Ankur Aggarwal
Net-Id: ankura2
CS 410
Fall 2021

NLP & TensorFlow – Tech Review

Introduction

Natural Language Processing

Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language.

The primary goal of NLP is to read, interpret, comprehend, and beneficially make sense of human language. Human speech, unlike computer language, is not necessarily exact and organized – it contains ambiguity, and complex factors such as slang, regional dialects, and social context may exist in its linguistic structure.

Google Translate uses natural language processing to translate across languages. Personal Assistant Apps like Siri, Alexa, and Bixby also uses NLP to make dialogues and comprehend speech.

Let’s talk about TensorFlow. Text-based apps, picture identification, voice search, and many other applications employ TensorFlow. TensorFlow finds its use in image identification in Facebook's image recognition technology. Apple's Siri also uses it for voice recognition. Google also uses TensorFlow in almost every app we use to improve our experience.

TensorFlow Overview

Machine learning is a complex discipline. But thanks to machine learning frameworks like TensorFlow that made the process of acquiring data, training models, serving predictions, and refining future results simpler than before, utilizing machine learning models is much less intimidating and complicated than before.

Ankur Aggarwal
Net-Id: ankura2
CS 410
Fall 2021

TensorFlow is an open-source software platform for machine learning. TensorFlow focuses on the training and inference of deep neural networks but has found uses in a wide range of applications. DistBelief, a proprietary machine-learning system based on deep learning neural networks, was built by Google Brain beginning in 2011. The DistBelief codebase was simplified and refactored into TensorFlow, a quicker, more robust application-grade framework. The team, led by Geoffrey Hinton, introduced extended backpropagation and other enhancements in 2009, allowing for the formation of neural networks with significantly improved accuracy, such as a 25% decrease in voice recognition mistakes. Since its inception, its use in both research and corporate applications developed quickly across several Alphabet entities.

Consider a world where there are no traffic accidents or incidences of road rage. Consider a world where every operation is successful with no human life lost to surgical mistakes. Consider a future where no kid is underprivileged and where those with special needs may enjoy the same standard of living like the rest of humankind.

NLP using TensorFlow

Overview

A good NLP system can execute a wide range of activities. You can ask Google to give weather updates or even ask how to say "How are you?" in any other language like Spanish. Below are some of the activities that a good NLP can perform using TensorFlow:

- **Tokenization**

  The process of tokenization is diving strings into tokens. tensorflow_text package provides tokenizers that are available for preprocessing text required for text-based models. If you tokenize using the TensorFlow graph, you won't have to worry about training and interference workflows and how to manage preprocessing scripts.

- **Word-sense Disambiguation (WSD) :** Word-sense-disambiguation (WSD) is a problem where we try to make sense of a word in a given phrase. Natural language is too ambiguous, where words can have a different context. This activity becomes extremely important to increase the effectiveness of complex systems like search engines.

- **Part-of-Speech (PoS) tagging: POS tagging is** the process of marking up a word in a text (corpus) as corresponding to a particular part of the speech. This activity is not an easy task, as some words can represent more than one part of the speech at different time. This phenomenon is not rare and make POS tagging complex.

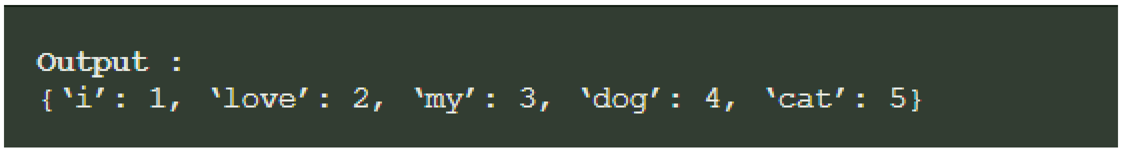
TensorFlow has built-in high-level API named tf.keras to define and train machine learning models and to make predictions. tf.keras is the TensorFlow variant of the open-source Keras API.

