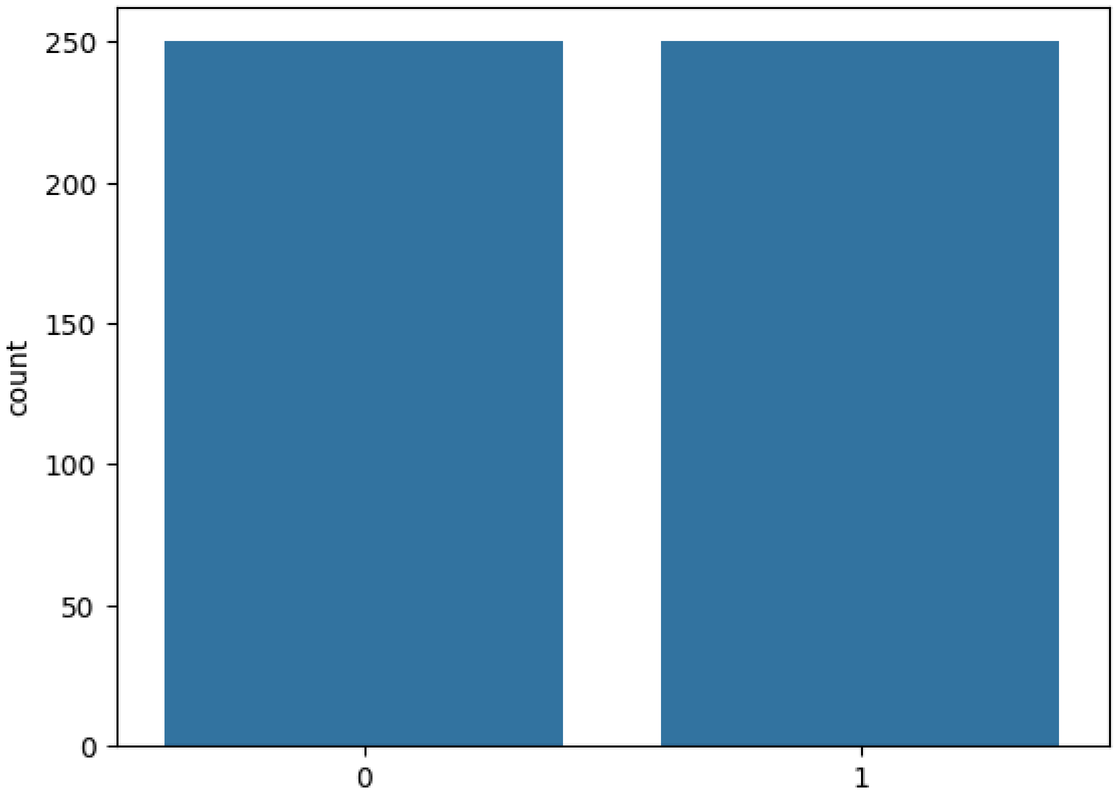
Below we load the dataset in memory. Since there is no seperate training set, we use dev set for training the model and evaluate on test set.

In [ ]:
```python
def parse_copa_dataset(split="test"):
    tree = ET.parse(f"{copa_data_dir}/copa-{split}.xml")
    root = tree.getroot()
    items = root.findall("item")

    data = []
    labels = []
    for item in items:
        data.append(
            {
                "question": item.get("asks-for"),
                "premise": item.find("p").text,
                "choice1": item.find("a1").text,
                "choice2": item.find("a2").text,
            }
        )
        labels.append(int(item.get("most-plausible-alternative")) - 1)

    return data, labels

train_data, train_labels = parse_copa_dataset("dev")
test_data, test_labels = parse_copa_dataset("test")

print(f"Number of Training Examples: {len(train_data)}")
print(f"Number of Test Examples: {len(test_data)}")
```

```
Number of Training Examples: 500
Number of Test Examples: 500
```

In [ ]:
```
sns.countplot(x = train_labels)
```

Out[ ]:
```
<Axes: ylabel='count'>
```



In [ ]:
```
# View a sample of the dataset
print("Example from dataset")
pprint(train_data[100], sort_dicts=False, indent=4)
print(f"Label: {train_labels[100]}")
```

```
Example from dataset
{   'question': 'effect',
    'premise': 'The teacher took roll.',
    'choice1': 'She identified the students that were absent.',
    'choice2': 'She gave her students a pop quiz.'}
Label: 0
```

As you can see, the dataset is pretty much very similar as SocialIQA, with the main difference being that we have two answer choices instead of three. Hence, we just need to concatenate choice1 and choice2, seperately with premise and question this time.

```
In [ ]: # Import the BertTokenizer from the library
        from transformers import BertTokenizer

        # Load a pre-trained BERT Tokenizer
        bert_tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

        example = train_data[100]
        premise = example["premise"]
        question = example["question"]
        choice1 = example["choice1"]
        choice2 = example["choice2"]

        pqc1 = premise + bert_tokenizer.sep_token + question + bert_tokenizer.sep
        pqc2 = premise + bert_tokenizer.sep_token + question + bert_tokenizer.sep

        print(pqc1)
        print(pqc2)

        tokenized_pqc1 = bert_tokenizer(pqc1)
        tokenized_pqc2 = bert_tokenizer(pqc2)
```

```
/Users/rajatjacob/.pyenv/versions/3.11.4/envs/nlp/lib/python3.11/site-pack
ages/huggingface_hub/file_download.py:1132: FutureWarning: `resume_downloa
d` is deprecated and will be removed in version 1.0.0. Downloads always re
sume when possible. If you want to force a new download, use `force_downlo
ad=True`.
  warnings.warn(
The teacher took roll.[SEP]effect[SEP]She identified the students that wer
e absent.
The teacher took roll.[SEP]effect[SEP]She gave her students a pop quiz.
```

Task 1: Setting up Custom Datasets and Dataloaders (4 Marks)

Task 1.1: Custom Dataset Class (2 Marks)

