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

May 13, 2024

Welcome to the Lab 2 of our course on Natural Language Processing. As the name suggests in this lab you will learn how to fine-tune a pretrained model like BERT on a downstream task to improve much more superior performance compared to the methods discussed so far. We will be working with the SocialIQA dataset this week, which is a multiple choice classification dataset designed to learn and measure social and emotional intelligence in NLP models.

This assignment will also make heavy use of the Transformers Library. Don't worry if you are not familiar with the library, we will discuss its usage in detail.

Note: Access to a GPU will be crucial for working on this assignment. So do select a GPU runtime in Colab before you start working.

Learning Outcomes from this Lab:

- Learn how to use Transformer library to load and fine-tune pre-trained langauge models
- Learn how to solve common sense reasoning problems using Masked Language Models like BERT

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)

```
from google.colab import drive
drive.mount('/content/gdrive')
siqa_data_dir = "gdrive/MyDrive/PlakshaTLF24-NLP/Lab02/data/socialiqa-train-dev/" #renamed basis directory organization on my drive.
```

```
Mounted at /content/gdrive
```

```
!ls -l gdrive/MyDrive/PlakshaTLF24-NLP/Lab02/data/socialiqa-train-dev/
```

```
total 8479
-rw------- 1 root root  476394 May 13 13:39 dev.jsonl
-rw------- 1 root root    5862 May 13 13:39 dev-labels.lst
```

```
-rw------- 1 root root 8098489 May 13 13:39 train.jsonl
-rw------- 1 root root  100230 May 13 13:39 train-labels.lst
```

```
# If using Colab, NO NEED TO INSTALL ANYTHING
# Install required libraries
# !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
```

```
# We start by importing libraries that we will be making use of in the
assignment.
import os
from functools import partial
import json
from pprint import pprint
import numpy as np
import pandas as pd
import torch
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
```

SocialIQA Dataset

We start by discussing the dataset that we will making use of in today's Lab. As described above SocialIQA was designed to learn and measure social and emotional intelligence in NLP models. It is a multiple choice classification task, where you are given a context of some social situation, a question about the context and then three possible answers to the questions. The task is to predict which of the three options answers the question given the context.

Below we load the dataset in memory

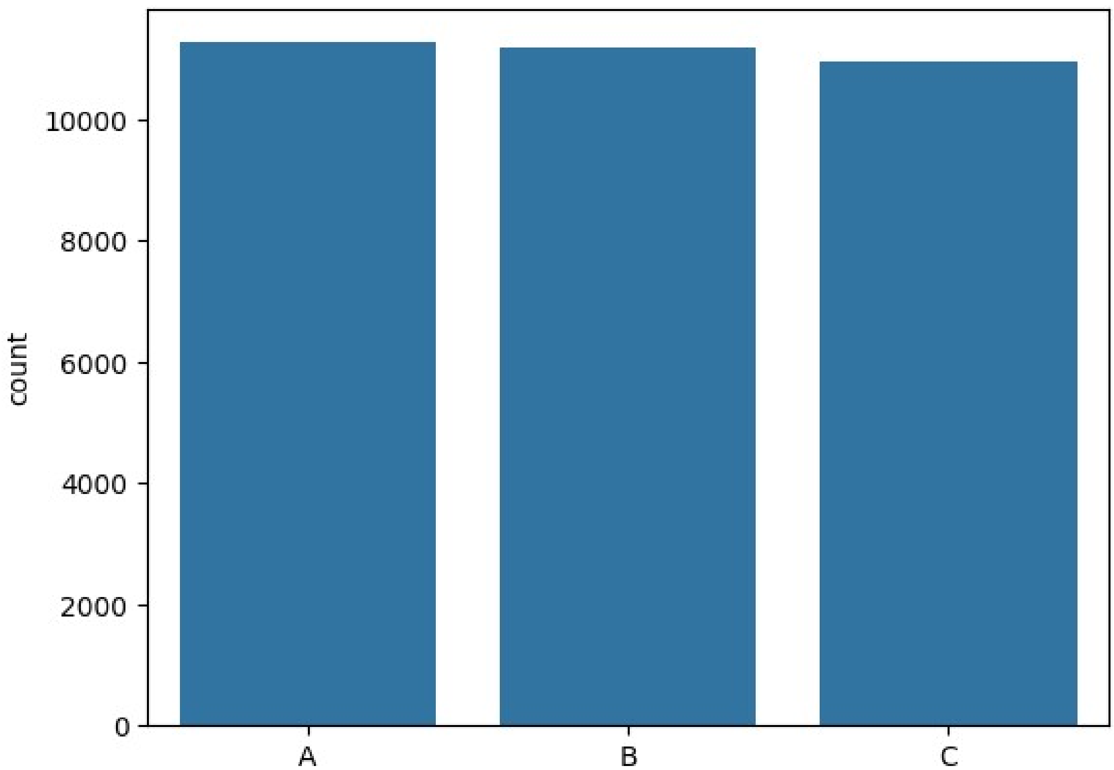
```
def load_siqa_data(split):

    # We first load the file containing context, question and answers
    with open(f"gdrive/MyDrive/PlakshaTLF24-NLP/Lab02/data/socialiqa-train-dev/{split}.jsonl") as f:
        data = [json.loads(jline) for jline in f.read().splitlines()]

    # We then load the file containing the correct answer for each question
    with open(f"gdrive/MyDrive/PlakshaTLF24-NLP/Lab02/data/socialiqa-train-dev/{split}-labels.lst") as f:
        labels = f.read().splitlines()

    labels_dict = {"1": "A", "2": "B", "3": "C"}
    labels = [labels_dict[label] for label in labels]

    return data, labels


train_data, train_labels = load_siqa_data("train")
dev_data, dev_labels = load_siqa_data("dev")
```

```
print(f"Number of Training Examples: {len(train_data)}")
print(f"Number of Validation Examples: {len(dev_data)}")
```

```
Number of Training Examples: 33410
Number of Validation Examples: 1954
```

```
sns.countplot(x = train_labels)
```

```
<Axes: ylabel='count'>
```




```
# 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
{   'context': "Jordan's dog peed on the couch they were selling and Jordan "
               'removed the odor as soon as possible.',
    'question': 'How would Jordan feel afterwards?',
    'answerA': 'selling a couch',
    'answerB': 'Disgusted',
    'answerC': 'Relieved'}
Label: B
```

```
train_data[500]
```

```
{'context': 'kendall was a person who kept her word so she got my money the other day.',
 'question': 'What will Others want to do next?',
 'answerA': 'resent kendall',
 'answerB': 'support kendall',
 'answerC': 'hate kendall'}
```

Task 1: Tokenization and Data Preperation (1 hour)

As discussed in the lectures, BERT and other pretrained language models use sub-word tokenization i.e. individual words can also be split into constituent subwords to reduce the vocabulary size. The Transformer library provides tokenizer for all the popular language models. Below we demonstrate how to create and use these tokenizers.

```
# Import the BertTokenizer from the library
from transformers import BertTokenizer

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

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```


{"model_id":"5524eb95b9aa41ecb89d492015b94b6a","version_major":2,"version_minor":0}

{"model_id":"03402e6fc94f4980a998c94b1da88254","version_major":2,"version_minor":0}

{"model_id":"e34594c4aaa64044935ecf2a807a1a2a","version_major":2,"version_minor":0}

{"model_id":"da3de40dd9ce452b872a7a6d2ede989e","version_major":2,"version_minor":0}

`BertTokenizer.from_pretrained` is used to load a pre-trained tokenizer. Notice that we provide the argument `"bert-base-uncased"` to the method. This refers to the variant of BERT that we want to use. The term "base" means we want to use the smaller BERT variant i.e. the one with 12 layers, and "uncased" refers to the fact that it treats upper-case and lower-case characters identically. There are 4 variants available for BERT which are: - `bert-base-uncased` - `bert-base-cased` - `bert-large-uncased` - `bert-large-cased` Now that we have loaded the tokenizer, let's see how to use it.

`tokenize` method can be used to split the text into sequence of tokens

```
bert_tokenizer.tokenize("kendall was a person who kept her word
exquisitely, so she got my money the other day")
```

```
['kendall',
 'was',
 'a',
 'person',
 'who',
 'kept',
 'her',
 'word',
 'exquisite',
 '##ly',
 ',',
 'so',
 'she',
 'got',
 'my',
 'money',
 'the',
 'other',
 'day']
```

Notice how the tokenizer not only splits the text into words but also subwords like "exquisitely" is split into "exquisite" and "ly".

Another use case of the tokenizer is to convert the tokens into indices. This is important because BERT and almost all language models takes as the inputs a sequence of token ids, which they use to map into embeddings. `convert_tokens_to_ids` method can be used to do this

```
sentence = "kendall was a person who kept her word exquisitely, so she
got my money the other day"
tokens = bert_tokenizer.tokenize(sentence)
token_ids = bert_tokenizer.convert_tokens_to_ids(tokens)
print(token_ids)
```

```
[14509, 2001, 1037, 2711, 2040, 2921, 2014, 2773, 19401, 2135, 1010,
2061, 2016, 2288, 2026, 2769, 1996, 2060, 2154]
```

The two steps can also be combined by simply calling the tokenizer object

```
pprint(bert_tokenizer(sentence), sort_dicts=False, indent=4)
```

```
{   'input_ids': [   101,
                     14509,
                     2001,
                     1037,
                     2711,
```

```
                     2040,
                     2921,
                     2014,
                     2773,
                     19401,
                     2135,
                     1010,
                     2061,
                     2016,
                     2288,
                     2026,
                     2769,
                     1996,
                     2060,
                     2154,
                     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],
    'attention_mask': [   1,
                          1,
                          1,
                          1,
                          1,
                          1,
                          1,
                          1,
                          1,
                          1,
                          1,
                          1,
```

```
                          1,
                          1,
                          1,
                          1,
                          1,
                          1,
                          1,
                          1,
                          1]}
```

Notice that it returns a bunch of things in addition to the ids. The `"input_ids"` are just the token ids that we obtained in the previous cell. However you will notice that it has a few additional ids, it starts with 101 and ends with 102. These are what we call special tokens and correspond the [CLS] and [SEP] tokens used by BERT. [CLS] token is mainly added to beginning of each sequence, and its representations are used to perform sequence classification. More on [SEP] token later.

`"token_type_ids"` contains which sequence does a particular token belongs to.

`"attention_mask`" is a mask vector that indicates if a particular token corresponds to padding. Padding is extremely important when we are dealing with variable length sequences, which is almost always the case. Through padding we can ensure that all the sequences in a batch are of same size. However, while processing the sequence we need ignore these padding tokens, hence a mask is required to identify such tokens.

We can tokenize a batch of sequences by just providing a list instead of a string while calling the tokenizer and later pad them using the `.pad` method.

```
batch_size = 4
sentence_batch = [train_data[i]["context"] for i in range(batch_size)]

#Tokenize the batch of sequences
tokenized_batch = bert_tokenizer(sentence_batch)

# Pad the tokenized batch
tokenized_batch_padded = bert_tokenizer.pad(tokenized_batch,
padding=True, max_length=32, return_tensors="pt")

input_ids = tokenized_batch_padded["input_ids"]
attn_mask = tokenized_batch_padded["attention_mask"]
print(f"Input Ids shape: {input_ids.shape}")
print(f"Attention Mask shape: {attn_mask.shape}")

pprint(f"Input Ids:\n {input_ids}\n")
pprint(f"Attention Mask:\n {attn_mask}\n")

Input Ids shape: torch.Size([4, 23])
Attention Mask shape: torch.Size([4, 23])
('Input Ids:\n'
 ' tensor([[  101,  7232,  2787,  2000,  2031,  1037, 26375,  1998,
```

```
5935,  '
 '2014,\n'
 '          2814,  2362,  1012,   102,     0,     0,     0,     0,
0,     '
 '0,\n'
 '             0,     0,     0],\n'
 '        [  101,  5553,  2734,  2000,  2507,  2041,  5841,  2005,
2019,  '
 '9046,\n'
 '          2622,  2012,  2147,  1012,   102,     0,     0,     0,
0,     '
 '0,\n'
 '             0,     0,     0],\n'
 '        [  101, 22712,  2001,  2019,  6739, 19949,  1998,  2001,
2006,  '
 '1996,\n'
 '          2300,  2007, 11928,  1012, 22712, 17395,  2098, 11928,
1005,  '
 '1055,\n'
 '          8103,  1012,   102],\n'
 '        [  101, 18403,  2435,  1037,  8549,  2000, 27970,  1005,
1055,  '
 '2365,\n'
 '          2043,  2027,  2020,  3110,  2091,  1012,   102,     0,
0,     '
 '0,\n'
 '             0,     0,     0]])\n')
('Attention Mask:\n'
 ' tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, '
 '0],\n'
 '        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0, 0, '
 '0],\n'
 '        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, '
 '1],\n'
 '        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0, '
 '0]])\n')

/usr/local/lib/python3.10/dist-packages/transformers/
tokenization_utils_base.py:2692: UserWarning: `max_length` is ignored
when `padding`=`True` and there is no truncation strategy. To pad to
max length, use `padding='max_length'`.
  warnings.warn(
```

Notice how 0s get appended to the input ids sequence, and the same is also reflected in the output of `attn_mask` where `0` indicates that the particular token was padded and `1` means otherwise. Setting `return_tensors="pt"` results in the outputs as torch tensors

Finally, for tasks involving reasoning over multiple sentences (like what we have for the SocialIQA dataset), it is common to seperate out each sentence using a [SEP] token:

We can achieve this by adding concatenating all sentences with the `[SEP]` token before calling the tokenizer

```
example = train_data[100]
context = example["context"]
question = example["question"]
answerA = example["answerA"]

# Concatenate the context, question and answerA
cqa = context + bert_tokenizer.sep_token + question +
bert_tokenizer.sep_token + answerA
print(cqa)

tokenized_cqa = bert_tokenizer(cqa)
pprint(tokenized_cqa, sort_dicts=False, indent=4)
```

```
Jordan's dog peed on the couch they were selling and Jordan removed
the odor as soon as possible.[SEP]How would Jordan feel afterwards?
[SEP]selling a couch
{   'input_ids': [   101,
                     5207,
                     1005,
                     1055,
                     3899,
                     21392,
                     2094,
                     2006,
                     1996,
                     6411,
                     2027,
                     2020,
                     4855,
                     1998,
                     5207,
                     3718,
                     1996,
                     19255,
                     2004,
                     2574,
                     2004,
                     2825,
                     1012,
                     102,
                     2129,
                     2052,
```

```
                          5207,
                          2514,
                          5728,
                          1029,
                          102,
                          4855,
                          1037,
                          6411,
                          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,
                          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,
                                 1,
                                 1,
                                 1,
                                 1,
                                 1,
                                 1,
                                 1,
                                 1]}
```

For the reasons that will become clear once we work on the modeling part, we need three input tensors for each dataset example, one for concatenating each answer with the context and question.

