

Module 6

Implementation using BERT Model



Which BERT Model we are using?

There are two different BERT models:

1. BERT **base**, which is a BERT model consists of 12 layers of Transformer encoder, 12 attention heads, 768 hidden size, and 110M parameters.
2. BERT **large**, which is a BERT model consists of 24 layers of Transformer encoder, 16 attention heads, 1024 hidden size, and 340 parameters.

BERT INPUT

BERT Input

BERT model expects a sequence of tokens (words) as an input. In each sequence of tokens, there are two special tokens that BERT would expect as an input:

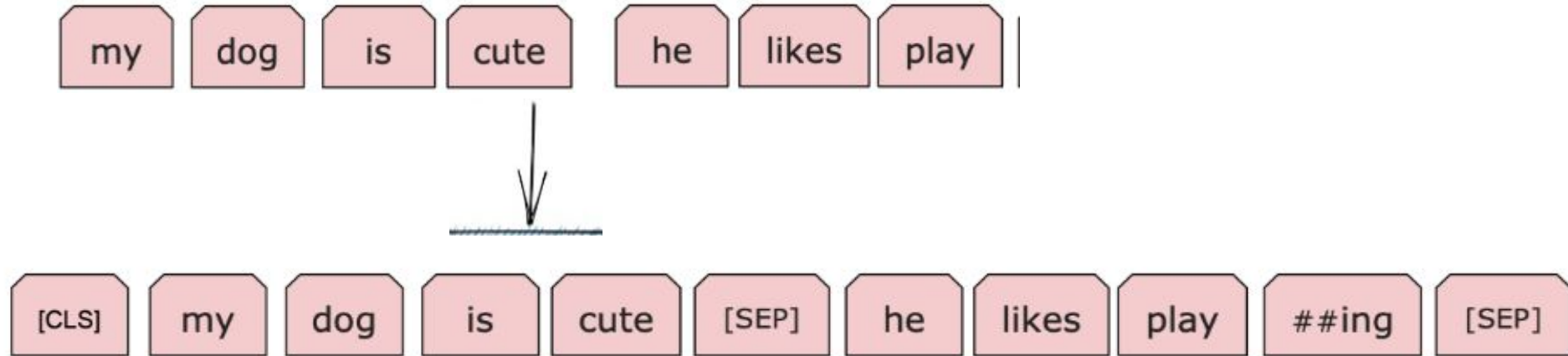
1. **[CLS]**: This is the first token of every sequence, which stands for classification token.
2. **[SEP]**: This is the token that makes BERT know which token belongs to which sequence. If we only have one sequence, then this token will be appended to the end of the sequence.

Input



BERT Tokenizer

BERT Tokenizer will do this



BERT Tokenizer parameters

- `padding` : to pad each sequence to the maximum length that you specify.
- `max_length` : the maximum length of each sequence. In this example we use 10, but for our actual dataset we will use 512, which is the maximum length of a sequence allowed for BERT.
- `truncation` : if True, then the tokens in each sequence that exceed the maximum length will be truncated.
- `return_tensors` : the type of tensors that will be returned. Since we're using Pytorch, then we use `pt`.

If you use Tensorflow, then you need to use `tf` .




```
tokenizer = BertTokenizer.from_pretrained(pretrained_model_name)

# Bert Tokenizer with example
example_text = 'I am doing the final module of NLP Interestship '

tokens = tokenizer.tokenize(example_text)
token_ids = tokenizer.convert_tokens_to_ids(tokens)
print(f' Sentence: {example_text}')
print(f'   Tokens: {tokens}')
print(f'Token IDs: {token_ids}')
```

✓ 0.4s

```
... Sentence: I am doing the final module of NLP Interestship
      Tokens: ['I', 'am', 'doing', 'the', 'final', 'module', 'of', 'NL', '##P', 'Interest', '##ship']
Token IDs: [146, 1821, 1833, 1103, 1509, 13196, 1104, 21239, 2101, 17067, 6607]
```

encode_plus

encode_plus

✓
0s

```
[42] sample = 'I am learning Final module of NLP'  
      encoding = tokenizer.encode_plus(sample,max_length=10, truncation = True)  
      for key, value in encoding.items():  
          print( '{} : {}'.format( key, value ) )
```

```
↳ input_ids : [101, 146, 1821, 3776, 3788, 13196, 1104, 21239, 2101, 102]  
   token_type_ids : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
   attention_mask : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

input_ids

1. is the id representation of each token. We can actually decode these input ids into the actual tokens as follows:

Token IDs: [146, 1821, 3776, 1509, 13196, 1104, 21239, 2101, 17067, 6607]

```
✓ [37] tokens = tokenizer.decode(token_ids)  
0s tokens
```


```
➞ 'I am learning final module of NLP Interestship'
```

token_type_ids

The second is `token_type_ids`, which is a binary mask that identifies in which sequence a token belongs. If we only have a single sequence, then all of the token type ids will be 0.

