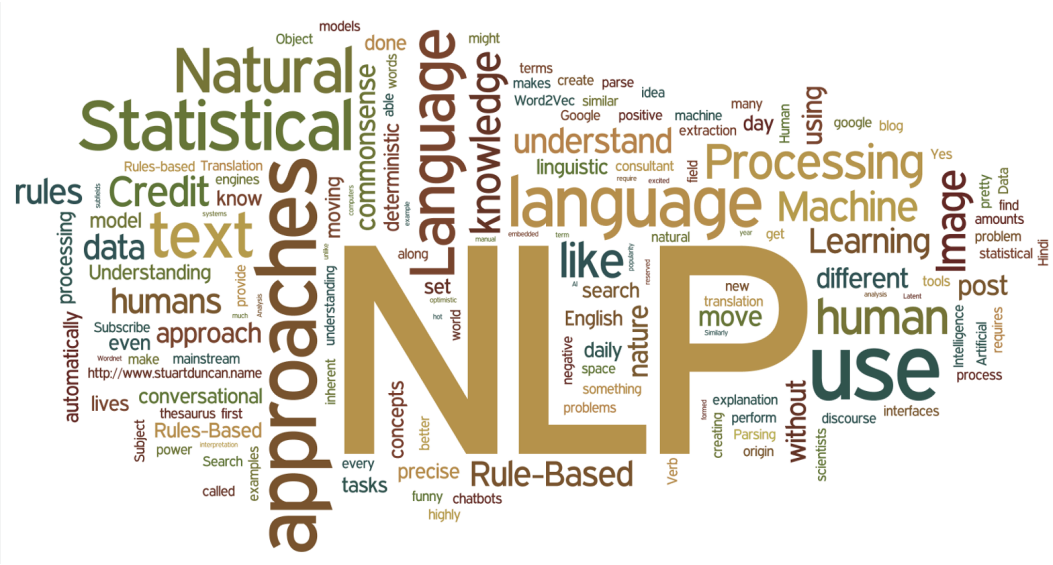
NLP with TensorFlow

Menyhárt Róbert

March 2020



1 Overview

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead.Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.

Natural language processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.

Sentiment analysis is the process of computationally identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writers attitude towards a target (a particular topic, a product, etc) is positive, neutral or negative.

The aim of the project is to create a software which will predict the positivity of a given review, by analyzing many different reviews of products and services from the internet.

The application will be written in Python using TensorFlow,a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.