```
example = train_data[100]
context = example["context"]
question = example["question"]
answerA = example["answerA"]

answerB = example["answerB"]
answerC = example["answerC"]

cqaA = context + bert_tokenizer.sep_token + question +
bert_tokenizer.sep_token + answerA
cqaB = context + bert_tokenizer.sep_token + question +
bert_tokenizer.sep_token + answerB
cqaC = context + bert_tokenizer.sep_token + question +
bert_tokenizer.sep_token + answerC
```

```
print(cqaA)
print(cqaB)
print(cqaC)

tokenized_cqaA = bert_tokenizer(cqaA)
tokenized_cqaB = bert_tokenizer(cqaB)
tokenized_cqaC = bert_tokenizer(cqaC)
```

```
Jordan's dog peed on the couch they were selling and Jordan removed
the odor as soon as possible.[SEP]How would Jordan feel afterwards?
[SEP]selling a couch
Jordan's dog peed on the couch they were selling and Jordan removed
the odor as soon as possible.[SEP]How would Jordan feel afterwards?
[SEP]Disgusted
Jordan's dog peed on the couch they were selling and Jordan removed
the odor as soon as possible.[SEP]How would Jordan feel afterwards?
[SEP]Relieved
```

Task 1.1: Custom Dataset Class

Now that we know how to use the hugging face tokenizers we can define the custom `torch.utils.Dataset` class like we did in the previous assignments to process and store the data as well as provides a way to iterate through the dataset. Implement the `SIQABertDataset` class below. Recall to create a custom class you need to implement 3 methods `__init__`, `__len__` and `__getitem__`.

```
from torch.utils.data import Dataset, DataLoader

class SIQABertDataset(Dataset):

    def __init__(self, data, labels, bert_variant = "bert-base-
uncased"):
        """
        Constructor for the `SST2BertDataset` class. Stores the
`sentences` and `labels` which can then be used by
        other methods. Also initializes the tokenizer

        Inputs:
            - data (list) : A list SIQA dataset examples
            - labels (list): A list of labels corresponding to each
example
            - bert_variant (str): A string indicating the variant of
BERT to be used.
        """
        self.label2label_id = {"A": 0, "B": 1, "C": 2}
        self.data = None
        self.labels = None
        self.tokenizer = None
```

```
        # YOUR CODE HERE
        self.data = data
        self.labels = labels
        self.tokenizer = BertTokenizer.from_pretrained(bert_variant)

        if (not self.data) or (not self.labels) or (not
self.tokenizer):
          raise NotImplementedError()

    def __len__(self):
        """
        Returns the length of the dataset
        """
        length = None

        # YOUR CODE HERE
        length = len(self.data)

        if length is None:
          raise NotImplementedError()

        return length

    def __getitem__(self, idx):
        """
        Returns the training example corresponding to review present
at the `idx` position in the dataset

        Inputs:
            - idx (int): Index corresponding to the review,label to be
returned

        Returns:
            - tokenized_input_dict (dict(str, dict)): A dictionary
corresponding to tokenizer outputs for the three resulting sequences
due to each answer choices as described above
            - label (int): Answer label for the corresponding
sentence. We will use 0, 1 and 2 to represent A, B and C respectively.

        Example Output:
            - tokenized_input_dict: {
                "A": {'input_ids': [101, 5207, 1005, 1055, 3899,
21392, 2094, 2006, 1996, 6411, 2027, 2020, 4855, 1998, 5207, 3718,
1996, 19255, 2004, 2574, 2004, 2825, 1012, 102, 2129, 2052, 5207,
2514, 5728, 1029, 102, 4855, 1037, 6411, 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, 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, 1, 1,
1, 1, 1, 1, 1, 1]},
```

```
                "B": {'input_ids': [101, 5207, 1005, 1055, 3899,
21392, 2094, 2006, 1996, 6411, 2027, 2020, 4855, 1998, 5207, 3718,
1996, 19255, 2004, 2574, 2004, 2825, 1012, 102, 2129, 2052, 5207,
2514, 5728, 1029, 102, 17733, 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, 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, 1, 1, 1, 1, 1, 1]},
                "C": {'input_ids': [101, 5207, 1005, 1055, 3899,
21392, 2094, 2006, 1996, 6411, 2027, 2020, 4855, 1998, 5207, 3718,
1996, 19255, 2004, 2574, 2004, 2825, 1012, 102, 2129, 2052, 5207,
2514, 5728, 1029, 102, 7653, 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, 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, 1, 1, 1, 1, 1, 1]}
            }
            - label: 0

        """

        tokenized_input_dict = {"A": None, "B": None, "C": None}
        label = None

        # YOUR CODE HERE
        example = self.data[idx]
        context = example["context"]
        question = example["question"]
        answerA = example["answerA"]
        answerB = example["answerB"]
        answerC = example["answerC"]

        cqaA = context + self.tokenizer.sep_token + question +
self.tokenizer.sep_token + answerA
        cqaB = context + self.tokenizer.sep_token + question +
self.tokenizer.sep_token + answerB
        cqaC = context + self.tokenizer.sep_token + question +
self.tokenizer.sep_token + answerC

        tokenized_input_dict["A"] = self.tokenizer(cqaA)
        tokenized_input_dict["B"] = self.tokenizer(cqaB)
        tokenized_input_dict["C"] = self.tokenizer(cqaC)

        label = self.label2label_id[self.labels[idx]]

        if label is None:
          raise NotImplementedError()

        return tokenized_input_dict, label

print("Running Sample Test Cases")
```

```
sample_dataset = SIQABertDataset(train_data[:2], train_labels[:2],
bert_variant="bert-base-uncased")

print(f"Sample Test Case 1: Checking if `__len__` is implemented
correctly")
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
correctly for `idx= 0`")
sample_idx = 0
tokenized_input_dict, label = sample_dataset.__getitem__(sample_idx)
expected_tokenized_input_dict = {'A': {'input_ids': [101, 7232, 2787,
2000, 2031, 1037, 26375, 1998, 5935, 2014, 2814, 2362, 1012, 102,
2129, 2052, 2500, 2514, 2004, 1037, 2765, 1029, 102, 2066, 7052, 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], '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]},
  'B': {'input_ids': [101, 7232, 2787, 2000, 2031, 1037, 26375, 1998,
5935, 2014, 2814, 2362, 1012, 102, 2129, 2052, 2500, 2514, 2004, 1037,
2765, 1029, 102, 2066, 6595, 2188, 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], '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]},
  'C': {'input_ids': [101, 7232, 2787, 2000, 2031, 1037, 26375, 1998,
5935, 2014, 2814, 2362, 1012, 102, 2129, 2052, 2500, 2514, 2004, 1037,
2765, 1029, 102, 1037, 2204, 2767, 2000, 2031, 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, 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, 1, 1]}}
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
```

```
correctly for `idx= 1`")
sample_idx = 1
tokenized_input_dict, label = sample_dataset.__getitem__(sample_idx)
expected_tokenized_input_dict =  {'A': {'input_ids': [101, 5553, 2734,
2000, 2507, 2041, 5841, 2005, 2019, 9046, 2622, 2012, 2147, 1012, 102,
2054, 2097, 2500, 2215, 2000, 2079, 2279, 1029, 102, 21090, 2007,
5553, 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, 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, 1]},
                                  'B': {'input_ids': [101, 5553, 2734,
2000, 2507, 2041, 5841, 2005, 2019, 9046, 2622, 2012, 2147, 1012, 102,
2054, 2097, 2500, 2215, 2000, 2079, 2279, 1029, 102, 2131, 2000, 2147,
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, 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,
1]},
                                'C': {'input_ids': [101, 5553, 2734,
2000, 2507, 2041, 5841, 2005, 2019, 9046, 2622, 2012, 2147, 1012, 102,
2054, 2097, 2500, 2215, 2000, 2079, 2279, 1029, 102, 7475, 2007, 1996,
14799, 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, 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, 1, 1]}}


expected_label = 1
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
correctly for `idx= 0` for a different bert-variant")
sample_dataset = SIQABertDataset(train_data[:2], train_labels[:2],
bert_variant="bert-base-cased")
sample_idx = 0
tokenized_input_dict, label = sample_dataset.__getitem__(sample_idx)
expected_tokenized_input_dict = {'A': {'input_ids': [101, 6681, 1879,
1106, 1138, 170, 2927, 3962, 27138, 1105, 5260, 1123, 2053, 1487, 119,
102, 1731, 1156, 8452, 1631, 1112, 170, 1871, 136, 102, 1176, 6546,
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, 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,
1]},
 'B': {'input_ids': [101, 6681, 1879, 1106, 1138, 170, 2927, 3962,
27138, 1105, 5260, 1123, 2053, 1487, 119, 102, 1731, 1156, 8452, 1631,
1112, 170, 1871, 136, 102, 1176, 6218, 1313, 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, 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, 1, 1]},
 'C': {'input_ids': [101, 6681, 1879, 1106, 1138, 170, 2927, 3962,
27138, 1105, 5260, 1123, 2053, 1487, 119, 102, 1731, 1156, 8452, 1631,
1112, 170, 1871, 136, 102, 170, 1363, 1910, 1106, 1138, 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, 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, 1, 1, 1, 1]}}
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:
 {'A': {'input_ids': [101, 7232, 2787, 2000, 2031, 1037, 26375, 1998,
5935, 2014, 2814, 2362, 1012, 102, 2129, 2052, 2500, 2514, 2004, 1037,
2765, 1029, 102, 2066, 7052, 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],
'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]}, 'B': {'input_ids': [101, 7232, 2787,
2000, 2031, 1037, 26375, 1998, 5935, 2014, 2814, 2362, 1012, 102,
2129, 2052, 2500, 2514, 2004, 1037, 2765, 1029, 102, 2066, 6595, 2188,
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], '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]},
'C': {'input_ids': [101, 7232, 2787, 2000, 2031, 1037, 26375, 1998,
```

```
5935, 2014, 2814, 2362, 1012, 102, 2129, 2052, 2500, 2514, 2004, 1037,
2765, 1029, 102, 1037, 2204, 2767, 2000, 2031, 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, 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, 1]}}
Expected tokenized_input_dict:
 {'A': {'input_ids': [101, 7232, 2787, 2000, 2031, 1037, 26375, 1998,
5935, 2014, 2814, 2362, 1012, 102, 2129, 2052, 2500, 2514, 2004, 1037,
2765, 1029, 102, 2066, 7052, 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],
'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]}, 'B': {'input_ids': [101, 7232, 2787,
2000, 2031, 1037, 26375, 1998, 5935, 2014, 2814, 2362, 1012, 102,
2129, 2052, 2500, 2514, 2004, 1037, 2765, 1029, 102, 2066, 6595, 2188,
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], '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]},
'C': {'input_ids': [101, 7232, 2787, 2000, 2031, 1037, 26375, 1998,
5935, 2014, 2814, 2362, 1012, 102, 2129, 2052, 2500, 2514, 2004, 1037,
2765, 1029, 102, 1037, 2204, 2767, 2000, 2031, 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, 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, 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:
 {'A': {'input_ids': [101, 5553, 2734, 2000, 2507, 2041, 5841, 2005,
2019, 9046, 2622, 2012, 2147, 1012, 102, 2054, 2097, 2500, 2215, 2000,
2079, 2279, 1029, 102, 21090, 2007, 5553, 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, 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]}, 'B': {'input_ids':
[101, 5553, 2734, 2000, 2507, 2041, 5841, 2005, 2019, 9046, 2622,
2012, 2147, 1012, 102, 2054, 2097, 2500, 2215, 2000, 2079, 2279, 1029,
102, 2131, 2000, 2147, 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, 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, 1]}, 'C': {'input_ids': [101, 5553,
2734, 2000, 2507, 2041, 5841, 2005, 2019, 9046, 2622, 2012, 2147,
1012, 102, 2054, 2097, 2500, 2215, 2000, 2079, 2279, 1029, 102, 7475,
2007, 1996, 14799, 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, 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, 1, 1]}}
Expected tokenized_input_dict:
 {'A': {'input_ids': [101, 5553, 2734, 2000, 2507, 2041, 5841, 2005,
2019, 9046, 2622, 2012, 2147, 1012, 102, 2054, 2097, 2500, 2215, 2000,
2079, 2279, 1029, 102, 21090, 2007, 5553, 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, 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]}, 'B': {'input_ids':
[101, 5553, 2734, 2000, 2507, 2041, 5841, 2005, 2019, 9046, 2622,
2012, 2147, 1012, 102, 2054, 2097, 2500, 2215, 2000, 2079, 2279, 1029,
102, 2131, 2000, 2147, 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, 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, 1]}, 'C': {'input_ids': [101, 5553,
2734, 2000, 2507, 2041, 5841, 2005, 2019, 9046, 2622, 2012, 2147,
1012, 102, 2054, 2097, 2500, 2215, 2000, 2079, 2279, 1029, 102, 7475,
2007, 1996, 14799, 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, 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, 1, 1]}}
label:
 1
Expected label:
 1
Sample Test Case Passed!
****************************************

