Final Project: Pun Location

Group 13：
0711239 李勝維 0711278 符凱喻 0812501 許瀚宇 0816038 林俊宇

Introduction

The attention mechanism of transformers makes generating proper embedding from data like text with contextual dependency possible. A few years ago, Google released a model called BERT, the architecture of which is also a transformer. In this project, we demonstrated an example of using a pre-trained Bert to locate a pun within a sentence.

Pre-processing and External resources

The pre-trained weights of BERT model is released by Google [(BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding)].

To use the pre-trained BERT model, we first need to generate three kinds of tensors from tokenized sentences, which are input_ids, attention_mask, and token_type_ids. In this example, we don’t need token_type_ids, which is used for tasks that require multi-sentences as inputs. Therefore, token_type_ids are set to be a list of 0. Then, input_ids are a unique int representation of words. Just enumerate all kinds of words, and make a dictionary to transform all words into int. attention_mask is needed as the dimension of model input is fixed. BERT needs to know how long the padded tensors are, so we then generate a list of len(tokens) 1s followed by (max_length - len(tokens)) 0s.

Your model design and concept (methods)

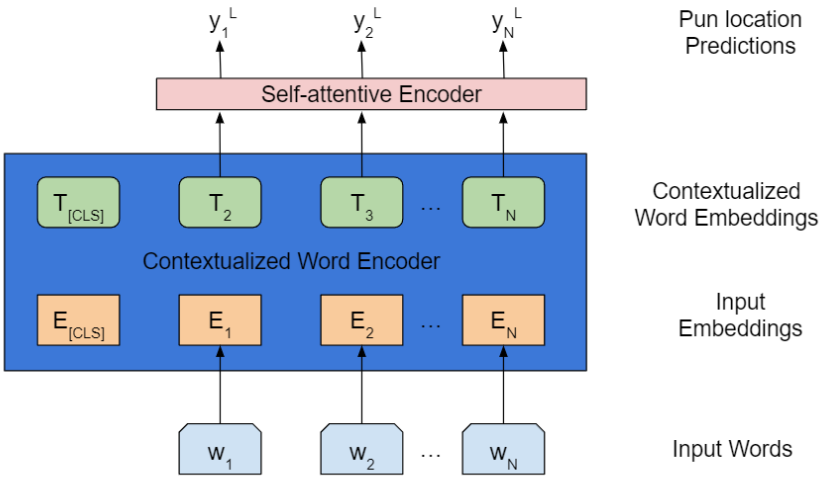
Input:

Text consists of a sequence of N words $\{w_1, w_2, ..., w_N\}$ and one of them is a pun.

Our goal:

For each word w_i, we would like to predict a binary label y_i^L that indicates if w_i is a punning word.

Framework:



Why contextualized word embeddings?

Contextualized word embeddings derive appropriate word representations with consideration of context words and capture the accurate semantics in the text.

For each word in the input text, we first derive contextualized word embedding. In our project, we choose BERT to help us achieve this part.
BERT deploys a multi-layer bidirectional encoder based on transformers with multi-head self-attention to model words in the text after integrating both word and position embeddings. Here, we denote T_i as the contextualized word embedding for the word w_i.

Then, a self-attentive encoder takes it as input to capture the overall semantics for pun location. For each word w_i, the self-attention mechanism estimates the attention score α_i. Hence, the self-attentive embedding vector for each word is computed by:

$$T_{i,\,attention} = \alpha_i * T_i$$

Finally, pun location is modeled as a binary classification task and the prediction y_i^L is generated by:

$$y_i^L = \text{argmax}\, F_L(T_{i,\,attention})_k,\ k \in \{0, 1\}$$

where $F_L()$ derives two logits for classifying if a word is a pun and we also optimize the model with cross-entropy loss.

Model details：

We follow BERT (BASE) to use 12 Transformer layers and self-attention heads.
model name: Bert-base-cased

```
class BertConfig(object):
    """Configuration class to store the configuration of a `BertModel`.
    """
    def __init__(self,
                 vocab_size_or_config_json_file,
                 hidden_size=768,
                 num_hidden_layers=12,
                 num_attention_heads=12,
                 intermediate_size=3072,
                 hidden_act="gelu",
                 hidden_dropout_prob=0.1,
                 attention_probs_dropout_prob=0.1,
                 max_position_embeddings=512,
                 type_vocab_size=2,
                 initializer_range=0.02):
```

(These settings could be found on https://huggingface.co/docs/transformers/model_doc/bert
Our code is modified from the source code provided by huggingface.)