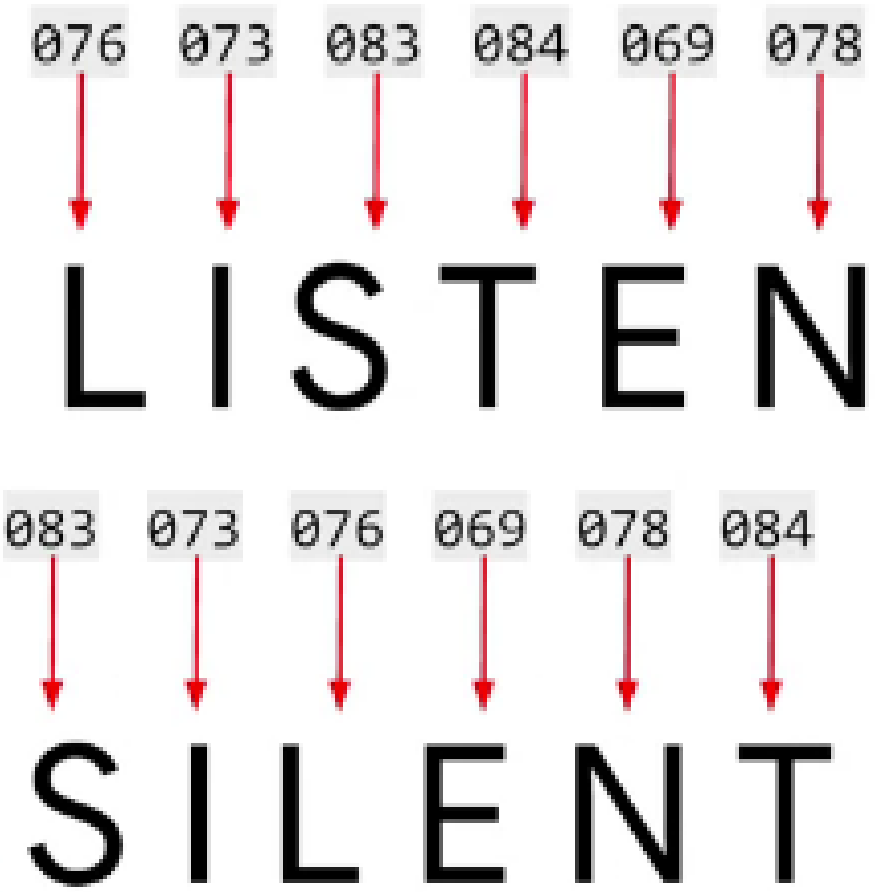
2 Main functionalities

2.1 Preprocessing data

There are countless online datasets from which we can acquire the training data for the system.Data from different sources can vary depending on who collected it, how it was collected, etc. Before the data can be fed to a neural network in order to train it, the data must be converted in a format the system will understand.TensorFlow offers many easy to use functionalities for this.

2.1.1.Encoding text data

We need to encode our data such that our system will be able to extract meaning from the input text. So, one possibility would be to encode every character by its ASCII code, but the problem is that the semantics of the word are not encoded in individual letters, for example:



Its easy to see that two words with almost the opposite meaning could be easily confused by our program, because they contain the same exact characters.We could take into consideration the order of the letters, but its not obvious how one could do that, and because human languages evolved fairly randomly it's not sure if it can even be done. A better encoding is not encoding every character in a word,

but encoding every word which ever appeared in our dataset.When an unkown word pops up for the first time in our training data, we assign it an identifier (number), and that number will uniquely identify that particular word for the whole existence of the system. For example:

I	**love**	**my**	**dog.**
1	**2**	**3**	**4**

I	**love**	**my**	**cat.**
1	**2**	**3**	**5**

With this encoding, the program can know that these two sentences are similar, and that they only differ in their last words.

Code:

```
{
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer

sentences = [
'i love my dog',
'i love my cat'
]

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

Interpretation:

1. Importing TensorFlow.
2. Importing Keras (a higher level API of TensorFlow)
3. From Keras, we import the text processing functionalities.
4. We store the training data in 'sentences'.
5. We initialize the Tokenizer, which will split our sentences in words.
6. It will get the 'num_words' most used words from our data.
7. With fit_on_texts, we feed our data to the Tokenizer.

8. 'word_index' is a dictionary containing the word:identifier pairs.

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

After the word index is set, we can encode any text based on that word index.The tokenizer will simply run through the text, and change any word from the index with its identifier.

```
sentences = [
    'i love my dog',
    'my cat loves my dog',
    'i like my job'
]
sequences = tokenizer.texts_to_sequences(sentences)
```

Will result in:

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

Notice that words that were not previously 'fit_on_texts' are not assigned a new identifier.To cater for this, we can add to the Tokenizer an Out Of Vocabulary token, which will be a placeholder for all the words missing from the word$_index$.

```
tokenizer = Tokenizer(num_words=100, oov_token= '<OOV>')
```

Note: the 'oov_token' will always take the identifier 1.The new word index is:

```
{'<OOV>' : 1,'i' : 2,'love' : 3,'my' : 4,'dog' : 5,'cat' : 6}
```

And the new sequences:

```
sequences = [[2, 3, 4, 5], [4, 6, 1, 4, 5], [2, 1, 4, 1]]
```

2.1.2.Fitting the data to a model

Our input values will have to be fed to the first (input) layer of a neural network, which has a fixed number of nodes.In order to do this, we need to convert the data to have a fixed length, i.e. pad the data with a value missing from the vocabulary for if we have less words, and cutting any extra words of if we have too many.This can be easily done with the 'pad_sequences' method:

```
padded_sequences = pad_sequences(sequences, max_length=6)
```

Now the sequences are ready to be fed to a model:

```
padded_sequences = [
    [0, 0, 2, 3, 4, 5],
    [0, 4, 6, 1, 4, 5],
    [0, 0, 2, 1, 4, 1]
]
```

2.1.3.Creating a model

Creating a model can be simply done by adding the needed layers (depending on the application) in order, the input layer being the first, and the output layer being the last.The Keras API contains the most used layers ready for use.For example:

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
]
)
```

