A Step-by-Step NLP Guide to Learn ELMo for Extracting Features from Text

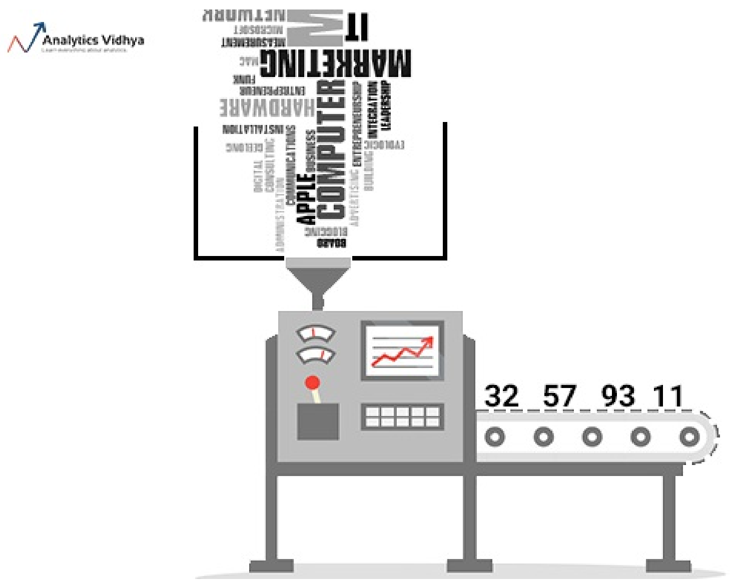
PRATEEK JOSHI, MARCH 11, 2019 LOGIN TO BOOKMARK THIS ARTICLE

Introduction

I work on different Natural Language Processing (NLP) problems (the perks of being a data scientist!). Each NLP problem is a unique challenge in its own way. That's just a reflection of how complex, beautiful and wonderful the human language is.

But one thing has always been a thorn in an NLP practitioner's mind is the inability (of machines) to understand the true meaning of a sentence. Yes, I'm talking about context. Traditional NLP techniques and frameworks were great when asked to perform basic tasks. Things quickly went south when we tried to add context to the situation.

The NLP landscape has significantly changed in the last 18 months or so. NLP frameworks like Google's BERT andZalando's Flair are able to parse through sentences and grasp the context in which they were written.



Embeddings from Language Models (ELMo)

One of the biggest breakthroughs in this regard came thanks to ELMo, a state-of-the-art NLP framework developed by AllenNLP. By the time you finish this article, you too will have become a big ELMo fan – just as I did.

In this article, we will explore ELMo (Embeddings from Language Models) and use it to build a mind-blowingNLP model using Python on a real-world dataset.

Note: This article assumes you are familiar with the different types of word embeddings and LSTM architecture. You can refer to the below articles to learn more about the topics:

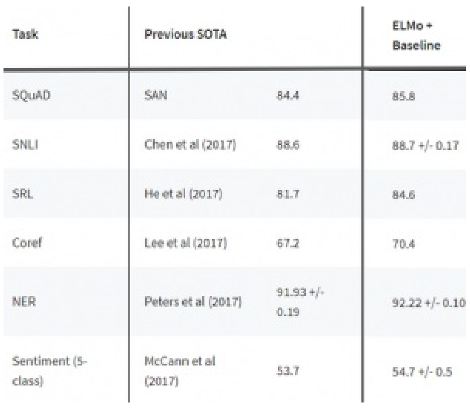
- *An Intuitive Understanding of Word Embeddings*
- *Essentials of Deep Learning : Introduction to Long Short Term Memory*

Table of Contents

What is ELMo?

No, the ELMo we are referring to isn't the character from Sesame Street! A classic example of the importance of context.

