How the Embedding Layers in BERT Were Implemented

___ Feb 20, 2019 · 5 min read
★

Introduction

In this article, I will explain the implementation details of the embedding layers in BERT, namely the Token Embeddings, Segment Embeddings, and the Position Embeddings.

Overview

Here's a diagram from the paper that aptly describes the function of each of the embedding layers in BERT:

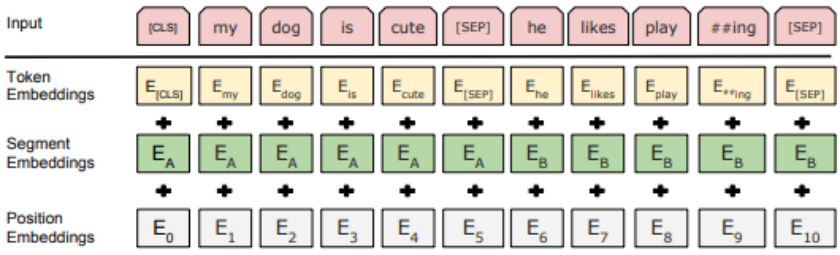


Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Like most deep learning models aimed at solving NLP-related tasks, BERT passes each input token (the words in the input text) through a Token Embedding layer so that each token is transformed into a vector representation. Unlike other deep learning models, BERT has additional

embedding layers in the form of Segment Embeddings and Position Embeddings. The reason for these additional embedding layers will become clear by the end of this article.

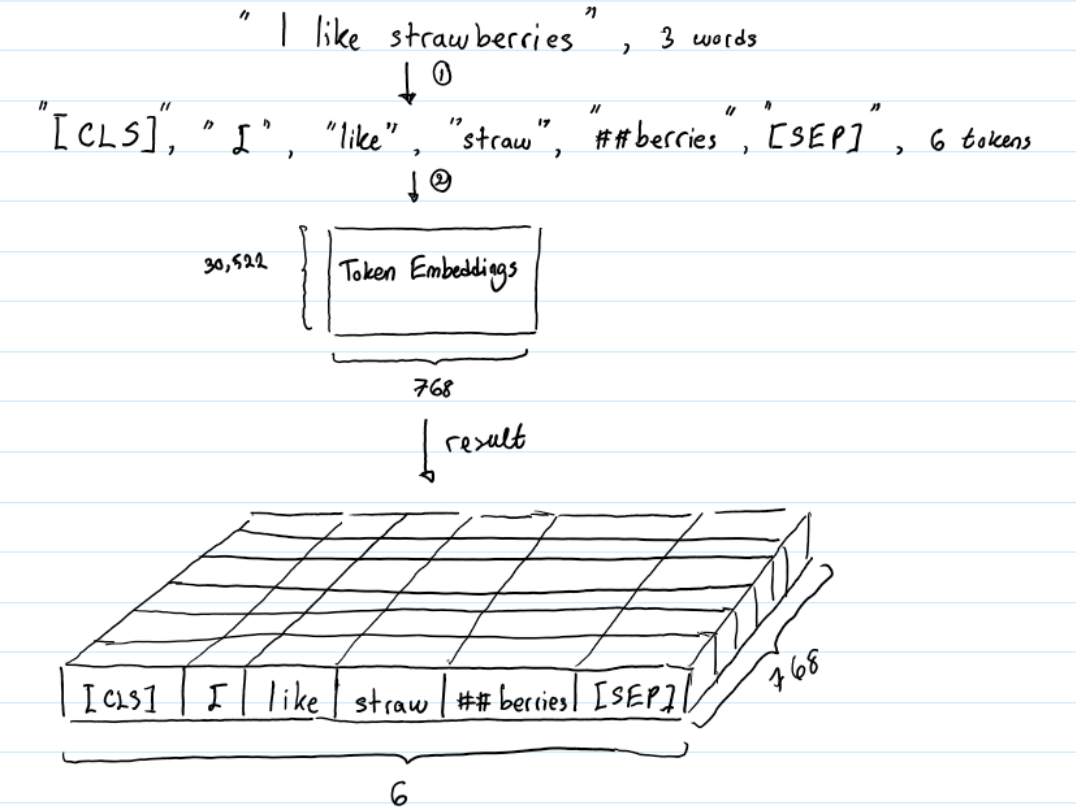
Token Embeddings

Purpose

As alluded to in the previous section, the role of the Token Embeddings layer is to transform words into vector representations of fixed dimension. In the case of BERT, each word is represented as a 768-dimensional vector.