2.1.3.Training the model

First, the model has to be compiled, and chosen a loss function, an optimizer and the metrics which will be evaluated during training.

```
model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['accuracy'])
```

After this, we can train the model by fitting the training data on it

```
model.fit(
    padded_sequences,
    training_labels,
    epochs=10,
    validation_data = (testing_padded, testing_labels_final)
)
```

- 'padded_sequences' contains the padded input sequences
- 'training_labels' contains the correct output for a given input
- 'epochs' number of times will our training_data be fed into the network (in a random order)
- 'validation_data' is data withheld from training, only used to estimate the models accuracy

2.1.4.Making predictions

Obviously the purpose of the training was to set up a model which can predict the sentiment (rating) of a text (review) without knowing what its output should be.This can be done with the models 'predict' method:

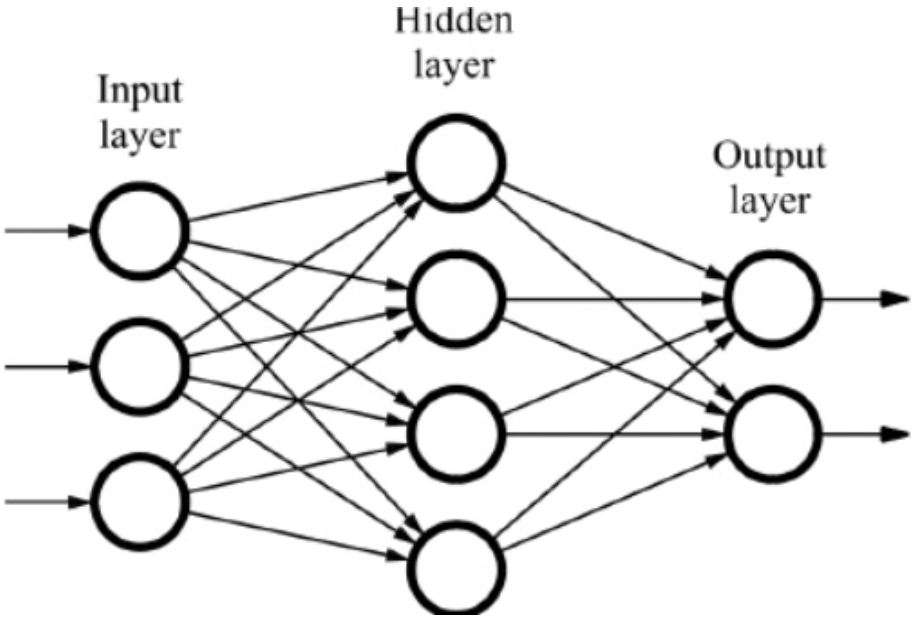
```
prediction = model.predict([test_review])
print(prediction[0])
```

Note:For the model to be able to predict for multiple cases at once, the 'predict' method takes as parameter a list of reviews, and returns a list of predictions.

3 Detailed description of the main algorithms

3.1 Neural networks

A model in TensorFlow basically implements a neural network, but how does a neural network (or ANN, Artificial Neural Network) work?



Briefly, a neural network consists of multiple neurons (nodes), each holding a value called activation value.The neurons are ordered in layers, the first of which is the input layer, the last one is the output layer, and the layers in between these two are called the hidden layers.Each neuron from a layer is connected to every neuron in the next layer, and each of this connection has a weight assigned. The neurons from the next layer will have as activation value the weighted sum of the activation values from the current layer.And so, a chain reaction starts, and every layer activates specific neurons in the next layer, up until the output layer.

A layer usually has some bias, which is subtracted from the weighted sum and it means that only above that value will any neuron in the layer be activated.In practice, the weighted sums between the layers are squished trough a function (called activation function) to keep them in a given range.For example, the famous Sigmoid function, which maps any value to a value between 0 and 1.

A neural networks "learns" by continuosly feeding into its input layer our data, compare the output of the network with the expected output, and slightly tweaking the weights and biases such that the output will get closer to the expected value, but in a way that it won't perform much worse for the previously "learned" examples.

