HELEN(MENGXIN) JI

Pre-trained BERT

27 MAR 2019

openai nlp

BERT is short for Bidirectional Encoder Representation from Transformers, which is the Encoder of the two-way Transformer, because the Decoder cannot get the information to be predicted. The main innovation for the model is in the pre-trained method, which uses Masked Language Model and Next Sentence Prediction to capture the word and sentence level representation respectively. This allows us to use a pre-trained BERT model by fine-tuning the same on downstream specific tasks such as sentiment classification, intent detection, question answering and more.

Pre-trained Task 1: Masked Language Model

Pre-trained Task 2; Next Sentence Prediction

Pre-trained Models

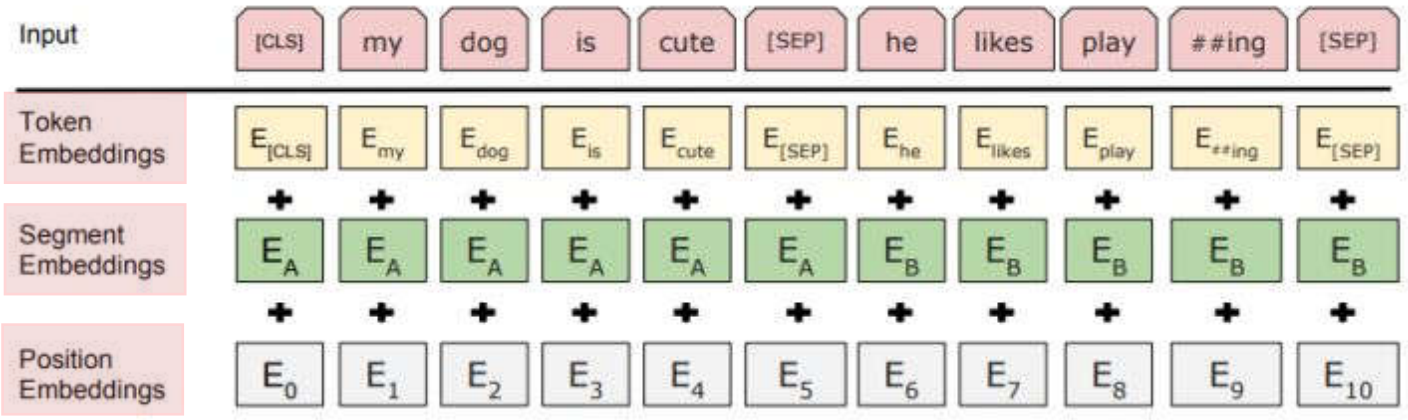
Google Research recently open-sourced implementation of BERT and also released the following pre-trained models:

- BERT-Base, Uncased: 12-layer, 768-hidden, 12-heads, 110M parameters
- BERT-Large, Uncased: 24-layer, 1024-hidden, 16-heads, 340M parameters
- BERT-Base, Cased: 12-layer, 768-hidden, 12-heads , 110M parameters
- BERT-Large, Cased: 24-layer, 1024-hidden, 16-heads, 340M parameters
- BERT-Base, Multilingual Cased (New, recommended): 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- BERT-Base, Chinese: Chinese Simplified and Traditional, 12-layer, 768-hidden, 12-heads, 110M parameters

In the following example, I will use bert-base-uncased pre-trained model.

Embedding

In BERT, the embedding is the summation of three types of embeddings:



where:

- Token Embeddings is a word vector, with the first word as the CLS flag, which can be used for classification tasks;
- Segment Embeddings is used to distinguish between two sentences, since pre-training is not just a language modeling but also a classification task with two sentences as input;
- Position Embedding is different from Transformer in the previous article.

We have to convert the input to the feature that is understood by BERT.

- input_ids: list of numerical ids for the tokenized text
- input_mask: will be set to 1 for real tokens and 0 for the padding tokens
- segment_ids: for our case, this will be set to the list of ones
- label_ids: one-hot encoded labels for the text

Tokenization

BERT-Base, uncased uses a vocabulary of 30,522 words. The processes of tokenization involves splitting the input text into list of tokens that are available in the vocabulary. In order to deal with the words not available in the vocabulary, BERT uses a technique called BPE based WordPiece tokenization.

Model Architecture

Here I use pre-trained BERT for binary sentiment analysis on Stanford Sentiment Treebank.

- BertEmbeddings: Input embedding layer
- BertEncoder: The 12 BERT attention layers
- Classifier: Our multi-label classifier with out_features=2, each corresponding to our 2 labels

```
- BertModel
    - embeddings: BertEmbeddings
        - word_embeddings: Embedding(28996, 768)
        - position_embeddings: Embedding(512, 768)
        - token_type_embeddings: Embedding(2, 768)
        - LayerNorm: FusedLayerNorm(torch.Size([768])
        - dropout: Dropout = 0.1
    - encoder: BertEncoder
        - 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 = 0.1
                - output: BertSelfOutput(
                        - dense: Linear(in_features=768, out_features=768, bias=True)
                        - LayerNorm: FusedLayerNorm(torch.Size([768]),
                        - dropout: Dropout =0.1

                - intermediate: BertIntermediate(
                        - dense): Linear(in_features=768, out_features=3072, bias=True)

                - output: BertOutput
                        - dense: Linear(in_features=3072, out_features=768, bias=True)
                        - LayerNorm: FusedLayerNorm(torch.Size([768])
                        - dropout: Dropout =0.1
        - pooler: BertPooler
                - dense: Linear(in_features=768, out_features=768, bias=True)
                - activation: Tanh()
        - dropout: Dropout =0.1
        - classifier: Linear(in_features=768, out_features = 2, bias=True)
```