Sample Test Case 4: Checking if `__getitem__` is implemented correctly
for `idx= 0` for a different bert-variant
```

{"model_id":"3f30874aaa6942319220a30c2e33186e","version_major":2,"version_minor":0}

{"model_id":"eb0b6ae457234ac1b2f4995dd3ceb9ad","version_major":2,"version_minor":0}

{"model_id":"56cd7cb15fb240c9af2b32a138c43d0e","version_major":2,"version_minor":0}

{"model_id":"90deb860c51c4915b019ab6e33b95e18","version_major":2,"version_minor":0}

```
tokenized_input_dict:
 {'A': {'input_ids': [101, 6681, 1879, 1106, 1138, 170, 2927, 3962,
27138, 1105, 5260, 1123, 2053, 1487, 119, 102, 1731, 1156, 8452, 1631,
1112, 170, 1871, 136, 102, 1176, 6546, 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, 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, 1]}, 'B': {'input_ids': [101,
```

```
6681, 1879, 1106, 1138, 170, 2927, 3962, 27138, 1105, 5260, 1123,
2053, 1487, 119, 102, 1731, 1156, 8452, 1631, 1112, 170, 1871, 136,
102, 1176, 6218, 1313, 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, 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, 1, 1]}, 'C': {'input_ids': [101, 6681,
1879, 1106, 1138, 170, 2927, 3962, 27138, 1105, 5260, 1123, 2053,
1487, 119, 102, 1731, 1156, 8452, 1631, 1112, 170, 1871, 136, 102,
170, 1363, 1910, 1106, 1138, 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, 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, 1, 1, 1, 1]}}
Expected tokenized_input_dict:
 {'A': {'input_ids': [101, 6681, 1879, 1106, 1138, 170, 2927, 3962,
27138, 1105, 5260, 1123, 2053, 1487, 119, 102, 1731, 1156, 8452, 1631,
1112, 170, 1871, 136, 102, 1176, 6546, 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, 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, 1]}, 'B': {'input_ids': [101,
6681, 1879, 1106, 1138, 170, 2927, 3962, 27138, 1105, 5260, 1123,
2053, 1487, 119, 102, 1731, 1156, 8452, 1631, 1112, 170, 1871, 136,
102, 1176, 6218, 1313, 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, 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, 1, 1]}, 'C': {'input_ids': [101, 6681,
1879, 1106, 1138, 170, 2927, 3962, 27138, 1105, 5260, 1123, 2053,
1487, 119, 102, 1731, 1156, 8452, 1631, 1112, 170, 1871, 136, 102,
170, 1363, 1910, 1106, 1138, 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, 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, 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

```
train_dataset = SIQABertDataset(train_data, train_labels,
bert_variant="bert-base-uncased")
dev_dataset = SIQABertDataset(dev_data, dev_labels,
bert_variant="bert-base-uncased")
```

Before we instantiate the dataloaders for iterating over the dataset like last time, we need define a collate function, that creates batches from a list of dataset examples. In the last class we didn't have to create one, because all of our examples were of the same size, but that's not the case anymore, and we need to pad the sequences so that they all are of same size. We have implemented the collate_fn for you below, but we recommend going through it step by step, as it is used often in practice.

```
def collate_fn(tokenizer, batch):
    """
    Collate function to be used when creating a data loader for the
SIQA dataset.
    :param tokenizer: The tokenizer to be used to tokenize the inputs.
    :param batch: A list of tuples of the form (tokenized_input_dict,
label)
    :return: A tuple of the form (tokenized_inputs_dict_batch,
labels_batch)
    """

    tokenized_inputsA_batch = []
    tokenized_inputsB_batch = []
    tokenized_inputsC_batch = []
    labels_batch = []
    for tokenized_inputs_dict, label in batch:
        tokenized_inputsA_batch.append(tokenized_inputs_dict["A"])
        tokenized_inputsB_batch.append(tokenized_inputs_dict["B"])
        tokenized_inputsC_batch.append(tokenized_inputs_dict["C"])
        labels_batch.append(label)

    #Pad the inputs
    tokenized_inputsA_batch = tokenizer.pad(tokenized_inputsA_batch,
padding=True, return_tensors="pt")
    tokenized_inputsB_batch = tokenizer.pad(tokenized_inputsB_batch,
padding=True, return_tensors="pt")
    tokenized_inputsC_batch = tokenizer.pad(tokenized_inputsC_batch,
padding=True, return_tensors="pt")

    # Convert labels list to a tensor
    labels_batch = torch.tensor(labels_batch)
    return (
        {"A": tokenized_inputsA_batch["input_ids"], "B":
tokenized_inputsB_batch["input_ids"], "C":
tokenized_inputsC_batch["input_ids"]},
        {"A": tokenized_inputsA_batch["attention_mask"], "B":
tokenized_inputsB_batch["attention_mask"], "C":
tokenized_inputsC_batch["attention_mask"]},
        labels_batch
    )
```

