Spelling Fixer
Rasika Bhalerao
Part 2

As with Part 1, this assignment is to write a spelling fixer. You will take user input and correct the spelling. The difference is that this spelling fixer will take into account the context of the surrounding words in the text.

Learning goals:

- Using BERT
- Design a spelling fixer algorithm that makes use of given tools

What to do:

1. Read the section below on useful Python code for BERT.
    a. In Google Colab, add a GPU by going to Edit → Notebook settings → Hardware accelerator → GPU. If resources are low at the time you try this, you may need to continue writing your code without a GPU, and then come back for the GPU to run it later. We will write the code such that it will automatically detect if there is a GPU or not, and then choose to run on the GPU or a CPU. (The GPU is faster.)
2. Design an algorithm to correct user text!
    a. Just like with Part 1, take some user input text, run your algorithm, and print the decoded text. You should decide the algorithm, including when to split the text into words. This is a more open-ended assignment!
    b. Some useful tools:
        i. The provided `log_prob()` function below gives (the log of) the probability for a given string of text.
        ii. Levenshtein distance is a useful way to measure the number of character edits needed to go from one word to another. For example, the Levenshtein distance between “provable” and “probable” is 1, because only one character needs to be changed. You may need to install the relevant Python package before importing it:
```
!pip install python-Levenshtein
from Levenshtein import distance
```
    c. If you are looking for inspiration, one example algorithm is to iterate through the text word-by-word, and for each word that is not in the dictionary:
        i. Find all words in the dictionary which are within a certain Levenshtein distance of it
        ii. Try replacing it with each word, and calculate the probability according to BERT
        iii. Output the sequence that had the highest probability
3. Test it out!

Some useful Python code for BERT:

Huggingface uses Pytorch (and Tensorflow, but we will use Pytorch).

```
import torch
```

Let’s detect if there is a GPU or not, and store the device in a variable called `device` so that we can use it to run things later.

```
if torch.cuda.is_available():
   device = torch.device("cuda")
   print('Using GPU ', torch.cuda.get_device_name(0))
else:
   device = torch.device("cpu")
   print('Using CPU')
```

We will use the transformers package from Huggingface. You may need to install it first before importing it.

```
!pip install transformers
from transformers import BertTokenizer, BertForMaskedLM
```

We will use the tokenizer and pretrained BERT model from Huggingface. They are “uncased,” so they will ignore case when calculating probabilites.

```
model = BertForMaskedLM.from_pretrained('bert-base-uncased', return_dict=True)
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
log_softmax = torch.nn.LogSoftmax(dim=0)
mask_token = tokenizer.mask_token
model.to(device)
```

You can test out the tokenizer with this (modified) example from Huggingface:

```
text_batch = ["I love Pixar.",
              "I don't care for Pixar.",
              "This is such a super duper long sentence with so many
words you can barely understand it oh my gosh"]
encoding = tokenizer(text_batch, return_tensors='pt', padding=True,
truncation=True)
input_ids = encoding['input_ids'].to(device)
attention_mask = encoding['attention_mask'].to(device)
```

Look at the contents of `input_ids` and `attention_mask`. You can see that shorter sentences have been padded with 0s at the end, and all tensors are the length of the longest sentence. BERT has a maximum length of 512 tokens. For our task, we will be using entire documents instead of sentences, and documents longer than 512 will be truncated.

To make sure all tensors are on the correct device, use `.to(device)` for every tensor. Pytorch will complain if you try computation with one tensor on a CPU and another on a GPU.

Here is a function that takes a sentence and returns the log probability of that sentence according to BERT:

```
# Takes a sentence and outputs the log probability
# of that sentence according to BERT
def log_prob(sentence: str):
   token_ids = tokenizer.encode(sentence, return_tensors='pt') # tokenize,
get list of token IDs
   mask_id = tokenizer.convert_tokens_to_ids(mask_token) # get mask ID

   sum_log_probs = 0
   # Get log prob of each token in sentence
   for i in range(len(token_ids)):
       # Make a copy of the tokenized sentence with just that one token
masked
       masked_token_ids = token_ids.clone().detach()
       masked_token_ids[0][i] = mask_id

       # Get log prob of the masked word
       output = model(masked_token_ids)
       log_probs = log_softmax(output[0].squeeze(0)[i])
       target_id = token_ids[0][i] # get the logsoftmax of the index
corresponding to the masked word
       log_prob_of_this_token = log_probs[target_id]

       sum_log_probs += log_prob_of_this_token.item()

   return sum_log_probs
```

You can test out the function like this:
`log_prob('This is a medium-siezd setnence wih a few typos')`

What to turn in:
Please submit these files:

- Your Python code
- A text or pdf file with your answers to these questions:
    - Questions specific to this assignment:
        - Is your model better at choosing the correct word than the model from Part 1? Why or why not?
        - Search online to learn which dataset was used to train this BERT model for us. How might their choice of dataset affect the performance of this spelling fixer algorithm?
        - There are social biases encoded in this BERT model, which it learned from the dataset on which it was trained. How might they affect your spelling fixer algorithm?
    - Questions we ask for every assignment:
        - How long did this assignment take you? (1 sentence)
        - Whom did you work with, and how? (1 sentence each)
            - Discussing the assignment with others is encouraged, as long as you don't share the code or answers.
        - Which resources did you use? (1 sentence each)
            - For each, please list the URL and a brief description of how it was useful.
        - A few sentences about:
            - What was the most difficult part of the assignment?
            - What was the most rewarding part of the assignment?
            - What did you learn doing the assignment?
        - Constructive and actionable suggestions for improving assignments, office hours, and class time are always welcome.