Similar to Lab 2, you will start by implementing a custom Dataset class for COPA dataset. The only difference will be that `__getitem__` should return tokenized outputs corresponding to the two choices choice1 and choice2, instead of three like in the case of SocialIQA dataset.

```
In [ ]: from torch.utils.data import Dataset, DataLoader


        class COPABertDataset(Dataset):

            def __init__(self, data, labels, bert_variant="bert-base-uncased"):
                """
                Constructor for the `COPABertDataset` class. Stores the `data` an
                other methods. Also initializes the tokenizer

                Inputs:
                    - data (list) : A list COPA dataset examples
                    - labels (list): A list of answer labels corresponding to eac
                    - bert_variant (str): A string indicating the variant of BERT
                """
                self.data = data
                self.labels = labels
                self.tokenizer = BertTokenizer.from_pretrained(bert_variant)

            def __len__(self):
                """
                Returns the length of the dataset
                """
                length = len(self.data)

                return length

            def __getitem__(self, idx):
                """
                Returns the training example corresponding to COPA example presen

                Inputs:
                    - idx (int): Index corresponding to the dataset example to be

                Returns:
                    - tokenized_input_dict (dict(str, dict)): A dictionary corres
                    - label (int): Answer label for the corresponding sentence. 0

                Example Output:
                    - tokenized_input_dict: {
                        "choice1": {'input_ids': [101, 5207, 1005, 1055, 3899, 21
                        "choice2": {'input_ids': [101, 5207, 1005, 1055, 3899, 21
                    }
                    - label: 0

                """

                record = self.data[idx]
                sentence1 = self.tokenizer.sep_token.join([record[k] for k in ('p
                sentence2 = self.tokenizer.sep_token.join([record[k] for k in ('p
                tokenized_input_dict = {
                    "choice1": self.tokenizer(sentence1),
                    "choice2": self.tokenizer(sentence2),
                }
                label = self.labels[idx]

                return tokenized_input_dict, label
```

In [ ]:
```
print("Running Sample Test Cases")

sample_dataset = COPABertDataset(train_data[:2], train_labels[:2], bert_v

print(f"Sample Test Case 1: Checking if `__len__` is implemented correctl
dataset_len= len(sample_dataset)
expected_len = 2
print(f"Dataset Length: {dataset_len}")
print(f"Expected Length: {expected_len}")
assert len(sample_dataset) == expected_len
print("Sample Test Case Passed!")
print("****************************************\n")

print(f"Sample Test Case 2: Checking if `__getitem__` is implemented corr
sample_idx = 0
tokenized_input_dict, label = sample_dataset.__getitem__(sample_idx)
```

```
expected_tokenized_input_dict = {'choice1': {'input_ids': [101, 2026, 230
 'choice2': {'input_ids': [101, 2026, 2303, 3459, 1037, 5192, 2058, 1996,
expected_label = 0
print(f"tokenized_input_dict:\n {tokenized_input_dict}")
print(f"Expected tokenized_input_dict:\n {expected_tokenized_input_dict}"
assert (expected_tokenized_input_dict == tokenized_input_dict)

print(f"label:\n {label}")
print(f"Expected label:\n {expected_label}")
assert expected_label == label

print("Sample Test Case Passed!")
print("****************************************\n")


print(f"Sample Test Case 3: Checking if `__getitem__` is implemented corr
sample_idx = 1
tokenized_input_dict, label = sample_dataset.__getitem__(sample_idx)
expected_tokenized_input_dict = {'choice1': {'input_ids': [101, 1996, 245
 'choice2': {'input_ids': [101, 1996, 2450, 25775, 2014, 2767, 1005, 1055

expected_label = 0
print(f"tokenized_input_dict:\n {tokenized_input_dict}")
print(f"Expected tokenized_input_dict:\n {expected_tokenized_input_dict}"
assert (expected_tokenized_input_dict == tokenized_input_dict)

print(f"label:\n {label}")
print(f"Expected label:\n {expected_label}")
assert expected_label == label

print("Sample Test Case Passed!")
print("****************************************\n")

print(f"Sample Test Case 4: Checking if `__getitem__` is implemented corr
sample_dataset = COPABertDataset(train_data[:2], train_labels[:2], bert_v
sample_idx = 0
tokenized_input_dict, label = sample_dataset.__getitem__(sample_idx)
expected_tokenized_input_dict = {'choice1': {'input_ids': [101, 1422, 140
 'choice2': {'input_ids': [101, 1422, 1404, 2641, 170, 6464, 1166, 1103,
expected_label = 0
print(f"tokenized_input_dict:\n {tokenized_input_dict}")
print(f"Expected tokenized_input_dict:\n {expected_tokenized_input_dict}"
assert (expected_tokenized_input_dict == tokenized_input_dict)

print(f"label:\n {label}")
print(f"Expected label:\n {expected_label}")
assert expected_label == label

print("Sample Test Case Passed!")
print("****************************************\n")
```

```
Running Sample Test Cases
Sample Test Case 1: Checking if `__len__` is implemented correctly
Dataset Length: 2
Expected Length: 2
Sample Test Case Passed!
****************************************

Sample Test Case 2: Checking if `__getitem__` is implemented correctly for
`idx= 0`
tokenized_input_dict:
 {'choice1': {'input_ids': [101, 2026, 2303, 3459, 1037, 5192, 2058, 1996,
5568, 1012, 102, 3426, 102, 1996, 3103, 2001, 4803, 1012, 102], 'token_typ
e_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'attent
ion_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}, 'ch
oice2': {'input_ids': [101, 2026, 2303, 3459, 1037, 5192, 2058, 1996, 556
8, 1012, 102, 3426, 102, 1996, 5568, 2001, 3013, 1012, 102], 'token_type_i
ds': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'attention
_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}}
Expected tokenized_input_dict:
 {'choice1': {'input_ids': [101, 2026, 2303, 3459, 1037, 5192, 2058, 1996,
5568, 1012, 102, 3426, 102, 1996, 3103, 2001, 4803, 1012, 102], 'token_typ
e_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'attent
ion_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}, 'ch
oice2': {'input_ids': [101, 2026, 2303, 3459, 1037, 5192, 2058, 1996, 556
8, 1012, 102, 3426, 102, 1996, 5568, 2001, 3013, 1012, 102], 'token_type_i
ds': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'attention
_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}}
label:
 0
Expected label:
 0
Sample Test Case Passed!
****************************************

Sample Test Case 3: Checking if `__getitem__` is implemented correctly for
`idx= 1`
tokenized_input_dict:
 {'choice1': {'input_ids': [101, 1996, 2450, 25775, 2014, 2767, 1005, 105
5, 3697, 5248, 1012, 102, 3426, 102, 1996, 2450, 2354, 2014, 2767, 2001, 2
183, 2083, 1037, 2524, 2051, 1012, 102], 'token_type_ids': [0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'attent
ion_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1]}, 'choice2': {'input_ids': [101, 1996, 2450, 25775, 201
4, 2767, 1005, 1055, 3697, 5248, 1012, 102, 3426, 102, 1996, 2450, 2371, 2
008, 2014, 2767, 2165, 5056, 1997, 2014, 16056, 1012, 102], 'token_type_id
s': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}}
Expected tokenized_input_dict:
 {'choice1': {'input_ids': [101, 1996, 2450, 25775, 2014, 2767, 1005, 105
5, 3697, 5248, 1012, 102, 3426, 102, 1996, 2450, 2354, 2014, 2767, 2001, 2
183, 2083, 1037, 2524, 2051, 1012, 102], 'token_type_ids': [0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'attent
ion_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1]}, 'choice2': {'input_ids': [101, 1996, 2450, 25775, 201
4, 2767, 1005, 1055, 3697, 5248, 1012, 102, 3426, 102, 1996, 2450, 2371, 2
008, 2014, 2767, 2165, 5056, 1997, 2014, 16056, 1012, 102], 'token_type_id
s': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}}
```

```
label:
 0
Expected label:
 0
Sample Test Case Passed!
****************************************

Sample Test Case 4: Checking if `__getitem__` is implemented correctly for
`idx= 0` for a different bert-variant
tokenized_input_dict:
 {'choice1': {'input_ids': [101, 1422, 1404, 2641, 170, 6464, 1166, 1103,
5282, 119, 102, 2612, 102, 1109, 3336, 1108, 4703, 119, 102], 'token_type_
ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'attentio
n_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}, 'choi
ce2': {'input_ids': [101, 1422, 1404, 2641, 170, 6464, 1166, 1103, 5282, 1
19, 102, 2612, 102, 1109, 5282, 1108, 2195, 119, 102], 'token_type_ids':
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'attention_mas
k': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}}
Expected tokenized_input_dict:
 {'choice1': {'input_ids': [101, 1422, 1404, 2641, 170, 6464, 1166, 1103,
5282, 119, 102, 2612, 102, 1109, 3336, 1108, 4703, 119, 102], 'token_type_
ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'attentio
n_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}, 'choi
ce2': {'input_ids': [101, 1422, 1404, 2641, 170, 6464, 1166, 1103, 5282, 1
19, 102, 2612, 102, 1109, 5282, 1108, 2195, 119, 102], 'token_type_ids':
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'attention_mas
k': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}}
label:
 0
Expected label:
 0
Sample Test Case Passed!
****************************************
```