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.

```
batch_size = 16
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True, collate_fn=partial(collate_fn, train_dataset.tokenizer))
dev_loader = DataLoader(dev_dataset, batch_size=batch_size,
shuffle=True, collate_fn=partial(collate_fn, dev_dataset.tokenizer))

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:
 {'A': tensor([[  101, 22712,  2170,  1000,  1996,  2158,  1000,
1010, 14407, 11404,
          2046,  1996,  3614,  1012,   102,  2339,  2106, 22712,
2079,  2023,
          1029,   102,  2025,  3693,  1996, 11700,   102,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101, 10555,  2001,  5287,  2000,  2011,  1037,  8606,
17220,  1998,
          2150,  2062,  6625,  1012,   102,  2129,  2052, 10555,
2514,  5728,
          1029,   102,  8363,   102,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101, 18961,  2973,  2006,  1996,  4534,  1012,  2016,
2356, 11928,
          2005,  2769,  2043,  2016,  2939,  2011,  1012,   102,
2129,  2052,
          2017,  6235, 18961,  1029,   102,  1037,  3532,  2711,
102,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101,  5863,  2170,  1996,  4614,  2043,  2002,  4384,
1996, 13742,
          3875,  1996, 21071,  1012,   102,  2054,  2097,  5863,
2215,  2000,
          2079,  2279,  1029,   102,  5466,  2019, 11355,  2012,
1996, 28019,
           102,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101,  7546,  3427,  2998,  2006,  2694,  2005,  1037,
```

```
2261,  2847,
          1998,  2001,  2200,  7622,  1012,   102,  2129,  2052,
7546,  2514,
          5728,  1029,   102, 11471,   102,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101,  3994,  4778,  2037,  2767,  2058,  2000,  3422,
1996,  4164,
          2006,  3477,  1011,  2566,  1011,  3193,  1012,   102,
2054,  2097,
          3994,  2215,  2000,  2079,  2279,  1029,   102,  2131,
2070, 27962,
          1998,  8974,  3201,  2000,  4521,  2096,  2027,  3422,
1996,  2265,
           102,     0],
        [  101,  2096,  2667,  2000,  8054,  2037,  2814,  2000,
3693,  1996,
          2012,  2277,  1010,  6683,  4207,  2014,  9029,  2055,
4676,  1012,
           102,  2054,  2097,  4148,  2000,  6683,  1029,   102,
8796,   102,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101, 22712,  2170,  1996,  2158,  2055,  1996,  3291,
2027,  2020,
          2383,  1012,   102,  2054,  2097,  2500,  2215,  2000,
2079,  2279,
          1029,   102,  2655,  1996,  2158,   102,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101,  7546,  2134,  1005,  1056,  3046,  2004,  2524,
2004,  2016,
          2288,  1998,  2288,  2353,  2173,  1012,   102,  2129,
2052,  2017,
          6235,  7546,  1029,   102, 12774,  1998, 11922,   102,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101, 18403,  1005,  1055, 10808,  2291,  3844,  2091,
2021,  2027,
          2018,  1037, 13788,  1012,   102,  2129,  2052,  2017,
6235, 18403,
          1029,   102,  6742,   102,     0,     0,     0,     0,
```

```
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101,  9036,  2001,  1040,  7274,  2571,  9048,  2278,
2061, 14509,
          2949,  2037,  4646,  2005,  1037,  3105,  2061,  2008,
9036,  2052,
          2131,  2019,  4357,  1998,  2265,  2037,  4813,  2059,
1012,   102,
          2129,  2052, 14509,  2514,  5728,  1029,   102,  2066,
1037,  2204,
          2767,   102],
        [  101, 22712,  2165, 27970,  1005,  1055,  2769,  2029,
12781,  2006,
          2010,  9715,  4600,  1012,   102,  2129,  2052, 22712,
2514,  5728,
          1029,   102,  2062,  4138,   102,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101, 18403,  2435,  9321,  2298,  2138,  2016,  2001,
2035,  2039,
          1999,  2014,  2449,  1012,   102,  2054,  2097, 18403,
2215,  2000,
          2079,  2279,  1029,   102,  5060,  2077,  2023,   102,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101,  9806,  2001,  4394,  2005,  4466,  2847,  1012,
2111,  2020,
          2559,  7249,  1012,  9806,  2170,  2188,  1012,   102,
2129,  2052,
          2500,  2514,  2004,  1037,  2765,  1029,   102,  9364,
1998, 11471,
           102,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101,  7627,  2001,  2012,  2188,  2007,  2037,  2155,
1010,  1998,
          7627,  2170,  2068,  2035,  2046,  1996,  2542,  2282,
1012,   102,
          2339,  2106,  7627,  2079,  2023,  1029,   102,  2359,
2000,  3422,
          1037,  2694,  2265,   102,     0,     0,     0,     0,
0,     0,
             0,     0],
```

```
        [  101, 10555,  2985,  2051,  2000,  5335,  2014,  7022,
1012,   102,
          2054,  2097, 10555,  2215,  2000,  2079,  2279,  1029,
102,  2156,
          2055,  2082,   102,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0]]), 'B': tensor([[  101, 22712,  2170,  1000,
1996,  2158,  1000,  1010, 14407, 11404,
          2046,  1996,  3614,  1012,   102,  2339,  2106, 22712,
2079,  2023,
          1029,   102,  3693,  1996, 11700,   102,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101, 10555,  2001,  5287,  2000,  2011,  1037,  8606,
17220,  1998,
          2150,  2062,  6625,  1012,   102,  2129,  2052, 10555,
2514,  5728,
          1029,   102,  6314,   102,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101, 18961,  2973,  2006,  1996,  4534,  1012,  2016,
2356, 11928,
          2005,  2769,  2043,  2016,  2939,  2011,  1012,   102,
2129,  2052,
          2017,  6235, 18961,  1029,   102,  2066,  2027,  2215,
2000,  4468,
         18961,   102,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101,  5863,  2170,  1996,  4614,  2043,  2002,  4384,
1996, 13742,
          3875,  1996, 21071,  1012,   102,  2054,  2097,  5863,
2215,  2000,
          2079,  2279,  1029,   102,  4604,  2037,  4937,  2000,
1996, 28019,
           102,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101,  7546,  3427,  2998,  2006,  2694,  2005,  1037,
2261,  2847,
          1998,  2001,  2200,  7622,  1012,   102,  2129,  2052,
7546,  2514,
          5728,  1029,   102,  1999, 15180,   102,     0,     0,
```

```
0,     0,
                 0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
                 0,     0],
             [  101,  3994,  4778,  2037,  2767,  2058,  2000,  3422,
1996,  4164,
              2006,  3477,  1011,  2566,  1011,  3193,  1012,   102,
2054,  2097,
              3994,  2215,  2000,  2079,  2279,  1029,   102,  2377,
1996,  3682,
              2096,  2027,  3422,  1996,  4164,   102,     0,     0,
0,     0,
                 0,     0],
             [  101,  2096,  2667,  2000,  8054,  2037,  2814,  2000,
3693,  1996,
              2012,  2277,  1010,  6683,  4207,  2014,  9029,  2055,
4676,  1012,
               102,  2054,  2097,  4148,  2000,  6683,  1029,   102,
2969, 19556,
               102,     0,     0,     0,     0,     0,     0,     0,
0,     0,
                 0,     0],
             [  101, 22712,  2170,  1996,  2158,  2055,  1996,  3291,
2027,  2020,
              2383,  1012,   102,  2054,  2097,  2500,  2215,  2000,
2079,  2279,
              1029,   102,  2831,  2000,  1996,  2158,   102,     0,
0,     0,
                 0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
                 0,     0],
             [  101,  7546,  2134,  1005,  1056,  3046,  2004,  2524,
2004,  2016,
              2288,  1998,  2288,  2353,  2173,  1012,   102,  2129,
2052,  2017,
              6235,  7546,  1029,   102,  4895, 18938, 21967,  1998,
13971,   102,
                 0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
                 0,     0],
             [  101, 18403,  1005,  1055, 10808,  2291,  3844,  2091,
2021,  2027,
              2018,  1037, 13788,  1012,   102,  2129,  2052,  2017,
6235, 18403,
              1029,   102,  2001,  5580,  2027,  2018,  1037, 13788,
102,     0,
                 0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
                 0,     0],
```

```
        [  101,  9036,  2001,  1040,  7274,  2571,  9048,  2278,
2061, 14509,
          2949,  2037,  4646,  2005,  1037,  3105,  2061,  2008,
9036,  2052,
          2131,  2019,  4357,  1998,  2265,  2037,  4813,  2059,
1012,   102,
          2129,  2052, 14509,  2514,  5728,  1029,   102,  2066,
1037, 11809,
          2767,   102],
        [  101, 22712,  2165, 27970,  1005,  1055,  2769,  2029,
12781,  2006,
          2010,  9715,  4600,  1012,   102,  2129,  2052, 22712,
2514,  5728,
          1029,   102,  2200,  3407,   102,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101, 18403,  2435,  9321,  2298,  2138,  2016,  2001,
2035,  2039,
          1999,  2014,  2449,  1012,   102,  2054,  2097, 18403,
2215,  2000,
          2079,  2279,  1029,   102,  8568,  2014,   102,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101,  9806,  2001,  4394,  2005,  4466,  2847,  1012,
2111,  2020,
          2559,  7249,  1012,  9806,  2170,  2188,  1012,   102,
2129,  2052,
          2500,  2514,  2004,  1037,  2765,  1029,   102,  7653,
1998,  8363,
           102,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101,  7627,  2001,  2012,  2188,  2007,  2037,  2155,
1010,  1998,
          7627,  2170,  2068,  2035,  2046,  1996,  2542,  2282,
1012,   102,
          2339,  2106,  7627,  2079,  2023,  1029,   102,  2018,
2019,  2590,
          8874,  2000,  2191,   102,     0,     0,     0,     0,
0,     0,
             0,     0],
        [  101, 10555,  2985,  2051,  2000,  5335,  2014,  7022,
1012,   102,
          2054,  2097, 10555,  2215,  2000,  2079,  2279,  1029,
102,  2022,
```

```
          1037,  2488,  3076,   102,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0]]), 'C': tensor([[  101, 22712,  2170,  1000,
1996,  2158,  1000,  1010, 14407, 11404,
          2046,  1996,  3614,  1012,   102,  2339,  2106, 22712,
2079,  2023,
          1029,   102,  2954,  2068,   102,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0],
        [  101, 10555,  2001,  5287,  2000,  2011,  1037,  8606,
17220,  1998,
          2150,  2062,  6625,  1012,   102,  2129,  2052, 10555,
2514,  5728,
          1029,   102,  4854,   102,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0],
        [  101, 18961,  2973,  2006,  1996,  4534,  1012,  2016,
2356, 11928,
          2005,  2769,  2043,  2016,  2939,  2011,  1012,   102,
2129,  2052,
          2017,  6235, 18961,  1029,   102,  1037,  2204,  6926,
102,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0],
        [  101,  5863,  2170,  1996,  4614,  2043,  2002,  4384,
1996, 13742,
          3875,  1996, 21071,  1012,   102,  2054,  2097,  5863,
2215,  2000,
          2079,  2279,  1029,   102, 10574,  1996, 10345,   102,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0],
        [  101,  7546,  3427,  2998,  2006,  2694,  2005,  1037,
2261,  2847,
          1998,  2001,  2200,  7622,  1012,   102,  2129,  2052,
7546,  2514,
          5728,  1029,   102,  7777,  2998,  1037,  2843,   102,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0],
```

```
        [  101,  3994,  4778,  2037,  2767,  2058,  2000,  3422,
1996,  4164,
          2006,  3477,  1011,  2566,  1011,  3193,  1012,   102,
2054,  2097,
          3994,  2215,  2000,  2079,  2279,  1029,   102,  5342,
2007,  2035,
          1996,  4597,  2125,  1998, 14694,  2701,  2043,  2014,
2767, 21145,
          2006,  1996,  2341,   102],
        [  101,  2096,  2667,  2000,  8054,  2037,  2814,  2000,
3693,  1996,
          2012,  2277,  1010,  6683,  4207,  2014,  9029,  2055,
4676,  1012,
           102,  2054,  2097,  4148,  2000,  6683,  1029,   102,
6848,  4676,
          2007,  2014,  2814,   102,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0],
        [  101, 22712,  2170,  1996,  2158,  2055,  1996,  3291,
2027,  2020,
          2383,  1012,   102,  2054,  2097,  2500,  2215,  2000,
2079,  2279,
          1029,   102,  9611,  1996,  3291,   102,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0],
        [  101,  7546,  2134,  1005,  1056,  3046,  2004,  2524,
2004,  2016,
          2288,  1998,  2288,  2353,  2173,  1012,   102,  2129,
2052,  2017,
          6235,  7546,  1029,   102, 12774,  1998,  2844,   102,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0],
        [  101, 18403,  1005,  1055, 10808,  2291,  3844,  2091,
2021,  2027,
          2018,  1037, 13788,  1012,   102,  2129,  2052,  2017,
6235, 18403,
          1029,   102,  2001,  8794,  2027,  2018,  1037, 13788,
102,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0],
        [  101,  9036,  2001,  1040,  7274,  2571,  9048,  2278,
2061, 14509,
          2949,  2037,  4646,  2005,  1037,  3105,  2061,  2008,
9036,  2052,
```

```
          2131,  2019,  4357,  1998,  2265,  2037,  4813,  2059,
1012,   102,
          2129,  2052, 14509,  2514,  5728,  1029,   102,  2200,
10791,   102,
             0,     0,     0,     0],
        [  101, 22712,  2165, 27970,  1005,  1055,  2769,  2029,
12781,  2006,
          2010,  9715,  4600,  1012,   102,  2129,  2052, 22712,
2514,  5728,
          1029,   102,  2200,  5905,   102,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0],
        [  101, 18403,  2435,  9321,  2298,  2138,  2016,  2001,
2035,  2039,
          1999,  2014,  2449,  1012,   102,  2054,  2097, 18403,
2215,  2000,
          2079,  2279,  1029,   102,  2131,  1037,  2047,  2767,
102,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0],
        [  101,  9806,  2001,  4394,  2005,  4466,  2847,  1012,
2111,  2020,
          2559,  7249,  1012,  9806,  2170,  2188,  1012,   102,
2129,  2052,
          2500,  2514,  2004,  1037,  2765,  1029,   102,  5506,
2012,  9806,
           102,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0],
        [  101,  7627,  2001,  2012,  2188,  2007,  2037,  2155,
1010,  1998,
          7627,  2170,  2068,  2035,  2046,  1996,  2542,  2282,
1012,   102,
          2339,  2106,  7627,  2079,  2023,  1029,   102,  2359,
2000,  2022,
          2187,  2894,   102,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0],
        [  101, 10555,  2985,  2051,  2000,  5335,  2014,  7022,
1012,   102,
          2054,  2097, 10555,  2215,  2000,  2079,  2279,  1029,
102,  2156,
          2065,  2009,  3271,   102,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
```

```
             0,      0,      0,      0]])}
batch_attn_mask:
 {'A': tensor([[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, 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, 1,
1, 1, 1, 1,
         0, 0, 0, 0, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
         1, 1, 1, 1, 1, 0, 0, 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, 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, 1, 1,
1, 1, 1, 1,
         1, 0, 0, 0, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 0, 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, 1, 1,
1, 1, 1, 1,
         1, 1, 0, 0, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
         1, 1, 1, 1, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
         0, 0, 0, 0, 0, 0, 0, 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, 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, 1,
         1, 0, 0, 0, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
         1, 1, 1, 1, 0, 0, 0, 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, 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, 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, 1, 1, 1, 1,
1, 1, 1, 0,
```

```
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]), 'B':
tensor([[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, 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, 1, 1,
1, 1, 1, 1,
           0, 0, 0, 0, 0, 0, 0, 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, 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, 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, 1, 1,
1, 1, 1, 1,
           1, 1, 0, 0, 0, 0, 0, 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, 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, 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, 1, 1,
1, 1, 1, 1,
           1, 1, 1, 0, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 0, 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, 1, 1,
1, 1, 1, 1,
           1, 1, 1, 1, 1, 0, 0, 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, 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, 1,
           1, 0, 0, 0, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
           1, 1, 1, 0, 0, 0, 0, 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, 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, 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, 1, 1, 1, 1,
1, 1, 1, 1,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]), 'C':
tensor([[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, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
          1, 1, 1, 1, 1, 0, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
          1, 1, 1, 1, 0, 0, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
          1, 1, 1, 1, 0, 0, 0, 0, 0, 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, 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, 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, 1,
1, 1, 1, 1,
          1, 1, 0, 0, 0, 0, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
          1, 1, 1, 1, 0, 0, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
          1, 1, 1, 1, 1, 0, 0, 0, 0, 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, 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],
         [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, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
          1, 1, 1, 1, 1, 0, 0, 0, 0, 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, 1, 1,
1, 1, 1, 1,
          1, 1, 1, 1, 1, 1, 1, 0, 0, 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, 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, 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,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]])}
```

```
batch_labels:
 tensor([2, 0, 0, 2, 1, 0, 2, 2, 1, 1, 0, 2, 1, 1, 1, 2])
```

Task 2: Implementing and Training BERT-based Multiple Choice Classifier (1 hour 30 minutes)

Similar to pretrained tokenizers, the transformers library also provide numerous pre-trained language models that can be fine-tuned on a wide variety of downstream tasks. We demonstrate usage of these models below.

```
# 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
```


{"model_id":"0d6e695936b14c2eb56a9d6d3810c92e","version_major":2,"version_minor":0}


```
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=True)
            (key): Linear(in_features=768, out_features=768,
bias=True)
            (value): Linear(in_features=768, out_features=768,
bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768,
bias=True)
            (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=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
  (pooler): BertPooler(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (activation): Tanh()
  )
)
```

