Deep Learning

Professor: Horacio Larreguy
TA: Eduardo Zago

ITAM Applied Research 2 / Economics Research Seminar,
11/01/2023

General Perspective

Introduction to NLP

Text Cleaning

Unsupervised Learning for NLP: Topic Modeling

Sentiment Analysis in Spanish

Mathematical Representations of Language

Transformer Revolution: BERT

Fine-Tuning BERT for Classification

What is Natural Language Processing?

- Field of study focused on enabling computers to **understand**, **interpret**, and **generate** human language
- At the intersection of computer science, Artificial Intelligence and Linguistics.
- Wide variety of applications: language translation, sentiment analysis, ChatGPT, speech recognition, etc.
- We need to start talking about **language** models.

Introductory Definitions

- Few definitions:
  1. **Document:** unit for NLP. Data set with 100 articles $\Longrightarrow$ Data Set with 100 documents.
  2. **Corpus:** collection of documents
  3. **Vocabulary:** collection of unique words and tokens inside a Corpus

Text Cleaning

- Before representing texts as numbers, we must remove and replace certain characters.
- We do this to increase our chances of getting a good **mathematical representation** of our texts.
- To motivate this let us look at the following example:

```
import pandas as pd

# Define the string to work on:
string1 = 'Creo que el ITAM es la mejor universidad del país #ITAM.
            'Aunque creo que no de todos los paises de
            'latinoamerica @AMLO.'
```

Text Cleaning, nltk package

- Let us introduce the Natural Language Toolkit (nltk) package.
- Go-to package for text analysis.
- Contains easy-to use interfaces to over 50 corpora, and very simple functions for text pre-processing.
- Let's install it:

```
!pip install nltk
```

Tokenization

- Let's first tokenize this string and keep the unique words as to obtain the vocabulary of our Corpus (which contains only one document).
- **Tokenization** refers to the process of breaking down the raw text into small chunks called tokens.
- Could be words, sentences, etc.

```
# Import the necessary packages:
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
```

Tokenization

- We first tokenize each sentence and then keep the unique words:

```
# Tokenization:
tokens = word_tokenize(string1)
unique_tokens = []
[unique_tokens.append(x) for x in tokens if x not in unique_tokens]
unique_tokens

## Output:
#['Creo', 'que', 'el', 'ITAM', 'es', 'la', 'mejor', 'universidad', 'de',
#'todas', 'las', 'del', 'país', '#', '.', 'Aunque', 'creo', 'no',
# 'todos', 'los','paises', 'latinoamerica', '@']
```

- Notice *creo* is repeated two times because of the upper case.

Text Cleaning

- Word repetition (that differ because of upper case letter, accents, etc) is a serious issue in NLP.
- The same word might be understood by the computer differently. The models would be inherently bad.
- Easily solvable:

```python
# Defining the function to remove accents:
import unidecode
def remove_accents(a):
    return unidecode.unidecode(a)

# Applying it to the string:
string2 = string1.lower()
string2 = remove_accents(string2)
string2

### Output:
# 'creo que el itam es la mejor universidad de todas las del pais
# #itam. aunque creo que no de todos los paises de latinoamerica @amlo.'
```

Regular Expressions

- Moreover, we might want to remove/replace characters/words that do not convey much meaning to the text.
- **Regular expressions** is a formal language to specify text strings.
- We can easily remove URLs, HTs and @ using it:

```python
# Remove mentions and hashtags
def remove_mentions_and_tags(text):
    text = re.sub(r'@\S*','',text)
    return re.sub(r'#\S*','',text)
string2 = remove_mentions_and_tags(string2)
string2

## Output:
# 'creo que el itam es la mejor universidad de todas las del pais
# aunque creo que no de todos los paises de latinoamerica '
```

Stopwords

- Finally, we might want to remove words or characters that are commonly used in a language but do not provide much meaning to the text.
- These are called **stopwords**.
- Spanish examples: *a*, *tu* and *y*.
- We do not have to define them, just load them using `nltk`

```
 from nltk.corpus import stopwords
stop_words = set(stopwords.words('spanish'))
```

Stopwords

- We remove them using the following algorithm:

```
# First we tokenize
word_tokens = word_tokenize(string2)

filtered_sentence = []

for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)

print(filtered_sentence)

### Output:
# ['creo', 'itam', 'mejor', 'universidad', 'todas', 'pais', '#', 'itam',
#'.', 'aunque', 'creo', 'paises', 'latinoamerica', '@', 'amlo', '.']
```

Topic Modeling

- Method of classification/grouping/clustering that enables us to find **natural groups of words** that belong to a certain topic.
- Helps us reduce the dimensionality of a data set that contains a big number of features.
- For NLP, each unique word could be considered a feature.
- The model that we will use is called **Latent Dirchlet Allocation** (LDA).

