CS 410 Final Documentation
Applying BERT on Facebook Children's Book Test

evanluo2 (Evan Luo), chiachi5 (Chia-Chi, Chen), tiehchu2 (Tieh Chu)

1) **An overview of the function of the code (i.e., what it does and what it can be used for).**

   Our task is to apply the BERT model on Facebook Children's Book Test Dataset. In the dataset, each question is constructed by taking 20 consecutive sentences from the book text and leaving the 21st as the query statement. A word from the query is selected and masked, and the model is tasked with selecting which word from the text (of the chosen type) should be used to fill this placeholder in the query.

   The Python code will be able to ingest a file with a set of questions. After parsing all the questions correctly, it would output the best word as the answer for each question.

   The main pipeline of our code is to preprocess data, predict a word using the BERT model, take the predicted word and compare it with one of the answer words provided using Word2Vec, then give the final answer.

2) **Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement.**

   1. **Preprocessing of data**

```
train_statement_string = list()
for problem in train_statement:
    final_string = ""
    for id, line in enumerate(problem):
        line = line.split(' ')
        line = line[1:]
        for i in range(len(line)):
            if not line[i][0].isalpha() and not line[i][-1].isalpha():
                line[i] = ''
            elif line[i][0].isalpha() and not line[i][-1].isalpha():
                line[i] = line[i][:-1]
            elif not line[i][0].isalpha() and line[i][-1].isalpha():
                line[i] = line[i][1:]
        final_string = final_string + ' '.join(line)
    train_statement_string.append(final_string)

train_question_string = list()
for problem in train_question:
    final_string = ""
    for word in problem.split(' '):
        if len(word) >= 5 and word[0:5] == 'XXXXX':
            word = '$'
        elif word[0].isalpha() == False and word[-1].isalpha() == False:
            word = ','
        final_string = final_string + word + ' '
    train_question_string.append(final_string[1:])
```

Before we feed our data into the model, we have to do a bit of preprocessing first. Due to the abundance of commas and periods in statements, we have to remove them in order to prevent them from affecting the model's ability to predict.

2. BERT model prediction

```python
def BERT(textA, textB):
    tokenizer = BertTokenizer.from_pretrained('bert-large-uncased')

    tokenized_textA = tokenizer.tokenize(textA)
    tokenized_textB = tokenizer.tokenize(textB)

    masked_index = len(tokenized_textA) + tokenized_textB.index('$')
    tokenized_text = tokenized_textA + tokenized_textB
    tokenized_text[masked_index] = '[MASK]'

    indexed_tokens = tokenizer.convert_tokens_to_ids(tokenized_text)
    segments_ids = [0] * len(tokenized_textA) + [1] * len(tokenized_textB)

    tokens_tensor = torch.tensor([indexed_tokens])
    segments_tensors = torch.tensor([segments_ids])

    model = BertForMaskedLM.from_pretrained('bert-large-uncased')
    model.eval()

    predictions = model(tokens_tensor, segments_tensors)

    predicted_index = torch.argmax(predictions[0, masked_index]).item()
    predicted_token = tokenizer.convert_ids_to_tokens([predicted_index])[0]
    print(predicted_token)

    return predicted_token
```

Here we try to predict a word from BERT's vocabulary that fits the blank in the question the most. First, we load the model tokenizer to tokenize our input. Then we mask the token that we try to predict with the model. We then convert the tokens to vocabulary indices and convert them into Pytorch tensors. Last, we predict the word using BertForMaskedLM, then return it.

3. Find similar word using Word2Vec

```python
def isplural(word):
    lemma = wnl.lemmatize(word, 'n')
    plural = True if word is not lemma else False
    return plural, lemma

def wordtovec(pred, choise):
    pred = isplural(pred)[1]
    WF_pred = wordnet.synsets(pred)

    score = dict()
    for c in choise:
        if pred == c:
            return c
        nc = isplural(c)[1]
        WF_c = wordnet.synsets(nc)
        if WF_pred and WF_c:
            score[c] = WF_pred[0].wup_similarity(WF_c[0])
            if score[c] == None:
                score[c] = 0

    return max(score.items(), key=operator.itemgetter(1))[0]
```

After predicting a word using BERT's vocabulary, we need to send the word to our Word2Vec function in order to compare the similarity between the predicted word and the provided answers. The answer with the highest similarity(score) will be returned as the final answer to the question. Note that we implemented an **isplural** function to change plural words to their singular form, as we observed an increase in performance after implementing the changes.

4. **Output chosen word and correct answer.**

```
def main():
    for idx in range(9):
        i = idx
        pred = BERT(train_statement_string[i], train_question_string[i])
        my_answer = wordtovec(pred, train_choice[i])
        print("Choice:", my_answer)
        print("Answer:", train_answer[i])
```

We output our answers and the correct answer.

3) **Documentation of the usage of the software including either documentation of usages of APIs or detailed instructions on how to install and run the software, whichever is applicable.**

First, clone our repository from github.

```
git clone git@github.com:daisy91530/CourseProject.git
```

Second, install the required packages. (pytorch, tqdm, boto3, requests, nltk)

```
pip install -r requirements.txt
```

Third, run our code with the following command.

```
python final_project.py
```

4) **Brief description of the contribution of each team member in the case of a multi-person team.**

The following are the contributions of each team member.

Task	Responsible Member
Parse and clean raw data	Evan Luo, Tieh Chu
Train and tune BERT model	Evan Luo, Chia-Chi
CBT answering model implementation	Chia-Chi, Chen, Tieh Chu
Fine-tune and demonstration	Evan Luo, Chia-Chi, Chen, Tieh Chu
Finish all documentation	Evan Luo, Chia-Chi, Chen, Tieh Chu