As you can see very similar to how we created pre-trained tokenizer, we can load a pretrained BERT model by calling `BertModel.from_pretrained(bert-base-uncased)`. This can actually be considered just a Pytorch `nn.Module` like `nn.Linear` and can be similarly plugged into a network architecture. Also, notice the model contains 12 BERT layers, where each layer consists of a Self Attention layer followed by a sequence of linear layers and activation functions (MLP), as we discussed when talking about Transformer architecture in the lecture.

```
sentence = "kendall was a person who kept her word exquisitely, so she
got my money the other day"
tokenizer_output = bert_tokenizer(sentence, return_tensors="pt")
input_ids, attn_mask = tokenizer_output["input_ids"],
tokenizer_output["attention_mask"]

output = bert_model(input_ids, attention_mask = attn_mask)
output

BaseModelOutputWithPoolingAndCrossAttentions(last_hidden_state=tensor(
[[[ 0.2823, -0.2353, -0.3529,  ..., -0.0834,  0.2548,  0.4870],
         [ 0.4055, -1.1768, -0.2842,  ..., -0.3740,  0.3920, -0.4480],
         [ 0.0377, -0.7788, -0.1174,  ..., -0.4201, -0.3078,  0.1824],
         ...,
         [-1.1595, -1.5650, -0.2526,  ..., -0.4569, -0.5474,  0.2315],
         [-1.0644, -0.5952, -0.3912,  ...,  0.2788, -0.0207, -0.1262],
         [ 0.5158,  0.4573,  0.0263,  ...,  0.1445, -0.6398, -
0.5258]]],
       grad_fn=<NativeLayerNormBackward0>), pooler_output=tensor([[-
7.1342e-01, -3.3350e-01, -4.4551e-01,  4.5434e-01,  5.0347e-01,
```

```
         -8.4619e-02,  4.8077e-01,  8.4160e-02, -2.2890e-01, -9.9974e-
01,
         -2.4549e-01,  7.9481e-01,  9.8030e-01, -2.8880e-02,  9.1588e-
01,
         -2.4136e-01,  5.3666e-01, -5.0656e-01,  2.2218e-01,  3.2123e-
01,
          6.4942e-01,  9.9757e-01,  4.7173e-01,  2.0766e-01,  3.6500e-
01,
          8.8725e-01, -4.2194e-01,  9.3109e-01,  9.1743e-01,  7.2114e-
01,
         -1.4355e-02, -1.8941e-02, -9.8951e-01,  3.0721e-02, -2.8992e-
01,
         -9.8191e-01,  1.7840e-01, -5.9347e-01,  1.1879e-01,  2.5965e-
01,
         -8.6462e-01,  1.4619e-01,  9.9952e-01, -4.7740e-01,  1.0605e-
01,
         -1.2106e-01, -9.9986e-01,  2.2875e-01, -8.5771e-01,  5.0007e-
01,
          2.7700e-01,  7.1537e-01,  9.0395e-02,  3.0770e-01,  3.6540e-
01,
         -7.2003e-02, -2.7237e-01, -2.9246e-02, -2.4881e-01, -4.1984e-
01,
         -6.4784e-01,  1.4369e-01, -5.4503e-01, -7.8703e-01,  4.1153e-
01,
          2.9400e-01, -1.3110e-01, -8.3965e-02,  6.9508e-03, -1.4154e-
01,
          6.8059e-01,  8.8732e-02,  1.5826e-02, -8.0095e-01,  1.2098e-
01,
          1.5737e-01, -5.1418e-01,  1.0000e+00,  4.2994e-02, -9.8423e-
01,
          1.7809e-01,  2.3727e-01,  3.4371e-01,  4.4902e-01, -2.5424e-
01,
         -1.0000e+00,  3.7378e-01, -2.9287e-03, -9.9117e-01,  1.0483e-
01,
          4.8637e-01, -2.1140e-01, -1.3496e-01,  4.0792e-01,  5.8095e-
02,
         -1.5223e-01, -2.3065e-01, -4.2501e-01, -1.0952e-01, -1.6091e-
01,
          2.2054e-01,  7.6145e-02, -4.6461e-02, -2.1545e-01,  2.0504e-
01,
         -4.3479e-01,  2.3062e-02,  4.2032e-01, -3.0370e-01,  5.1830e-
01,
          3.7295e-01, -1.9882e-01,  2.8470e-01, -9.4566e-01,  3.9429e-
01,
         -2.5140e-01, -9.8011e-01, -4.6321e-01, -9.9085e-01,  6.2031e-
01,
          3.0362e-02, -2.4470e-01,  9.4980e-01,  4.4613e-01,  1.0056e-
01,
          1.2657e-01, -4.9239e-01, -1.0000e+00, -3.8979e-01,  1.7306e-
```

```
02,
          6.2905e-02,  7.5699e-03, -9.6858e-01, -9.6126e-01,  3.2523e-
01,
          9.4696e-01,  2.4653e-02,  9.9795e-01, -1.0900e-01,  9.2609e-
01,
          2.7076e-01, -2.3979e-01,  2.4420e-03, -4.1432e-01,  3.1651e-
01,
         -2.5690e-01, -5.1866e-02,  1.3838e-01, -1.3608e-01,  1.5090e-
02,
         -2.7091e-01,  1.0166e-02, -6.0731e-02, -9.0828e-01, -2.3963e-
01,
          9.5270e-01, -1.2421e-01, -5.1564e-01,  4.5463e-01, -9.4697e-
02,
          1.9799e-02,  6.7040e-01,  3.5773e-01,  3.0287e-01, -3.0182e-
01,
          3.2714e-01, -3.3120e-01,  4.4667e-01, -5.9530e-01,  3.9171e-
01,
          2.3601e-01, -1.9295e-01, -1.1556e-01, -9.7889e-01, -2.1943e-
01,
          1.7873e-01,  9.8326e-01,  6.1193e-01,  1.2009e-01,  4.6305e-
01,
         -2.2367e-01,  5.1362e-01, -9.4201e-01,  9.8198e-01,  2.3595e-
02,
          1.6000e-01, -1.6324e-01,  2.4670e-01, -8.0511e-01, -3.3924e-
01,
          4.8846e-01, -5.1100e-01, -7.3509e-01,  4.6975e-02, -3.3179e-
01,
         -1.8198e-01, -4.1462e-01,  1.0465e-01, -2.1432e-01, -3.4277e-
01,
          9.7266e-02,  9.3661e-01,  7.2161e-01,  4.8405e-01, -3.6871e-
01,
          3.1814e-01, -8.4417e-01, -4.6677e-01,  2.3358e-02,  8.2281e-
02,
         -3.5102e-02,  9.9018e-01, -2.8120e-01,  1.5138e-01, -8.7715e-
01,
         -9.8174e-01, -1.8074e-01, -8.3024e-01, -1.6251e-01, -4.6465e-
01,
          4.1388e-01, -5.0448e-01,  7.9151e-03,  1.6464e-01, -8.4324e-
01,
         -5.9927e-01,  3.1949e-01, -1.7097e-01,  3.2296e-01, -2.7389e-
01,
          9.0340e-01,  6.0541e-01, -4.7819e-01, -3.6939e-01,  9.1910e-
01,
         -2.9349e-01, -7.2003e-01,  3.8927e-01, -1.0532e-01,  5.2019e-
01,
         -4.1045e-01,  9.4339e-01,  6.6610e-01,  4.9983e-01, -8.7163e-
01,
          1.0599e-02, -6.2514e-01,  5.1351e-02,  7.7707e-04, -5.0177e-
01,
```

```
          2.8171e-01,  4.3400e-01,  2.9356e-01,  7.4288e-01, -4.1969e-
02,
          8.6796e-01, -9.1876e-01, -9.4036e-01, -7.8274e-01,  9.0406e-
02,
         -9.8649e-01,  1.0411e-01,  1.7797e-01, -1.0403e-01, -2.1737e-
01,
         -1.7412e-01, -9.4650e-01,  3.6575e-01, -4.9280e-02,  9.1307e-
01,
         -3.6831e-01, -6.6574e-01, -4.2411e-01, -9.2306e-01, -2.2497e-
01,
         -1.3000e-01,  6.1594e-02, -1.6496e-01, -9.4146e-01,  4.3231e-
01,
          4.3532e-01,  4.4700e-01, -1.5150e-01,  9.6835e-01,  9.9996e-
01,
          9.6499e-01,  8.9856e-01,  4.0772e-01, -9.9106e-01, -7.2477e-
01,
          9.9990e-01, -8.8650e-01, -9.9999e-01, -8.8209e-01, -3.2363e-
01,
         -1.0676e-02, -1.0000e+00, -6.8102e-02,  2.3007e-01, -8.1374e-
01,
          1.3577e-01,  9.7153e-01,  9.1199e-01, -1.0000e+00,  7.6972e-
01,
          9.3895e-01, -4.7636e-01,  7.2471e-01, -1.8839e-01,  9.6564e-
01,
          2.1396e-01,  3.6877e-01, -6.0258e-02,  2.6505e-01, -5.1107e-
01,
         -4.2091e-01,  1.9413e-02, -2.5991e-01,  9.7003e-01, -2.1713e-
02,
         -4.3161e-01, -8.6556e-01,  2.8448e-01,  5.9014e-02, -4.4330e-
01,
         -9.4856e-01, -1.1291e-01,  2.5513e-02,  3.7114e-01,  9.9050e-
03,
          5.2219e-02, -3.4138e-01, -3.1813e-02, -3.0110e-01, -5.4604e-
02,
          5.3167e-01, -9.1741e-01, -2.2246e-01, -1.0530e-02, -4.1509e-
01,
          3.0833e-01, -9.7295e-01,  9.5327e-01, -2.9202e-01,  5.2656e-
01,
          1.0000e+00, -1.5967e-02, -7.8224e-01,  2.5205e-01,  8.0371e-
02,
         -1.1654e-01,  1.0000e+00,  5.2342e-01, -9.7992e-01, -4.5646e-
01,
          4.4127e-01, -3.6342e-01, -5.4687e-01,  9.9663e-01, -1.6571e-
01,
         -2.4606e-01,  7.1765e-02,  9.8694e-01, -9.8773e-01,  9.2531e-
01,
         -7.8466e-01, -9.7499e-01,  9.5526e-01,  9.4014e-01, -3.1455e-
02,
         -3.7029e-01, -8.3566e-02,  7.2885e-02,  7.8481e-02, -8.5670e-
```

```
01,
          2.4106e-01,  1.2382e-01,  2.2440e-02,  9.0840e-01,  4.9766e-
02,
         -4.8200e-01,  1.0067e-01, -3.3606e-01,  2.7572e-01,  5.1258e-
01,
          3.4788e-01,  2.0840e-02, -4.1822e-02,  2.2078e-02, -5.3827e-
01,
         -9.6365e-01,  5.3595e-01,  1.0000e+00,  1.7988e-01,  2.2127e-
01,
          1.1171e-01,  4.2887e-02, -2.8482e-01,  2.7275e-01,  2.8186e-
01,
         -1.9890e-01, -6.9904e-01,  5.4585e-01, -8.2020e-01, -9.8801e-
01,
          3.9557e-01,  1.4115e-01, -9.0601e-02,  9.9788e-01,  5.2465e-
02,
          8.7988e-02, -6.1069e-02,  8.0411e-01, -1.0107e-01,  4.5896e-
02,
          2.8721e-01,  9.7186e-01, -5.0391e-02,  4.3895e-01,  5.3262e-
01,
         -3.7363e-01, -1.4349e-01, -5.3335e-01, -1.7289e-01, -9.3699e-
01,
          2.4583e-01, -9.5441e-01,  9.4962e-01,  6.8467e-01,  2.8843e-
01,
          2.7844e-02,  1.9399e-01,  1.0000e+00, -5.5154e-01,  2.7470e-
01,
          7.3334e-01,  2.1665e-01, -9.9376e-01, -6.3638e-01, -4.0457e-
01,
          8.1165e-02, -1.0932e-01, -1.6521e-01,  1.1090e-01, -9.6194e-
01,
          1.4183e-01,  2.6996e-01, -9.0304e-01, -9.8854e-01, -1.8411e-
01,
         -1.4559e-01,  1.0233e-01, -8.8277e-01, -4.2014e-01, -5.6760e-
01,
          2.0938e-01, -7.8962e-02, -9.2779e-01,  3.7535e-01, -3.2771e-
01,
          3.7110e-01, -1.7455e-02,  4.4611e-01,  3.7664e-01,  8.9600e-
01,
         -1.4597e-01, -4.1541e-02, -2.2711e-02, -5.6408e-01,  4.9451e-
01,
         -6.0685e-01, -5.7528e-01,  1.6080e-02,  1.0000e+00, -2.4831e-
01,
          4.7874e-01,  3.2856e-01,  4.0354e-01,  3.5562e-02,  7.1661e-
02,
          5.1131e-01,  1.7113e-01, -5.2168e-04, -2.9695e-01,  7.3751e-
01,
         -1.8529e-01,  4.4554e-01,  2.2911e-01,  2.8562e-02,  7.7319e-
01,
          5.5270e-01,  1.0272e-01,  2.6211e-01, -1.2720e-03,  9.7222e-
01,
```

```
         -2.8842e-02,  5.6738e-02, -2.8775e-01, -1.6172e-02, -1.9353e-
01,
          4.9592e-01,  1.0000e+00,  8.4379e-02, -1.5369e-01, -9.8806e-
01,
         -3.3429e-01, -7.0185e-01,  9.9986e-01,  7.6013e-01, -6.0292e-
01,
          3.8539e-01,  2.5654e-01, -1.1947e-01,  3.1904e-01, -2.9968e-
02,
         -8.2039e-02, -3.6788e-02, -4.1350e-02,  9.4251e-01, -3.8563e-
01,
         -9.6644e-01, -1.8887e-01,  3.3731e-01, -9.5321e-01,  9.9444e-
01,
         -3.1648e-01, -7.6926e-02, -2.2873e-01, -1.2125e-01, -8.2212e-
01,
         -1.8324e-01, -9.8104e-01,  3.3368e-03,  6.5977e-02,  9.6485e-
01,
          6.2124e-02, -4.1838e-01, -8.8928e-01,  4.6512e-01,  1.6997e-
01,
         -4.9288e-01, -9.0313e-01,  9.4608e-01, -9.6796e-01,  4.0968e-
01,
          9.9998e-01,  2.5584e-01, -4.0539e-01,  1.3442e-01, -1.9956e-
01,
          2.1901e-01, -7.6149e-02,  4.1793e-01, -9.3571e-01, -2.5734e-
01,
         -2.6220e-02,  1.9381e-01, -1.0859e-02,  1.0624e-02,  5.5476e-
01,
          1.3884e-01, -3.7362e-01, -5.1091e-01, -2.0794e-02,  2.3886e-
01,
          4.4453e-01, -1.9578e-01,  2.4322e-02,  1.2190e-01,  4.5708e-
02,
         -8.9249e-01, -2.3091e-01, -2.3275e-01, -9.9933e-01,  3.8215e-
01,
         -1.0000e+00,  2.4109e-01, -3.3744e-01, -9.5531e-02,  7.7688e-
01,
          6.7944e-01,  5.0124e-01, -5.4854e-01, -4.4367e-01,  7.4304e-
01,
          6.9326e-01, -1.1128e-01,  5.8066e-02, -5.1514e-01, -1.8508e-
02,
          4.7066e-02, -3.9159e-02, -1.2526e-01,  6.4872e-01, -2.8116e-
01,
          1.0000e+00,  9.4141e-02, -1.6344e-01, -8.3718e-01,  9.6640e-
02,
         -1.5380e-01,  9.9999e-01, -4.3615e-01, -9.4652e-01,  1.8444e-
01,
         -3.2900e-01, -7.3108e-01,  2.8296e-01, -1.6438e-01, -5.5741e-
01,
         -5.1496e-01,  9.4393e-01,  1.5646e-01, -4.8041e-01,  3.7670e-
01,
         -7.6502e-02, -3.4085e-01, -2.3005e-01,  4.6745e-01,  9.8561e-
```

```
01,
        1.8930e-01,  6.5902e-01, -1.4607e-01, -5.6216e-02,  9.6718e-
01,
        2.0109e-01, -4.4703e-01,  2.6056e-03,  1.0000e+00,  2.5435e-
01,
       -8.5391e-01,  1.4343e-01, -9.7004e-01,  6.7542e-02, -9.1234e-
01,
        2.3352e-01, -1.0207e-01,  9.0571e-01, -1.2382e-01,  8.9837e-
01,
       -2.5035e-01, -9.6540e-02, -1.3500e-02,  7.9872e-02,  3.4886e-
01,
       -9.0798e-01, -9.8696e-01, -9.8399e-01,  2.2660e-01, -2.6349e-
01,
       -1.7568e-03,  2.5412e-01, -4.6728e-02,  3.2300e-01,  2.3782e-
01,
       -1.0000e+00,  9.4718e-01,  2.8214e-01,  5.5079e-01,  9.5964e-
01,
        4.3230e-01,  2.9824e-01,  1.1796e-01, -9.8534e-01, -9.1499e-
01,
       -2.2060e-01, -2.2493e-01,  4.2716e-01,  4.1828e-01,  7.7361e-
01,
        2.5652e-01, -4.4088e-01, -5.4283e-01, -1.9669e-01, -9.2347e-
01,
       -9.9092e-01,  1.6564e-01, -2.1975e-02, -7.5007e-01,  9.5135e-
01,
       -4.2565e-01,  4.1158e-02,  3.4651e-01, -3.7918e-01,  5.3755e-
01,
        6.6361e-01, -1.7167e-01, -1.6186e-01,  3.5413e-01,  8.6273e-
01,
        5.4597e-01,  9.7368e-01, -3.1056e-01,  3.8867e-01, -3.5970e-
01,
        3.0381e-01,  7.2659e-01, -8.7832e-01,  4.5919e-02,  5.9937e-
02,
        1.7157e-01,  1.6686e-01, -1.6294e-01, -7.8675e-01,  4.3243e-
02,
       -2.4215e-01,  9.7457e-02, -2.9032e-01,  2.1996e-01, -2.2742e-
01,
        4.9331e-02, -3.6432e-01, -2.3407e-01,  5.0983e-01, -1.0163e-
01,
        9.1016e-01,  5.7243e-01,  4.2006e-02, -4.0053e-01, -9.6790e-
03,
       -1.7138e-01, -8.6321e-01,  6.2138e-01,  2.0165e-01,  2.1101e-
01,
        1.2278e-01, -1.7986e-01,  8.6453e-01, -5.4444e-01, -3.0941e-
01,
       -3.3675e-01, -3.4412e-01,  7.0476e-01, -5.8997e-01, -3.5156e-
01,
       -5.8885e-02,  4.5485e-01,  1.9294e-01,  9.9796e-01, -1.2719e-
01,
```

```
        -3.9385e-01, -3.1919e-01, -2.4795e-01,  2.6886e-01,  3.8581e-
02,
        -1.0000e+00,  1.7946e-01, -1.6161e-01, -9.5156e-02, -2.4447e-
01,
         3.1448e-01, -3.1113e-01, -9.0355e-01, -1.8078e-02,  5.0788e-
01,
         4.1148e-01, -4.0674e-01, -3.7335e-01,  4.8784e-01, -2.1414e-
01,
         8.2619e-01,  8.2344e-01, -1.7268e-01,  6.6992e-01,  5.1202e-
01,
        -4.5926e-01, -5.4121e-01,  9.0145e-01]],
grad_fn=<TanhBackward0>), hidden_states=None, past_key_values=None,
attentions=None, cross_attentions=None)
```