We can now create Dataset instances for both training and dev datasets

In [ ]:
```python
train_dataset = COPABertDataset(train_data, train_labels, bert_variant="b
test_dataset = COPABertDataset(test_data, test_labels, bert_variant="bert
```

Task 1.2: Custom `collate_fn` Class (2 Marks)

Similar to Lab 2, you will now implement a custom `colate_fn` for the `COPABertDataset` class. Remember, a `colate_fn` informs a dataloader on how to construct batches from a list of sequences in a batch. Implement `collate_fn` that takes a batch which is a list of tuples of the form `(tokenized_input_dict, label)` and constructs the batches of following:

- `input_ids_dict` : A dictionary containing batch of `input_ids` tensors corresponding to both choice1 and choice2. You will need to do padding here,
- `attn_mask_dict` : A dictionary containing batch of `attention_mask` tensors corresponding to both choice1 and choice2. You will need to do padding here,

- `labels` : A tensor of shape [batch_size,] containing the labels for each example in the batch

In [ ]:
```
def collate_fn(tokenizer, batch):
    """
    Collate function to be used when creating a data loader for the COPA
    :param tokenizer: The tokenizer to be used to tokenize the inputs.
    :param batch: A list of tuples of the form (tokenized_input_dict, lab
    :return:
        - A tuple of the form (input_ids_dict, attn_mask_dict, labels) as
    """

    tokenized_batch_dict = {
        "choice1": [],
        "choice2": []
    }

    labels_batch = []
    for tokenized_inputs_dict, label in batch:
        for choiceKey in ('choice1', 'choice2'):
            tokenized_batch_dict[choiceKey].append(tokenized_inputs_dict[
        labels_batch.append(label)

    for choiceKey in ('choice1', 'choice2'):
            tokenized_batch_dict[choiceKey] = tokenizer.pad(tokenized_bat

    labels_batch = torch.tensor(labels_batch)

    return (
        {k: v['input_ids'] for k, v in tokenized_batch_dict.items()},
        {k: v['attention_mask'] for k, v in tokenized_batch_dict.items()}
        labels_batch
    )
```

In [ ]:
```
print("Running Sample Test Cases")

sample_dataset = COPABertDataset(train_data[:2], train_labels[:2], bert_v
batch = [sample_dataset.__getitem__(0), sample_dataset.__getitem__(1)]


print(f"Sample Test Case 1: Checking if the return output of `collate_fn`
colated_batch = collate_fn(bert_tokenizer, batch)
print(f"Output type: {type(colated_batch)}")
assert (type(colated_batch) == tuple)
print(f"Tuple Length: {len(colated_batch)}")
assert (len(colated_batch) == 3)
print(f"Tuple 0th element type: {type(colated_batch[0])}")
assert (type(colated_batch[0]) == dict)
print(f"Tuple 1st element type: {type(colated_batch[1])}")
assert (type(colated_batch[1]) == dict)
print(f"Tuple 2nd element type: {type(colated_batch[2])}")
assert (type(colated_batch[2]) == torch.Tensor)
print("Sample Test Case Passed!")
print("****************************************\n")

print(f"Sample Test Case 2: Checking if the return output of `collate_fn`
print(f"Tuple 0th element shape for choice1: {colated_batch[0]['choice1']
assert (colated_batch[0]['choice1'].shape == torch.Size([2, 27]))
print(f"Tuple 0th element shape for choice2: {colated_batch[0]['choice2']
assert (colated_batch[0]['choice2'].shape == torch.Size([2, 27]))

print(f"Tuple 1st element shape for choice1: {colated_batch[1]['choice1']
assert (colated_batch[1]['choice1'].shape == torch.Size([2, 27]))
print(f"Tuple 1st element shape for choice2: {colated_batch[1]['choice2']
assert (colated_batch[1]['choice2'].shape == torch.Size([2, 27]))

print(f"Tuple 2nd element shape: {colated_batch[2].shape}")
assert (colated_batch[2].shape == torch.Size([2]))

print("Sample Test Case Passed!")
print("****************************************\n")

print(f"Sample Test Case 3: Checking if the return output of `collate_fn`
tup0_choice1_expected = torch.tensor([[  101,  2026,  2303,  3459,  1037,
           102,  3426,   102,  1996,  3103,  2001,  4803,  1012,   102,
             0,     0,     0,     0,     0,     0,     0],
        [  101,  1996,  2450, 25775,  2014,  2767,  1005,  1055,  3697,
          1012,   102,  3426,   102,  1996,  2450,  2354,  2014,  2767,
          2183,  2083,  1037,  2524,  2051,  1012,   102]])
print(f"Tuple 0th element predicted values for choice1: {colated_batch[0]
print(f"Tuple 0th element expected values for choice1: {tup0_choice1_expe
assert (torch.allclose(colated_batch[0]['choice1'], tup0_choice1_expected
Running Sample Test Cases
Sample Test Case 1: Checking if the return output of `collate_fn` is of th
e correct type
Output type: <class 'tuple'>
Tuple Length: 3
Tuple 0th element type: <class 'dict'>
Tuple 1st element type: <class 'dict'>
Tuple 2nd element type: <class 'torch.Tensor'>
Sample Test Case Passed!
****************************************

Sample Test Case 2: Checking if the return output of `collate_fn` is of th
e correct shape
Tuple 0th element shape for choice1: torch.Size([2, 27])
Tuple 0th element shape for choice2: torch.Size([2, 27])
Tuple 1st element shape for choice1: torch.Size([2, 27])
Tuple 1st element shape for choice2: torch.Size([2, 27])
Tuple 2nd element shape: torch.Size([2])
Sample Test Case Passed!
****************************************

Sample Test Case 3: Checking if the return output of `collate_fn` is of th
e correct values
Tuple 0th element predicted values for choice1: tensor([[  101,  2026,  23
03,  3459,  1037,  5192,  2058,  1996,  5568,  1012,
           102,  3426,   102,  1996,  3103,  2001,  4803,  1012,   102,
0,
             0,     0,     0,     0,     0,     0,     0],
        [  101,  1996,  2450, 25775,  2014,  2767,  1005,  1055,  3697,  5
248,
          1012,   102,  3426,   102,  1996,  2450,  2354,  2014,  2767,  2
001,
          2183,  2083,  1037,  2524,  2051,  1012,   102]])
Tuple 0th element expected values for choice1: tensor([[  101,  2026,  230
3,  3459,  1037,  5192,  2058,  1996,  5568,  1012,
           102,  3426,   102,  1996,  3103,  2001,  4803,  1012,   102,
0,
             0,     0,     0,     0,     0,     0,     0],
        [  101,  1996,  2450, 25775,  2014,  2767,  1005,  1055,  3697,  5
248,
          1012,   102,  3426,   102,  1996,  2450,  2354,  2014,  2767,  2
001,
          2183,  2083,  1037,  2524,  2051,  1012,   102]])
```