ELMo is a novel way to represent words in vectors or embeddings. These word embeddings are helpful in achieving state-of-the-art (SOTA) results in several NLP tasks:



Task	Previous SOTA		ELMo + Baseline
SQuAD	SAN	84.4	85.8
SNLI	Chen et al (2017)	88.6	88.7 +/- 0.17
SRL	He et al (2017)	81.7	84.6
Coref	Lee et al (2017)	67.2	70.4
NER	Peters et al (2017)	91.93 +/- 0.19	92.22 +/- 0.10
Sentiment (5-class)	McCann et al (2017)	53.7	54.7 +/- 0.5

NLP scientists globally have started using ELMo for various NLP tasks, both in research as well as the industry. You must check out the original ELMo research paper here – https://arxiv.org/pdf/1802.05365.pdf. I don't usually ask people to read research papers because they can often come across as heavy and complex but I'm making an exception for ELMo. This one is a really cool explanation of how ELMo was designed.

Understanding how ELMo works

Let's get an intuition of how ELMo works underneath before we implement it in Python. Why is this important?

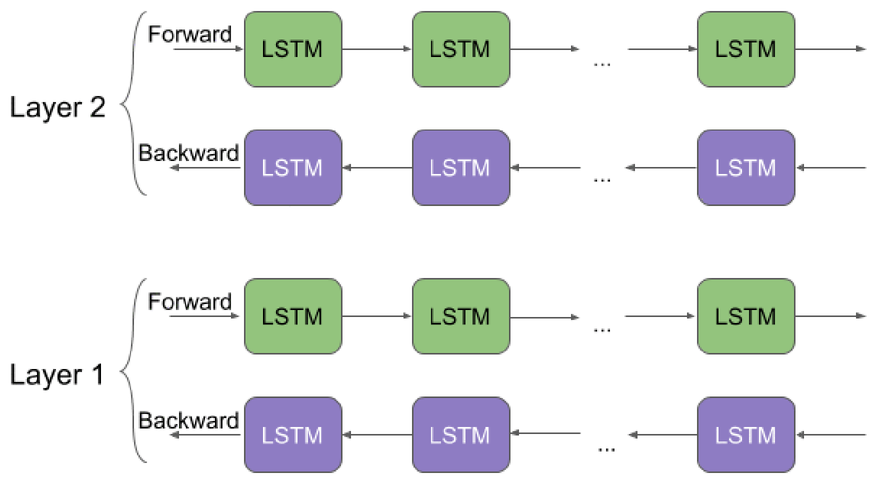
Well, picture this. You've successfully copied the ELMo code from GitHub into Python and managed to build a model on your custom

Well, picture this. You've successfully copied the ELMo code from GitHub into Python and managed to build a model on your custom text data. You get average results so you need to improve the model. How will you do that if you don't understand the architecture of ELMo? What parameters will you tweak if you haven't studied about it?

This line of thought applies to all machine learning algorithms. You need not get into their derivations but you should always know enough to play around with them and improve your model.

Now, let's come back to how ELMo works.

As I mentioned earlier, ELMo word vectors are computed on top of a two-layer bidirectional language model (biLM). This biLM model has two layers stacked together. Each layer has 2 passes — forward pass and backward pass:



- The architecture above uses a character-level convolutional neural network (CNN) to represent words of a text string into raw word vectors
- These raw word vectors act as inputs to the first layer of biLM
- The forward pass contains information about a certain word and the context (other words) before that word
- The backward pass contains information about the word and the context after it
- This pair of information, from the forward and backward pass, forms the intermediate word vectors
- These intermediate word vectors are fed into the next layer of biLM
- The final representation (ELMo) is the weighted sum of the raw word vectors and the 2 intermediate word vectors

As the input to the biLM is computed from characters rather than words, it captures the inner structure of the word. For example, the biLM will be able to figure out that terms like *beauty* and *beautiful* are related at some level without even looking at the context they often appear in. Sounds incredible!

How is ELMo different from other word embeddings?

Unlike traditional word embeddings such as word2vec and GLoVe, the ELMo vector assigned to a token or word is actually a function of the entire sentence containing that word. Therefore, the same word can have different word vectors under different contexts.

I can imagine you asking – how does knowing that help me deal with NLP problems? Let me explain this using an example.