As you can see, calling `bert_model` returns a bunch of different things. Let's go through them one by one and understand

```
last_hidden_state = output.last_hidden_state
print(f"input_ids shape: {input_ids.shape}")
print(f"last_hidden_state shape: {last_hidden_state.shape}")

input_ids shape: torch.Size([1, 21])
last_hidden_state shape: torch.Size([1, 21, 768])
```

For an input of shape `[1,21]` which just means a single sequence of 21 tokens, last_hidden_state is a tensor of shape `[1, 21, 768]` denoting the contextual embedding of each of the 21 tokens in the sequence. These representations can be then used for solving a downstream task, by adding a linear layer or MLP layer on top. These can be useful for sequence labelling type of tasks.

```
pooler_output = output.pooler_output
print(f"input_ids shape: {input_ids.shape}")
print(f"pooler_output shape: {pooler_output.shape}")

input_ids shape: torch.Size([1, 21])
pooler_output shape: torch.Size([1, 768])
```

`pooler_output` is an aggregate representation of the entire sentence and can be thought of as a sentence embedding. It is obtained by passing the representation of the [CLS] token through a linear layer. This can be useful for sentence-level tasks like sentiment analysis as well as multiple choice classification tasks etc.

Apart from these two we can also obtain other values by providing additional arguments. Like if we want to obtain attention maps which can be useful for interpretating the model's behavior, we can just specify `output_attentions=True` while calling the model

```
output = bert_model(input_ids, attention_mask = attn_mask,
output_attentions=True)
attentions = output.attentions
```

```
print(f"Data type of attentions output: {type(attentions)}")
print(f"Number of elements: {len(attentions)}")
print(f"Shape of individual element: {attentions[0].shape}")
print(f"Example attention map: {attentions[0][0,0]}")
```

```
Data type of attentions output: <class 'tuple'>
Number of elements: 12
Shape of individual element: torch.Size([1, 12, 21, 21])
Example attention map: tensor([[0.0365, 0.0188, 0.0239, 0.0918,
0.0375, 0.0409, 0.0390, 0.0327, 0.0256,
          0.0260, 0.0488, 0.0387, 0.0532, 0.0188, 0.0251, 0.0408,
0.0321, 0.0840,
          0.0793, 0.0262, 0.1801],
         [0.0178, 0.0460, 0.0240, 0.0306, 0.0482, 0.0194, 0.0510,
0.0919, 0.0260,
          0.0818, 0.0425, 0.0482, 0.0159, 0.0576, 0.0703, 0.1526,
0.0337, 0.0405,
          0.0194, 0.0371, 0.0454],
         [0.0527, 0.0466, 0.0899, 0.0241, 0.0464, 0.0245, 0.0645,
0.0587, 0.0256,
          0.0401, 0.0298, 0.0606, 0.0302, 0.0735, 0.1222, 0.0573,
0.0297, 0.0204,
          0.0224, 0.0539, 0.0268],
         [0.0425, 0.0528, 0.0454, 0.0338, 0.0402, 0.0353, 0.0515,
0.0666, 0.0397,
          0.0464, 0.0336, 0.0518, 0.0492, 0.0609, 0.0529, 0.0731,
0.0509, 0.0580,
          0.0381, 0.0492, 0.0280],
         [0.0492, 0.1514, 0.0892, 0.0078, 0.0237, 0.0282, 0.0478,
0.0639, 0.0288,
          0.0346, 0.0376, 0.0427, 0.0386, 0.0865, 0.0633, 0.0612,
0.0589, 0.0099,
          0.0217, 0.0326, 0.0225],
         [0.0328, 0.0851, 0.0669, 0.0210, 0.0342, 0.0278, 0.0618,
0.0965, 0.0244,
          0.0388, 0.0382, 0.0554, 0.0281, 0.0809, 0.0671, 0.0634,
0.0372, 0.0276,
          0.0302, 0.0289, 0.0536],
         [0.0390, 0.0402, 0.1059, 0.0350, 0.0359, 0.0453, 0.0443,
0.0639, 0.0261,
          0.0437, 0.0524, 0.0837, 0.0440, 0.0611, 0.0661, 0.0560,
0.0265, 0.0284,
          0.0247, 0.0368, 0.0410],
         [0.0637, 0.0480, 0.0736, 0.0488, 0.0387, 0.0301, 0.0386,
0.1177, 0.0174,
          0.0286, 0.0322, 0.0448, 0.0351, 0.0715, 0.0595, 0.0795,
0.0202, 0.0495,
          0.0239, 0.0297, 0.0488],
         [0.0379, 0.0930, 0.0471, 0.0373, 0.0634, 0.0240, 0.0501,
0.0724, 0.0077,
```

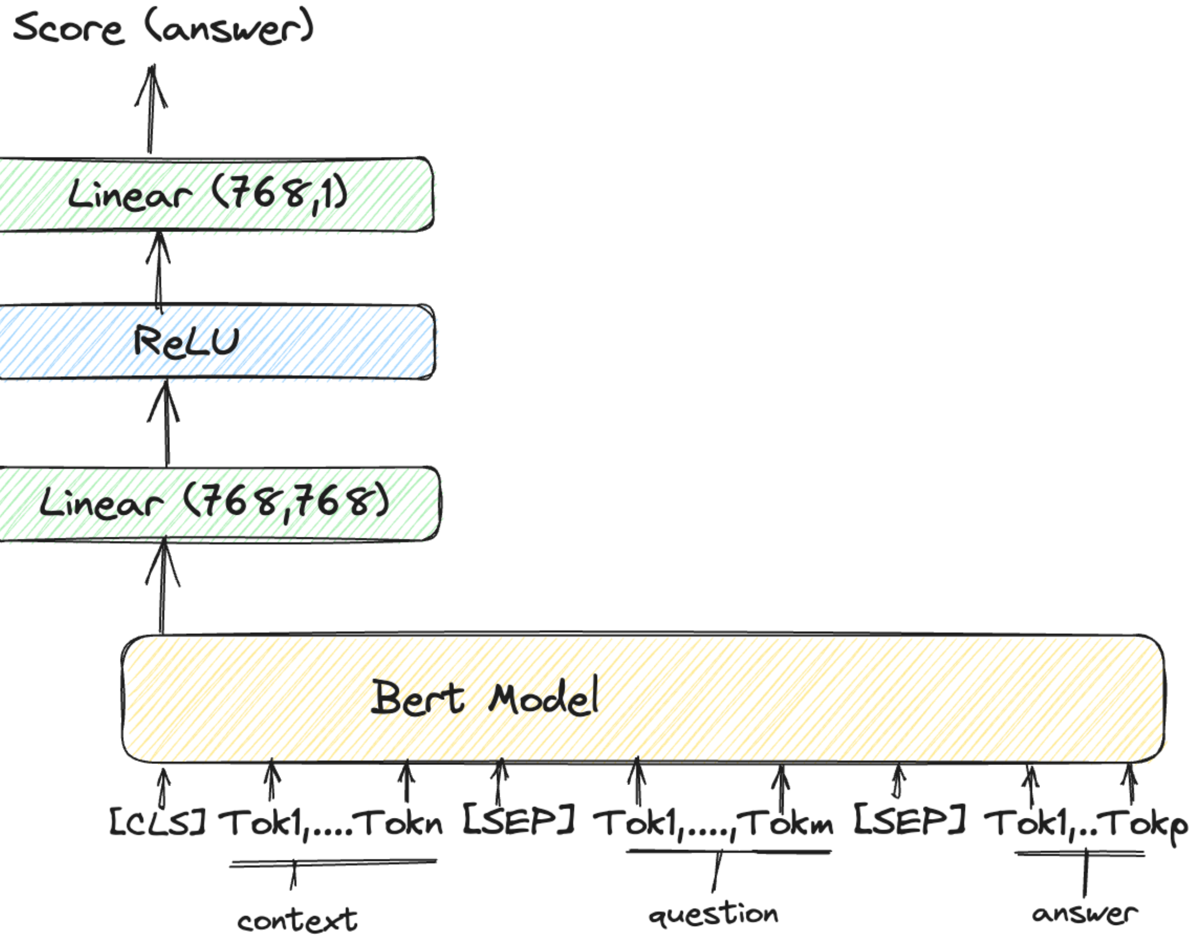
```
          0.0576, 0.1025, 0.0672, 0.0309, 0.0510, 0.0276, 0.0513,
0.0243, 0.0229,
          0.0249, 0.0429, 0.0641],
         [0.0287, 0.1002, 0.0683, 0.0142, 0.0493, 0.0313, 0.0819,
0.0516, 0.0350,
          0.0476, 0.0602, 0.0453, 0.0288, 0.0470, 0.1209, 0.0661,
0.0310, 0.0140,
          0.0197, 0.0225, 0.0367],
         [0.0285, 0.0755, 0.0461, 0.0240, 0.0530, 0.0329, 0.0516,
0.0549, 0.0636,
          0.0900, 0.0263, 0.0202, 0.0290, 0.0448, 0.0628, 0.0496,
0.0781, 0.0263,
          0.0399, 0.0470, 0.0559],
         [0.0478, 0.0464, 0.0582, 0.0576, 0.0513, 0.0424, 0.0580,
0.0508, 0.0362,
          0.0565, 0.0378, 0.0387, 0.0565, 0.0367, 0.0458, 0.0434,
0.0477, 0.0464,
          0.0433, 0.0513, 0.0475],
         [0.0320, 0.0923, 0.0587, 0.0300, 0.0370, 0.0399, 0.0736,
0.1011, 0.0183,
          0.0490, 0.0473, 0.0639, 0.0212, 0.0637, 0.0596, 0.0799,
0.0234, 0.0308,
          0.0306, 0.0149, 0.0330],
         [0.0335, 0.0893, 0.1025, 0.0227, 0.0361, 0.0223, 0.0555,
0.1325, 0.0104,
          0.0403, 0.0382, 0.0476, 0.0227, 0.0636, 0.0781, 0.0949,
0.0216, 0.0269,
          0.0159, 0.0252, 0.0202],
         [0.0544, 0.1423, 0.0956, 0.0254, 0.0281, 0.0320, 0.0471,
0.0546, 0.0136,
          0.0673, 0.0418, 0.0395, 0.0531, 0.0422, 0.0400, 0.1038,
0.0275, 0.0331,
          0.0201, 0.0151, 0.0234],
         [0.0403, 0.0612, 0.0584, 0.0240, 0.0489, 0.0368, 0.0703,
0.0585, 0.0444,
          0.0276, 0.0329, 0.0323, 0.0453, 0.0484, 0.1187, 0.0948,
0.0410, 0.0257,
          0.0229, 0.0184, 0.0490],
         [0.0249, 0.0599, 0.0252, 0.0498, 0.0809, 0.0517, 0.0329,
0.0799, 0.0372,
          0.0461, 0.0336, 0.0263, 0.0299, 0.0659, 0.0677, 0.0610,
0.0249, 0.0287,
          0.0350, 0.0519, 0.0864],
         [0.0298, 0.0571, 0.0516, 0.0481, 0.0414, 0.0276, 0.0582,
0.0824, 0.0294,
          0.0469, 0.0406, 0.0386, 0.0444, 0.0660, 0.0562, 0.0832,
0.0321, 0.0636,
          0.0414, 0.0343, 0.0273],
         [0.0209, 0.0511, 0.0464, 0.0348, 0.0476, 0.0474, 0.0860,
```

```
0.0561, 0.0389,
         0.0365, 0.0628, 0.0366, 0.0508, 0.0486, 0.0764, 0.0663,
0.0431, 0.0291,
         0.0349, 0.0329, 0.0528],
        [0.0228, 0.1058, 0.1149, 0.0203, 0.0668, 0.0299, 0.0469,
0.0448, 0.0492,
         0.0439, 0.0376, 0.0540, 0.0458, 0.0640, 0.0903, 0.0373,
0.0353, 0.0134,
         0.0297, 0.0104, 0.0368],
        [0.0342, 0.0267, 0.0292, 0.0788, 0.0409, 0.0381, 0.0354,
0.0651, 0.0243,
         0.0478, 0.0643, 0.0683, 0.0461, 0.0384, 0.0317, 0.0849,
0.0342, 0.0737,
         0.0565, 0.0304, 0.0511]], grad_fn=<SelectBackward0>)
```