Now that we have defined the collate_fn, lets create the dataloaders. It is common to use smaller batch size while fine-tuning these big models, as they occupy quite a lot of memory.

In [ ]:
```python
batch_size = 16
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=T
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=Tru
```

In [ ]:
```python
batch_input_ids, batch_attn_mask, batch_labels = next(iter(train_loader))
print(f"batch_input_ids:\n {batch_input_ids}")
print(f"batch_attn_mask:\n {batch_attn_mask}")
print(f"batch_labels:\n {batch_labels}")
```

```
batch_input_ids:
 {'choice1': tensor([[  101,  1996,  2450,  2359,  2000,  2022,  1037,  34
60,  1012,   102,
          3466,   102,  2016,  4716,  1996,  2902,  1012,   102,     0,
0,
             0,     0,     0,     0,     0],
        [  101,  1996,  2158,  2001,  2439,  1012,   102,  3466,   102,  2
002,
          2356,  2005,  7826,  1012,   102,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0],
        [  101,  1996,  2155, 14475,  1996,  3347,  4783,  4226,  1012,
102,
          3426,   102,  1996, 19939,  2170,  2005, 12642,  1012,   102,
0,
             0,     0,     0,     0,     0],
        [  101,  1996,  3076,  2001,  1999,  1037,  5481,  2000,  2131,  2
000,
          2082,  2006,  2051,  1012,   102,  3466,   102,  2002,  2187,  2
010,
          8775,  2012,  2188,  1012,   102],
        [  101,  1996,  2269,  2404,  2010,  2684,  1999,  2014, 27244,  2
121,
          1012,   102,  3426,   102,  2016,  4342,  2000,  3328,  1012,
102,
             0,     0,     0,     0,     0],
        [  101,  1996, 13170, 11041,  1999,  2049,  5806,  1012,   102,  3
426,
           102,  2009,  8823,  1037,  3869,  1012,   102,     0,     0,
0,
             0,     0,     0,     0,     0],
        [  101,  1996,  2158, 23133,  1996,  4169,  1012,   102,  3466,
102,
          2002,  2371,  1999, 15180,  1012,   102,     0,     0,     0,
0,
             0,     0,     0,     0,     0],
        [  101,  1996,  3677,  2366,  4596,  2000,  2010,  6368,  1012,
102,
          3466,   102,  2010,  6368,  2020, 24665, 20113,  1012,   102,
0,
             0,     0,     0,     0,     0],
        [  101,  1045,  2985,  1996,  2154,  2012,  1996,  4770,  1012,
102,
          3466,   102,  1045, 11867, 27361,  2026, 10792,  1012,   102,
0,
             0,     0,     0,     0,     0],
        [  101,  1996, 19785,  2766,  1996, 20777,  1012,   102,  3466,
102,
          1996, 20777,  2234,  2041,  1997,  1996,  5800,  1012,   102,
0,
             0,     0,     0,     0,     0],
        [  101,  1996,  6501,  4370,  3147,  1012,   102,  3426,   102,  2
009,
          2001, 19351, 28405,  1012,   102,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0],
        [  101,  1996,  2158,  4711,  2041,  1996,  7852,  1012,   102,  3
426,
           102,  2009,  2001,  4840,  1012,   102,     0,     0,     0,
0,
              0,     0,     0,     0,     0],
            [  101,  1045,  8461,  1996,  8079,  1999,  1996,  5800,  1012,  
102,
             3466,   102,  1045, 27129,  1996,  5800,  1012,   102,     0,     
0,
                0,     0,     0,     0,     0],
            [  101,  1996,  5776,  2001,  2139, 10536,  7265,  3064,  1012,  
102,
             3466,   102,  1996,  6821,  7718,  2010, 22259,  2229,  1012,  
102,
                0,     0,     0,     0,     0],
            [  101,  1996,  2611,  4375,  2091,  2014,  4253,  2000,  2014,  3
920,
             2905,  1012,   102,  3426,   102,  1996,  4253,  2020, 11937, 14
795,
             1012,   102,     0,     0,     0],
            [  101,  1996,  2611,  2371, 14849,  1012,   102,  3466,   102,  2
016,
             2439,  2014,  5703,  1012,   102,     0,     0,     0,     0,     
0,
                0,     0,     0,     0,     0]]), 'choice2': tensor([[  101,  
1996,  2450,  2359,  2000,  2022,  1037,  3460,  1012,   102,
             3466,   102,  2016,  2253,  2000,  2966,  2082,  1012,   102,     
0,
                0,     0,     0,     0,     0],
            [  101,  1996,  2158,  2001,  2439,  1012,   102,  3466,   102,  2
002,
             3881,  1037,  4949,  1012,   102,     0,     0,     0,     0,     
0,
                0,     0,     0,     0,     0],
            [  101,  1996,  2155, 14475,  1996,  3347,  4783,  4226,  1012,  
102,
             3426,   102,  2009,  2001,  1037,  6209,  5353,  1012,   102,     
0,
                0,     0,     0,     0,     0],
            [  101,  1996,  3076,  2001,  1999,  1037,  5481,  2000,  2131,  2
000,
             2082,  2006,  2051,  1012,   102,  3466,   102,  2002,  2716,  2
010,
             6265,  2000,  2082,  1012,   102],
            [  101,  1996,  2269,  2404,  2010,  2684,  1999,  2014, 27244,  2
121,
             1012,   102,  3426,   102,  2002,  2001,  5458,  1997,  4755,  2
014,
             1012,   102,     0,     0,     0],
            [  101,  1996, 13170, 11041,  1999,  2049,  5806,  1012,   102,  3
426,
              102,  2009, 11156,  1037, 15267,  1012,   102,     0,     0,     
0,
                0,     0,     0,     0,     0],
            [  101,  1996,  2158, 23133,  1996,  4169,  1012,   102,  3466,  
102,
             2002,  7798,  1012,   102,     0,     0,     0,     0,     0,     
0,
                0,     0,     0,     0,     0],
            [  101,  1996,  3677,  2366,  4596,  2000,  2010,  6368,  1012,  
102,
             3466,   102,  2010,  6368,  2253,  7501,  1012,   102,     0,     
0,
                0,     0,     0,     0,     0],
        [  101,  1045,  2985,  1996,  2154,  2012,  1996,  4770,  1012,
102,
          3466,   102,  2026,  2227,  2288,  3103,  8022,  2098,  1012,
102,
             0,     0,     0,     0,     0],
        [  101,  1996, 19785,  2766,  1996, 20777,  1012,   102,  3466,
102,
          1996, 20777,  2550,  8079,  1012,   102,     0,     0,     0,
0,
             0,     0,     0,     0,     0],
        [  101,  1996,  6501,  4370,  3147,  1012,   102,  3426,   102,  1
045,
          8250,  2009,  1999,  1996, 18097,  1012,   102,     0,     0,
0,
             0,     0,     0,     0,     0],
        [  101,  1996,  2158,  4711,  2041,  1996,  7852,  1012,   102,  3
426,
           102,  2009,  2001, 26729,  1012,   102,     0,     0,     0,
0,
             0,     0,     0,     0,     0],
        [  101,  1045,  8461,  1996,  8079,  1999,  1996,  5800,  1012,
102,
          3466,   102,  1996,  8079, 11867, 25849,  1012,   102,     0,
0,
             0,     0,     0,     0,     0],
        [  101,  1996,  5776,  2001,  2139, 10536,  7265,  3064,  1012,
102,
          3466,   102,  1996,  6821,  2435,  2032,  2019,  4921,  1012,
102,
             0,     0,     0,     0,     0],
        [  101,  1996,  2611,  4375,  2091,  2014,  4253,  2000,  2014,  3
920,
          2905,  1012,   102,  3426,   102,  2016,  2041, 17603,  2860,  1
996,
          4253,  1012,   102,     0,     0],
        [  101,  1996,  2611,  2371, 14849,  1012,   102,  3466,   102,  2
016,
          2106, 14082,  1012,   102,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0]])}
batch_attn_mask:
 {'choice1': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0, 0, 0, 0, 0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1,
         1],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0]]), 'choice2': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1,
         1],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
0, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0,
         0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0]])}
batch_labels:
 tensor([1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0])
```

