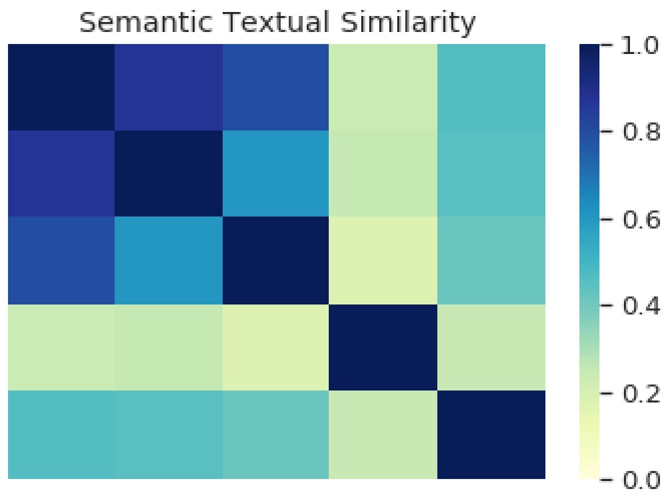

Use-cases of Google's Universal Sentence Encoder in Production

Sambit Mahapatra Follow
Jan 24, 2019 · 6 min read

Before building any Deep Learning model in Natural Language Processing (NLP), text embedding plays a major role. The text embedding converts text (words or sentences) into a numerical vector.

Why do we convert texts into vectors?

A vector is an array of numbers of a particular dimension. A vector of size 5×1 contain 5 numbers and we can think of it as a point in 5D space. If there are two vectors each of dimension 5, they can be thought of two points in a 5D space. Thus we can calculate how close or distant those two vectors are, depending on the distance measure between them.

Hence, lots of efforts in machine learning research are bring put to converting data into a vector as once data is converted into a vector, we can say two data points are similar or not by calculating their distance. Techniques like Word2vec and Glove do that by converting a word to vector. Thus the corresponding vector of “cat” will be closer to “dog” than “eagle”. But while embedding a sentence, along with words the context of the whole sentence needs to be captured in that vector. This is where the “Universal Sentence Encoder” comes into the picture.

The Universal Sentence Encoder encodes text into high dimensional vectors that can be used for text classification, semantic similarity, clustering, and other natural language tasks. The pre-trained Universal Sentence Encoder is publicly available in Tensorflow-hub. It comes with two variations i.e. one trained with **Transformer encoder** and other trained with **Deep Averaging Network (DAN)**. The two have a trade-off of accuracy and computational resource requirement. While the one with Transformer encoder has higher accuracy, it is computationally more intensive. The one with DNA encoding is computationally less expensive and with little lower accuracy.

Here, we will go with the **transformer encoder** version. It worked well for us while running it along with 5 other heavy deep learning models in a 5 GB ram instance. Also, we could be able to train a classifier with 1.5 Million data using this version of Universal Sentence Encoder at embedding level. Few use cases of Universal Sentence Encoder I have come across are :

1. As an embedding layer at the start of a deep learning model.
2. Performing classification by finding semantically similar sentences.
3. Getting rid of duplicate sentences or phrases before analysis.

Let’s see how to use the pre-trained Universal Sentence Encoder available at Tensorflow-hub, for above-mentioned use cases in python.

First, let’s import the required libraries:

```
import tensorflow as tf
import tensorflow_hub as hub
```

While using in production, we need to download the pre-trained Universal Sentence Encoder to local so that each time we call it won't be downloaded.

```
#download the model to local so it can be used again and again
!mkdir ../sentence_wise_email/module/module_useT
# Download the module, and uncompress it to the destination folder.
!curl -L "https://tfhub.dev/google/universal-sentence-encoder-
large/3?tf-hub-format=compressed" | tar -zxvC
../sentence_wise_email/module/module_useT
```

Here, "../sentence_wise_email/module/module_useT" is the folder where the sentence encoder files are downloaded. The encoder is optimized for greater-than-word length text, hence can be applied to sentences, phrases or short paragraphs.

For example (official site example):

```
embed = hub.Module("../sentence_wise_email/module/module_useT")

# Compute a representation for each message, showing various lengths
supported.
word = "Elephant"
sentence = "I am a sentence for which I would like to get its
embedding."
paragraph = (
    "Universal Sentence Encoder embeddings also support short
paragraphs. "
    "There is no hard limit on how long the paragraph is. Roughly,
the longer "
    "the more 'diluted' the embedding will be.")
messages = [word, sentence, paragraph]

# Reduce logging output.
tf.logging.set_verbosity(tf.logging.ERROR)

with tf.Session() as session:
    session.run([tf.global_variables_initializer(),
tf.tables_initializer()])
    message_embeddings = session.run(embed(messages))

for i, message_embedding in
enumerate(np.array(message_embeddings).tolist()):
        print("Message: {}".format(messages[i]))
        print("Embedding size: {}".format(len(message_embedding)))
        message_embedding_snippet = ", ".join((str(x) for x in
message_embedding[:3]))
```

```
        print("Embedding[{},...]\n".
              format(message_embedding_snippet))
```

The output it gives :

```
Message: Elephant
Embedding size: 512
Embedding: [0.04498474299907684, -0.05743394419550896,
0.002211471786722541, ...]

Message: I am a sentence for which I would like to get its
embedding.
Embedding size: 512
Embedding: [0.05568016692996025, -0.009607920423150063,
0.006246279925107956, ...]

Message: Universal Sentence Encoder embeddings also support short
paragraphs. There is no hard limit on how long the paragraph is.
Roughly, the longer the more 'diluted' the embedding will be.
Embedding size: 512
Embedding: [0.03874940797686577, 0.0765201598405838,
-0.0007945669931359589, ...]
```

As it can be seen whether it is a word, sentence or phrase, the sentence encoder is able to give an embedding vector of size 512.