Suppose we have a couple of sentences:

1. I ***read*** the book yesterday.
2. Can you ***read*** the letter now?

Take a moment to ponder the difference between these two. The verb “read” in the first sentence is in the past tense. And the same verb transforms into present tense in the second sentence. This is a case of **Polysemy** wherein a word could have multiple meanings or senses.

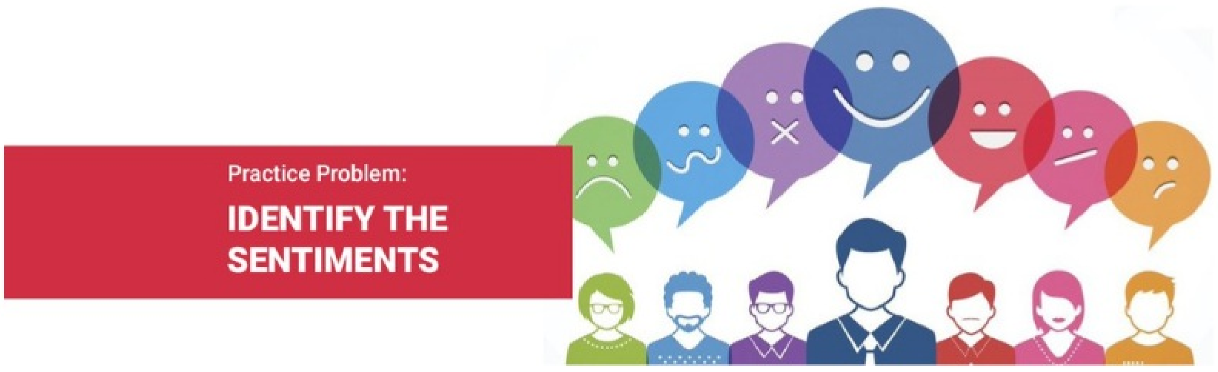
Language is such a wonderfully complex thing.

Traditional word embeddings come up with the same vector for the word “read” in both the sentences. Hence, the system would fail to distinguish between the polysemous words. These word embeddings just cannot grasp the context in which the word was used.

ELMo word vectors successfully address this issue. ELMo word representations take the entire input sentence into equation for calculating the word embeddings. Hence, the term “read” would have different ELMo vectors under different context.

Implementation: ELMo for Text Classification in Python

And now the moment you have been waiting for – implementing ELMo in Python! Let’s take this step-by-step.



1. Understanding the Problem Statement

The first step towards dealing with any data science challenge is defining the problem statement. It forms the base for our future actions.

For this article, we already have the problem statement in hand:

Sentiment analysis remains one of the key problems that has seen extensive application of natural language processing (NLP). This time around, given the tweets from customers about various tech firms who manufacture and sell mobiles, computers, laptops, etc., the task is to identify if the tweets have a negative sentiment towards such companies or products.

It is clearly a binary text classification task wherein we have to predict the sentiments from the extracted tweets.

2. About the Dataset

Here’s a breakdown of the dataset we have:

- The train set contains 7,920 tweets
- The test set contains 1,953 tweets

You can download the dataset from this page. *Note that you will have to register or sign-in to do so.*

Caution: Most profane and vulgar terms in the tweets have been replaced with “$&@*#”. However, please note that the dataset might still contain text that could be considered profane, vulgar, or offensive.

Alright, let's fire up our favorite Python IDE and get coding!