Task 2: Implementing and Training BERT-based Multiple Choice Classifier (6 Marks)

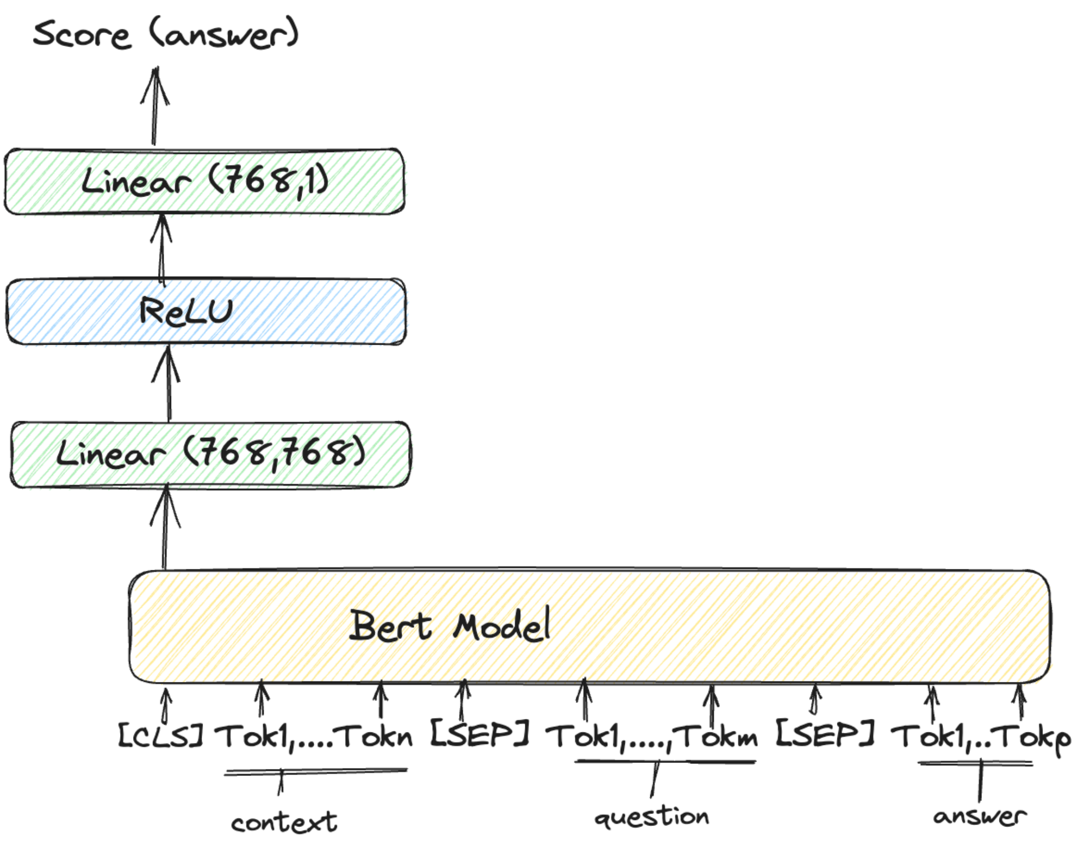
In [ ]:

```python
# Import BertModel from the library
from transformers import BertModel

# Create an instance of pretrained BERT
bert_model = BertModel.from_pretrained("bert-base-uncased")
bert_model
```

```
/Users/rajatjacob/.pyenv/versions/3.11.4/envs/nlp/lib/python3.11/site-pack
ages/huggingface_hub/file_download.py:1132: FutureWarning: `resume_downloa
d` is deprecated and will be removed in version 1.0.0. Downloads always re
sume when possible. If you want to force a new download, use `force_downlo
ad=True`.
  warnings.warn(
Out[ ]: BertModel(
          (embeddings): BertEmbeddings(
            (word_embeddings): Embedding(30522, 768, padding_idx=0)
            (position_embeddings): Embedding(512, 768)
            (token_type_embeddings): Embedding(2, 768)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (encoder): BertEncoder(
            (layer): ModuleList(
              (0-11): 12 x BertLayer(
                (attention): BertAttention(
                  (self): BertSelfAttention(
                    (query): Linear(in_features=768, out_features=768, bias=Tru
        e)
                    (key): Linear(in_features=768, out_features=768, bias=True)
                    (value): Linear(in_features=768, out_features=768, bias=Tru
        e)
                    (dropout): Dropout(p=0.1, inplace=False)
                  )
                  (output): BertSelfOutput(
                    (dense): Linear(in_features=768, out_features=768, bias=Tru
        e)
                    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine
        =True)
                    (dropout): Dropout(p=0.1, inplace=False)
                  )
                )
                (intermediate): BertIntermediate(
                  (dense): Linear(in_features=768, out_features=3072, bias=True)
                  (intermediate_act_fn): GELUActivation()
                )
                (output): BertOutput(
                  (dense): Linear(in_features=3072, out_features=768, bias=True)
                  (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=T
        rue)
                  (dropout): Dropout(p=0.1, inplace=False)
                )
              )
            )
          )
          (pooler): BertPooler(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (activation): Tanh()
          )
        )
```