In the rech review we will do hands-on on how we can train NLP models

Tokenization

Representing the words in a way that a computer can process them, with a view to later training a Neural network that can understand their meaning. This process is called tokenization.

```
#
Importing
required
libraries
         import tensorflow as tf
         from tensorflow import keras
         from tensorflow.keras.preprocessing.text import Tokenizer


         # List of sample sentences that we want to tokenize
         sentences = ['I love my dog',
                      'I love my cat',
                      ]

         # intializing a tokenizer that can index
         # num_words is the maximum number words that can be kept
         # tokenizer will automatically help in choosing most frequent words
         tokenizer = Tokenizer(num_words = 100)

         # fitting the sentences to using created tokenizer object
         tokenizer.fit_on_texts(sentences)

         # the full list of words is available as the tokenizer's word index
         word_index = tokenizer.word_index

         # the result will be a dictionary, key being the words and the values being the token
         for that word
         print(word_index)
```

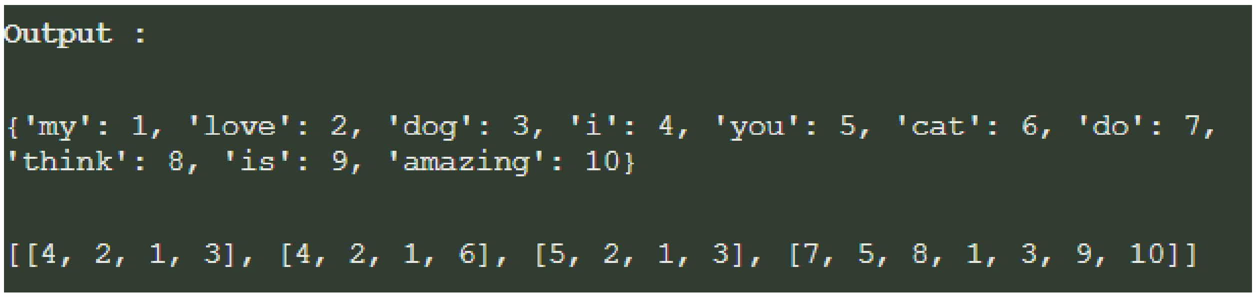


```
Output :
{'i': 1, 'love': 2, 'my': 3, 'dog': 4, 'cat': 5}
```

Sequencing

Now that our words are represented like this, next, we need to represent our sentences by a sequence of numbers in the correct order. Then we will have data ready for processing by a neural network to understand or maybe even generate new text. Let’s look at how we can manage this sequencing using TensorFlow tools

```
sentences
= ['I
love my
dog',
                      'I love my cat',
                      'you love my dog!',
                      'Do you think my dog is amazing?',
                      ]


         tokenizer = Tokenizer(num_words = 100)
         tokenizer.fit_on_texts(sentences)
         word_index = tokenizer.word_index


         # this creates sequence of tokens representing each
         sentence
         sequences = tokenizer.texts_to_sequences(sentences)


         print(word_index)
         print(sequences)
```



```
Output :

{'my': 1, 'love': 2, 'dog': 3, 'i': 4, 'you': 5, 'cat': 6, 'do': 7,
'think': 8, 'is': 9, 'amazing': 10}

[[4, 2, 1, 3], [4, 2, 1, 6], [5, 2, 1, 3], [7, 5, 8, 1, 3, 9, 10]]
```

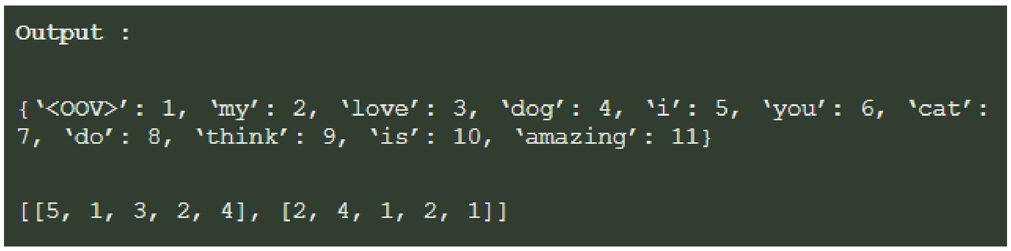
Ankur Aggarwal
Net-Id: ankura2
CS 410
Fall 2021

Unseen Words

So we can imagine that we need a huge word index to handle sentences that are not in the training set. But in order not to lose the length of the sequence, there is also a little trick that we can use. Let's take a look at that.