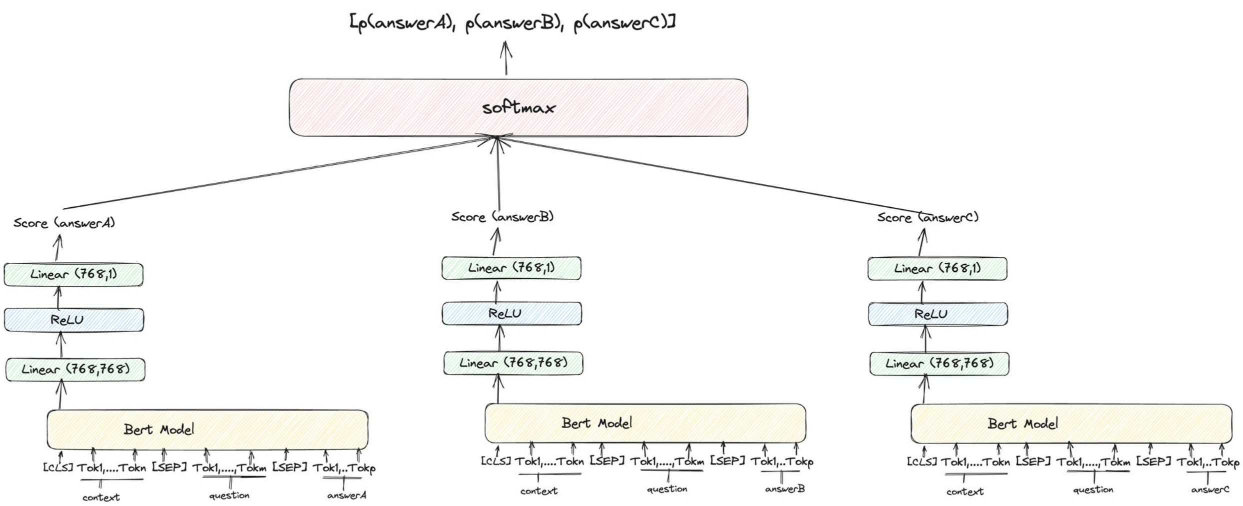
As you can see `attentions` is a tuple containing 12 elements which corresponds to the attention maps of each of the 12 layers in the network. Further each layer's attention maps also contains 12 attention maps corresponding to 12 heads in each layer. A single attention map as you can see is a 18x18 matrix representing the attention pattern for all the tokens in the sequence

Task 2.1: Implementing BERT-based Classifier for Multiple Choice Classification

In this task you will implement a bert-based classifier in Pytorch very similar to how we created bag of word classifiers in the previous assignments. The architecture of the model is as follows:



Essentially, what we have here is a model that takes a context and question, and scores a particular answer (denoted as a score(a)). At the backbone we have the BERT model, using which we obtain the contextualized representation of the [context, 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. To predict the correct answer, score each of the three answers, obtain their scores and normalize them 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:

```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 output i.e. (Linear(d_hidden, d_hidden) -> ReLU -> Linear(d_hidden, 1)),
        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 = None
        self.mlp_layer = None
        self.log_softmax_layer = None

        # YOUR CODE HERE
        self.bert_layer = BertModel.from_pretrained(bert_variant)
        self.mlp_layer = nn.Sequential(
            nn.Linear(d_hidden, d_hidden),
            nn.ReLU(),
            nn.Linear(d_hidden, 1)
        )
```

```
        self.log_softmax_layer = nn.LogSoftmax(dim=-1)

    def forward(self, input_ids_dict, attn_mask_dict):
        """
        Forward Passes the inputs through the network and obtains the
prediction

        Inputs:
            - input_ids_dict (dict(str,torch.tensor)): A dictionary
containing input_ids corresponding to each answer choice. Keys are A,
B and C and value is a torch tensor of shape [batch_size, seq_len]
                                        representing the sequence of
token ids
            - attn_mask_dict (dict(str,torch.tensor)): A dictionary
containing attention mask corresponding to each answer choice. Keys
are A, B and C and value is a torch tensor of shape [batch_size,
seq_len]

        Returns:
          - output (torch.tensor): A torch tensor of shape
[batch_size,] obtained after passing the input to the network


        Hints:
            1. Recall which of the outputs from BertModel is
appropriate for the sentence classification task and how to access it.
            2. `torch.cat` might come in handy before performing
softmax
        """
        output = None
        key_outs = []
        # YOUR CODE HERE
        for key in input_ids_dict.keys():
            bert_output = self.bert_layer(input_ids_dict[key],
attention_mask = attn_mask_dict[key])
            pooler_output = bert_output.pooler_output
            mlp_output = self.mlp_layer(pooler_output)
            key_outs.append(mlp_output)

        output = self.log_softmax_layer(torch.cat(key_outs, dim=-1))

        return output

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))
```

```python
bert_out = model(batch_input_ids, batch_attn_mask).detach().numpy()
expected_bert_out = np.array([[-1.1189675, -1.0885007, -1.0886753],
                              [-1.1045516, -1.0834142, -1.108049 ],
                              [-1.1027125, -1.0822924, -1.1110513],
                              [-1.1008494, -1.0936636, -1.1013424],
                              [-1.0921422, -1.0974907, -1.1062546],
                              [-1.0798943, -1.1088552, -1.1073538],
                              [-1.1030427, -1.0939085, -1.0989065],
                              [-1.0971034, -1.097092 , -1.1016482],
                              [-1.131921 , -1.0825679, -1.0821619],
                              [-1.0961349, -1.1014836, -1.0982255],
                              [-1.0979307, -1.0836827, -1.1144608],
                              [-1.1034715, -1.0959275, -1.0964555],
                              [-1.1019452, -1.0958116, -1.0980899],
                              [-1.1050864, -1.0986389, -1.0921533],
                              [-1.1013198, -1.0821339, -1.112621 ],
                              [-1.1027979, -1.0906712, -1.1024152]],)
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(dev_loader))
bert_out = model(batch_input_ids, batch_attn_mask).detach().numpy()
expected_bert_out = np.array([[-1.1005359, -1.1009303, -1.094384 ],
                              [-1.073251 , -1.1178819, -1.1052346],
                              [-1.1025076, -1.094363 , -1.098983 ],
                              [-1.1236262, -1.1056151, -1.0674216],
                              [-1.0999551, -1.1014045, -1.0944905],
                              [-1.0953273, -1.0959654, -1.1045709],
                              [-1.1084402, -1.0971687, -1.0903118],
                              [-1.099349 , -1.1130908, -1.0836148],
                              [-1.1031718, -1.0897288, -1.1029954],
                              [-1.0929244, -1.1077557, -1.0952206],
                              [-1.0995092, -1.0998485, -1.0964826],
                              [-1.1419646, -1.1081928, -1.0479565],
                              [-1.1052557, -1.0851235, -1.1055952],
                              [-1.0840428, -1.1084775, -1.1034834],
                              [-1.0872697, -1.1025085, -1.1061592],
                              [-1.1060572, -1.0939908, -1.095831 ]])
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
Model Output: [[-1.1189675 -1.0885006 -1.0886753]
 [-1.1045516 -1.0834142 -1.108049 ]
 [-1.1027124 -1.0822923 -1.1110513]
 [-1.1008494 -1.0936637 -1.1013424]
 [-1.0921423 -1.0974909 -1.1062545]
 [-1.0798944 -1.1088551 -1.1073539]
 [-1.1030428 -1.0939085 -1.0989064]
 [-1.0971036 -1.0970919 -1.1016482]
 [-1.1319212 -1.082568  -1.0821619]
 [-1.0961349 -1.1014836 -1.0982256]
 [-1.0979306 -1.0836825 -1.1144607]
 [-1.1034716 -1.0959275 -1.0964555]
 [-1.101945  -1.0958116 -1.0980897]
 [-1.1050864 -1.0986388 -1.0921534]
 [-1.1013197 -1.082134  -1.1126211]
 [-1.102798  -1.0906713 -1.1024151]]
Expected Output: [[-1.1189675 -1.0885007 -1.0886753]
 [-1.1045516 -1.0834142 -1.108049 ]
 [-1.1027125 -1.0822924 -1.1110513]
 [-1.1008494 -1.0936636 -1.1013424]
 [-1.0921422 -1.0974907 -1.1062546]
 [-1.0798943 -1.1088552 -1.1073538]
 [-1.1030427 -1.0939085 -1.0989065]
 [-1.0971034 -1.097092  -1.1016482]
 [-1.131921  -1.0825679 -1.0821619]
 [-1.0961349 -1.1014836 -1.0982255]
 [-1.0979307 -1.0836827 -1.1144608]
 [-1.1034715 -1.0959275 -1.0964555]
 [-1.1019452 -1.0958116 -1.0980899]
 [-1.1050864 -1.0986389 -1.0921533]
 [-1.1013198 -1.0821339 -1.112621 ]
 [-1.1027979 -1.0906712 -1.1024152]]
Test Case Passed! :)
******************************

Sample Test Case 2
Model Output: [[-1.100536  -1.1009303 -1.094384 ]
 [-1.0732511 -1.1178818 -1.1052345]
 [-1.1025076 -1.094363  -1.098983 ]
 [-1.1236264 -1.1056153 -1.0674216]
 [-1.0999552 -1.1014045 -1.0944903]
 [-1.0953273 -1.0959655 -1.1045707]
 [-1.10844   -1.0971686 -1.0903118]
 [-1.0993489 -1.1130905 -1.0836148]
 [-1.1031718 -1.0897288 -1.1029955]
```

```
 [-1.0929244 -1.1077558 -1.0952207]
 [-1.0995094 -1.0998485 -1.0964826]
 [-1.1419643 -1.1081928 -1.0479566]
 [-1.1052556 -1.0851237 -1.1055952]
 [-1.0840428 -1.1084775 -1.1034834]
 [-1.0872697 -1.1025085 -1.1061593]
 [-1.1060572 -1.0939908 -1.0958309]]
Expected Output: [[-1.1005359 -1.1009303 -1.094384 ]
 [-1.073251  -1.1178819 -1.1052346]
 [-1.1025076 -1.094363  -1.098983 ]
 [-1.1236262 -1.1056151 -1.0674216]
 [-1.0999551 -1.1014045 -1.0944905]
 [-1.0953273 -1.0959654 -1.1045709]
 [-1.1084402 -1.0971687 -1.0903118]
 [-1.099349  -1.1130908 -1.0836148]
 [-1.1031718 -1.0897288 -1.1029954]
 [-1.0929244 -1.1077557 -1.0952206]
 [-1.0995092 -1.0998485 -1.0964826]
 [-1.1419646 -1.1081928 -1.0479565]
 [-1.1052557 -1.0851235 -1.1055952]
 [-1.0840428 -1.1084775 -1.1034834]
 [-1.0872697 -1.1025085 -1.1061592]
 [-1.1060572 -1.0939908 -1.095831 ]]
Test Case Passed! :)
******************************
```

Task 2.2: Training and Evaluating the Model

Now that we have implemented the custom Dataset and a BERT based classifier model, we can start training and evaluating the model. This time we will modify the training loop slightly. At the end of each training epoch we will now evaluate on the validation data and check the accuracy. Based on this we will select the best model across the epochs that obtains highest validation accuracy. You will need to implement the `train` and `evaluate` functions below.