Task 2.1: Implementing BERT-based Classifier for Multiple Choice Classification (2 Marks)

We will be using trhe same exact architecture as SocialIQA dataset here as well i.e. we have the BERT model as the backbone, using which we obtain the contextualized representation of the [premise, question, answer] sequence. We then use the [CLS] token's embedding as the sequence representation and feed it to a 2 layer MLP (Linear(768, 768) -> ReLU -> Linear(768, 1)) that scores the answer.



The only change this time will be that to predict the correct answer, we need to score each of the two choices each instead of the three answers. Afterwards, like last time we ormalize the scores for the choices by applying softmax, that gives us the probability of each option being the correct answer.

Implement the architecture and forward pass in `BertMultiChoiceClassifierModel` class below:

In [ ]:
```python
class BertMultiChoiceClassifierModel(nn.Module):

    def __init__(self, d_hidden = 768, bert_variant = "bert-base-uncased"
        """
        Define the architecture of Bert-Based mulit-choice classifier.
        You will mainly need to define 3 components, first a BERT layer
        using `BertModel` from transformers library,
        a two layer MLP layer to map the representation from Bert to the
        and a log sftmax layer to map the scores to a probabilities

        Inputs:
            - d_hidden (int): Size of the hidden representations of bert
            - bert_variant (str): BERT variant to use
        """
        super(BertMultiChoiceClassifierModel, self).__init__()
        self.bert_layer = BertModel.from_pretrained(bert_variant)
        self.mlp_layer = nn.Sequential(nn.Linear(d_hidden, d_hidden), nn.
        self.log_softmax_layer = nn.LogSoftmax()

    def forward(self, input_ids_dict, attn_mask_dict):
        """
        Forward Passes the inputs through the network and obtains the pre
        Inputs:
            - input_ids_dict (dict(str,torch.tensor)): A dictionary conta
                                                       representing the sequence of toke
            - attn_mask_dict (dict(str,torch.tensor)): A dictionary conta

        Returns:
          - output (torch.tensor): A torch tensor of shape [batch_size,]

        """
        out = []
        for i in ('choice1', 'choice2'):
            input_ids = input_ids_dict[i]
            attn_mask = attn_mask_dict[i]
            bert_out = self.bert_layer(input_ids, attention_mask=attn_mas
            mlp_out = self.mlp_layer(bert_out)
            out.append(mlp_out)

        return self.log_softmax_layer(torch.cat(out, axis=1))

        return out
```

