Simple BERT using TensorFlow 2.0

Gergely D. Németh
Oct 31, 2019 · 3 min read ★

This story shows a simple usage of the BERT [1] embedding using TensorFlow 2.0. As TensorFlow 2.0 has been released recently, the module aims to use easy, ready-to-use models based on the high-level Keras API. The previous usage of BERT was described in a long Notebook implementing a Movie Review prediction. In this story, we will see a simple BERT embedding generator using Keras and the latest TensorFlow and TensorFlow Hub modules. All codes are available on Google Colab.

My previous stories used the `bert-embedding` module to generate sentence-level and token-level embeddings using the pre-trained uncased BERT base model. Here, we will implement this module's usage with only a few steps.

TensorFlow 2.0

Module imports

We will use the latest TensorFlow (2.0+) and TensorFlow Hub (0.7+), therefore, it might need an upgrade in the system. For the model creation, we use the high-level Keras API Model class (newly integrated to tf.keras).

The BERT tokenizer is still from the BERT python module (bert-for-tf2).

```
import tensorflow_hub as hub
import tensorflow as tf
import bert
FullTokenizer = bert.bert_tokenization.FullTokenizer
from tensorflow.keras.models import Model       # Keras is the new high level API for TensorFlow
import math
```

bert-colab-dependencies.py hosted with ♡ by GitHub view raw

Imports of the project

The model

We will implement a model based on the example on TensorFlow Hub. Here, we can see that the `bert_layer` can be used in a more complex model similarly as any other Keras layer.

The goal of this model is to use the pre-trained BERT to generate the embedding vectors. Therefore, we need only the required inputs for the BERT layer and the model has only the BERT layer as a hidden layer. Of course, inside the BERT layer, there is a more complex architecture.

The `hub.KerasLayer` function imports the pre-trained model as a Keras layer.

```
max_seq_length = 128  # Your choice here.
input_word_ids = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32,
                                       name="input_word_ids")
input_mask = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32,
                                   name="input_mask")
segment_ids = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32,
                                    name="segment_ids")
bert_layer = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1",
                            trainable=True)
pooled_output, sequence_output = bert_layer([input_word_ids, input_mask, segment_ids])

model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs=[pooled_output, sequenc
```

bert-colab-model.py hosted with ♡ by GitHub view raw

BERT embedding model in Keras

Preprocessing

The BERT layer requires 3 input sequence:

- Token ids: for every token in the sentence. We restore it from the BERT vocab dictionary
- Mask ids: for every token to mask out tokens used only for the sequence padding (so every sequence has the same length).
- Segment ids: 0 for one-sentence sequence, 1 if there are two sentences in the sequence and it is the second one (see the original paper or the corresponding part of the BERT on GitHub for more details: `convert_single_example` in the `run_classifier.py`).

```
def get_masks(tokens, max_seq_length):
    """Mask for padding"""
    if len(tokens)>max_seq_length:
        raise IndexError("Token length more than max seq length!")
    return [1]*len(tokens) + [0] * (max_seq_length - len(tokens))


def get_segments(tokens, max_seq_length):
    """Segments: 0 for the first sequence, 1 for the second"""
    if len(tokens)>max_seq_length:
        raise IndexError("Token length more than max seq length!")
    segments = []
    current_segment_id = 0
    for token in tokens:
        segments.append(current_segment_id)
        if token == "[SEP]":
            current_segment_id = 1
    return segments + [0] * (max_seq_length - len(tokens))


def get_ids(tokens, tokenizer, max_seq_length):
    """Token ids from Tokenizer vocab"""
    token_ids = tokenizer.convert_tokens_to_ids(tokens)
    input_ids = token_ids + [0] * (max_seq_length-len(token_ids))
    return input_ids
```

bert-colab-preprocessing.py hosted with ♡ by GitHub view raw

Functions to generate the input based on the tokens and the max sequence length

Prediction

With these steps, we can generate BERT contextualised embedding vectors for our sentences! Don't forget to add `[CLS]` and `[SEP]` separator tokens to keep the original format!

```
s = "This is a nice sentence."
stokens = tokenizer.tokenize(s)
stokens = ["[CLS]"] + stokens + ["[SEP]"]

input_ids = get_ids(stokens, tokenizer, max_seq_length)
input_masks = get_masks(stokens, max_seq_length)
input_segments = get_segments(stokens, max_seq_length)

pool_embs, all_embs = model.predict([[input_ids],[input_masks],[input_segments]])
```

bert-colab-prediction.py hosted with ♡ by GitHub view raw

Bert Embedding Generator in use

Pooled Embedding as Sentence-level embedding

The original paper suggests the use of the `[CLS]` separator as a representation of the whole sentence because every sentence has a `[CLS]` token and as it is a contextualised embedding, this can represent the whole sentence. In my previous works, I also used this token's embedding as sentence-level representation. The `bert_layer` from TensorFlow Hub returns with a different pooled output for the representation of the entire input sequence.

To compare the two embeddings, let's use cosine similarity. The difference between the pooled embedding and the first token's embedding in the sample sentence *"This is a nice sentence."* is 0.0276.

Summary

This story introduces a simple, high-level Keras based TensorFlow 2.0 usage of the BERT embedding model. Other models like ALBERT are also available on TensorFlow Hub.

All codes of this story can be accessed on Google Colab.

References

[1] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Learn NMT with BERT stories

1. BLEU-BERT-y: Comparing sentence scores
2. Visualisation of embedding relations (word2vec, BERT)
3. Machine Translation: A Short Overview
4. Identifying the right meaning of the words using BERT
5. Machine Translation: Compare to SOTA
6. Simple BERT using TensorFlow 2.0

Machine Learning | Artificial Intelligence | Bert | TensorFlow | NLP