Technical Review: TensorFlow to implement TF-IDF and other methods

Chengbo Jiang
CS410 Fall 2020

What is TensorFlow?

Currently, the most famous deep learning library in the world is Google's TensorFlow. Google product uses machine learning in all of its products to improve the search engine, translation, image captioning or recommendations.

TensorFlow Architecture

Tensorflow architecture works in three parts:

- **Preprocessing the data**
- **Build the model**
- **Train and estimate the model**

It is called Tensorflow because it takes input as a multi-dimensional array, also known as tensors. You can construct a sort of flowchart of operations (called a Graph) that you want to perform on that input. The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output.

Why is TensorFlow popular?

TensorFlow is the best library of all because it is built to be accessible for everyone. Tensorflow library incorporates different API to built at scale deep learning architecture like CNN or RNN. TensorFlow is based on graph computation; it allows the developer to visualize the construction of the neural network with Tensorboad. This tool is helpful to debug the program. Finally, Tensorflow is built to be deployed at scale. It runs on CPU and GPU.
Tensorflow attracts the largest popularity on GitHub compare to the other deep learning framework.
Therefore it is practical to review TensorFlow application in NLP.

TensorFlow: tft.tfidf

tft.tfidf(x, vocab_size, smooth=True, name=None)

The term frequency of a term in a document is calculated as (count of term in document) / (document size)
The inverse document frequency of a term is, by default, calculated as 1 + log((corpus size + 1) / (count of documents containing term + 1)).

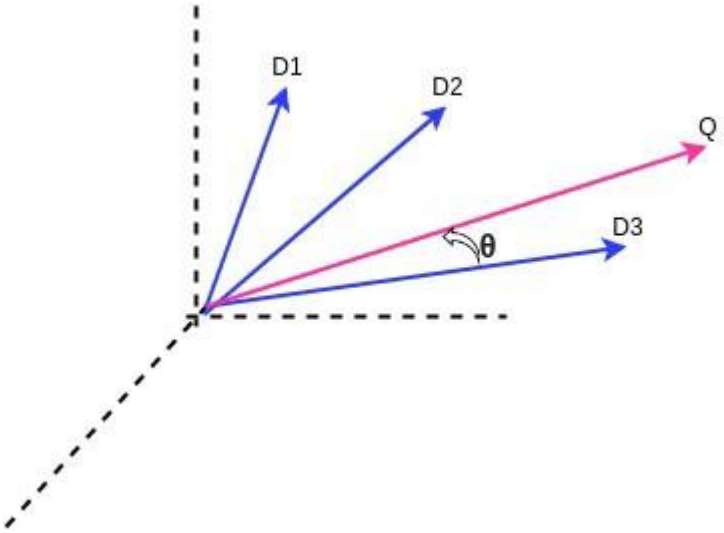
Args	
`x`	A `SparseTensor` representing int64 values (most likely that are the result of calling `compute_and_apply_vocabulary` on a tokenized string).
`vocab_size`	An int - the count of vocab used to turn the string into int64s including any OOV buckets.
`smooth`	A bool indicating if the inverse document frequency should be smoothed. If True, which is the default, then the idf is calculated as 1 + log((corpus size + 1) / (document frequency of term + 1)). Otherwise, the idf is 1 +log((corpus size) / (document frequency of term)), which could result in a division by zero error.
`name`	(Optional) A name for this operation.

Returns
Two `SparseTensors` with indices [index_in_batch, index_in_bag_of_words]. The first has values vocab_index, which is taken from input `x`. The second has values tfidf_weight.

Document Search Review (Theory from CS410 and Google Universal sentence encoder)

- **Calculating the ranking by using cosine similarity**

It is the most common metric used to calculate the similarity between document text from input keywords/sentences. Mathematically, it measures the cosine of the angle b/w two vectors projected in a multi-dimensional space.



Cosine Similarity b/w document to query

In the above diagram, have 3 document vector value and one query vector in space. when we are calculating the cosine similarity b/w above 3 documents. The most similarity value will be D3 document from three documents.

- **TF-IDF:**

TF-IDF stands for "Term Frequency — Inverse Document Frequency". This is a technique to calculate the weight of each word signifies the importance of the word in the document and corpus. This algorithm is mostly using for the retrieval of information and text mining field.