3. Import Libraries

Import the libraries we'll be using throughout our notebook:

```python
import pandas as pd
import numpy as np
import spacy
from tqdm import tqdm
import re
import time
import pickle
pd.set_option('display.max_colwidth', 200)
```

import_libraries_elmo.py hosted with ❤ by **GitHub** **view raw**

4. Read and Inspect the Data

```
# read data
train = pd.read_csv("train_2kmZucJ.csv")
test = pd.read_csv("test_oJQbWVk.csv")

train.shape, test.shape
```

Output: ((7920, 3), (1953, 2))

The train set has 7,920 tweets while the test set has only 1,953. Now let's check the class distribution in the train set:

```
train['label'].value_counts(normalize = True)
```

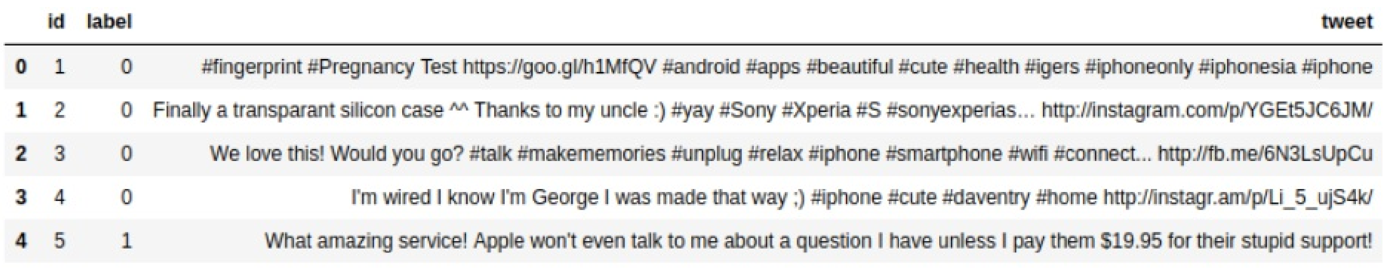
Output:

```
0    0.744192
1    0.255808
Name: label, dtype: float64
```

Here, 1 represents a negative tweet while 0 represents a non-negative tweet.

Let's take a quick look at the first 5 rows in our train set:

```
train.head()
```



	id	label	tweet
0	1	0	#fingerprint #Pregnancy Test https://goo.gl/h1MfQV #android #apps #beautiful #cute #health #igers #iphoneonly #iphonesia #iphone
1	2	0	Finally a transparant silicon case ^^ Thanks to my uncle :) #yay #Sony #Xperia #S #sonyexperias… http://instagram.com/p/YGEt5JC6JM/
2	3	0	We love this! Would you go? #talk #makememories #unplug #relax #iphone #smartphone #wifi #connect... http://fb.me/6N3LsUpCu
3	4	0	I'm wired I know I'm George I was made that way ;) #iphone #cute #daventry #home http://instagr.am/p/Li_5_ujS4k/
4	5	1	What amazing service! Apple won't even talk to me about a question I have unless I pay them $19.95 for their stupid support!

We have three columns to work with. The column 'tweet' is the independent variable while the column 'label' is the target variable.

5. Text Cleaning and Preprocessing

We would have a clean and structured dataset to work with in an ideal world. But things are not that simple in NLP (yet).

We need to spend a significant amount of time cleaning the data to make it ready for the model building stage. Feature extraction from the text becomes easy and even the features contain more information. You'll see a meaningful improvement in your model's performance the better your data quality becomes.

So let's clean the text we've been given and explore it.

There seem to be quite a few URL links in the tweets. They are not telling us much (if anything) about the sentiment of the tweet so let's remove them.

```python
# remove URL's from train and test
train['clean_tweet'] = train['tweet'].apply(lambda x: re.sub(r'http\S+', '', x))

test['clean_tweet'] = test['tweet'].apply(lambda x: re.sub(r'http\S+', '', x))
```

remove_url_elmo.py hosted with by **GitHub** **view raw**

We have used Regular Expressions (or RegEx) to remove the URLs.