LDA

- **Basic assumption:** each of the documents can be represented by the distribution of topics, which in turn can be represented by some word distribution.
- Our metric will be how interpretable these topics are.
- Very subjective. We will refer to the best fit as the model with the **most humanly interpretable results**.

LDA in Python

- We will introduce two new packages for text analysis: `SpaCy` and `gensim`.
- Both contain tons of easy to use models and Corpora for different languages.
- Let's install them:

```
!pip install pyLDAvis -qq
!pip install -qq -U gensim
!pip install spacy -qq

# Load the English corpora:
!python -m spacy download en_core_web_md -qq
```

LDA in Python

- We load them into the kernel:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import spacy
import pyLDAvis.gensim_models
pyLDAvis.enable_notebook()# Visualise inside a notebook
import en_core_web_md
from gensim.corpora.dictionary import Dictionary
from gensim.models import LdaMulticore
from gensim.models import CoherenceModel
```

Dataset

- We will use a data set of Facebook and Twitter posts labeled as fake by AfricaCheck:

```
fake = pd.read_parquet("../data/training_fake_final.parquet")
fake.head()
```

Figure: Fake Data Set

	text	label
0	Jacob Zuma and wife admitted to hospital after...	False
1	Apparently Bisi Olatilo and Bolu Akin-Olugbade...	False
2	@lyne_ian @Jeremy05749458 3 hours ago Pope Fra...	False
3	“We commend President Buhari for the swiftness...	False
4	So the hippo that was in Fourways has been sla...	False

Text Cleaning using SpaCy

- We will clean our text using the `SpaCy` package.
- We will **lemmatize** the text, meaning we will reduce words to their root form, which is called **lemma**.

```
# Our spaCy model:
nlp = en_core_web_md.load()

# Tags I want to remove from the text
removal= ['ADV','PRON','CCONJ','PUNCT','PART','DET','ADP','SPACE', 'NUM', 'SYM']
tokens = []

for summary in nlp.pipe(fake['text']):
    proj_tok = [token.lemma_.lower() for token in summary if token.pos_
    not in removal and not token.is_stop and token.is_alpha]
    tokens.append(proj_tok)

# Add tokens to new column
fake['tokens'] = tokens
fake['tokens']

#### Output:
#0 [jacob, zuma, wife, admit, hospital, contract]
```

Dictionary

- We generate our dictionary of unique tokens, to then generate our Corpus.

```
dictionary = Dictionary(fake['tokens'])
print(dictionary.token2id)

#### Output:
#{'admit': 0, 'contract': 1, 'hospital': 2, 'jacob': 3, 'wife': 4,...}
```

- And we filter words with low frequency:

```
# Filter dictionary
dictionary.filter_extremes(no_below=5, no_above=0.5, keep_n=2000)

# Create corpus
corpus = [dictionary.doc2bow(doc) for doc in fake['tokens']]
```

5-Topic Model

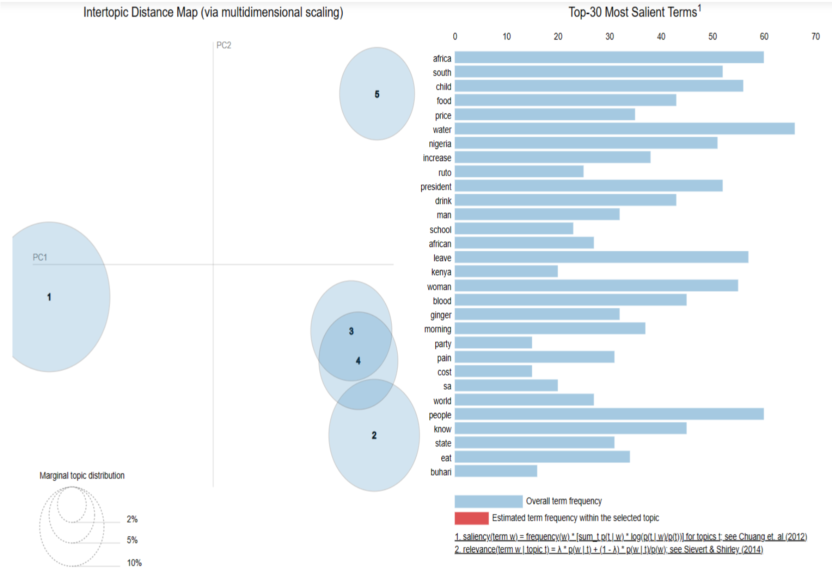
- We estimate the first model, one with 5 topics

```
lda_model5 = LdaMulticore(corpus=corpus, id2word=dictionary,
        iterations=100, num_topics=5,
        workers = 4, passes=100)
```

- And visualize inside the notebook:

```
lda_display = pyLDAvis.gensim_models.prepare(lda_model5,
        corpus, dictionary)
pyLDAvis.display(lda_display)
```

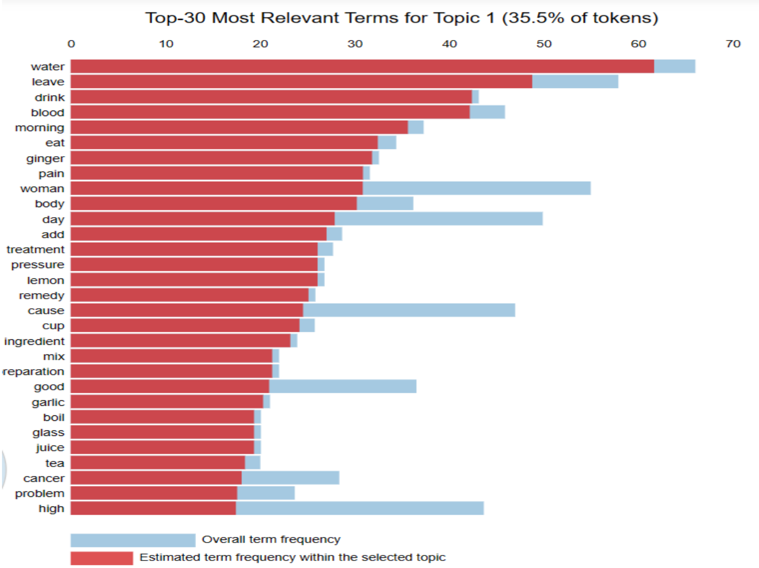
Visualizing Results



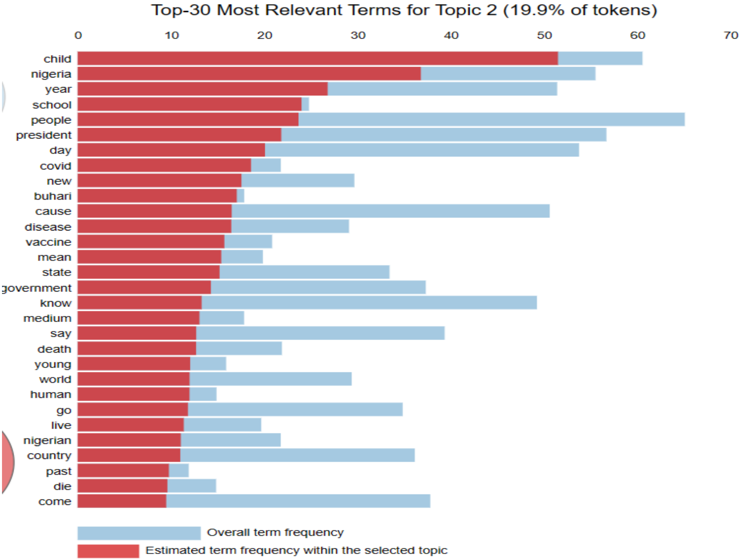
Metrics, Intertopic Distance

- We first get the **Intertopic Distance** between groups.
- This provides us information on how different each group is by looking at the number of similar words in them.
- We do not want Topics that overlap a lot, like 3 and 4.
- It is also useful to visualize the most relevant terms for each group.
- This can help us assign a human interpretable topic to them:

Visualizing Specific Topics



Visualizing Specific Topics



Human Interpretable Topics

- **Topic 1** contains tons of medical words, but also spices, food and other supplements used in Traditional Medicine.
- This topic probably refers to Health Misinformation, using Traditional Medicine.
- Tons of posts of people curing cancer using herbs, reducing blood pressure using tea, etc.
- **Topic 2** contains words such as vaccine, COVID, government and disease.
- Probably refers to Health Misinformation, using Modern Medicine.

Coherence Metric

- Although most of the result analysis is subjective, there is a metric used to evaluate these types of models called **Coherence**.
- The **Coherence** scores a single topic by measuring the degree of semantic similarity between high scoring words in the topic.
- Helps distinguish between topics that are **semantically interpretable** and topics that are **artifacts of statistical inference**.

Coherence Score

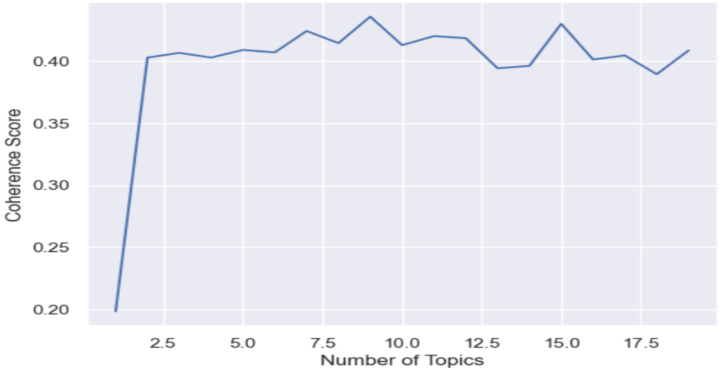
- We estimate 20 different Topic Models, by changing the number of topics:

```
# Coherence score using C_v:
topics = []
score = []
for i in range(1,20,1):
    lda_model = LdaMulticore(corpus=corpus, id2word=dictionary,
            iterations=10, num_topics=i, workers = 4,
            passes=10, random_state=100)
    cm = CoherenceModel(model=lda_model, texts = fake['tokens'],
            corpus=corpus, dictionary=dictionary, coherence='c_v')
    topics.append(i)
    score.append(cm.get_coherence())
```

Coherence Score

- We can graph the results:

```
_=plt.plot(topics, score)
_=plt.xlabel('Number of Topics')
_=plt.ylabel('Coherence Score')
plt.show()
```



Coherence Score

- If we decided which model to use by this metric alone, we would have to choose the 9-Topic Model.
- However, are the topics humanly interpretable?
- It is important to note that **optimizing for coherence may not yield human interpretable topics**.
- We actually get the same human interpretable topics as the 5-Topic Model, but we add 4 more overlapping topics[1]
- Thus not a good fit!

[1] Check the notebook.

Sentiment Analysis in Spanish

- While working with Neural Networks, we more often than not, will not have enough data points to train complicated models.
- However, we can use trained models (trained for our specific task) and apply them to our own small data set.
- In this chapter we will see how to use a `keras`' model for Sentiment Analysis in Spanish, to predict the sentiment of posts.

Sentiment Analysis in Python

- `sentiment-spanish` is a Python library that uses convolutional neural networks to predict the sentiment of spanish sentences.
- The model was trained using over 800000 reviews of users of the pages eltenedor, decathlon, tripadvisor, filmaffinity and ebay.
- Returns a number between 0 and 1, the closer to 0 the more negative the text is, and viceversa.[2]
- Let's install it:

```
!pip install sentiment-analysis-spanish
```

[2] For more information visit the package documentation.

Sentiment Analysis in Python

- Load the necessary packages:

```
import spacy
import nltk

from nltk.corpus import stopwords
from unicodedata import normalize

import re
import scipy as sc

from nltk.tokenize import word_tokenize, sent_tokenize
from keras.preprocessing.text import text_to_word_sequence
from sentiment_analysis_spanish import sentiment_analysis
from spacy.lang.es import Spanish

# Load the models:
# For lemmatization in Spanish
nlp = spacy.load('es_core_news_lg')}
# For sentiment analysis
from sentiment_analysis_spanish import sentiment_analysis
```

Data Set, Feminist-related Hate Speech

- We will use 20 Tweets containing the word *feminazi* that we scrapped to analyze the prevalence of hate speech in Social Media during the 8M Marches[3]

```
# Load the data:
df = pd.read_parquet('../data/hate_speech.parquet')

# Clean it:
df = clean_data(df, 'text')
df = df[~((df['text_clean'].isna()) |
        (df['text_clean']==''))].reset_index(drop=True)
df[['author_id', 'text','text_clean']].head()
```

[3] Text cleaning functions are defined in the notebook.

Data Set, Feminist-related Hate Speech

- Which looks like this, once cleaned:

	author_id	text	text_clean
0	203579995	RT @Lady_Chiyome: Femenina, nunca #FEMINAZI 🤮\...	rt femenina nunca
1	1358301364384890880	@PabloEchenique @IreneMontero Prefiero escucha...	prefiero escuchar personas con mas neuronas qu...
2	232758195	RT @danielalozanocu: Todo el año: feminazi, lo...	rt todo el an feminazi loca abortera deberia m...
3	1325368543614013440	Feminismo≠Feminazi\nFeminismo es igualdad\nUn ...	feminismo feminazi feminismo es igualdad un ho...
4	551967420	RT @jmrivas6911: RADFEM\n\nHay cavada una trin...	rt radfem hay cavada una trinchera entre el od...

Functions

- Now we can define the functions to do the prediction in our Data Frame.

```
# Function to calculate the sentiment of a text
def sentiment_metrics(text, sentiments, sentiment_score=True):
    try:
        if sentiment_score:
            sentimiento = sentiments.sentiment(text)
        else:
            sentimiento = np.nan
    except:
        sentimiento = np.nan, np.nan
    return sentimiento

# Function to work with Data Frames:
def compute_sentiment(df, text):
    # Instantiate the model:
    sentiments = sentiment_analysis.SentimentAnalysisSpanish()
    df["sentiment_score"] = df.apply(
        lambda x: sentiment_metrics(x[text], sentiments),
        axis=1,
        )
    df_sentiment = pd.DataFrame(df["sentiment_score"].values.tolist(),
    index=df.index)
    df_sentiment.rename(columns={0: "sentiment_score"}, inplace=True)
    df_sentiment = pd.concat(
        [df, df_sentiment["sentiment_score"]], axis=1)
    return(df_sentiment)
```

Sentiment Prediction

- We do the prediction for our 10 Tweets:

```
df = compute_sentiment(df, 'text_clean')
df[['author_id', 'text','text_clean', 'sentiment_score']].head()
```

Figure: Sentiment Prediction

	author_id	text	text_clean	sentiment_score
0	203579995	RT @Lady_Chiyome: Femenina, nunca #FEMINAZI 🤮\...	rt femenina nunca	0.152140
1	1358301364384890880	@PabloEchenique @IreneMontero Prefiero escucha...	prefiero escuchar personas con mas neuronas qu...	0.006189
2	232758195	RT @danielalozanocu: Todo el año: feminazi, lo...	rt todo el an feminazi loca abortera deberia m...	0.069884
3	1325368543614013440	Feminismo≠Feminazi\nFeminismo es igualdad\nUn ...	feminismo feminazi feminismo es igualdad un ho...	0.000006
4	551967420	RT @jmrivas6911: RADFEM\n\nHay cavada una trin...	rt radfem hay cavada una trinchera entre el od...	0.004116

Mathematical Representations of Language

- How are we going to feed text into a Neural Net that does Mathematical Computations?
- Language is a complex and dynamic aspect of human communication
- Involves the use of symbols, sounds, and grammar rules to convey meaning.
- Incredibly hard task, but some advancement has been made:
  1. **One-Hot Encoding** vectors.
  2. Probabilistic models such as **n-grams**.
  3. Neural language models such as **Word Embeddings**.

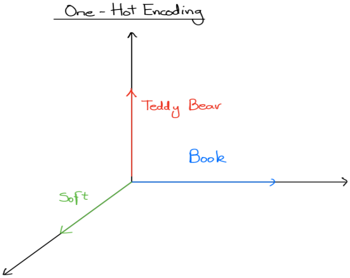
One-Hot Encoded Vectors

- **One-Hot Encoding** is representing each word as dummies.
- 1 for the specific index that word is assigned to, 0 in other case.
- The length of the vectors is equal to the size of the Vocabulary.
- Examples:

$$x^{Bear} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, x^{Book} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, x^{Soft} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, x^{Teddy} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

One-Hot Encoded Vectors, Graphically

- We can look at the examples graphically:



Also notice that:

$$(x^{Soft})^T x^{Bear} = (x^{Bear})^T x^{Book} = 0 \tag{1}$$

Main Disadvantages

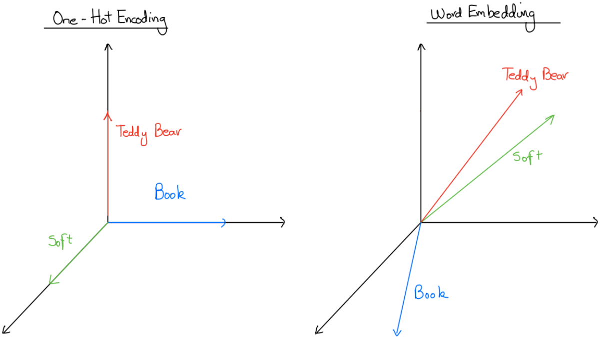
- Languages have millions of words.
- We would need vectors of millions of entries, and matrices of millions of vectors, to correctly define a whole Corpus.
- These vectors are **orthogonal** to each other.
- Thus, not capable of **extracting relationships** in the meaning of words.
- Incapable of contextualizing words or sentences.

Word Embeddings

- **Word embeddings** are numeric representations (vectors) of words, in which words with similar meaning result in similar representations.
- They take into account words similarity.
- For the Teddy, Bear and Book example we would have something like this:

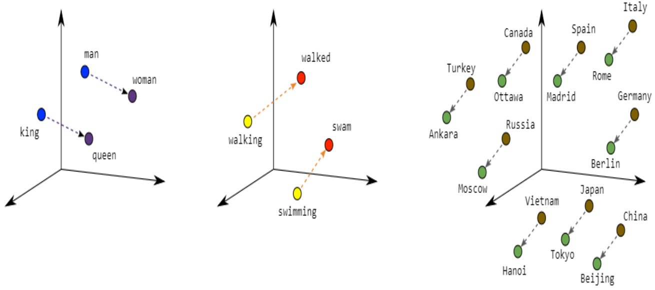
$$x^{Bear} = \begin{bmatrix} 2.3 \\ 3.4 \\ 0.4 \\ 5.2 \end{bmatrix}, x^{Book} = \begin{bmatrix} -2 \\ -0.05 \\ -.9 \\ -3 \end{bmatrix}, x^{Soft} = \begin{bmatrix} .5 \\ 3 \\ 4.12 \\ 1.2 \end{bmatrix}, x^{Teddy} = \begin{bmatrix} 2 \\ 3.3 \\ .65 \\ 0.03 \end{bmatrix}$$

Word Embeddings, Graphically



- Words that often go together, like Teddy Bear and Soft, are actually related geometrically.

Word Embeddings, Graphically



Male-Female Verb Tense Country-Capital

- Amazingly great at capturing semantic relations.
- Linear Combinations of them yield related words.