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

    Inputs:
        - model (BertMultiChoiceClassifierModel): BERT based multiple
choice classifier model to be evaluated
        - test_dataloader (torch.utils.DataLoader): A dataloader
defined over the test dataset

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

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


    model = model.to(device)
    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().to(device)

            # Step 1: Compute model's prediction on the test batch
(Note here you need to get the final prediction from the model's
output)
            preds = None
            # YOUR CODE HERE
            preds = model(input_ids_dict, attn_mask_dict)

            # Step 2: then compute accuracy and store it in
batch_accuracy
            batch_accuracy = 0
            # YOUR CODE HERE - Referred to Assignment 1
            preds = torch.argmax(preds, dim=1)
            pred_accuracy = torch.sum(preds == labels).item()
            batch_accuracy = pred_accuracy / len(labels)

            accuracy += batch_accuracy

    accuracy = accuracy / len(test_dataloader)
    return accuracy


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

    Inputs:
        - model (BertMultiChoiceClassifierModel): BERT based classifer
model to be trained
```

```
        - train_dataloader (torch.utils.DataLoader): A dataloader
defined over the training dataset
        - val_dataloader (torch.utils.DataLoader): A dataloader
defined over the validation dataset
        - 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' (for using gpu) or 'cpu'

    Returns:
        - best_model (BertMultiChoiceClassifierModel): model
corresponding to the highest validation accuracy (checked at the end
of each epoch)
        - best_val_accuracy (float): Validation accuracy corresponding
to the best epoch
    """
    epoch_loss = 0
    model = model.to(device)

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

    # 1. Define Loss function and optimizer
    loss_fn = None
    optimizer = None
    # YOUR CODE HERE
    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
value changes as we train the model.
        # 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 log-probabilities
```

```
            model_outs = None
            # YOUR CODE HERE
            model_outs = model(input_ids_dict, attn_mask_dict)

            # Step 4: Compute the loss and perform backward pass
            loss = None
            # YOUR CODE HERE
            loss = loss_fn(model_outs, labels)
            loss.backward()

            # Step 5: Take optimizer step
            # YOUR CODE HERE
            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 store the validation accuracy in `val_accurracy`
        val_accuracy = 0
        # YOUR CODE HERE
        val_accuracy = evaluate(model, val_dataloader, device)

        # 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_loss} | Validation Accuracy: {val_accuracy}")

    return best_model, best_val_accuracy

torch.manual_seed(42)
print("Training on 100 data points for sanity check")
sample_data = train_data[:100]
sample_labels = train_labels[:100]
sample_dataset = SIQABertDataset(sample_data, sample_labels)
sample_dataloader = DataLoader(sample_dataset, batch_size=4,
collate_fn=partial(collate_fn, sample_dataset.tokenizer))

model = BertMultiChoiceClassifierModel()
best_model, best_val_acc = train(model, sample_dataloader,
sample_dataloader, num_epochs = 5, device = "cuda")
print(f"Best Validation Accuracy: {best_val_acc}")
print(f"Expected Best Validation Accuracy: {1.0}")
```

```
Training on 100 data points for sanity check
```

{"model_id":"b327d02baf7f42fcaed457593e29d0ec","version_major":2,"version_minor":0}

```
Epoch 0 completed | Average Training Loss: 1.099404911994934 |
Validation Accuracy: 0.78
```

{"model_id":"a91243a797d241fc8c0d695e45224c5e","version_major":2,"version_minor":0}

```
Epoch 1 completed | Average Training Loss: 1.005476462841034 |
Validation Accuracy: 0.88
```

{"model_id":"1be91cbd7e9b411ca815c84711a036bd","version_major":2,"version_minor":0}

```
Epoch 2 completed | Average Training Loss: 0.5784622395038604 |
Validation Accuracy: 0.96
```

{"model_id":"646b36a6ac2f46579cc389ceeacb67dc","version_major":2,"version_minor":0}

```
Epoch 3 completed | Average Training Loss: 0.34795909315347673 |
Validation Accuracy: 1.0
```

{"model_id":"46ecd1f4965a4d7d97705e20d1b02a88","version_major":2,"version_minor":0}

```
Epoch 4 completed | Average Training Loss: 0.10256962414830922 |
Validation Accuracy: 1.0
Best Validation Accuracy: 1.0
Expected Best Validation Accuracy: 1.0
```

You can expect the validation accuracy of 1.0 by the end of training. This is so high because we trained on just 100 examples and just use those for validation for a sanity check. This is often done to debug the model and training loop. Let's now train on the entire dataset. This can take some time approximately 50 minutes per epoch, since we are fine-tuning all the 12 layers of BERT.

```
model = BertMultiChoiceClassifierModel()
best_model, best_val_acc = train(model, train_loader, dev_loader,
num_epochs = 2, device = "cuda")
```

{"model_id":"7e0d8e989a6044a0b93fd167a3ffbbe3","version_major":2,"version_minor":0}

```
Epoch 0 completed | Average Training Loss: 0.6980315272875982 |
Validation Accuracy: 0.6102642276422764
```

{"model_id":"18d1752139da4f4bb996e31feebea3bb","version_major":2,"version_minor":0}

```
Epoch 1 completed | Average Training Loss: 0.4188179671693042 |
Validation Accuracy: 0.6072154471544715
```

You should expect about ~61% validation accuracy (random classifier will have an accuracy of 33%), which is around what's reported in the SocialIQA paper. Note that this is a much more complex task than the news classification that we had in the last lab. You can further improve the performance by using bigger models like bert-base-large or roberta-large.

Now that we have a model ready for the task, we can save it on disk, so we can use it later (This will come handy for Assignment2)

```
# Save the best model
save_dir = "models/siqa_bert-base-uncased/"
if not os.path.exists(save_dir):
    os.makedirs(save_dir)

torch.save(best_model.state_dict(), f"{save_dir}/model.pt")
```

Task 2.3: Making Predictions from scratch

Similar to assignment 1, implement the function `predict_siqa` that takes as input the context, question and answers and runs them through the BERT classifier model to obtain the prediction.

```
def predict_text(siqa_instance, model, tokenizer,device = "cpu"):
    """
    Predicts the correct answer for a piece of a Social IQA instance
using the BERT classifier model

    Inputs:
        - siqa_instance (dict(str, str)): An SIQA instance containing
the context, question and the three answer choices.
        - model (BertMultiChoiceClassifierModel): Fine-tuned BERT
based classifer model
        - tokenizer (BertTokenizer): Pre-trained BERT tokenizer
    Returns:
        - pred_label (float): Predicted answer for `siqa_instance`
    """

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

    pred_label = None

    input_ids_dict = None
    attn_mask_dict = None
    # Step 1: Tokenize the [sentence, question, answer] triplet using
the tokenizer and create input_ids_dict and attn_mask_dict, as done in
the Dataset class
    # (Don't forget to convert the lists to tensors, torch.Tensor()
```

```
can come handy or just use return_tensors = "pt" while calling the
tokenizer)
    # YOUR CODE HERE
    context = siqa_instance["context"]
    question = siqa_instance["question"]
    answerA = siqa_instance["answerA"]
    answerB = siqa_instance["answerB"]
    answerC = siqa_instance["answerC"]

    tokenized_input_dict = {"A": None, "B": None, "C": None}

    cqaA = context + tokenizer.sep_token + question +
tokenizer.sep_token + answerA
    cqaB = context + tokenizer.sep_token + question +
tokenizer.sep_token + answerB
    cqaC = context + tokenizer.sep_token + question +
tokenizer.sep_token + answerC

    tokenized_input_dict["A"] = tokenizer(cqaA, return_tensors="pt")
    tokenized_input_dict["B"] = tokenizer(cqaB, return_tensors="pt")
    tokenized_input_dict["C"] = tokenizer(cqaC, return_tensors="pt")


    input_ids_dict = {key: tokenized_input_dict[key]
["input_ids"].to(device) for key in tokenized_input_dict.keys()}
    attn_mask_dict = {key: tokenized_input_dict[key]
["attention_mask"].to(device) for key in tokenized_input_dict.keys()}


    # Step 2: Feed the input_ids_dict and attn_mask_dict to the model
and get the final predictions
    # (Don't forget torch.no_grad())
    pred_label = None
    # YOUR CODE HERE
    with torch.no_grad():
        pred_label = model(input_ids_dict, attn_mask_dict)
        pred_label = torch.argmax(pred_label, dim=1).item()

    # Step 3: Make the predicted human readable i.e. convert 0 to A, 1
to B and 2 to C
    pred_label_hr = None
    # YOUR CODE HERE
    mapping = {0: "A", 1: "B", 2: "C"}
    pred_label_hr = mapping[pred_label]

    return pred_label_hr

print("Running Sample Test Case. If the implementation is correct, we
should get the same accuracy as best_val_acc above, by predicting on
each example of the dev data")
```

```
preds = [
    predict_text(siqa_instance, best_model, bert_tokenizer, device =
"cuda")
    for siqa_instance in tqdm(dev_data)
]
test_case_accuracy = (np.array(preds) == np.array(dev_labels)).mean()
print(f"Accuracy by calling `predict_text`: {test_case_accuracy}")
print(f"Expected Accuracy: {best_val_acc}")

assert np.allclose(test_case_accuracy, best_val_acc, 1e-2)
print("Test Case Passed! :)")
print("******************************\n")
```

```
Running Sample Test Case. If the implementation is correct, we should
get the same accuracy as best_val_acc above, by predicting on each
example of the dev data
```

{"model_id":"2cf3f65bffaa4c4b8511270c99e3e6d2","version_major":2,"version_minor":0}

```
Accuracy by calling `predict_text`: 0.6074718526100307
Expected Accuracy: 0.6102642276422764
Test Case Passed! :)
******************************
```

```
idx = 0
sample_data= dev_data[idx]
predicted_label = predict_text(sample_data, best_model,
bert_tokenizer)
expected_label = "C"
pprint(sample_data, sort_dicts=False, indent = 4)
print(f"Predicted Label: {predicted_label}")
print(f"Gold Label: {dev_labels[idx]}")
print("**********************************\n")

idx = 100
sample_data= dev_data[idx]
predicted_label = predict_text(sample_data, best_model,
bert_tokenizer)
expected_label = "C"
pprint(sample_data, sort_dicts=False, indent = 4)
print(f"Predicted Label: {predicted_label}")
print(f"Gold Label: {dev_labels[idx]}")
print("**********************************\n")


idx = 200
sample_data= dev_data[idx]
predicted_label = predict_text(sample_data, best_model,
bert_tokenizer)
```

```
expected_label = "C"
pprint(sample_data, sort_dicts=False, indent = 4)
print(f"Predicted Label: {predicted_label}")
print(f"Gold Label: {dev_labels[idx]}")

print("**********************************\n")

{   'context': "Tracy didn't go home that evening and resisted Riley's "
               'attacks.',
    'question': 'What does Tracy need to do before this?',
    'answerA': 'make a new plan',
    'answerB': 'Go home and see Riley',
    'answerC': 'Find somewhere to go'}
Predicted Label: C
Gold Label: C
**********************************

{   'context': 'Robin left food out for the animals in her backyard to come '
               'and enjoy.',
    'question': 'What will Robin want to do next?',
    'answerA': 'chase the animals away',
    'answerB': 'watch the animals eat',
    'answerC': 'go out in the backyard'}
Predicted Label: B
Gold Label: B
**********************************

{   'context': 'As usual, Aubrey went to the park but this morning, he met a '
               'stranger at the park who jogged with him.',
    'question': 'What will Others want to do next?',
    'answerA': 'win against Aubrey',
    'answerB': 'have a nice jog',
    'answerC': 'go to eat'}
Predicted Label: B
Gold Label: B
**********************************
```