Note: *You can learn more about Regex in this article.*

We'll go ahead and do some routine text cleaning now.

```python
# remove punctuation marks
punctuation = '!"#$%&()*+-/:;<=>?@[\\]^_`{|}~'

train['clean_tweet'] = train['clean_tweet'].apply(lambda x: ''.join(ch for ch in x if ch not in set(punctuation)))
test['clean_tweet'] = test['clean_tweet'].apply(lambda x: ''.join(ch for ch in x if ch not in set(punctuation)))

# convert text to lowercase
train['clean_tweet'] = train['clean_tweet'].str.lower()
test['clean_tweet'] = test['clean_tweet'].str.lower()

# remove numbers
train['clean_tweet'] = train['clean_tweet'].str.replace("[0-9]", " ")
test['clean_tweet'] = test['clean_tweet'].str.replace("[0-9]", " ")

# remove whitespaces
train['clean_tweet'] = train['clean_tweet'].apply(lambda x:' '.join(x.split()))
test['clean_tweet'] = test['clean_tweet'].apply(lambda x: ' '.join(x.split()))
```

text_preprocessing_elmo.py hosted with by **GitHub** **view raw**

I'd also like to normalize the text, aka, perform text normalization. This helps in reducing a word to its base form. For example, the base form of the words 'produces', 'production', and 'producing' is **'product'**. It happens quite often that multiple forms of the same word are not really that important and we only need to know the base form of that word.

We will lemmatize (normalize) the text by leveraging the popular spaCy library.

```python
# import spaCy's language model
nlp = spacy.load('en', disable=['parser', 'ner'])

# function to lemmatize text
def lemmatization(texts):
    output = []
    for i in texts:
        s = [token.lemma_ for token in nlp(i)]
        output.append(' '.join(s))
    return output
```

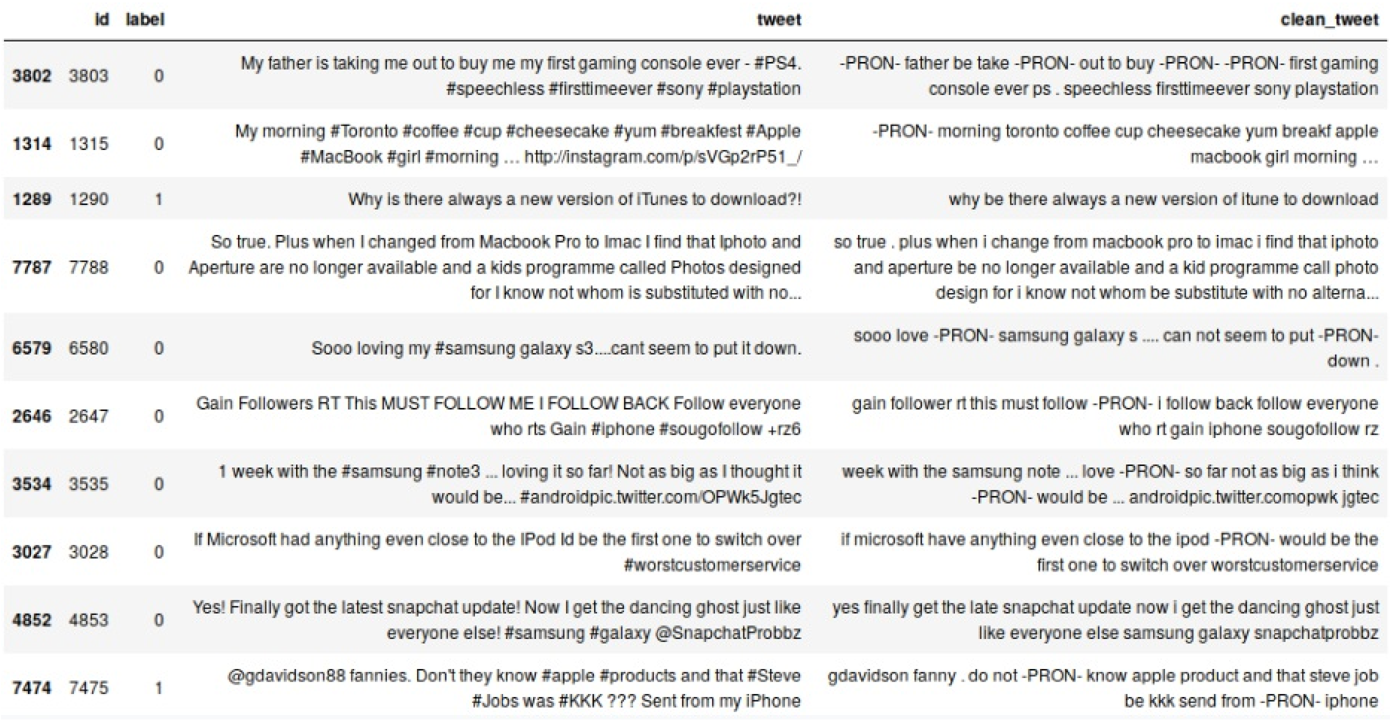
text_normalization_elmo.py hosted with by **GitHub** **view raw**