```
[8] example_text = 'I am learning final module of NLP'  
     print(bert_input['token_type_ids'])
```

```
tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```



attention_mask

The third is `attention_mask`, which is a binary mask that identifies whether a token is a real word or just padding. If the token contains **[CLS]**, **[SEP]**, or any real word, then the mask would be 1. Meanwhile, if the token is just padding or **[PAD]**, then the mask would be 0.

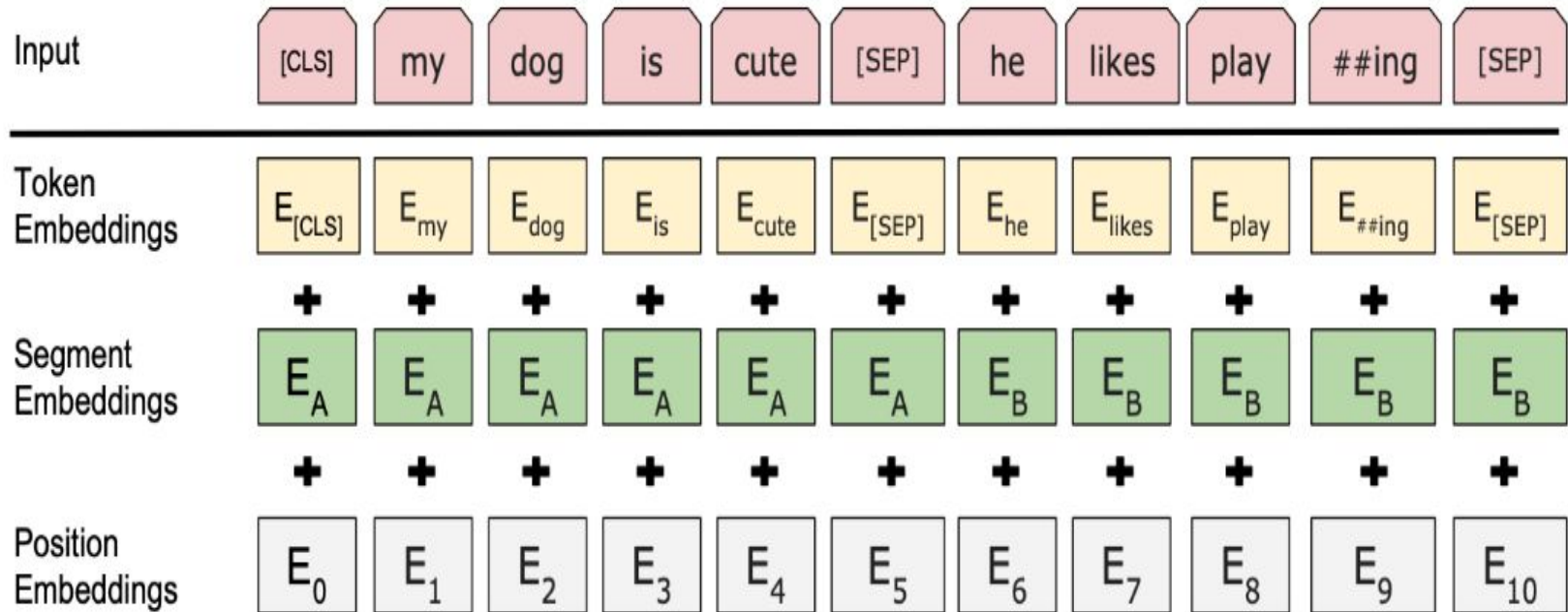
```
[14] example_text = 'I am learning final module of NLP'  
      print(bert_input['attention_mask'])  
  
      tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

BERT Embeddings

In the case of BERT, it creates three embeddings for

- Token,
- Segments and
- Position.







Build a Dataset Class for our
Quora dataset.


```
class QuestionDataset(Dataset):

    def __init__(self, questions, targets, tokenizer, max_len):
        self.questions = questions
        self.targets = targets
        self.tokenizer = tokenizer
        self.max_len = max_len


    def __len__(self):
        return len(self.questions)

    def __getitem__(self, item):
        question = str(self.questions[item])
        target = self.targets[item]
        encoding = self.tokenizer.encode_plus(
            question,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            pad_to_max_length=True,
            return_attention_mask=True,
            return_tensors='pt',
            truncation=True
        )
```

Model Building

```
class QuestionClassifier(nn.Module):
    def __init__(self, n_classes):
        super(QuestionClassifier, self).__init__()
        self.bert = BertModel.from_pretrained(pretrained_model_name)
        self.drop = nn.Dropout(p=0.3)
        self.out = nn.Linear(self.bert.config.hidden_size, n_classes)
    def forward(self, input_ids, attention_mask):
        _, pooled_output = self.bert(
            input_ids=input_ids,
            attention_mask=attention_mask,
            return_dict=False,
        )
        output = self.drop(pooled_output)
        return self.out(output)
```

✓ 0.0s

- 
1. Training -> training_fun
 2. Evaluate -> evaluate_model
 3. Predictions -> predictions
 4. Compute metrics -> F1 Score



Thank You