In [ ]:

```
print(f"Running Sample Test Cases!")
torch.manual_seed(42)
model = BertMultiChoiceClassifierModel()

print("Sample Test Case 1")
batch_input_ids, batch_attn_mask, batch_labels = next(iter(train_loader))
bert_out = model(batch_input_ids, batch_attn_mask).detach().numpy()
expected_bert_out = np.array([[-0.69547975, -0.6908201 ],
                             [-0.6995947,  -0.68674093],
                             [-0.68830335, -0.6980145 ],
                             [-0.6899294,  -0.6963753 ],
                             [-0.6987722,  -0.68755364],
                             [-0.7084117,  -0.6781122 ],
                             [-0.6960533,  -0.6902495 ],
                             [-0.6748525,  -0.71178275],
                             [-0.6868652,  -0.699469  ],
                             [-0.68641526, -0.6999247 ],
                             [-0.6998995,  -0.68644005],
                             [-0.70553845, -0.6809075 ],
                             [-0.7043667,  -0.6820522 ],
                             [-0.6675567,  -0.7194098 ],
                             [-0.700077,   -0.6862651 ],
                             [-0.72046393, -0.6665568 ]])
print(f"Model Output: {bert_out}")
print(f"Expected Output: {expected_bert_out}")

assert bert_out.shape == expected_bert_out.shape
assert np.allclose(bert_out, expected_bert_out, 1e-4)
print("Test Case Passed! :)")
print("******************************\n")

print("Sample Test Case 2")
batch_input_ids, batch_attn_mask, batch_labels = next(iter(test_loader))
bert_out = model(batch_input_ids, batch_attn_mask).detach().numpy()
expected_bert_out = np.array([[-0.6993066,  -0.6870254 ],
                             [-0.7057758,  -0.68067616],
                             [-0.670608,   -0.71620613],
                             [-0.6946109,  -0.69168556],
                             [-0.6930771,  -0.69321734],
                             [-0.6869541,  -0.6993789 ],
                             [-0.68364906, -0.7027364 ],
                             [-0.68642354, -0.6999164 ],
                             [-0.6944856,  -0.69181055],
                             [-0.6879125,  -0.6984094 ],
                             [-0.7094514,  -0.67710465],
                             [-0.68425775, -0.7021164 ],
                             [-0.6869471,  -0.6993859 ],
                             [-0.69160426, -0.6946925 ],
                             [-0.68354183, -0.7028456 ],
                             [-0.69150895, -0.69478804]])
print(f"Model Output: {bert_out}")
print(f"Expected Output: {expected_bert_out}")

assert bert_out.shape == expected_bert_out.shape
assert np.allclose(bert_out, expected_bert_out, 1e-4)
print("Test Case Passed! :)")
print("******************************\n")
```

```
Running Sample Test Cases!
Sample Test Case 1
```

```
/Users/rajatjacob/.pyenv/versions/3.11.4/envs/nlp/lib/python3.11/site-pack
ages/torch/nn/modules/module.py:1532: UserWarning: Implicit dimension choi
ce for log_softmax has been deprecated. Change the call to include dim=X a
s an argument.
  return self._call_impl(*args, **kwargs)
```

```
Model Output: [[-0.69547975 -0.69082004]
 [-0.6995947  -0.68674093]
 [-0.68830335 -0.6980145 ]
 [-0.6899294  -0.6963753 ]
 [-0.6987723  -0.6875535 ]
 [-0.7084117  -0.6781121 ]
 [-0.6960532  -0.6902497 ]
 [-0.67485255 -0.7117827 ]
 [-0.68686515 -0.699469  ]
 [-0.686415   -0.699925  ]
 [-0.69989955 -0.68644005]
 [-0.7055386  -0.6809075 ]
 [-0.70436656 -0.68205225]
 [-0.6675569  -0.7194096 ]
 [-0.700077   -0.6862651 ]
 [-0.7204638  -0.6665569 ]]
Expected Output: [[-0.69547975 -0.6908201 ]
 [-0.6995947  -0.68674093]
 [-0.68830335 -0.6980145 ]
 [-0.6899294  -0.6963753 ]
 [-0.6987722  -0.68755364]
 [-0.7084117  -0.6781122 ]
 [-0.6960533  -0.6902495 ]
 [-0.6748525  -0.71178275]
 [-0.6868652  -0.699469  ]
 [-0.68641526 -0.6999247 ]
 [-0.6998995  -0.68644005]
 [-0.70553845 -0.6809075 ]
 [-0.7043667  -0.6820522 ]
 [-0.6675567  -0.7194098 ]
 [-0.700077   -0.6862651 ]
 [-0.72046393 -0.6665568 ]]
Test Case Passed! :)
******************************

Sample Test Case 2
Model Output: [[-0.69930667 -0.68702537]
 [-0.7057755  -0.6806763 ]
 [-0.67060804 -0.71620595]
 [-0.69461083 -0.6916856 ]
 [-0.69307727 -0.69321716]
 [-0.6869541  -0.69937885]
 [-0.68364906 -0.70273644]
 [-0.6864234  -0.69991654]
 [-0.69448566 -0.6918104 ]
 [-0.6879122  -0.6984097 ]
 [-0.7094514  -0.67710465]
 [-0.68425757 -0.7021165 ]
 [-0.6869472  -0.6993859 ]
 [-0.69160426 -0.6946925 ]
 [-0.68354166 -0.7028458 ]
 [-0.69150895 -0.69478804]]
Expected Output: [[-0.6993066  -0.6870254 ]
 [-0.7057758  -0.68067616]
 [-0.670608   -0.71620613]
 [-0.6946109  -0.69168556]
 [-0.6930771  -0.69321734]
 [-0.6869541  -0.6993789 ]
 [-0.68364906 -0.7027364 ]
 [-0.68642354 -0.6999164 ]
 [-0.6944856  -0.69181055]
 [-0.6879125  -0.6984094 ]
 [-0.7094514  -0.67710465]
 [-0.68425775 -0.7021164 ]
 [-0.6869471  -0.6993859 ]
 [-0.69160426 -0.6946925 ]
 [-0.68354183 -0.7028456 ]
 [-0.69150895 -0.69478804]]
Test Case Passed! :)
******************************
```

Task 2.2: Training and Evaluating the Model (3 Marks)

Now that we have implemented the custom Dataset and a BERT based classifier model, we can start training and evaluating the model as in Lab 2. You will need to implement the `train` and `evaluate` functions below.

In [ ]:
```python
def evaluate(model, test_dataloader, device = "cpu"):
    """
    Evaluates `model` on test dataset

    Inputs:
        - model (BertMultiChoiceClassifierModel): A BERT based multiple c
        - test_dataloader (torch.utils.DataLoader): A dataloader defined

    Returns:
        - accuracy (float): Average accuracy over the test dataset
    """

    model.eval()
    model = model.to(device)
    accuracy = 0

    with torch.no_grad():
        for test_batch in test_dataloader:

            # Read the batch from dataloader
            input_ids_dict, attn_mask_dict, labels = test_batch

            # Send all values of dicts to device
            for key in input_ids_dict.keys():
                input_ids_dict[key] = input_ids_dict[key].to(device)
                attn_mask_dict[key] = attn_mask_dict[key].to(device)
            labels = labels.float()

            # Step 1: Compute model's prediction on the test batch (Note
            preds = model(input_ids_dict, attn_mask_dict).detach().numpy(

            # Step 2: then compute accuracy and store it in batch_accurac
            batch_accuracy = (preds == labels).sum().item() / len(labels)

            accuracy += batch_accuracy

    accuracy = accuracy / len(test_dataloader)
    return accuracy
```

```
def train(model, train_dataloader, test_dataloader,
          lr = 1e-5, num_epochs = 3,
          device = "cpu"):
    """
    Runs the training loop. Define the loss function as BCELoss like the
    and optimizer as Adam and traine for `num_epochs` epochs.

    Inputs:
        - model (BertMultiChoiceClassifierModel):  A BERT based multiple
        - train_dataloader (torch.utils.DataLoader): A dataloader defined
        - test_dataloader (torch.utils.DataLoader): A dataloader defined
        - lr (float): The learning rate for the optimizer
        - num_epochs (int): Number of epochs to train the model for.
        - device (str): Device to train the model on. Can be either 'cuda

    Returns:
        - test_accuracy (float): Test accuracy corresponding to the last
        Note that we are not doing model selection here since we do not h
          It is not a good practice to do model selection on the test set
    """
    epoch_loss = 0
    model = model.to(device)

    best_val_accuracy = float("-inf")
    best_model = None

    # 1. Define Loss function and optimizer
    loss_fn = nn.NLLLoss()
    optimizer = Adam(model.parameters(), lr=lr)

    # Iterate over `num_epochs`
    for epoch in range(num_epochs):
        epoch_loss = 0 # We can use this to keep track of how the loss va
        # Iterate over each batch using the `train_dataloader`
        for train_batch in tqdm(train_dataloader):

            # Zero out any gradients stored in the previous steps
            optimizer.zero_grad()

            # Read the batch from dataloader
            input_ids_dict, attn_mask_dict, labels = train_batch

            # Send all values of dicts to device
            for key in input_ids_dict.keys():
                input_ids_dict[key] = input_ids_dict[key].to(device)
                attn_mask_dict[key] = attn_mask_dict[key].to(device)
            labels = labels.to(device)

            # Step 3: Feed the input features to the model to get outputs
            model_outs = model(input_ids_dict, attn_mask_dict).to(device)

            # Step 4: Compute the loss and perform backward pass
            loss = loss_fn(model_outs, labels)
            loss.backward()

            # Step 5: Take optimizer step
            optimizer.step()

            # Store loss value for tracking
```

```
                epoch_loss += loss.item()

            epoch_loss = epoch_loss / len(train_dataloader)
            # Step 6. Evaluate on validation data by calling `evaluate` and s
            total = 0
            correct = 0
            for val_batch in test_dataloader:
                input_ids_dict, attn_mask_dict, labels = val_batch
                for key in input_ids_dict.keys():
                    input_ids_dict[key] = input_ids_dict[key].to(device)
                    attn_mask_dict[key] = attn_mask_dict[key].to(device)
                labels = labels.to(device)
                pred = torch.argmax(model(input_ids_dict, attn_mask_dict).to(
                correct += (pred==labels).sum().item()
                total += len(labels)
            val_accuracy = total / correct

            # Model selection
            if val_accuracy > best_val_accuracy:
                best_val_accuracy = val_accuracy
                best_model = copy.deepcopy(model) # Create a copy of model

            print(f"Epoch {epoch} completed | Average Training Loss: {epoch_l

        return best_val_accuracy
```