The number of times a word appears in a document divided by the total number of words in the document. Every document has its term frequency.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

The log of the number of documents divided by the number of documents that contain the word w. Inverse data frequency determines the weight of rare words across all documents in the corpus.

$$idf(w) = log(\frac{N}{df_t})$$

Lastly, the TF-IDF is simply the TF multiplied by IDF.

TF-IDF = Term Frequency (TF) * Inverse Document Frequency (IDF)

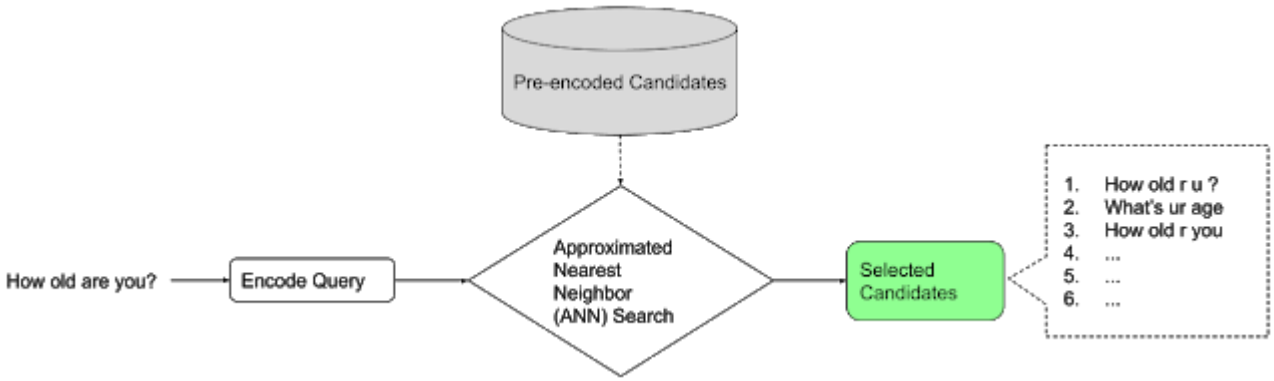
$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

- **Introduction Google USE**

  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 others trained with Deep Averaging Network (DAN). They are pre-trained on a large corpus and can be used in a variety of tasks (sentimental analysis, classification and so on). 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 expensive. The one with DNA encoding is computationally less expensive and with little lower accuracy.

  Here we are using Second one DAN Universal sentence encoder as available in this URL:- Google USE DAN Model

  Both models take a word, sentence or a paragraph as input and output a 512-dimensional vector.



  A prototypical semantic retrieval pipeline, used for textual similarity.
  Before using the TensorFlow-hub model.

Example of implementation

Dataset:

Here, we are using 20newsgroup dataset to the analysis of a text search engine giving input keywords/sentences input.

```
In [1]: import pandas as pd

In [2]: news = pd.read_json('https://raw.githubusercontent.com/zayedrais/DocumentSearchEngine/master/
data/newsgroups.json')

In [3]: news
Out[3]:
                                               content  ...              target_names
0      From: lerxst@wam.umd.edu (where's my thing)\nS...  ...                 rec.autos
1      From: guykuo@carson.u.washington.edu (Guy Kuo)...  ...     comp.sys.mac.hardware
2      From: twillis@ec.ecn.purdue.edu (Thomas E Will...  ...     comp.sys.mac.hardware
3      From: jgreen@amber (Joe Green)\nSubject: Re: W...  ...            comp.graphics
4      From: jcm@head-cfa.harvard.edu (Jonathan McDow...  ...                sci.space
...                                                  ...  ...                      ...
11309  From: jim.zisfein@factory.com (Jim Zisfein) \n...  ...                sci.med
11310  From: ebodin@pearl.tufts.edu\nSubject: Screen ...  ...     comp.sys.mac.hardware
11311  From: westes@netcom.com (Will Estes)\nSubject:...  ...  comp.sys.ibm.pc.hardware
11312  From: steve@hcrlgw (Steven Collins)\nSubject: ...  ...            comp.graphics
11313  From: gunning@cco.caltech.edu (Kevin J. Gunnin...  ...          rec.motorcycles

[11314 rows x 3 columns]
```

Data cleaning:

```
In [5]: import re

In [6]: for i,txt in enumerate(news['content']):
   ...:     subject = re.findall('Subject:(.*\n)',txt)
   ...:     if (len(subject) !=0):
   ...:         news.loc[i,'Subject'] =str(i)+' '+subject[0]
   ...:     else:
   ...:         news.loc[i,'Subject'] ='NA'
   ...: df_news =news[['Subject','content']]
```

There are additional steps of processing of data such as:

- Conversion the text into a lower form
- **Word tokenization** is the process to divide the sentence into the form of a word.
- **Stop words** are the most commonly occurring words which don't give any additional value to the document vector. in-fact removing these will increase computation and space efficiency.
- **Lemmatisation** is a way to reduce the word to root synonym of a word. Unlike Stemming, Lemmatisation makes sure that the reduced word is again a dictionary word (word present in the same language).

Generated TF-IDF by using TfidfVectorizer from Sklearn

```
In [8]: import pandas as pd
   ...: import numpy as np
   ...: import os
   ...: import re
   ...: import operator
   ...: import nltk
   ...: from nltk.tokenize import word_tokenize
   ...: from nltk import pos_tag
   ...: from nltk.corpus import stopwords
   ...: from nltk.stem import WordNetLemmatizer
   ...: from collections import defaultdict
   ...: from nltk.corpus import wordnet as wn
   ...: from sklearn.feature_extraction.text import TfidfVectorizer
```

```
In [10]: from sklearn.feature_extraction.text import TfidfVectorizer
    ...: import operator## Create Vocabulary
    ...: vocabulary = set()for doc in df_news.Clean_Keyword:
    ...:     vocabulary.update(doc.split(','))vocabulary = list(vocabulary)# Intializating the tfIdf
model
    ...: tfidf = TfidfVectorizer(vocabulary=vocabulary)# Fit the TfIdf model
    ...: tfidf.fit(df_news.Clean_Keyword)# Transform the TfIdf model
    ...: tfidf_tran=tfidf.transform(df_news.Clean_Keyword)
```

```
In [13]: def gen_vector_T(tokens):Q = np.zeros((len(vocabulary)))
    ...:     x= tfidf.transform(tokens)
    ...:     #print(tokens[0].split(','))
    ...:     for token in tokens[0].split(','):
    ...:         #print(token)
    ...:         try:
    ...:             ind = vocabulary.index(token)
    ...:             Q[ind]  = x[0, tfidf.vocabulary_[token]]
    ...:         except:
    ...:             pass
    ...:     return Q
```

Cosine Similarity function for the calculation

```
In [11]: def cosine_sim(a, b):
    ...:     cos_sim = np.dot(a, b)/(np.linalg.norm(a)*np.linalg.norm(b))
    ...:     return cos_sim
    ...:
```

Cosine Similarity b/w document to query function

```
In [12]: def cosine_similarity_T(k, query):
    ...:     preprocessed_query = preprocessed_query = re.sub("\W+", " ", query).strip()
    ...:     tokens = word_tokenize(str(preprocessed_query))
    ...:     q_df = pd.DataFrame(columns=['q_clean'])
    ...:     q_df.loc[0,'q_clean'] =tokens
    ...:     q_df['q_clean'] =wordLemmatizer(q_df.q_clean)
    ...:     d_cosines = []
    ...:
    ...:     query_vector = gen_vector_T(q_df['q_clean'])
    ...:     for d in tfidf_tran.A:
    ...:         d_cosines.append(cosine_sim(query_vector, d))
    ...:
    ...:     out = np.array(d_cosines).argsort()[-k:][::-1]
    ...:     #print("")
    ...:     d_cosines.sort()
    ...:     a = pd.DataFrame()
    ...:     for i,index in enumerate(out):
    ...:         a.loc[i,'index'] = str(index)
    ...:         a.loc[i,'Subject'] = df_news['Subject'][index]
    ...:     for j,simScore in enumerate(d_cosines[-k:][::-1]):
    ...:         a.loc[j,'Score'] = simScore
    ...:     return a
    ...:
```

Testing the function

```
In [14]: cosine_similarity_T(10,'computer science')
```

	index	Subject	Score
0	2231	2231 Computer Engr vs Computer Science	0.396137
1	10340	10340 Science and Methodology	0.302603
2	4173	4173 Rawlins debunks creationism	0.284936
3	4326	4326 Computer Engr vs Computer Science	0.284256
4	6921	6921 Automatic layout of state diagrams	0.260732
5	7618	7618 Solution Why do I need xrdb m when Xdefau...	0.251170