Lemmatize tweets in both the train and test sets:

```
train['clean_tweet'] = lemmatization(train['clean_tweet'])
test['clean_tweet'] = lemmatization(test['clean_tweet'])
```

Let's have a quick look at the original tweets vs our cleaned ones:

```
train.sample(10)
```



	id	label	tweet	clean_tweet
3802	3803	0	My father is taking me out to buy me my first gaming console ever - #PS4. #speechless #firsttimeever #sony #playstation	-PRON- father be take -PRON- out to buy -PRON- -PRON- first gaming console ever ps . speechless firsttimeever sony playstation
1314	1315	0	My morning #Toronto #coffee #cup #cheesecake #yum #breakfest #Apple #MacBook #girl #morning … http://instagram.com/p/sVGp2rP51_/	-PRON- morning toronto coffee cup cheesecake yum breakf apple macbook girl morning …
1289	1290	1	Why is there always a new version of iTunes to download?!	why be there always a new version of itune to download
7787	7788	0	So true. Plus when I changed from Macbook Pro to Imac I find that Iphoto and Aperture are no longer available and a kids programme called Photos designed for I know not whom is substituted with no...	so true . plus when i change from macbook pro to imac i find that iphoto and aperture be no longer available and a kid programme call photo design for i know not whom be substitute with no alterna...
6579	6580	0	Sooo loving my #samsung galaxy s3....cant seem to put it down.	sooo love -PRON- samsung galaxy s .... can not seem to put -PRON- down .
2646	2647	0	Gain Followers RT This MUST FOLLOW ME I FOLLOW BACK Follow everyone who rts Gain #iphone #sougofollow +rz6	gain follower rt this must follow -PRON- i follow back follow everyone who rt gain iphone sougofollow rz
3534	3535	0	1 week with the #samsung #note3 ... loving it so far! Not as big as I thought it would be... #androidpic.twitter.com/OPWk5Jgtec	week with the samsung note ... love -PRON- so far not as big as i think -PRON- would be ... androidpic.twitter.comopwk jgtec
3027	3028	0	If Microsoft had anything even close to the IPod Id be the first one to switch over #worstcustomerservice	if microsoft have anything even close to the ipod -PRON- would be the first one to switch over worstcustomerservice
4852	4853	0	Yes! Finally got the latest snapchat update! Now I get the dancing ghost just like everyone else! #samsung #galaxy @SnapchatProbbz	yes finally get the late snapchat update now i get the dancing ghost just like everyone else samsung galaxy snapchatprobbz
7474	7475	1	@gdavidson88 fannies. Don't they know #apple #products and that #Steve #Jobs was #KKK ??? Sent from my iPhone	gdavidson fanny . do not -PRON- know apple product and that steve job be kkk send from -PRON- iphone

Check out the above columns closely. The tweets in the 'clean_tweet' column appear to be much more legible than the original tweets.

However, I feel there is still plenty of scope for cleaning the text. I encourage you to explore the data as much as you can and find more insights or irregularities in the text.

6. Brief Intro to TensorFlow Hub

Wait, what does TensorFlow have to do with our tutorial?

TensorFlow Hub is a library that enables transfer learning by allowing the use of many machine learning models for different tasks. ELMo is one such example. That's why we will access ELMo via TensorFlow Hub in our implementation.