What kind of word sense representation used and experimented in your model

作者在contexual word embedding使用了常見的NLP NN model BERT，在句子拆成tokens，加上[CLS]和[SEP]，之後產生positional embedding（arange句子長度後用nn.Embedding產生）、word embedding，進入BERT後產生Contextualized Word Embeddings

What problem did you face during the homework and how you solved（ALL）

- Cannot fit a large batch into the VRAM:
  Solved by using a technique called gradient accumulation. With gradient accumulation, the model updates after multiple batches thus creating a larger batch virtually.
- Program crash for an unknown reason:
  Use Colab instead. Though the program crashes on my computer, it works fine on Colab.
- BERT has a huge number of parameters, hence training it on a small dataset would lead to overfitting:
  Using the pre-trained model and try to “fine-tune” it for the current dataset.

Experiment results

model 預設設定：

batch size：32
epoch：3
learning rate：0.00005
pretrained model：bert-base-cased

batch size	score
16	0.90000
32	0.90625
64	0.86250

epoch	score
2	0.88125
3	0.90625
4	0.87500

learning rate	score
0.00003	0.85625
0.00005	0.90625
0.00008	0.85625
0.00010	0.83125

pretrained model	score
bert-base-uncased	0.88125
bert-base-cased	0.90625

除了上述的測試之外，還有另外進行兩項測試。首先是觀察不同random seed的結果，在多次測試後得出誤差範圍為0.00625。其次是我們另外找來的一篇f1-score很好的論文——“The Boating Store Had Its Best Sail Ever”: Pronunciation-attentive Contextualized Pun Recognition，這篇論文與我們做的model差別在於多了Pronunciation Embedding的部分，所以我們嘗試把Pronunciation Embedding加進去，但在測試後發現兩者的結果完全相同，我們認為這是因為其實不需要Pronunciation Embedding便能辨識大部分的pun，又因為這次的test data非常少，只有320個，所以比較難看出差異。

Error Analysis and Discussion (discussion)

在使用一部分的train作為validation後發現error主要具有以下特徵:

1. 與pun相關或是決定pun的字, 例如:
    a. Scientists have created a flea from a scratch.
       Pun : scratch
       Predict : flea
    b. My friend came around for dinner. We ate for 20 minutes, then he fainted again.
       Pun : came
       Predict : fainted
    c. I'm a sap for tree jokes.
       Pun : sap
       Predict : tree

上述例子中可以看出因為flea所以scratch是pun, 因為fainted所以came是pun, 因為tree所以sap是pun。我們推測是因為model可以偵測到跟pun相關的字組, 但是無法正確的從字組中選出pun。

2. 慣用語方面的pun, 例如:
    a. My job as head chef at a top rated restaurant is in jeopardy because my latest culinary creation was called a recipe for disaster.
    b. The astronauts said their experience on the moon was out of this world.

上述例子的pun都發生在慣用語中, 它們是慣用語的一部份, 但是在慣用語外又有另一層意思。我們推測因為慣用語是一組字形成獨立的意思, 而model主要是判斷字句的關係, 難以判斷慣用語, 所以也就判斷不出pun。

3. 少見或不存在的字義, 例如:
    a. She was only a fruit vendor's daughter, but, my, she had big melons.
    b. ''Waiter, there's a fly in my soup!'' ''Force of habit, sir, the chef used to be a tailor.
    c. If an animal loses its tail it should shop at a retail store.