By using the OOV(out of vocabulary) token property, and setting it as something that you would not expect to see in the corpus, like "<OOV>", this word is never used anywhere, so we can use a word that we can assume never appears in a text. Then the tokenizer will create a token for that and replaces words that it doesn't recognize with the out of vocabulary token instead. It's simple but effective. Let's look at an example.

```
sentences
= ['I
love my
dog',
                      'I love my cat',
                      'you love my dog!',
                      'Do you think my dog is amazing?',
                      ]

         # adding a "out of vocabulary" word to the tokenizer
         tokenizer = Tokenizer(num_words = 100,oov_token="<OOV>")
         tokenizer.fit_on_texts(sentences)
         word_index = tokenizer.word_index
         sequences = tokenizer.texts_to_sequences(sentences)

         test_data = ['i really love my dog',
                      'my dog loves my manatee',
                      ]

         test_seq = tokenizer.texts_to_sequences(test_data)

         print(word_index)
         print(test_seq)
```



```
Output :

{'<OOV>': 1, 'my': 2, 'love': 3, 'dog': 4, 'i': 5, 'you': 6, 'cat':
7, 'do': 8, 'think': 9, 'is': 10, 'amazing': 11}

[[5, 1, 3, 2, 4], [2, 4, 1, 2, 1]]
```

Padding the sequence

A simple solution is padding. For this, we will use pad_sequences imported for the sequence module of tensorflow.keras.preprocessing. As the name suggests, we can use it to pad our sequence. So we just need to pass sequences to pad_sequence function and the rest is done for us.

```
from
tensorflow.keras.preprocessing.sequence
import pad_sequences

                                       sentences = ['I love my dog',
                                                    'I love my cat',
                                                    'you love my dog!',
                                                    'Do you think my dog is amazing?',
                                                    ]

                                       tokenizer = Tokenizer(num_words =
                                       100,oov_token="<OOV>")
                                       tokenizer.fit_on_texts(sentences)
                                       word_index = tokenizer.word_index
                                       sequences = tokenizer.texts_to_sequences(sentences)

                                       # padding sequences
                                       padded = pad_sequences(sequences)

                                       print(word_index)
                                       print(sequences)
                                       print(padded)
```

```
Output:

{'<OOV>': 1, 'my': 2, 'love': 3, 'dog': 4, 'i': 5, 'you': 6, 'cat':
7, 'do': 8, 'think': 9, 'is': 10, 'amazing': 11}

[[5, 3, 2, 4], [5, 3, 2, 7], [6, 3, 2, 4], [8, 6, 9, 2, 4, 10, 11]]

[[ 0  0  0  5  3  2  4]
 [ 0  0  0  5  3  2  7]
 [ 0  0  0  6  3  2  4]
 [ 8  6  9  2  4 10 11]]
```

Splitting data to train and test sets

Now that we have 3 lists one with our labels, one with the text, and one with the URLs, we start doing our familiar steps tokenizing and sequencing the words. But calling tokenizer.fit_on_texts with the headlines, we 'll create tokens for every word in the corpus and we'll see them in the word index.

Now there's a problem here. We don't have a split in the data for training and testing. We just have a list of 26,09 sequences. Fortunately, python makes it super easy for us to slice this up.

```
training_size
= 20000

             training_sentences = sentences[0:training_size]
             testing_sentences = sentences[training_size:]

             training_labels = labels [0:training_size]
             testing_labels = labels [training_size:]
```

Reference

https://en.wikipedia.org/wiki/TensorFlow

https://www.mygreatlearning.com/blog/what-is-tensorflow-machine-learning-library-explained/#1

https://aqsakausar30.medium.com/nlp-in-tensorflow-all-you-need-for-a-kickstart-3293d7d2630e

Natural Language Processing with TensorFlow by Thushan Ganegedara

https://www.tensorflow.org/text/guide/tokenizers