How to use in Rest API

While using it in Rest API, you have to call it multiple times. Calling the module and session, again and again, will be very time-consuming. (~16s for each call from our testing). One thing can be done is to call the module and create the session at the start, and continue reusing it. (The first call takes ~16s and then consecutive calls in ~.3s).

```
#Function so that one session can be called multiple times.
#Useful while multiple calls need to be done for embedding.
import tensorflow as tf
import tensorflow_hub as hub
def embed_useT(module):
    with tf.Graph().as_default():
        sentences = tf.placeholder(tf.string)
        embed = hub.Module(module)
        embeddings = embed(sentences)
        session = tf.train.MonitoredSession()
    return lambda x: session.run(embeddings, {sentences: x})

embed_fn = embed_useT('../sentence_wise_email/module/module_useT')
messages = [
    "we are sorry for the inconvenience",
    "we are sorry for the delay",
    "we regret for your inconvenience",
    "we don't deliver to baner region in pune",
    "we will get you the best possible rate"
]
embed_fn(messages)
```

The output is a matrix of dimension 5*512. (each sentence is a vector of size 512). Since the values are normalized, the inner product of encodings can be treated as a similarity matrix.

```
encoding_matrix = embed_fn(messages)
import numpy as np
np.inner(encoding_matrix, encoding_matrix)
```

The output is:

```
array([[1.        , 0.87426376, 0.8004891 , 0.23807861, 0.46469775],
       [0.87426376, 1.0000001 , 0.60501504, 0.2508136 , 0.4493388 ],
       [0.8004891 , 0.60501504, 0.9999998 , 0.1784874 , 0.4195464 ],
       [0.23807861, 0.2508136 , 0.1784874 , 1.0000001 , 0.24955797],
       [0.46469775, 0.4493388 , 0.4195464 , 0.24955797, 1.0000002
]],
      dtype=float32)
```

As it can be seen here, the similarity between "we are sorry for the inconvenience" and "we are sorry for the delay" is 0.87 (row-1, col-2), while the similarity between "we are sorry for the inconvenience" and "we will get you the best possible rate" is 0.46 (row-1, col-5), which is amazing. There are other ways of finding the similarity score from encodings like cosine similarity, Manhattan distance etc. (code is there in my Github repo mentioned at the end of the article).

Removing duplicate texts

While working on a question-answer verification system, one of the major problems was the repeating of answer statements. Traversing through the available data at us, we found the sentences having the semantic similarity score > 0.8 (calculated through the inner product of encodings as mentioned above) are actually duplicate statements so we removed them.

```
#It takes similarity matrix (generated from sentence encoder) as
input and gives index of redundant statements
def redundant_sent_idx(sim_matrix):
    dup_idx = []
    for i in range(sim_matrix.shape[0]):
        if i not in dup_idx:
            tmp = [t+i+1 for t in list(np.where( sim_matrix[i][i+1:]
> 0.8 )[0])]
            dup_idx.extend(tmp)
    return dup_idx
#indexes of duplicate statements.
dup_indexes  = redundant_sent_idx(np.inner(encoding_matrix,
                                           encoding_matrix))

unique_messages = np.delete(np.array(messages), dup_indexes)
```

Now the unique messages were:

```
array(['we are sorry for the inconvenience',
       "we don't deliver to baner region in pune",
       'we will get you the best possible rate'], dtype='<U40')
```

Basically, it discarded the statements “we are sorry for the delay” and “we regret for your inconvenience” as they are the duplicate of sentence 1.

Performing classification by finding semantically similar sentences

While building an answer evaluation system, we came across the problem of detecting the follow-up statements. But there was not enough data to train a statistical model. We could solve the problem, thanks to the Universal Sentence Encoder. What approach we followed is, create a matrix of encodings of all the available data. Then get the encoding of user input and see if it is more than 60% similar to any of the available data, take it as a follow-up. A simple greetings identification can be:

```
greets = ["What's up?",
 'It is a pleasure to meet you.',
 'How do you do?',
 'Top of the morning to you!',
 'Hi',
 'How are you doing?',
 'Hello',
 'Greetings!',
 'Hi, How is it going?',
 'Hi, nice to meet you.',
 'Nice to meet you.']
greet_matrix = embed_fn(greets)
test_text = "Hey, how are you?"
test_embed = embed_fn([test_text])
np.inner(test_embed, greet_matrix)
sim_matrix  = np.inner(test_embed, greet_matrix)
if sim_matrix.max() > 0.8:
    print("it is a greetings")
else:
    print("it is not a greetings")
```

sambit9238/Deep-Learning

Implementations of Deep Learning techniques in the fields of NLP, Computer Vision etc. - sambit9238/Deep-Learning

github.com

https://tfhub.dev/google/universal-sentence-encoder-large/3

Universal Sentence Encoder

We present models for encoding sentences into embedding vectors that specifically target transfer learning to other NLP...

arxiv.org

Machine Learning   Tensorflow Hub   Transfer Learning   NLP   Deep Learning

About   Help   Legal