TensorFlow Hub

Before we do anything else though, we need to install TensorFlow Hub. You must install or upgrade your TensorFlow package to at least 1.7 to use TensorFlow Hub:

```
$ pip install "tensorflow>=1.7.0"
$ pip install tensorflow-hub
```

7. Preparing ELMo Vectors

We will now import the pretrained ELMo model. A note of caution – the model is over 350 mb in size so it might take you a while to download this.

```
import tensorflow_hub as hub
import tensorflow as tf

elmo = hub.Module("https://tfhub.dev/google/elmo/2", trainable=True)
```

I will first show you how we can get ELMo vectors for a sentence. All you have to do is pass a list of string(s) in the object **elmo**.

```python
# just a random sentence
x = ["Roasted ants are a popular snack in Columbia"]

# Extract ELMo features
embeddings = elmo(x, signature="default", as_dict=True)["elmo"]

embeddings.shape
```

elmo_example.py hosted with ♥ by **GitHub** **view raw**

Output: TensorShape([Dimension(1), Dimension(8), Dimension(1024)])

The output is a 3 dimensional tensor of shape (1, 8, 1024):

- The first dimension of this tensor represents the number of training samples. This is 1 in our case
- The second dimension represents the maximum length of the longest string in the input list of strings. Since we have only 1 string in our input list, the size of the 2nd dimension is equal to the length of the string – 8
- The third dimension is equal to the length of the ELMo vector

Hence, **every word in the input sentence has an ELMo vector of size 1024.**

Let's go ahead and extract ELMo vectors for the cleaned tweets in the train and test datasets. However, to arrive at the vector representation of an entire tweet, we will take the mean of the ELMo vectors of constituent terms or tokens of the tweet.

Let's define a function for doing this:

```python
def elmo_vectors(x):
  embeddings = elmo(x.tolist(), signature="default", as_dict=True)["elmo"]

  with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    sess.run(tf.tables_initializer())
    # return average of ELMo features
    return sess.run(tf.reduce_mean(embeddings,1))
```

elmo_vectors_func.py hosted with ♥ by **GitHub** **view raw**

You might run out of computational resources (memory) if you use the above function to extract embeddings for the tweets in one go. As a workaround, split both train and test set into batches of 100 samples each. Then, pass these batches sequentially to the function **elmo_vectors( )**.

I will keep these batches in a list:

```
list_train = [train[i:i+100] for i in range(0,train.shape[0],100)]
list_test = [test[i:i+100] for i in range(0,test.shape[0],100)]
```

Now, we will iterate through these batches and extract the ELMo vectors. Let me warn you, this will take a long time.

```
# Extract ELMo embeddings
elmo_train = [elmo_vectors(x['clean_tweet']) for x in list_train]
elmo_test = [elmo_vectors(x['clean_tweet']) for x in list_test]
```

Once we have all the vectors, we can concatenate them back to a single array:

```
elmo_train_new = np.concatenate(elmo_train, axis = 0)
elmo_test_new = np.concatenate(elmo_test, axis = 0)
```

I would advice you to save these arrays as it took us a long time to get the ELMo vectors for them. We will save them as pickle files:

```
# save elmo_train_new
pickle_out = open("elmo_train_03032019.pickle","wb")
pickle.dump(elmo_train_new, pickle_out)
pickle_out.close()

# save elmo_test_new
pickle_out = open("elmo_test_03032019.pickle","wb")
pickle.dump(elmo_test_new, pickle_out)
pickle_out.close()
```

save_pickle_elmo.py hosted with ♥ by **GitHub** **view raw**

Use the following code to load them back:

```
# load elmo_train_new
pickle_in = open("elmo_train_03032019.pickle", "rb")
elmo_train_new = pickle.load(pickle_in)

# load elmo_train_new
pickle_in = open("elmo_test_03032019.pickle", "rb")
elmo_test_new = pickle.load(pickle_in)
```

load_pickle_elmo.py hosted with ♥ by **GitHub** **view raw**

8. Model Building and Evaluation

Let's build our NLP model with ELMo!