Implementation

Suppose the input text is “I like strawberries”. Here’s a diagram describing the role of the Token Embeddings layer (there is a minor typo in this diagram, see this comment from Acha for details):



The input text is first tokenized before it gets passed to the Token Embeddings layer. Additionally, extra tokens are added at the start ([CLS])

and end ([SEP]) of the tokenized sentence. The purpose of these tokens are to serve as an input representation for classification tasks and to separate a pair of input texts respectively (more details in the next section).

The tokenization is done using a method called WordPiece tokenization. This is a data-driven tokenization method that aims to achieve a balance between vocabulary size and out-of-vocab words. This is way “strawberries” has been split into “straw” and “berries”. A detailed description of this method is beyond the scope of this article. The interested reader may refer to section 4.1 in Wu et al. (2016) and Schuster & Nakajima (2012). The use of WordPiece tokenization enables BERT to only store 30,522 “words” in its vocabulary and very rarely encounter out-of-vocab words in the wild when tokenizing English texts.

Top highlight

The Token Embeddings layer will convert each wordpiece token into a 768-dimensional vector representation. This results in our 6 input tokens being converted into a matrix of shape (6, 768) or a tensor of shape (1, 6, 768) if we include the batch axis.

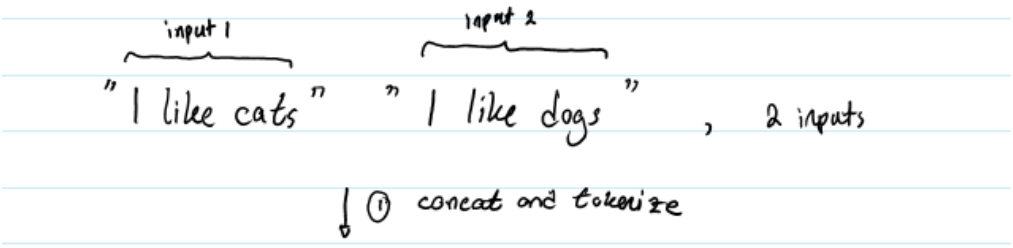
Segment Embeddings

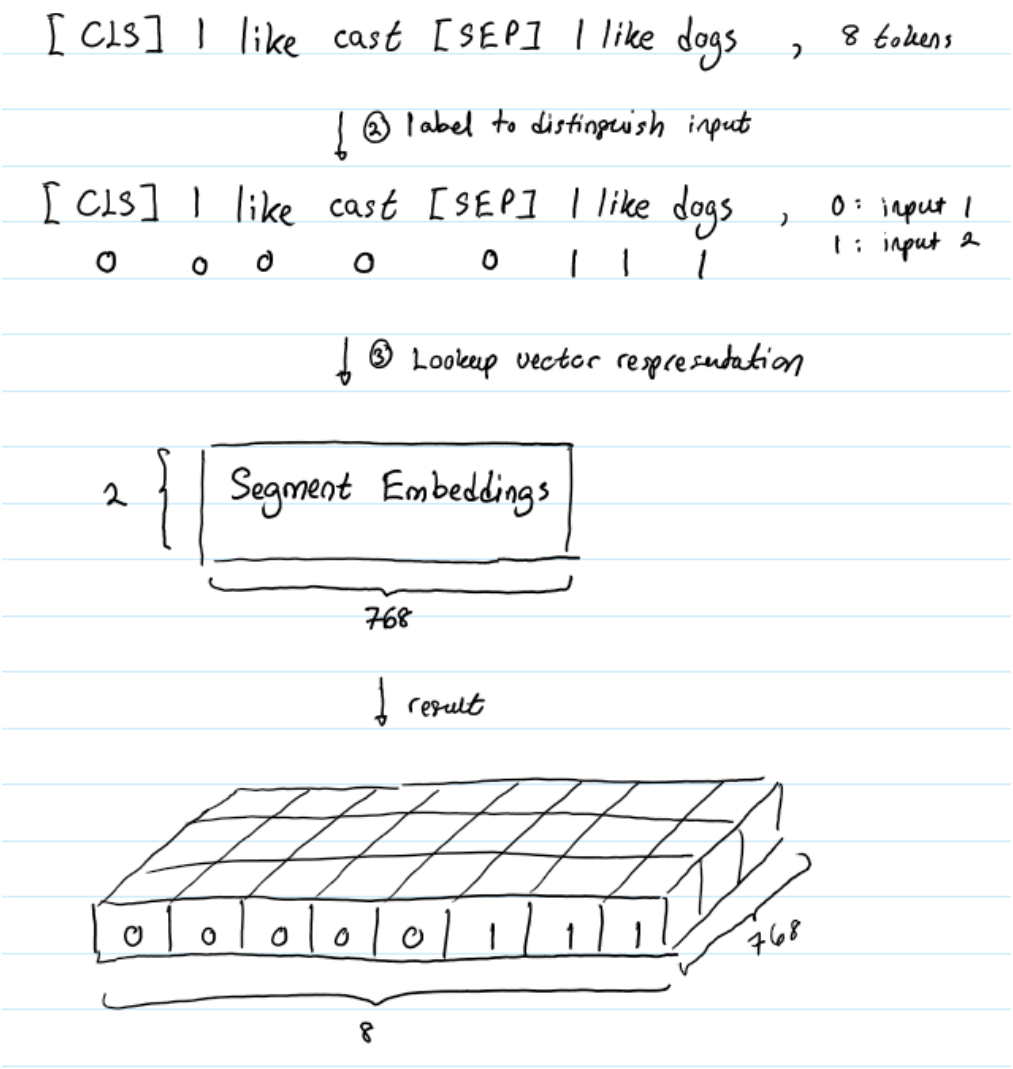
Purpose

BERT is able to solve NLP tasks that involve text classification given a pair of input texts. An example of such a problem is classifying whether two pieces of text are semantically similar. The pair of input text are simply concatenated and fed into the model. So how does BERT distinguishes the inputs in a given pair? The answer is Segment Embeddings.

Implementation

Suppose our pair of input text is (“I like cats”, “I like dogs”). Here’s how Segment Embeddings help BERT distinguish the tokens in this input pair:





The Segment Embeddings layer only has 2 vector representations. The first vector (index 0) is assigned to all tokens that belong to input 1 while the last vector (index 1) is assigned to all tokens that belong to input 2. If an input consists only of one input sentence, then its segment embedding will just be the vector corresponding to index 0 of the Segment Embeddings table.