3.2 Flatten layer

It simply flattens the data, for example if the layers input is:

```
[["Hello "], ["World"], ["!"]]
```

It will output to the next layer:

```
["Hello World!"]
```

3.3 Dense layer

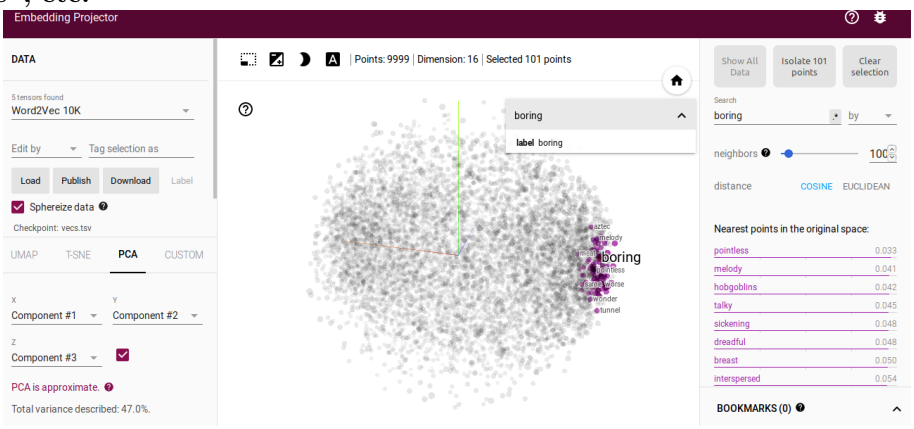
It is the classic neural network layer. It has a number of nodes, and an activation function through which the weighted sum will be passed.

3.4 Embedding layer

This is the layer which will perform the sentiment analysis itself.Embedding consists of vectors for each word with their associated sentiments.For example we have a movie review dataset, now depending on our labels, the embedding layer will learn 'meaning' of different words in a way that similar words will have similar vectors — good and nice will have similar vectors, bad and awful will have similar vectors. The embedding_dim defines the dimension of the vector space in which each word is represented. Each word will have a vector in this n dimensional space.During learning, the words with similar meaning (i.e. often appearing in similar contexts) will be slightly rotated towards each other, and away from the furthest vectors (the ones with opposite meaning).This way, the angle between two vectors will represent how close they are in meaning.

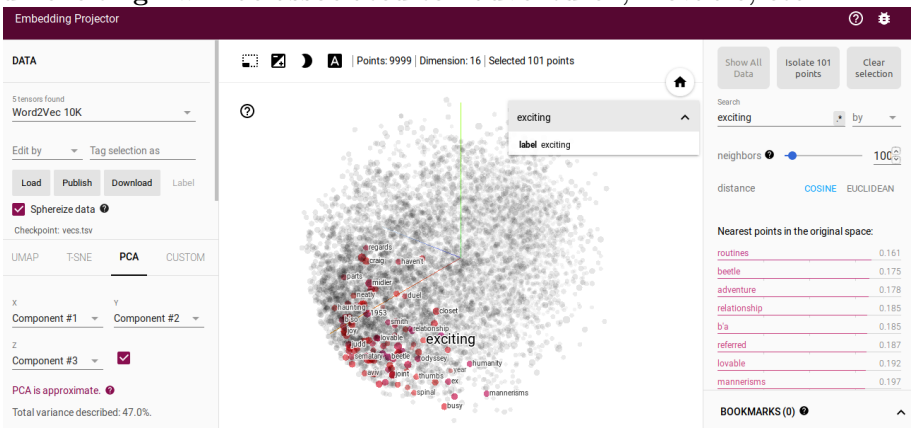
4 Examples

The word ”boring” will have similar meaning to ”dreadful”, ”pointless”, etc.



”

And ”exciting” will be associated to ”adventure”, ”lovable, etc.”



”

5 Proposed problem:general specification, source of data, related work

A human-written text is given to the system, which will try to predict the degree of satisfaction of the person. The satisfaction will be represented on a scale of 0-5 stars (because most companies like Yelp, Amazon, Google use this format) allowing for floating point values. The accuracy of the guesses will be raised by training the system with newly acquired data.

Datasets:

- Yelp Dataset: https://www.yelp.com/dataset/documentation/main
- Amazon Review Data: http://jmcauley.ucsd.edu/data/amazon/
- Movie Review Data: http://www.cs.cornell.edu/people/pabo/movie-review-data/
- Multi-Domain Reviews: http://www.cs.jhu.edu/ mdredze/datasets/sentiment/
- ...and many more