In [ ]:

```
torch.manual_seed(0)
model = BertMultiChoiceClassifierModel()
test_acc = train(model, train_loader, test_loader, num_epochs = 10, lr =
```

```
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 0 completed | Average Training Loss: 0.6893896795809269 | Validation
Accuracy: 1.6129032258064515
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 1 completed | Average Training Loss: 0.6642300561070442 | Validation
Accuracy: 1.4836795252225519
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 2 completed | Average Training Loss: 0.5603887150064111 | Validation
Accuracy: 1.4619883040935673
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 3 completed | Average Training Loss: 0.3591609923169017 | Validation
Accuracy: 1.5105740181268883
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 4 completed | Average Training Loss: 0.20868938486091793 | Validatio
n Accuracy: 1.4749262536873156
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 5 completed | Average Training Loss: 0.10893460421357304 | Validatio
n Accuracy: 1.4836795252225519
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 6 completed | Average Training Loss: 0.0718339525628835 | Validation
Accuracy: 1.492537313432836
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 7 completed | Average Training Loss: 0.04862447260529734 | Validatio
n Accuracy: 1.5015015015015014
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 8 completed | Average Training Loss: 0.0328431484522298 | Validation
Accuracy: 1.5151515151515151
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 9 completed | Average Training Loss: 0.026169340359047055 | Validati
on Accuracy: 1.5151515151515151
```

You should expect about ~65% test accuracy. Note that the model quickly overfits to the dataset in this case, i.e. the training loss reduces dramatically, but there isn't much improvement in the test accuracy after the first epoch. This happens because our training data consists of just 500 examples, which is usually not sufficient for training these large models. Next, we try out a simple strategy to improve the performance drastically.

Task 2.3: Continued Fine-tuning of BERT trained on SocialIQA Dataset (1 Mark)

In Lab2, we fine-tuned BERT on SocialIQA, which is also a common sense reasoning task and has a much larger training set. Deep learning models exhibhit a remarkable property of transfer learning where we can leverage a model trained on task to transfer it's knowledge for learning a new task much more effectively. The idea is that training on SocialIQA dataset would have endowed our model with some common-sense reasoning capabilities, which we can leverage to learn COPA task as well.

Below, you are needed to load the model that you trained in Lab2 and the train that model instead. You can read about how to load pre-trained models in pytorch here. Once you load the model, to train it, just call the `train` function as before.

In [ ]:
```
model = BertMultiChoiceClassifierModel()
model_path = "../Lab02/models/siqa_bert-base-uncased/model.pt" # Change t

# Step 1: Load the pre-trained model weights
# YOUR CODE HERE
model.load_state_dict(torch.load(model_path))
model.eval()


# Step 2: Train the loaded model on COPA dataset
test_acc = train(model, train_loader, test_loader, num_epochs = 10, lr =
test_acc
```

```
/Users/rajatjacob/.pyenv/versions/3.11.4/envs/nlp/lib/python3.11/site-pack
ages/huggingface_hub/file_download.py:1132: FutureWarning: `resume_downloa
d` is deprecated and will be removed in version 1.0.0. Downloads always re
sume when possible. If you want to force a new download, use `force_downlo
ad=True`.
  warnings.warn(
  0%|          | 0/32 [00:00<?, ?it/s]
/Users/rajatjacob/.pyenv/versions/3.11.4/envs/nlp/lib/python3.11/site-pack
ages/torch/nn/modules/module.py:1532: UserWarning: Implicit dimension choi
ce for log_softmax has been deprecated. Change the call to include dim=X a
s an argument.
  return self._call_impl(*args, **kwargs)
Epoch 0 completed | Average Training Loss: 0.6216099723242223 | Validation
Accuracy: 1.3477088948787062
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 1 completed | Average Training Loss: 0.3392861606553197 | Validation
Accuracy: 1.3513513513513513
  0%|          | 0/32 [00:00<?, ?it/s]
```

```
Epoch 2 completed | Average Training Loss: 0.14957202458754182 | Validatio
n Accuracy: 1.3736263736263736
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 3 completed | Average Training Loss: 0.042198095994535834 | Validati
on Accuracy: 1.366120218579235
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 4 completed | Average Training Loss: 0.015410947991767898 | Validati
on Accuracy: 1.36986301369863
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 5 completed | Average Training Loss: 0.008942162043240387 | Validati
on Accuracy: 1.366120218579235
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 6 completed | Average Training Loss: 0.005849872732142103 | Validati
on Accuracy: 1.3736263736263736
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 7 completed | Average Training Loss: 0.004159900086960988 | Validati
on Accuracy: 1.36986301369863
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 8 completed | Average Training Loss: 0.0026933540830214042 | Validat
ion Accuracy: 1.366120218579235
  0%|          | 0/32 [00:00<?, ?it/s]
Epoch 9 completed | Average Training Loss: 0.002068763826173381 | Validati
on Accuracy: 1.358695652173913
```

```
Out[ ]:  1.3736263736263736
```

You should expect ~77% accuracy with this, which is quite a large increase over the original 65% accuracy that we obtained by training the model from scratch. This illustrates the effectiveness of transfer-learning for NLP tasks, specially when the two tasks are related as they were in this case. Transfer learning had been the dominant paradigm in NLP since 2018. However, recently we have been witnessing a new paradigm emerge called "Prompting", which has taken the NLP community and in many ways the whole world by a storm. In the next lab and assignments, we will learn more about this new paradigm.