Position Embeddings

Purpose

BERT consists of a stack of Transformers (Vaswani et al. 2017) and broadly speaking, Transformers do not encode the sequential nature of their inputs. The Motivation section in this blog post explains what I mean in greater detail. To summarize, having position embeddings will allow BERT to understand that given an input text like:

> *I think, therefore I am*

the first “I” should not have the same vector representation as the second “I”.

Implementation

BERT was designed to process input sequences of up to length 512. The authors incorporated the sequential nature of the input sequences by having BERT learn a vector representation for each position. This means that the Position Embeddings layer is a lookup table of size (512, 768) where the first row is the vector representation of any word in the first position, the second row is the vector representation of any word in the second position, etc. Therefore, if we have an input like “Hello world” and “Hi there”, both “Hello” and “Hi” will have identical position embeddings since they are the first word in the input sequence. Similarly, both “world” and “there” will have the same position embedding.

Combining Representations

We have seen that a tokenized input sequence of length n will have three distinct representations, namely:

- Token Embeddings with shape (1, n, 768) which are just vector representations of words
- Segment Embeddings with shape (1, n, 768) which are vector representations to help BERT distinguish between paired input sequences.
- Position Embeddings with shape (1, n, 768) to let BERT know that the inputs its being fed with have a temporal property.

These representations are summed element-wise to produce a single representation with shape (1, n, 768). This is the input representation that is passed to BERT’s Encoder layer.

Conclusion

In this article, I have described the purpose of each of BERT’s embedding

layers and their implementation. Let me know in the comments if you have any questions.

References

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding; Devlin et al. 2018.

Google's Neural Machine Translation System: Briding the Gap between Human and Machine Translation; Wu et al. 2016

Japanese and Korean Voice Search; Schuster and Nakajima. 2012.

Attention Is All You Need; Vaswani et al. 2017.