BERT

- The BERT model was pre-trained in a **self-supervised fashion.**
- Initially built (and trained) for **Masked Language Modelling** and **Next sentence prediction** tasks.
- Pre-trained on BookCorpus, a dataset consisting of 11,038 unpublished books and English Wikipedia.
- BERT has already been pre-trained, which means this model has already learnt the **inner structure of the English language**

BERT

- How can we use it? Fine-Tune it and re-train it on your own, smaller data-set.
- What makes BERT so powerful and special?
- Two main things:

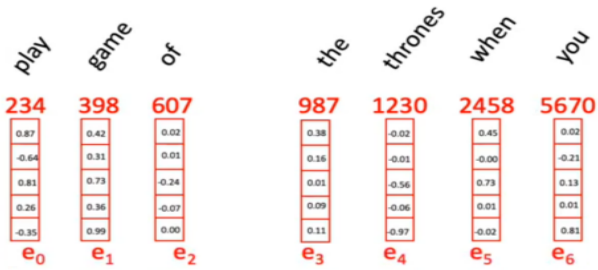
1. The way it uses and estimates two different word embeddings.
2. The Attention Mechanism.

BERT Token Embeddings

- Standard word embeddings that represent each token (word or subword) in the input text.
- This is the first layer of the NN, and it is named the **Embeddings Layer.**
- The Embedding layer maps integers into One-Hot-Encoded Vectors with length equal to the whole Corpus.
- Then, this vectors are mapped into a dense vector representation (word embeddings).
- Note this vectors are **trainable weights**.

BERT Token Embeddings

- Already trained in a gigantic data set.
- Can be used as input in other models to represent numerically an entirely different data set.
- The author sets the length of this vectors to 512.

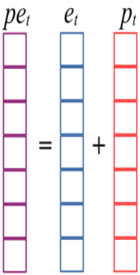


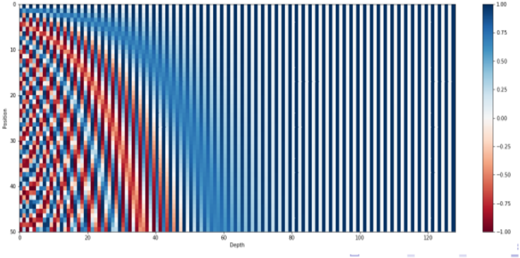
BERT Positional Embeddings

- Used to compensate for the lack of recurrence operations.
- Transformers process all embeddings at once: parallel.
- Position information is used through wave frequencies, using a cosine functions.
- The farther apart from the beginning, the smaller the amplitude of the wave.
- These then are ranked by amplitude of waves to provide positional information to the Word Embeddings.
- These are also **trainable weights.**

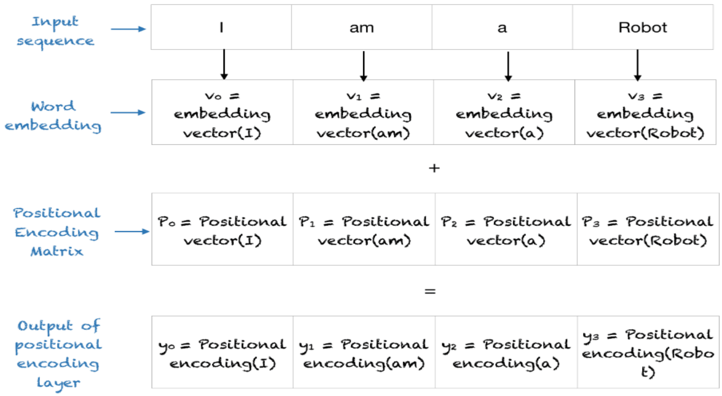
BERT Positional Embeddings

- Position embeddings are added to the token embeddings to provide the model with this positional information.





BERT Final Embeddings



The positional encoding layer in the transformer

Attention Mechanism

- BERT uses a self-attention mechanism to understand the context of each word in a sentence.
- Self-attention allows BERT to **weigh the importance of each word in a sentence** based on the context of the entire sentence.
- In the simplest of terms, Attention involves generating a score for each word.
- The score indicates how relevant the word is to the current word being processed.
- Helps solve the problem of losing context.
- Allows the model to look at all the past hidden states (words).

Attention Mechanism

- By using self-attention, BERT can capture complex relationships between words in a sentence and generate high-quality representations of them.

	<start>	I	am	no	man	<end>
<start>	1	0	0	0	0	0
I	0.01	0.99	0	0	0	0
am	0.001	0.004	0.995	0	0	0
no	0.003	0.004	0.003	0.99	0	0
man	0.003	0.003	0.04	0.02	0.93	0
<end>	0.001	0.001	0.001	0.001	0.001	0.995

Fine-Tuning BERT using Tensorflow

- In this section we will take a look at how to Fine-Tune BERT to solve a Classification problem: Is a post verifiable or not?
- Let's first install all necessary packages