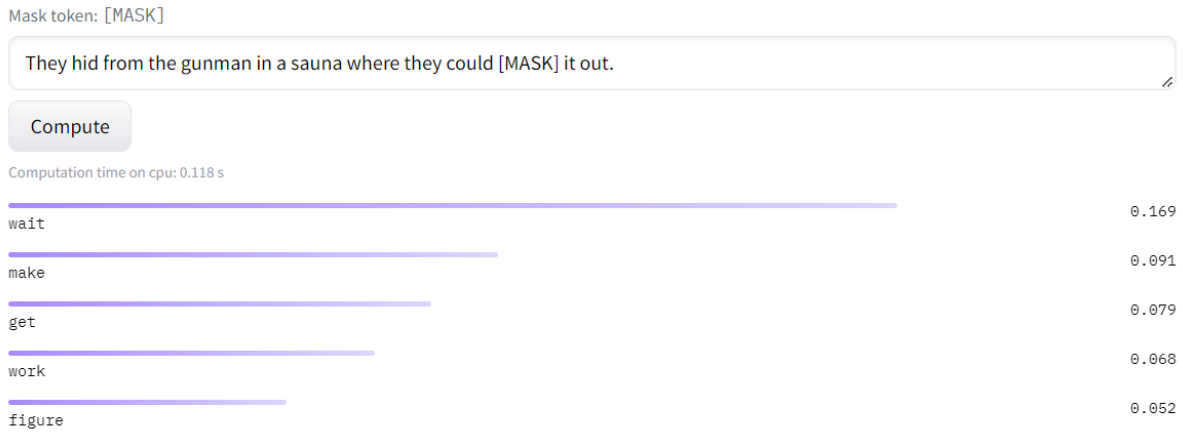
上述例子中的pun分別為melon、fly、retail, 其中melon有少見字義指大肚子, fly有少見字義指褲襠拉鍊, retail甚至是拆字拆成re-tail。我們推測model很難甚至沒有能力辨識這種少見或不存在的字義, 導致model找不出pun。

Compare and implement unsupervised method and supervised method

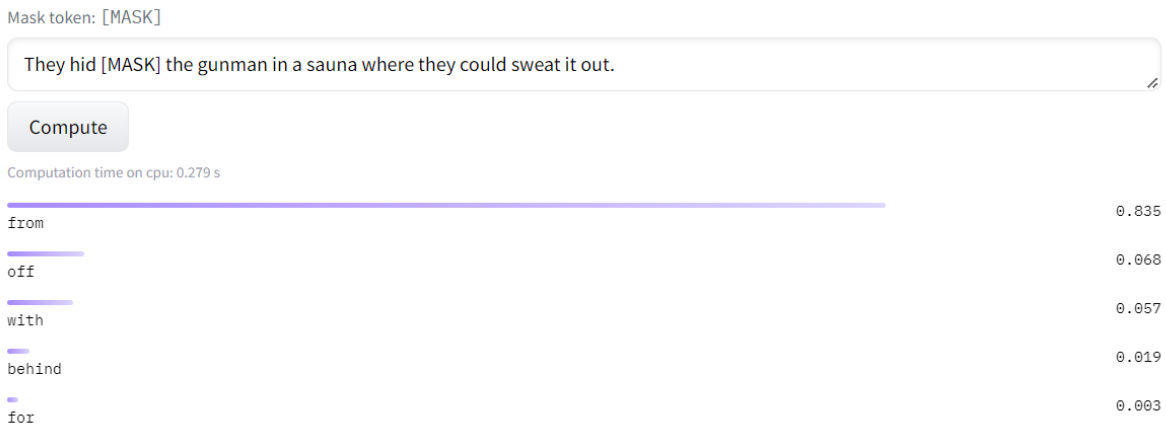
The unsupervised method

We choose the BERT language model (without any fine-tuning) as the unsupervised method and the proposed method as the comparing supervised method.
For every word in the input sequence, we mask the word and let the LM predicts the word. Since "puns" have multiple meanings, we assume that if the language model predicts two words (or more) with high probability, this may lead to a detected pun.

Examples:



Multiple words with high probability → may be a pun.



Only one word with high probability → may not be a pun.

Experiment Results:

Method	score
Unsupervised (BERT LM)	0.73125
Supervised	0.90625

The unsupervised method performs unexpectedly well, but there is more implicit knowledge about puns that our language model cannot capture well. The result also demonstrates the importance of fine-tuning BERT for downstream tasks.

Conclusion

The BERT model generates powerful contextualized embeddings that are useful for downstream tasks. However, it still cannot recognize some phrases and rare words due to the limitation of the size of our training data.