We will use the ELMo vectors of the train dataset to build a classification model. Then, we will use the model to make predictions on the test set. But before all of that, split **elmo_train_new** into training and validation set to evaluate our model prior to the testing phase.

```
from sklearn.model_selection import train_test_split

xtrain, xvalid, ytrain, yvalid = train_test_split(elmo_train_new,
                                                  train['label'],
                                                  random_state=42,
                                                  test_size=0.2)
```

split_data_elmo.py hosted with ♥ by **GitHub** **view raw**

Since our objective is to set a baseline score, we will build a simple logistic regression model using ELMo vectors as features:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
```

```python
from sklearn.metrics import f1_score

lreg = LogisticRegression()
lreg.fit(xtrain, ytrain)
```

train_model_elmo.py hosted with ♥ by **GitHub** **view raw**

Prediction time! First, on the validation set:

```
preds_valid = lreg.predict(xvalid)
```

We will evaluate our model by the F1 score metric since this is the official evaluation metric of the contest.

```
f1_score(yvalid, preds_valid)
```

Output: 0.789976

The F1 score on the validation set is pretty impressive. Now let's proceed and make predictions on the test set:

```
# make predictions on test set
preds_test = lreg.predict(elmo_test_new)
```

Prepare the submission file which we will upload on the contest page:

```python
# prepare submission dataframe
sub = pd.DataFrame({'id':test['id'], 'label':preds_test})

# write predictions to a CSV file
sub.to_csv("sub_lreg.csv", index=False)
```

create_submission.py hosted with ♥ by **GitHub** **view raw**

These predictions give us a score of **0.875672** on the public leaderboard. That is frankly pretty impressive given that we only did fairly basic text preprocessing and used a very simple model. Imagine what the score could be with more advanced techniques. Try them out on your end and let me know the results!

What else we can do with ELMo?

We just saw first hand how effective ELMo can be for text classification. If coupled with a more sophisticated model, it would surely give an even better performance. The application of ELMo is not limited just to the task of text classification. You can use it whenever you have to vectorize text data.

Below are a few more NLP tasks where we can utilize ELMo:

- Machine Translation
- Language Modeling
- Text Summarization
- Named Entity Recognition
- Question-Answering Systems

End Notes

ELMo is undoubtedly a significant progress in NLP and is here to stay. Given the sheer pace at which research in NLP is progressing, other new state-of-the-art word embeddings have also emerged in the last few months, like Google BERT and Falando's Flair. Exciting times ahead for NLP practitioners!

I strongly encourage you to use ELMo on other datasets and experience the performance boost yourself. If you have any questions or want to share your experience with me and the community, please do so in the comments section below. You should also check out the below NLP related resources if you're starting out in this field:

- [Natural Language Processing (NLP) course]()
- [Certified Program: Natural Language Processing (NLP) for Beginners]()

You can also read this article on Analytics Vidhya's Android APP GET IT ON Google Play

Share this:

Hewlett Packard Enterprise
MÊME PAS PEUR DU CLOUD
Avec notre expertise en cloud hybride, Mercedes-AMG Petronas Motorsport peut gagner en agilité et rapidité.
En savoir plus

TAGS : ELMO, NATURAL LANGUAGE PROCESSING, NLP, PYTHON, WORD EMBEDDING

NEXT ARTICLE

5 Amazing Deep Learning Frameworks Every Data Scientist Must Know! (with Illustrated Infographic)

•••

PREVIOUS ARTICLE

29 Inspiring Women Blazing a Trail in the Data Science World

Prateek Joshi

Data Scientist at Analytics Vidhya with multidisciplinary academic background and experience in BFSI domain. Passionate about learning and implementing Data Science techniques for social good.

17 COMMENTS

SANJOY DATTA Reply

March 11, 2019 at 12:51 pm

This line in the lemmatization(texts) function is not working:

s = [token.lemma_ for token in nlp(i)]

name 'nlp is not defined'

Have run all the code upto this function.