```
# A dependency of the preprocessing for BERT inputs
!pip install -q tensorflow-text
# For the AdamW optimizer from
!pip install -q tf-models-official tensorflow/models
!pip install bert-for-tf2
!pip install sentencepiece
```

Loading Packages

- And loading them to the kernel:

```
import os
import shutil
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text as text
import re
from official.nlp import optimization  # to create AdamW optmizer
tf.get_logger().setLevel('ERROR')

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Embedding, Dense, Dropout

# Data cleaning
from drive.MyDrive.bert_train.src.utils.clean import *
import nltk
nltk.download('stopwords')
```

Data Set

- We import the data set.

```
# Importing the data:
df = pd.read_parquet('drive/MyDrive/bert_train/6-labeled/2-training_2_classes.parquet')
df
```

	text	URL	label
0	Controversial cross dresser Idris Okuneye aka ...	https://www.facebook.com/161915460542267/posts...	Verifiable
1	Tinubu's presidency will produce wealth, prosp...	https://www.facebook.com/161915460542267/posts...	Not Verifiable
2	PDP was corrupt in 2015 and Nigerians needed t...	https://www.facebook.com/100044230039479/posts...	Not Verifiable
3	Underwater energy is WoW	https://twitter.com/MrOdanz/status/15447596547...	Not Verifiable
4	Mbappe's ego is getting way too annoying. Man ...	https://twitter.com/MrOdanz/status/15591191764...	Not Verifiable
...	...	...	...
1072	With 523 points, Karim Benzema wins UEFA POTY ...	https://twitter.com/MrOdanz/status/15628684937...	Verifiable
1073	My mum never changed her surname after marriag...	https://twitter.com/MrOdanz/status/15619785734...	Not Verifiable
1074	Heroic welcome for Raila as he votes in Kibera	https://www.facebook.com/178342827608/posts/59...	Verifiable
1075	Take this Homemade drink for 5days and say Goo...	https://www.facebook.com/peterfatomilolaoffici...	Verifiable
1076	#KenyaDecides Results Update: Raila: 129,751 -...	https://www.facebook.com/100044230039479/posts...	Verifiable

1077 rows × 3 columns

Functions

- We define the functions to clean the text.
- Notice we won't remove stopwords or lemmatize text, since BERT can use them to understand better the context.

```
# Defining the functions to clean the text
TAG_RE = re.compile(r'<[^>]+>')
def preprocess_text(sen):
    sentence = TAG_RE.sub('', sen) # html tags
    # punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sentence)
    # single character
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)
    sentence = re.sub(r'\s+', ' ', sentence) # multiple spaces
return sentence.lower()
```

Functions

- We clean the text and generate our X (text, no chosen features) and our y (labels, 1 if *verifiable* 0 if *not verifiable*).

```
# Cleaning the text and generating the data sets:
sentences = list(df['text'])
text = np.array([str.encode(preprocess_text(sen)) for sen
        in sentences], dtype=object)

y = df['label']
y = np.array(list(map(lambda x: 1 if x=="Verifiable"
        else 0, y)), dtype='int32')
text[:3]

### Output:
# 1 'controversial cross dresser idris okuneye aka bobrisky has
# unveiled his plans for his st birthday ',
# 2 'tinubu presidency will produce wealth prosperity'
# 3 'pdp was corrupt in and nigerians needed to remove them from power '
```

Train Test Split

- We divide our data set into training and validation:

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(text,
            y, test_size=0.20)
```

- Until now we have already seen how to do all of this.
- Now, we will define the model, download and instantiate it with the already trained weights, define the hyper-parameters and train it.

Pre-Processing Layer

- We begin defining the pre-processing layer:

```python
# URLs to download the weights for both layers (encoder and pre-process):
tfhub_handle_encoder = "https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/2"
tfhub_handle_preprocess = "https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3"

# Defining the input layer:
text_inputs = [tf.keras.layers.Input(shape=(), dtype=tf.string)]

# Download the Pre-Processor weights:
preprocessor = hub.load(tfhub_handle_preprocess)
tokenize = hub.KerasLayer(preprocessor.tokenize)
tokenized_inputs = [tokenize(segment) for segment in text_inputs]

# Compile the whole input layer:
bert_pack_inputs = hub.KerasLayer(preprocessor.bert_pack_inputs,
        arguments=dict(seq_length=256))
```

Attention Layer

- Then the Encoder Layer, or Attention Layers:

```
 # Bridge between tokens and embeddings

encoder_inputs = bert_pack_inputs(tokenized_inputs)

# Download and define the encoder layers
encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder') # BERT embedding 

embedded = encoder(encoder_inputs)
```

Fine-Tuning, Final Layer

- Finally, we fine-tune the model by adding a regularizer and a final dense layer, which would do the classification.
- We will use a linear activation function and DropOut for regularization:

```
 # Connection between the trained model and the additional layers:
net = embedded['pooled_output']

# Adding DropOut
net = tf.keras.layers.Dropout(0.1)(net)

# Adding final Dense Layer (Classifier)
net = tf.keras.layers.Dense(1, activation=None, name='classifier')(net)

# Final Compilation:
model_BERT = tf.keras.Model(text_inputs, net)
```

- We define the final set of hyper-parameters: loss function, optimization algorithm, learning rate and epochs:

```
epochs = 8

init_lr = 3e-5
optimizer = optimization.create_optimizer(init_lr=init_lr,
        optimizer_type='adamw')

model_BERT.compile(optimizer=optimizer,
        loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
        metrics=tf.metrics.BinaryAccuracy())
```

- We also define a callback function to go back to the model with the best accuracy (Early Stopping):

```
checkpoint_path = 'drive/MyDrive/bert_train/models/tensor/cp7.cpkt'
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
    save_weights_only=True,
    save_best_only=True,
    verbose=1)
```

Training the Model

- And we can finally train the model:

```
history = model_BERT.fit(x=x_train, y=y_train, epochs=8,
       validation_data = (x_test, y_test),
       callbacks=[cp_callback])
```

Performance Evaluation, Learning Curve

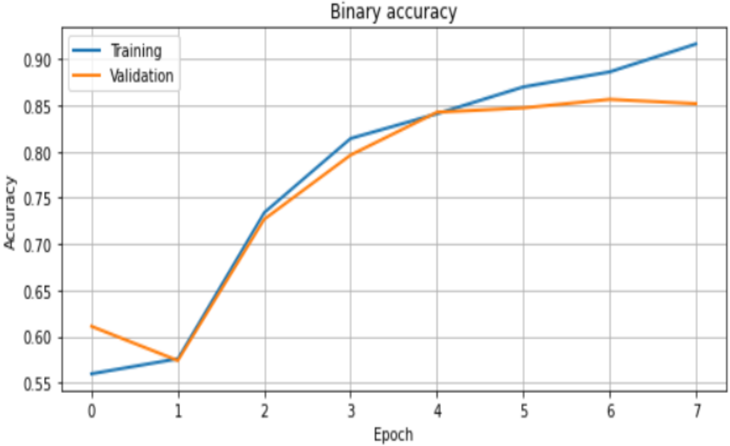
- We evaluate the best model using the usual metrics. First the learning curve:

```
 # Plot loss
plt.figure(figsize=(20, 4))

plt.title('Binary accuracy')
plt.plot(history.history['binary_accuracy'],
        label='Training', linewidth=2)
plt.plot(history.history['val_binary_accuracy'],
        label='Validation', linewidth=2)
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid(True)

plt.show()
```

Performance Evaluation, Learning Curve



Performance Evaluation, Accuracy

- We recover the best model weights to check the accuracy:
- We obtained an accuracy of 84% in validation, which is pretty good for this type of task:

```
latest = tf.train.latest_checkpoint(checkpoint_dir)
model_BERT.load_weights(latest)

# Re-evaluate the model
loss, acc = model_BERT.evaluate(x_test, y_test, verbose=2)
print("Restored model, accuracy: {:5.2f}%".format(100 * acc))

### Output:
# 84.26%
```

Performance Evaluation, Accuracy

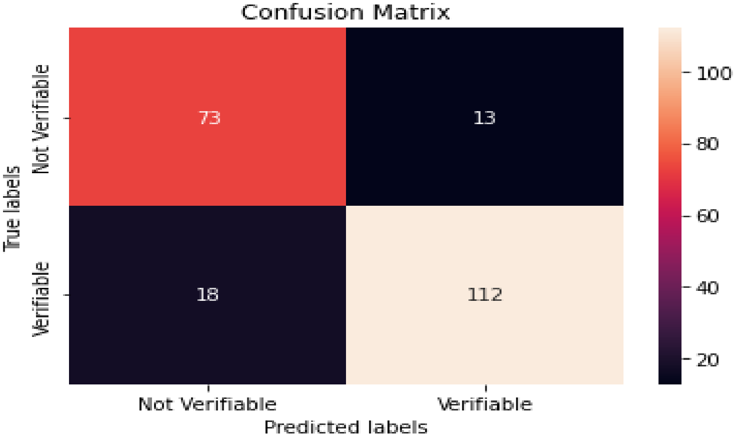
- Finally we can check which class is the most problematic, by plotting the confusion matrix:

```
cm = tf.math.confusion_matrix(y_test, y_pred_nn)

ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax);

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Not Verifiable', 'Verifiable']);
ax.yaxis.set_ticklabels(['Not Verifiable', 'Verifiable'])
```

Performance Evaluation, Confusion Matrix



QUESTIONS: