ZEAL EDUCATION SOCIETY's

ZEAL COLLEGE OF ENGINEERING AND RESEARCH, NARHE, PUNE

DEPARTMENT OF COMPUTER ENGINEERING

SEMESTER-II

[A.Y. : 2022 - 2023]

Laboratory Practice-VI (410256)

Natural Language Processing 410252(A)

LABORATORY MANUAL

Department Vision and Mission

INSTITUTE VISION	To impart value added technological education through pursuit of academic excellence, research and entrepreneurial attitude.
INSTITUTE MISSION	**M1:** To achieve academic excellence through innovative teaching and learning process. **M2:** To imbibe the research culture for addressing industry and societal needs**.** **M3:** To provide conducive environment for building the entrepreneurial skills. **M4:** To produce competent and socially responsible professionals with core human values.

DEPARTMENT VISION	To emerge as a department of repute in Computer Engineering which produces competent professionals and entrepreneurs to lead technical and betterment of mankind.
DEPARTMENT MISSION	**M1:** To strengthen the theoretical and practical aspects of the learning process by teaching applications and hands on practices using modern tools and FOSS technologies. **M2:** To endavor innovative interdisciplinary research and entrepreneurship skills to serve the needs of Industry and Society. **M3:** To enhance industry academia dialog enabling students to inculcate professional skills. **M4:** To incorporate social and ethical awareness among the students to make them conscientious professionals.

Department
Program Educational Objectives (PEOs)

PEO1: To Impart fundamentals in science, mathematics and engineering to cater the needs of society andIndustries.
PEO2: Encourage graduates to involve in research, higher studies, and/or to become entrepreneurs.
PEO3: To Work effectively as individuals and as team members in a multidisciplinary environment withhigh ethical values for the benefit of society.

Savitribai Phule Pune University Fourth Year of Computer Engineering (2019 course) 410256: Laboratory Practice-VI		
Teaching Scheme: PR: 02 Hours/Week	**Credit** 01	**Examination Scheme:** TW: 50 Marks

Course Objectives:

- To understand the fundamental concepts and techniques of natural language processing (NLP)

Course Outcomes:

On completion of the course, student will be able to-

CO1:	Apply basic principles of elective subjects to problem solving and modelling.
CO2:	Use tools and techniques in the area of software development to build mini projects.
CO3:	Design and develop applications on subjects of their choice.
CO4:	Generate and manage deployment, administration & security.

List of Assignments

Sr. No.	Title
	Group 1
1	Perform tokenization (Whitespace, Punctuation-based, Treebank, Tweet, MWE) using NLTK library. Use porter stemmer and snowball stemmer for stemming. Use any technique for lemmatization. **Input / Dataset –use any sample sentence**
2	Perform bag-of-words approach (count occurrence, normalized count occurrence), TF-IDF on data. Create embeddings using Word2Vec. **Dataset to be used:** https://www.kaggle.com/datasets/CooperUnion/cardataset
3	Perform text cleaning, perform lemmatization (any method), remove stop words (any method), label encoding. Create representations using TF-IDF. Save outputs. **Dataset:https://github.com/PICT-NLP/BE-NLP-Elective/blob/main/3-Preprocessing/News_dataset.pickle**
4	Create a transformer from scratch using the Pytorch library
5	Morphology is the study of the way words are built up from smaller meaning bearing units. Study and understand the concepts of morphology by the use of add delete table

Group 2	
6	**Mini Project** (Fine tune transformers on your preferred task) Finetune a pretrained transformer for any of the following tasks on any relevant dataset of your choice: • Neural Machine Translation • Classification • Summarization
7	**Mini Project** - POS Taggers For Indian Languages
8	**Mini Project** -Feature Extraction using seven moment variants
9	**Mini Project** -Feature Extraction using Zernike Moments

Group 1

Assignment No 1

Problem Statement:

Perform tokenization (Whitespace, Punctuation-based, Treebank, Tweet, MWE) using NLTK library. Use porter stemmer and snowball stemmer for stemming. Use any technique for lemmatization.

Input / Dataset –use any sample sentence

Objective:

To understand the fundamental concepts and techniques of natural language processing (NLP).

CO Relevance: CO1

Contents for Theory:

Introduction to NLP (Natural Language Processing)

Computers speak their own language, the binary language. Thus, they are limited in how they can interact with us humans; expanding their language and understanding our own is crucial to set them free from their boundaries.

NLP is an abbreviation for natural language processing, which encompasses a set of tools, routines, and techniques computers can use to process and understand human communications. Not to be confused with speech recognition, NLP deals with understanding the meaning of words other than interpreting audio signals into those words.

If you think NLP is just a futuristic idea, you may be shocked to know that we are likely to interact with NLP every day when we perform queries in Google when we use translators online when we talk with Google Assistant or Siri. NLP is everywhere, and to implement it in your projects is now very reachable thanks to libraries such as NLTK, which provide a huge abstraction of the complexity.

```
In [1]: pip install nltk
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: nltk in /home/dipali/.local/lib/python3.8/site-packages (3.8.1)
Requirement already satisfied: joblib in /home/dipali/.local/lib/python3.8/site-packages (from nltk) (1.2.0)
Requirement already satisfied: regex>=2021.8.3 in /home/dipali/.local/lib/python3.8/site-packages (from nltk) (2022.
10.31)
Requirement already satisfied: click in /home/dipali/.local/lib/python3.8/site-packages (from nltk) (8.1.3)
Requirement already satisfied: tqdm in /home/dipali/.local/lib/python3.8/site-packages (from nltk) (4.64.1)

[notice] A new release of pip is available: 23.0 -> 23.0.1
[notice] To update, run: python3 -m pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

```
In [9]: nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /home/dipali/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Out[9]: True
```

1. Tokenization

Tokenization is the process of breaking text into smaller pieces called tokens. These smaller pieces can be sentences, words, or sub-words.

For example, the sentence “I won” can be tokenized into two word-tokens “I” and “won”.

Sentence Tokenization

In [4]:
```
import nltk
from nltk.tokenize import sent_tokenize
```

In [5]:
```
text=" India is a unique country with diversity. Unity is diversity is the main slogan of the country."
```

In [6]:
```
print(sent_tokenize(text))
```

```
[' India is a unique country with diversity.', 'Unity is diversity is the main slogan of the country.']
```

Word Tokenization

In [7]:
```
import nltk
from nltk.tokenize import word_tokenize
```

In [8]:
```
print(word_tokenize(text))
```

```
['India', 'is', 'a', 'unique', 'country', 'with', 'diversity', '.', 'Unity', 'is', 'diversity', 'is', 'the', 'main',
'slogan', 'of', 'the', 'country', '.']
```

A. Whitespace tokenization

A WhitespaceTokenizer is a tokenizer that splits on and discards only whitespace characters.

In [9]:
```
print(f'Whitespace tokenization = {text.split()}')
```

```
Whitespace tokenization = ['India', 'is', 'a', 'unique', 'country', 'with', 'diversity.', 'Unity', 'is', 'diversity
', 'is', 'the', 'main', 'slogan', 'of', 'the', 'country.']
```

B. Punctuation-based tokenization

Punctuation-based tokenization is slightly more advanced than whitespace-based tokenization since it splits on whitespace and punctuations and also retains the punctuations.

In [10]:
```
from nltk.tokenize import wordpunct_tokenize
```

In [11]:
```
print(f'Punctuation-based tokenization = {wordpunct_tokenize(text)}')
```

```
Punctuation-based tokenization = ['India', 'is', 'a', 'unique', 'country', 'with', 'diversity', '.', 'Unity', 'is',
'diversity', 'is', 'the', 'main', 'slogan', 'of', 'the', 'country', '.']
```

C. Default/TreebankWordTokenizer

The default tokenization method in NLTK involves tokenization using regular expressions as defined in the Penn Treebank (based on English text). It assumes that the text is already split into sentences.

In [12]:
```
from nltk.tokenize import TreebankWordTokenizer
```

In [13]:
```
sentence="What's your name?"
```

In [14]:
```
tokenizer = TreebankWordTokenizer()
print(f'Default/Treebank tokenization = {tokenizer.tokenize(sentence)}')
```

```
Default/Treebank tokenization = ['What', "'s", 'your', 'name', '?']
```

D.TweetTokenizer

When we want to apply tokenization in text data like tweets, the tokenizers mentioned above can't produce practical tokens. Through this issue, NLTK has a rule based tokenizer special for tweets. We can split emojis into different words if we need them for tasks like sentiment analysis.

a. Install emoji library

In [15]:
```
pip install emoji --upgrade
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: emoji in /home/dipali/.local/lib/python3.8/site-packages (2.2.0)

[notice] A new release of pip is available: 23.0 -> 23.0.1
[notice] To update, run: python3 -m pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

In [16]:
```
import emoji
```

In [17]:
```
print(emoji.emojize('Hi Everyone! :grinning_face:'))
```

```
Hi Everyone! 😀
```

In [18]:
```
sentence1= emoji.emojize('Hi Everyone! :grinning_face:')
```

In [19]:
```
from nltk.tokenize import TweetTokenizer
```

In [20]:
```
tokenizer = TweetTokenizer()
print(f'Tweet-rules based tokenization = {tokenizer.tokenize(sentence1)}')
```

```
Tweet-rules based tokenization = ['Hi', 'Everyone', '!', '😀']
```

E. MWETokenizer

NLTK's multi-word expression tokenizer (MWETokenizer) provides a function add_mwe() that allows the user to enter multiple word expressions before using the tokenizer on the text. More simply, it can merge multi-word expressions into single tokens.

In [21]:
```
sentence2="Hope, is the only thing stronger than fear! Hunger Games"
```

In [22]:
```
print(word_tokenize(sentence2))
```

```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', 'Hunger', 'Games']
```

In [23]:
```
from nltk.tokenize import MWETokenizer
```

In [24]:
```
tokenizer = MWETokenizer()
tokenizer.add_mwe(('Hunger', 'Games'))
print(f'Multi-word expression (MWE) tokenization = {tokenizer.tokenize(word_tokenize(sentence2))}')
```

```
Multi-word expression (MWE) tokenization = ['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear',
'!', 'Hunger_Games']
```

2. Stemming and Lemmatization

A. Stemming

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. Often when searching text for a certain keyword, it helps if the search returns variations of the word. For instance, searching for "boat" might also return "boats" and "boating". Here, "boat" would be the stem for [boat, boater, boating, boats]. Stemming is a somewhat crude method for cataloging related words; it essentially chops off letters from the end until the stem is reached.

i) Porter Stemmer

In [25]:
```
# Import the toolkit and the full Porter Stemmer library
import nltk

from nltk.stem.porter import *
p_stemmer = PorterStemmer()
words = ['run','runner','running','ran','runs','easily','fairly']
for word in words:
    print(word+' --> '+p_stemmer.stem(word))
```

```
run --> run
runner --> runner
running --> run
ran --> ran
runs --> run
easily --> easili
fairly --> fairli
```

ii) Snowball Stemmer

In [26]:
```
from nltk.stem.snowball import SnowballStemmer

# The Snowball Stemmer requires that you pass a language parameter
s_stemmer = SnowballStemmer(language='english')
words = ['run','runner','running','ran','runs','easily','fairly']
for word in words:
    print(word+' --> '+s_stemmer.stem(word))
```

```
run --> run
runner --> runner
running --> run
ran --> ran
runs --> run
easily --> easili
fairly --> fair
```

B. Lemmatization

In contrast to stemming, lemmatization looks beyond word reduction and considers a language's full vocabulary to apply a morphological analysis to words. The lemma of 'was' is 'be' and the lemma of 'mice' is 'mouse'.

Lemmatization is typically seen as much more informative than simple stemming, which is why Spacy has opted to only have Lemmatization available instead of Stemming

Install Spacy

In [27]:
```
!pip3 install spacy
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: spacy in /home/dipali/.local/lib/python3.8/site-packages (3.5.0)
Requirement already satisfied: thinc<8.2.0,>=8.1.0 in /home/dipali/.local/lib/python3.8/site-packages (from spacy)
(8.1.7)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /home/dipali/.local/lib/python3.8/site-packages (from spac
y) (2.0.8)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /home/dipali/.local/lib/python3.8/site-packages (from spacy)
(4.64.1)
Requirement already satisfied: typer<0.8.0,>=0.3.0 in /home/dipali/.local/lib/python3.8/site-packages (from spacy)
(0.7.0)
Requirement already satisfied: setuptools in /home/dipali/.local/lib/python3.8/site-packages (from spacy) (67.0.0)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /home/dipali/.local/lib/python3.8/site-packages (from spacy)
```

Download the model for English Langauge

In [28]:
```
!python3 -m spacy download en_core_web_sm
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting en-core-web-sm==3.5.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.5.0/en_core_web_sm-3.5.0-
py3-none-any.whl (12.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 1.5 MB/s eta 0:00:00m eta 0:00:01[36m0:00:01m
Requirement already satisfied: spacy<3.6.0,>=3.5.0 in /home/dipali/.local/lib/python3.8/site-packages (from en-core-
web-sm==3.5.0) (3.5.0)
Requirement already satisfied: setuptools in /home/dipali/.local/lib/python3.8/site-packages (from spacy<3.6.0,>=3.
5.0->en-core-web-sm==3.5.0) (67.0.0)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /home/dipali/.local/lib/python3.8/site-packages (from spac
y<3.6.0,>=3.5.0->en-core-web-sm==3.5.0) (2.0.8)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /home/dipali/.local/lib/python3.8/site-packages (from spacy<3.
6.0,>=3.5.0->en-core-web-sm==3.5.0) (2.0.7)
Requirement already satisfied: numpy>=1.15.0 in /home/dipali/.local/lib/python3.8/site-packages (from spacy<3.6.0,>=
3.5.0->en-core-web-sm==3.5.0) (1.24.1)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.11.0,>=1.7.4 in /home/dipali/.local/lib/python3.8/site-packa
ges (from spacy<3.6.0,>=3.5.0->en-core-web-sm==3.5.0) (1.10.4)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /home/dipali/.local/lib/python3.8/site-packages (from sp
acy<3.6.0,>=3.5.0->en-core-web-sm==3.5.0) (1.0.9)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /home/dipali/.local/lib/python3.8/site-packages (from spacy<3.
6.0,>=3.5.0->en-core-web-sm==3.5.0) (4.64.1)
```

In [29]:
```
#Perform standard imports:
import spacy
# Load English tokenizer, tagger, parser and NER
nlp = spacy.load('en_core_web_sm')
def show_lemmas(text):
    for token in text:
        print(f'{token.text:{12}} {token.pos_:{6}} {token.lemma:<{22}} {token.lemma_}')
```

In [30]:
```
doc = nlp(u"I am a runner running in a race because I love to run since I ran today.")

show_lemmas(doc)
```

```
I            PRON   4690420944186131903    I
am           AUX    10382539506755952630   be
a            DET    11901859001352538922   a
runner       NOUN   12640964157389618806   runner
running      VERB   12767647472892411841   run
in           ADP    3002984154512732771    in
a            DET    11901859001352538922   a
race         NOUN   8048469955494714898    race
because      SCONJ  16950148841647037698   because
I            PRON   4690420944186131903    I
love         VERB   3702023516439754181    love
to           PART   3791531372978436496    to
run          VERB   12767647472892411841   run
since        SCONJ  10066841407251338481   since
I            PRON   4690420944186131903    I
ran          VERB   12767647472892411841   run
today        NOUN   11042482332948150395   today
.            PUNCT  12646065887601541794   .
```

Conclusion- In this way we have performed tokenization using NLTK. And porter and snowball stemming. Using SpaCy library performed lemmatization.

Viva Questions

1. What is difference between porter and snowball stemmer?
2. What is lemmatization?
3. Differentiate between lemmatization and stemming.
4. What are different python libraries used for lemmatization.
5. Why do we need tokenization?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	

Group 1

Assignment No:2

Title of the Assignment:

Perform bag-of-words approach (count occurrence, normalized count occurrence), TF-IDF on data. Create embeddings using Word2Vec.
Dataset to be used: https://www.kaggle.com/datasets/CooperUnion/cardataset

Objective of the Assignment: To understand the fundamental concepts and techniques of natural language processing (NLP).

CO Relevance: CO1

Theory:

Word Embedding

It is an approach for representing words and documents. Word Embedding or Word Vector is a numeric vector input that represents a word in a lower-dimensional space. It allows words with similar meaning to have a similar representation. They can also approximate meaning. Word Embeddings are the texts converted into numbers and there may be different numerical representations of the same text.

In [1]:
```
pip install pandas
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas in /home/dipali/.local/lib/python3.8/site-packages (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /home/dipali/.local/lib/python3.8/site-packages (from pandas) (2022.
7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /home/dipali/.local/lib/python3.8/site-packages (from panda
s) (2.8.2)
Requirement already satisfied: numpy>=1.20.3 in /home/dipali/.local/lib/python3.8/site-packages (from pandas) (1.24.
1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from python-dateutil>=2.8.1->pandas) (1.1
4.0)

[notice] A new release of pip is available: 23.0 -> 23.0.1
[notice] To update, run: python3 -m pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
pip install sklearn
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: sklearn in /home/dipali/.local/lib/python3.8/site-packages (0.0.post1)

[notice] A new release of pip is available: 23.0 -> 23.0.1
[notice] To update, run: python3 -m pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
pip install scikit-learn
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: scikit-learn in /home/dipali/.local/lib/python3.8/site-packages (1.2.1)
Requirement already satisfied: joblib>=1.1.1 in /home/dipali/.local/lib/python3.8/site-packages (from scikit-learn)
(1.2.0)
Requirement already satisfied: scipy>=1.3.2 in /home/dipali/.local/lib/python3.8/site-packages (from scikit-learn)
(1.10.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /home/dipali/.local/lib/python3.8/site-packages (from scikit-
learn) (3.1.0)
Requirement already satisfied: numpy>=1.17.3 in /home/dipali/.local/lib/python3.8/site-packages (from scikit-learn)
(1.24.1)

[notice] A new release of pip is available: 23.0 -> 23.0.1
[notice] To update, run: python3 -m pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

Bag of words(BOW)

Bag of words is a simple and popular technique for feature extraction from text. Bag of word model processes the text to find how many times each word appeared in the sentence. This is also called as vectorization.

Steps for creating BOW

1. Tokenize the text into sentences
2. Tokenize sentences into words
3. Remove punctuation or stop words
4. Convert the words to lower text
5. Create the frequency distribution of words

In the code below, use CountVectorizer, it tokenizes a collection of text documents, builds a vocabulary of known words, and encodes new documents using that vocabulary.

In [11]:
```
#Creating frequency distribution of words using nltk
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
text="""Achievers are not afraid of Challenges, rather they relish them,
        thrive in them, use them. Challenges makes is stronger.
        Challenges makes us uncomfortable. If you get comfortable with uncomfort then you will grow.
        Challenge the challenge. """
#Tokenize the sentences from the text corpus
tokenized_text=sent_tokenize(text)#using CountVectorizer and removing stopwords in english language
cv1= CountVectorizer(lowercase=True,stop_words='english')#fitting the tonized senetnecs to the countvectorizer
text_counts=cv1.fit_transform(tokenized_text)# printing the vocabulary and the frequency distribution pf vocabulary i
print(cv1.vocabulary_)
print(text_counts.toarray())
```

```
{'achievers': 0, 'afraid': 1, 'challenges': 3, 'relish': 7, 'thrive': 9, 'use': 12, 'makes': 6, 'stronger': 8, 'unco
mfortable': 11, 'comfortable': 4, 'uncomfort': 10, 'grow': 5, 'challenge': 2}
[[1 1 0 1 0 0 0 1 0 1 0 0 1]
 [0 0 0 1 0 0 1 0 1 0 0 0 0]
 [0 0 0 1 0 0 1 0 0 0 0 1 0]
 [0 0 0 0 1 1 0 0 0 0 1 0 0]
 [0 0 2 0 0 0 0 0 0 0 0 0 0]]
```

Count Occurrence

Counting word occurrence. The reason behind of using this approach is that keyword or important signal will occur again and again. So if the number of occurrence represent the importance of word. More frequency means more importance.

In [12]:
```
import collections
import pandas as pd
import numpy as np


from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

In [13]:
```
doc = "India is my country. India is very beautiful country."

count_vec = CountVectorizer()
count_occurs = count_vec.fit_transform([doc])
count_occur_df = pd.DataFrame((count, word) for word, count in zip(count_occurs.toarray().tolist()[0],
                 count_vec. get_feature_names_out()))
count_occur_df.columns = ['Word', 'Count']
count_occur_df.sort_values('Count', ascending=False)
count_occur_df.head()
```

Out[13]:

	Word	Count
0	beautiful	1
1	country	2
2	india	2
3	is	2
4	my	1

Normalized Count Occurrence

If you think that high frequency may dominate the result and causing model bias. Normalization can be apply to pipeline easily.

In [14]:
```
doc = "India is my country. India is very beautiful country."

norm_count_vec = TfidfVectorizer(use_idf=False, norm='l2')
norm_count_occurs = norm_count_vec.fit_transform([doc])
norm_count_occur_df = pd.DataFrame((count, word) for word, count in zip(
    norm_count_occurs.toarray().tolist()[0], norm_count_vec. get_feature_names_out()))
norm_count_occur_df.columns = ['Word', 'Count']
norm_count_occur_df.sort_values('Count', ascending=False, inplace=True)
norm_count_occur_df.head()
```

Out[14]:

	Word	Count
1	country	0.516398
2	india	0.516398
3	is	0.516398
0	beautiful	0.258199
4	my	0.258199

TF-IDF

TF-IDF take another approach which is believe that high frequency may not able to provide much information gain. In another word, rare words contribute more weights to the model.

Word importance will be increased if the number of occurrence within same document (i.e. training record). On the other hand, it will be decreased if it occurs in corpus (i.e. other training records).

In [15]:
```
doc = "India is my country. India is very beautiful country."
tfidf_vec = TfidfVectorizer()
tfidf_count_occurs = tfidf_vec.fit_transform([doc])
tfidf_count_occur_df = pd.DataFrame((count, word) for word, count in zip(
    tfidf_count_occurs.toarray().tolist()[0], tfidf_vec. get_feature_names_out()))
tfidf_count_occur_df.columns = ['Word', 'Count']
tfidf_count_occur_df.sort_values('Count', ascending=True, inplace=True)
tfidf_count_occur_df.head()
```

Out[15]:

	Word	Count
0	beautiful	0.258199
4	my	0.258199
5	very	0.258199
1	country	0.516398
2	india	0.516398

Introduction to Word2Vec

Word2vec is one of the most popular technique to learn word embeddings using a two-layer neural network. Its input is a text corpus and its output is a set of vectors. Word embedding via word2vec can make natural language computer-readable, then further implementation of mathematical operations on words can be used to detect their similarities. A well-trained set of word vectors will place similar words close to each other in that space. For instance, the words women, men, and human might cluster in one corner, while yellow, red and blue cluster together in another.

There are two main training algorithms for word2vec, one is the continuous bag of words(CBOW), another is called skip-gram. The major difference between these two methods is that CBOW is using context to predict a target word while skip-gram is using a word to predict a target context. Generally, the skip-gram method can have a better performance compared with CBOW method, for it can capture two semantics for a single word. For instance, it will have two vector representations for Apple, one for the company and another for the fruit.

Gensim Python Library Introduction

Gensim is an open source python library for natural language processing and it was developed and is maintained by the Czech natural language processing researcher Radim Řehůřek. Gensim library will enable us to develop word embeddings by training our own word2vec models on a custom corpus either with CBOW of skip-grams algorithms.

In [1]:
```
!pip install --upgrade gensim
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting gensim
  Downloading gensim-4.3.0-cp38-cp38-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (24.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.1/24.1 MB 586.1 kB/s eta 0:00:00m eta 0:00:01[36m0:00:02
Collecting FuzzyTM>=0.4.0
  Downloading FuzzyTM-2.0.5-py3-none-any.whl (29 kB)
Requirement already satisfied: numpy>=1.18.5 in /home/dipali/.local/lib/python3.8/site-packages (from gensim) (1.24.
1)
Requirement already satisfied: smart-open>=1.8.1 in /home/dipali/.local/lib/python3.8/site-packages (from gensim)
(6.3.0)
Requirement already satisfied: scipy>=1.7.0 in /home/dipali/.local/lib/python3.8/site-packages (from gensim) (1.10.
1)
```

Download the data

Dataset Description

This vehicle dataset includes features such as make, model, year, engine, and other properties of the car. We will use these features to generate the word embeddings for each make model and then compare the similarities between different make model.

In [2]:
```
!wget https://raw.githubusercontent.com/PICT-NLP/BE-NLP-Elective/main/2-Embeddings/data.csv
```

```
--2023-03-01 21:22:40--  https://raw.githubusercontent.com/PICT-NLP/BE-NLP-Elective/main/2-Embeddings/data.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.108.133, 185.199.109.13
3, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1475504 (1.4M) [text/plain]
Saving to: 'data.csv.1'

data.csv.1          100%[===================>]   1.41M   587KB/s    in 2.5s

2023-03-01 21:22:43 (587 KB/s) - 'data.csv.1' saved [1475504/1475504]
```

Implementation of Word Embedding with Gensim

In [3]:
```
import pandas as pd
```

In [4]:
```
df = pd.read_csv('data.csv')
df.head()
```

Out[4]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	Vehicle Style	highway MPG	city mpg	Popularity
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	Coupe	26	19	3916
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Convertible	28	19	3916
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	Coupe	28	20	3916
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Coupe	28	18	3916
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible	28	18	3916

Data Preprocessing

Since the purpose of this tutorial is to learn how to generate word embeddings using genism library, we will not do the EDA and feature selection for the word2vec model for the sake of simplicity.

Genism word2vec requires that a format of 'list of lists' for training where every document is contained in a list and every list contains lists of tokens of that document. At first, we need to generate a format of 'list of lists' for training the make model word embedding. To be more specific, each make model is contained in a list and every list contains lists of features of that make model.

To achieve this, we need to do the following things

Create a new column for Make Model

In [5]:
```
df['Maker_Model']= df['Make']+ " " + df['Model']
```

Generate a format of ' list of lists' for each Make Model with the following features: Engine Fuel Type, Transmission Type, Driven_Wheels, Market Category, Vehicle Size, Vehicle Style.

In [6]:
```
df1 = df[['Engine Fuel Type','Transmission Type','Driven_Wheels','Market Category','Vehicle Size', 'Vehicle Style', '

df2 = df1.apply(lambda x: ','.join(x.astype(str)), axis=1)

df_clean = pd.DataFrame({'clean': df2})

sent = [row.split(',') for row in df_clean['clean']]
```

In [36]:
```
df_clean
```

Out[36]:

	clean
0	premium unleaded (required),MANUAL,rear wheel ...
1	premium unleaded (required),MANUAL,rear wheel ...
2	premium unleaded (required),MANUAL,rear wheel ...
3	premium unleaded (required),MANUAL,rear wheel ...
4	premium unleaded (required),MANUAL,rear wheel ...
...	...
11909	premium unleaded (required),AUTOMATIC,all whee...
11910	premium unleaded (required),AUTOMATIC,all whee...
11911	premium unleaded (required),AUTOMATIC,all whee...
11912	premium unleaded (recommended),AUTOMATIC,all w...
11913	regular unleaded,AUTOMATIC,front wheel drive,L...

11914 rows × 1 columns

Genism word2vec Model Training

We can train the genism word2vec model with our own custom corpus as following:

In [13]:
```
from gensim.models.word2vec import Word2Vec
```

Let's try to understand the hyperparameters of this model.

1. `vector_size` : The number of dimensions of the embeddings and the default is 100.
2. `window` : The maximum distance between a target word and words around the target word. The default window is 5.
3. `min_count` : The minimum count of words to consider when training the model; words with occurrence less than this count will be ignored. The default for min_count is 5.
4. `workers` : The number of partitions during training and the default workers is 3.
5. `sg` : The training algorithm, either CBOW(0) or skip gram(1). The default training algorithm is CBOW.

After training the word2vec model, we can obtain the word embedding directly from the training model as following.

In [29]:
```
model = Word2Vec(sent, min_count=1,vector_size= 50,workers=3, window =3, sg= 1)
```

Save the model

In [30]:
```
model.save("word2vec.model")
```

Load the model

In [31]:
```
model = Word2Vec.load("word2vec.model")
```

After training the word2vec model, we can obtain the word embedding directly from the training model as following.

In [32]:
```
model.wv['Toyota Camry']
```

Out[32]:
```
array([ 0.01411632,  0.14784585,  0.01885239, -0.10753247, -0.07146065,
       -0.22338827,  0.01374025,  0.30203253, -0.09214514, -0.08290682,
        0.04862676,  0.03026489,  0.11046173, -0.02939197, -0.03781607,
        0.17612398,  0.15454124,  0.29968512, -0.13931143, -0.29023492,
       -0.05522037, -0.0564889 ,  0.25777268,  0.07147302,  0.17070875,
        0.00251983, -0.03870121,  0.393018  , -0.04162066, -0.00246239,
        0.01573551,  0.02139783,  0.03586031,  0.00898253,  0.085445  ,
       -0.11928873,  0.1799241 , -0.02834527,  0.04921703,  0.08003329,
        0.10614782, -0.03156348, -0.22044587,  0.09316084,  0.37852806,
        0.0510865 , -0.0260468 , -0.15805483,  0.00056193,  0.01605724],
      dtype=float32)
```

In [33]:
```
sims = model.wv.most_similar('Toyota Camry', topn=10)
sims
```

Out[33]:
```
[('Chevrolet Malibu', 0.9873985648155212),
 ('Nissan Sentra', 0.9869186878204346),
 ('Mazda 6', 0.9854127764701843),
 ('Buick Verano', 0.9837399125099182),
 ('Toyota Avalon', 0.9836648106575012),
 ('Nissan Altima', 0.9834702610969543),
 ('Pontiac Grand Am', 0.9833094477653503),
 ('Chevrolet Cruze', 0.9826680421829224),
 ('Suzuki Verona', 0.9806841611862183),
 ('Suzuki Kizashi', 0.9794840812683105)]
```

Calculate similarity between two words

```
In [34]: model.wv.similarity('Toyota Camry','Mazda 6')
Out[34]: 0.98541266

In [35]: model.wv.similarity('Dodge Dart','Mazda 6')
Out[35]: 0.97065187
```

Conclusion: In this way we have implement Bag-of-Word (BOW), Tf-Idf appoarch by using python library. And word2vec model implemented successfully by using genism library.

Viva Questions:

1. What is Tf-Idf?
2. Differentiate between continuous-bags-of-words and skip-gram.
3. What is mean by Word Embedding? And what are techniques of word embedding?
4. Why word embedding is required?
5. Which library used in word embedding?

.

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	

Group 1

Assignment No 3

Title of the Assignment:
Perform text cleaning, perform lemmatization (any method), remove stop words (any method), label encoding. Create representations using TF-IDF. Save outputs.

Dataset: https://github.com/PICT-NLP/BE-NLP-Elective/blob/main/3-Preprocessing/News_dataset.pickle

Objective of the Assignment: To understand the fundamental concepts and techniques of natural language processing (NLP).

CO Relevance: CO1

Import libraries and load data

In [15]:
```
#Importing Libraries

import pickle
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import chi2
import numpy as np
```

In [16]:
```
#Accessing document uploaded

path_df = "News_dataset.pickle"

with open(path_df, 'rb') as data:
    df = pickle.load(data)
```

In [17]:
```
#checking data

df.head()
```

Out[17]:

	File_Name	Content	Category	Complete_Filename	id	News_length
0	001.txt	Ad sales boost Time Warner profit\r\n\r\nQuart...	business	001.txt-business	1	2569
1	002.txt	Dollar gains on Greenspan speech\r\n\r\nThe do...	business	002.txt-business	1	2257
2	003.txt	Yukos unit buyer faces loan claim\r\n\r\nThe o...	business	003.txt-business	1	1557
3	004.txt	High fuel prices hit BA's profits\r\n\r\nBriti...	business	004.txt-business	1	2421
4	005.txt	Pernod takeover talk lifts Domecq\r\n\r\nShare...	business	005.txt-business	1	1575

In [18]:
```
#Chcking article

df.loc[1]['Content']
```

Out[18]:
```
'Dollar gains on Greenspan speech\r\n\r\nThe dollar has hit its highest level against the euro in almost three month
s after the Federal Reserve head said the US trade deficit is set to stabilise.\r\n\r\nAnd Alan Greenspan highlighte
d the US government\'s willingness to curb spending and rising household savings as factors which may help to reduce
it. In late trading in New York, the dollar reached $1.2871 against the euro, from $1.2974 on Thursday. Market conce
rns about the deficit has hit the greenback in recent months. On Friday, Federal Reserve chairman Mr Greenspan\'s sp
eech in London ahead of the meeting of G7 finance ministers sent the dollar higher after it had earlier tumbled on t
he back of worse-than-expected US jobs data. "I think the chairman\'s taking a much more sanguine view on the curren
t account deficit than he\'s taken for some time," said Robert Sinche, head of currency strategy at Bank of America
in New York. "He\'s taking a longer-term view, laying out a set of conditions under which the current account defici
t can improve this year and next."\r\n\r\nWorries about the deficit concerns about China do, however, remain. Chin
a\'s currency remains pegged to the dollar and the US currency\'s sharp falls in recent months have therefore made C
hinese export prices highly competitive. But calls for a shift in Beijing\'s policy have fallen on deaf ears, despit
e recent comments in a major Chinese newspaper that the "time is ripe" for a loosening of the peg. The G7 meeting is
thought unlikely to produce any meaningful movement in Chinese policy. In the meantime, the US Federal Reserve\'s de
cision on 2 February to boost interest rates by a quarter of a point - the sixth such move in as many months - has o
pened up a differential with European rates. The half-point window, some believe, could be enough to keep US assets
looking more attractive, and could help prop up the dollar. The recent falls have partly been the result of big budg
et deficits, as well as the US\'s yawning current account gap, both of which need to be funded by the buying of US b
onds and assets by foreign firms and governments. The White House will announce its budget on Monday, and many comme
ntators believe the deficit will remain at close to half a trillion dollars.'
```

1. Text cleaning and preparation

In [19]:
```
#Text cleaning

df['Content_Parsed_1'] = df['Content'].str.replace("\r", " ")
df['Content_Parsed_1'] = df['Content_Parsed_1'].str.replace("\n", " ")
df['Content_Parsed_1'] = df['Content_Parsed_1'].str.replace("    ", " ")
df['Content_Parsed_1'] = df['Content_Parsed_1'].str.replace('"', '')
```

In [20]:
```
#Text preparation

df['Content_Parsed_2'] = df['Content_Parsed_1'].str.lower()         #all to lower case

punctuation_signs = list("?:!.,;")                                  #remove punctuations
df['Content_Parsed_3'] = df['Content_Parsed_2']

for punct_sign in punctuation_signs:
    df['Content_Parsed_3'] = df['Content_Parsed_3'].str.replace(punct_sign, '')

df['Content_Parsed_4'] = df['Content_Parsed_3'].str.replace("'s", "")       #remove possessive pronouns
```

```
/tmp/ipykernel_4042/3467828116.py:9: FutureWarning: The default value of regex will change from True to False in a f
uture version. In addition, single character regular expressions will *not* be treated as literal strings when regex
=True.
  df['Content_Parsed_3'] = df['Content_Parsed_3'].str.replace(punct_sign, '')
```

a) Use any 1 method for Lemmatization

In [21]:
```
#Stemming and Lemmatization

nltk.download('punkt')
nltk.download('wordnet')

nltk.download('averaged_perceptron_tagger')
from nltk.corpus import wordnet
```

```
[nltk_data] Downloading package punkt to /home/dipali/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /home/dipali/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/dipali/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

1st method for lemmatization

In [22]:
```
#Stemming and Lemmatization

wordnet_lemmatizer = WordNetLemmatizer()
nrows = len(df)
lemmatized_text_list = []

for row in range(0, nrows):

    # Create an empty list containing lemmatized words
    lemmatized_list = []

    # Save the text and its words into an object
    text = df.loc[row]['Content_Parsed_4']
    text_words = text.split(" ")

    # Iterate through every word to lemmatize
    for word in text_words:
        lemmatized_list.append(wordnet_lemmatizer.lemmatize(word, pos="v"))

    # Join the list
    lemmatized_text = " ".join(lemmatized_list)

    # Append to the list containing the texts
    lemmatized_text_list.append(lemmatized_text)

df['Content_Parsed_5'] = lemmatized_text_list
```

In [23]:
```
df['Content_Parsed_5']
```

Out[23]:
```
0       ad sales boost time warner profit quarterly pr...
1       dollar gain on greenspan speech the dollar hav...
2       yukos unit buyer face loan claim the owners of...
3       high fuel price hit ba profit british airways ...
4       pernod takeover talk lift domecq share in uk d...
                              ...
2220    bt program to beat dialler scam bt be introduc...
2221    spam e-mail tempt net shoppers computer users ...
2222    be careful how you code a new european directi...
2223    us cyber security chief resign the man make su...
2224    lose yourself in online game online role play ...
Name: Content_Parsed_5, Length: 2225, dtype: object
```

2nd method for lemmatization

In [24]:
```
lemmatizer = WordNetLemmatizer()

# function to convert nltk tag to wordnet tag
def nltk_tag_to_wordnet_tag(nltk_tag):
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

def lemmatize_sentence(sentence):
    #tokenize the sentence and find the POS tag for each token
    nltk_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
    #tuple of (token, wordnet_tag)
    wordnet_tagged = map(lambda x: (x[0], nltk_tag_to_wordnet_tag(x[1])), nltk_tagged)
    lemmatized_sentence = []
    for word, tag in wordnet_tagged:
        if tag is None:
            #if there is no available tag, append the token as is
            lemmatized_sentence.append(word)
        else:
            #else use the tag to lemmatize the token
            lemmatized_sentence.append(lemmatizer.lemmatize(word, tag))
    return " ".join(lemmatized_sentence)

nrows = len(df)
lemmatized_text_list = []

for row in range(0, nrows):
    lemmatized_text = lemmatize_sentence(df.loc[row]['Content_Parsed_4'])
    lemmatized_text_list.append(lemmatized_text)

df['Content_Parsed_5'] = lemmatized_text_list
```

In [25]:
```
df['Content_Parsed_5']
```

Out[25]:
```
0       ad sale boost time warner profit quarterly pro...
1       dollar gain on greenspan speech the dollar hav...
2       yukos unit buyer face loan claim the owner of ...
3       high fuel price hit ba profit british airway h...
4       pernod takeover talk lift domecq share in uk d...
                              ...
2220    bt program to beat dialler scam bt be introduc...
2221    spam e-mails tempt net shopper computer user a...
2222    be careful how you code a new european directi...
2223    us cyber security chief resign the man make su...
2224    lose yourself in online gaming online role pla...
Name: Content_Parsed_5, Length: 2225, dtype: object
```

b) Use any 1 method for stop word

In [26]:
```
#Downloading

nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /home/dipali/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[26]: True

In [27]:
```
#Removing stop words

stop_words = list(stopwords.words('english'))
```

1st Method

In [28]:
```
df['Content_Parsed_6'] = df['Content_Parsed_5']

for stop_word in stop_words:

    regex_stopword = r"\b" + stop_word + r"\b"
    df['Content_Parsed_6'] = df['Content_Parsed_6'].str.replace(regex_stopword, '')
```

```
/tmp/ipykernel_4042/2995545048.py:6: FutureWarning: The default value of regex will change from True to False in a f
uture version.
  df['Content_Parsed_6'] = df['Content_Parsed_6'].str.replace(regex_stopword, '')
```

In [29]:
```
df.loc[5]['Content_Parsed_6']
```

Out[29]:
```
'japan narrowly escape recession japan economy teeter   brink   technical recession   three month  september figure
show revised figure indicate growth   01 % -   similar-sized contraction   previous quarter   annual basis  data sug
gest annual growth   02 % suggest  much  hesitant recovery   previously  think  common technical definition   recess
ion  two successive quarter  negative growth  government  keen  play   worrying implication   data  maintain  view
japan economy remain   minor adjustment phase   upward climb   monitor development carefully say economy minister h
eizo takenaka    face   strengthen yen make export less competitive  indication  weaken economic condition ahead obs
erver  less sanguine  paint  picture   recovery much patchy  previously think say paul sheard economist  lehman brot
her  tokyo improvement   job market apparently  yet  fee   domestic demand  private consumption   02 %   third quart
er'
```

2nd Method

In [30]:
```
stop_list_final=[]
nrows = len(df)
stopwords_english = stopwords.words('english')

for row in range(0, nrows):

    # Create an empty list containing no stop words
    stop_list = []

    # Save the text and its words into an object
    text = df.loc[row]['Content_Parsed_5']
    text_words = text.split(" ")

    # Iterate through every word to remove stopwords
    for word in text_words:
        if (word not in stopwords_english):
          stop_list.append(word)

    # Join the list
    stop_text = " ".join(stop_list)

    # Append to the list containing the texts
    stop_list_final.append(stop_text)

df['Content_Parsed_6'] = stop_list_final
```

In [31]:
```
df.loc[5]['Content_Parsed_6']
```

Out[31]:
```
'japan narrowly escape recession japan economy teeter brink technical recession three month september figure show re
vised figure indicate growth 01 % - similar-sized contraction previous quarter annual basis data suggest annual grow
th 02 % suggest much hesitant recovery previously think common technical definition recession two successive quarter
negative growth government keen play worrying implication data maintain view japan economy remain minor adjustment p
hase upward climb monitor development carefully say economy minister heizo takenaka face strengthen yen make export
less competitive indication weaken economic condition ahead observer less sanguine paint picture recovery much patch
y previously think say paul sheard economist lehman brother tokyo improvement job market apparently yet fee domestic
demand private consumption 02 % third quarter'
```

In [32]:
```
#Checking data

df.head(1)
```

Out[32]:

	File_Name	Content	Category	Complete_Filename	id	News_length	Content_Parsed_1	Content_Parsed_2	Content_Parsed_3	Content_Parsed_4	Content
0	001.txt	Ad sales boost Time Warner profit\r \n\r\nQuart...	business	001.txt-business	1	2569	Ad sales boost Time Warner profit Quarterly pr...	ad sales boost time warner profit quarterly pr...	ad sales boost time warner profit quarterly pr...	ad sales boost time warner profit quarterly pr...	ad sale w qua

In [33]:
```
#Removing the old content_parsed columns

list_columns = ["File_Name", "Category", "Complete_Filename", "Content", "Content_Parsed_6"]
df = df[list_columns]

df = df.rename(columns={'Content_Parsed_6': 'Content_Parsed'})
```

In [34]:
```
df.head()
```

Out[34]:

	File_Name	Category	Complete_Filename	Content	Content_Parsed
0	001.txt	business	001.txt-business	Ad sales boost Time Warner profit\r\n\r\nQuart...	ad sale boost time warner profit quarterly pro...
1	002.txt	business	002.txt-business	Dollar gains on Greenspan speech\r\n\r\nThe do...	dollar gain greenspan speech dollar hit high l...
2	003.txt	business	003.txt-business	Yukos unit buyer faces loan claim\r\n\r\nThe o...	yukos unit buyer face loan claim owner embattl...
3	004.txt	business	004.txt-business	High fuel prices hit BA's profits\r\n\r\nBriti...	high fuel price hit ba profit british airway b...
4	005.txt	business	005.txt-business	Pernod takeover talk lifts Domecq\r\n\r\nShare...	pernod takeover talk lift domecq share uk drin...

2. Label coding

In [35]:
```
#Generating new column for Category codes

category_codes = {
    'business': 0,
    'entertainment': 1,
    'politics': 2,
    'sport': 3,
    'tech': 4
}

# Category mapping
df['Category_Code'] = df['Category']
df = df.replace({'Category_Code':category_codes})
```

In [36]:
```
df.head()
```

Out[36]:

	File_Name	Category	Complete_Filename	Content	Content_Parsed	Category_Code
0	001.txt	business	001.txt-business	Ad sales boost Time Warner profit\r\n\r\nQuart...	ad sale boost time warner profit quarterly pro...	0
1	002.txt	business	002.txt-business	Dollar gains on Greenspan speech\r\n\r\nThe do...	dollar gain greenspan speech dollar hit high l...	0
2	003.txt	business	003.txt-business	Yukos unit buyer faces loan claim\r\n\r\nThe o...	yukos unit buyer face loan claim owner embattl...	0
3	004.txt	business	004.txt-business	High fuel prices hit BA's profits\r\n\r\nBriti...	high fuel price hit ba profit british airway b...	0
4	005.txt	business	005.txt-business	Pernod takeover talk lifts Domecq\r\n\r\nShare...	pernod takeover talk lift domecq share uk drin...	0

3. Train - test split

In [37]:
```
X_train, X_test, y_train, y_test = train_test_split(df['Content_Parsed'],
                                                    df['Category_Code'],
                                                    test_size=0.15,
                                                    random_state=8)
```

4. Text representation

TF-IDF Vectors

unigrams & bigrams corresponding to a particular category

In [38]:
```
# Parameter election
ngram_range = (1,2)
min_df = 10
max_df = 1.
max_features = 300
```

In [39]:
```
tfidf = TfidfVectorizer(encoding='utf-8',
                        ngram_range=ngram_range,
                        stop_words=None,
                        lowercase=False,
                        max_df=max_df,
                        min_df=min_df,
                        max_features=max_features,
                        norm='l2',
                        sublinear_tf=True)

features_train = tfidf.fit_transform(X_train).toarray()
labels_train = y_train
print(features_train.shape)

features_test = tfidf.transform(X_test).toarray()
labels_test = y_test
print(features_test.shape)
```

```
(1891, 300)
(334, 300)
```

In [41]:
```
from sklearn.feature_selection import chi2
import numpy as np

for Product, category_id in sorted(category_codes.items()):
    features_chi2 = chi2(features_train, labels_train == category_id)
    indices = np.argsort(features_chi2[0])
    feature_names = np.array(tfidf.get_feature_names_out())[indices]
    unigrams = [v for v in feature_names if len(v.split(' ')) == 1]
    bigrams = [v for v in feature_names if len(v.split(' ')) == 2]
    print("# '{}' category:".format(Product))
    print("  . Most correlated unigrams:\n. {}".format('\n. '.join(unigrams[-5:])))
    print("  . Most correlated bigrams:\n. {}".format('\n. '.join(bigrams[-2:])))
    print("")
```

```
# 'business' category:
  . Most correlated unigrams:
. price
. market
. economy
. growth
. bank
  . Most correlated bigrams:
. last year
. year old

# 'entertainment' category:
  . Most correlated unigrams:
. best
. music
. star
. award
. film
  . Most correlated bigrams:
. mr blair
. prime minister
```

```
# 'politics' category:
  . Most correlated unigrams:
. blair
. party
. election
. tory
. labour
  . Most correlated bigrams:
. prime minister
. mr blair

# 'sport' category:
  . Most correlated unigrams:
. side
. player
. team
. game
. match
  . Most correlated bigrams:
. say mr
. year old

# 'tech' category:
  . Most correlated unigrams:
. mobile
. software
. technology
. computer
. user
  . Most correlated bigrams:
. year old
. say mr
```

```
In [42]: bigrams
Out[42]: ['tell bbc', 'last year', 'mr blair', 'prime minister', 'year old', 'say mr']
```

Conclusion:

Hence, we perform text cleaning, lemmatization, remove stop words, label encoding and create representations using TF-IDF.

Viva Question:

1. What is lemmatization?
2. Differentiate between lemmatization and stemming?
3. How calculate TF-IDF?
4. What is pickle library?
5. How perform text cleaning?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	

Group 1

Assignment No 4

Title of the Assignment:
Create a transformer from scratch using the Pytorch library.

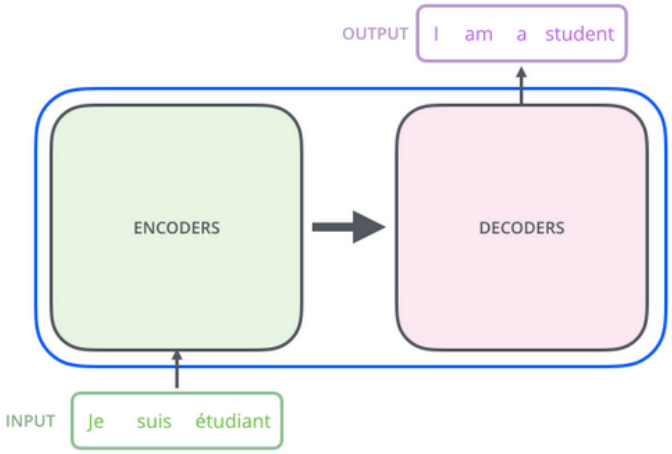
Objective of the Assignment: To understand the fundamental concepts and techniques of natural language processing (NLP).

CO Relevance: CO1

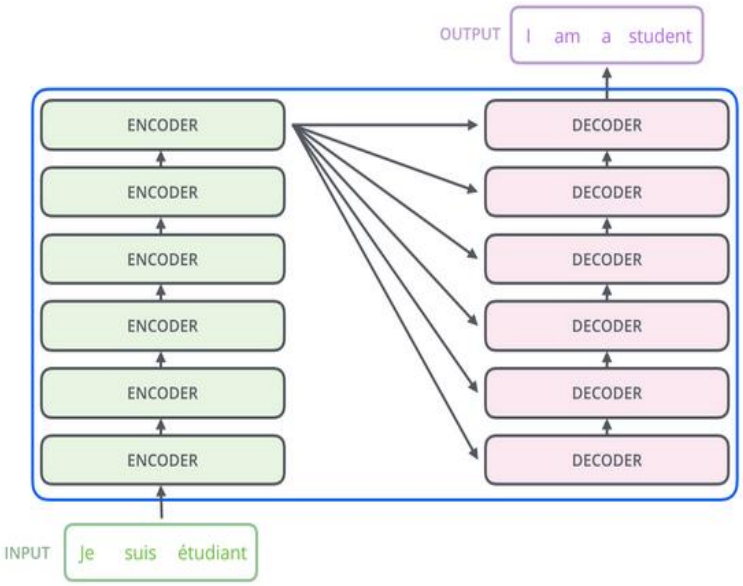
Transformers From Scratch Using Pytorch

1. Introduction

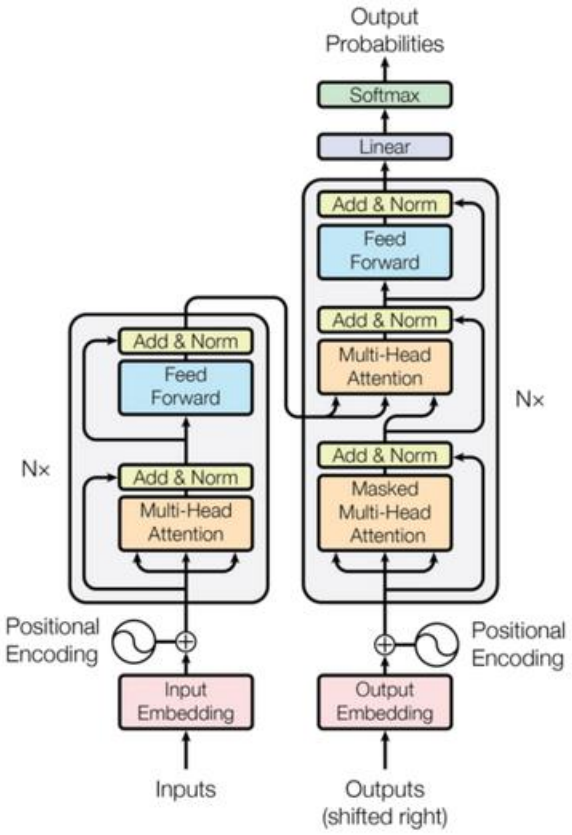
Implement transformers in "Attention is all you need paper" from scratch using Pytorch. Basically transformer have an encoder-decoder architecture. It is common for language translation models.



The above image shows a language translation model from French to English. Actually we can use stack of encoder(one in top of each) and stack of decoders as below:



Before going further Let us see a full fledged image of our attention model.



2. Import Libraries

In [1]:
```
pip install torchvision
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: torchvision in /home/dipali/.local/lib/python3.8/site-packages (0.15.1)
Collecting torch==2.0.0
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
pip install torchtext==0.10.0
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: torchtext==0.10.0 in /home/dipali/.local/lib/python3.8/site-packages (0.10.0)
Requirement already satisfied: torch==1.9.0 in /home/dipali/.local/lib/python3.8/site-packages (from torchtext==0.1
0.0) (1.9.0)
Requirement already satisfied: numpy in /home/dipali/.local/lib/python3.8/site-packages (from torchtext==0.10.0) (1.
24.1)
Requirement already satisfied: requests in /usr/lib/python3/dist-packages (from torchtext==0.10.0) (2.22.0)
Requirement already satisfied: tqdm in /home/dipali/.local/lib/python3.8/site-packages (from torchtext==0.10.0) (4.6
4.1)
Requirement already satisfied: typing-extensions in /home/dipali/.local/lib/python3.8/site-packages (from torch==1.
9.0->torchtext==0.10.0) (4.4.0)

[notice] A new release of pip is available: 23.0 -> 23.0.1
[notice] To update, run: python3 -m pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
# importing required libraries
import torch.nn as nn
import torch
import torch.nn.functional as F
import math,copy,re
import warnings
import pandas as pd
import numpy as np
import seaborn as sns
import torchtext
import matplotlib.pyplot as plt
warnings.simplefilter("ignore")
print(torch.__version__)
```

```
1.9.0+cu102
```

3. Basic components

Positional Encoding

Next step is to generate positional encoding. Inorder for the model to make sense of the sentence, it needs to know two things about the each word.

- what does the word mean?
- what is the position of the word in the sentence.

In "attention is all you need paper" author used the following functions to create positional encoding. On odd time steps a cosine function is used and in even time steps a sine function is used.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

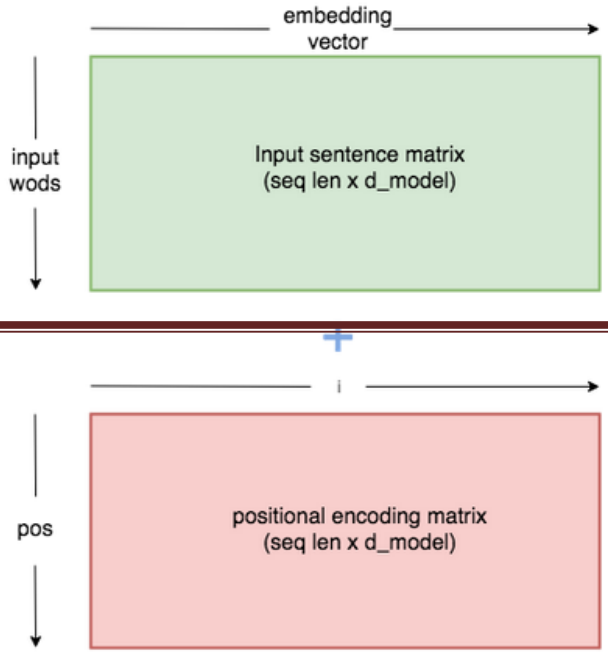
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

```
pos -> refers to order in the sentence
i -> refers to position along embedding vector dimension
```

Positinal embedding will generate a matrix of similar to embedding matrix. It will create a matrix of dimension sequence length x embedding dimension. For each token(word) in sequence, we will find the embedding vector which is of dimension 1 x 512 and it is added with the correspondng positional vector which is of dimension 1 x 512 to get 1 x 512 dim out for each word/token.

for eg: if we have batch size of 32 and seq length of 10 and let embedding dimension be 512. Then we will have embedding vector of dimension 32 x 10 x 512. Similarly we will have positional encoding vector of dimension 32 x 10 x 512. Then we add both.

```
        return out
```



In [5]:
```
# register buffer in Pytorch ->
# If you have parameters in your model, which should be saved and restored in the state_dict,
# but not trained by the optimizer, you should register them as buffers.


class PositionalEmbedding(nn.Module):
    def __init__(self,max_seq_len,embed_model_dim):
        """
        Args:
            seq_len: length of input sequence
            embed_model_dim: demension of embedding
        """
        super(PositionalEmbedding, self).__init__()
        self.embed_dim = embed_model_dim

        pe = torch.zeros(max_seq_len,self.embed_dim)
        for pos in range(max_seq_len):
            for i in range(0,self.embed_dim,2):
                pe[pos, i] = math.sin(pos / (10000 ** ((2 * i)/self.embed_dim)))
                pe[pos, i + 1] = math.cos(pos / (10000 ** ((2 * (i + 1))/self.embed_dim)))
        pe = pe.unsqueeze(0)
        self.register_buffer('pe', pe)


    def forward(self, x):
        """
        Args:
            x: input vector
        Returns:
            x: output
        """

        # make embeddings relatively larger
        x = x * math.sqrt(self.embed_dim)
        #add constant to embedding
        seq_len = x.size(1)
        x = x + torch.autograd.Variable(self.pe[:,:seq_len], requires_grad=False)
        return x
```

Self Attention

Let me give a glimpse on Self Attention and Multihead attention

What is self attention?

Suppose we have a sentence "Dog is crossing the street because it saw the kitchen".What does it refers to here? It's easy to understand for the humans that it is Dog. But not for the machines.

As model proceeses each word, self attention allows it to look at other positions in the input sequence for clues. It will creates a vector based on dependency of each word with the other.

Let us go through a step by step illustration of self attention.

- **Step 1:** The first step in calculating self-attention is to create three vectors from each of the encoder's input vectors (in this case, the embedding of each word). So for each word, we create a Query vector, a Key vector, and a Value vector. Each of the vector will be of dimension 1x64.

Since we have a multihead attention we will have 8 self attention heads I will explain the code with 8 attention head in mind

- **Step 2:** Second step is to calculate the score. ie, we will multiply query marix with key matrix. [Q x K.t]

  code hint:
  Suppose our key,query and value dimension be 32x10x8x64. Before proceeding further, we will transpose each of them for multiplication convinience (32x8x10x64). Now multiply query matrix with transpose key matrix. ie (32x8x10x64) x (32x8x64x10) -> (32x8x10x10).

- **Step 3:** Now divide the output matrix with square root of dimension of key matrix and then apply Softmax over it.

  code hint: we will divide 32x8x10x10 vector by 8 ie, by square root of 64 (dimension of key matrix)

- **Step 4:** Then this gets multiply it with value matrix.

  code hint:
  After step 3 our output will be of dimension 32x8x10x10. Now muliply it with value matrix (32x8x10x64) to get output of dimension (32x8x10x64).Here 8 is the number of attention heads and 10 is the sequence length.Thus for each word we have 64 dim vector.

- **Step 5:** Once we have this we will pass this through a linear layer. This forms the output of multihead attention.

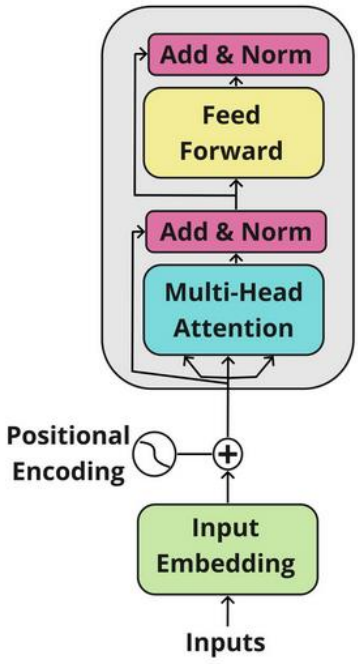
  code hint:
  (32x8x10x64) vector gets transposed to (32x10x8x64) and then reshaped as (32x10x512).Then it is passed through a linear layer to get output of (32x10x512).

Now you got an idea on how multihead attention works. You will be more clear once you go through the implementation part of it.

In [6]:
```
class MultiHeadAttention(nn.Module):
    def __init__(self, embed_dim=512, n_heads=8):
        """
        Args:
            embed_dim: dimension of embeding vector output
            n_heads: number of self attention heads
        """
        super(MultiHeadAttention, self).__init__()

        self.embed_dim = embed_dim    #512 dim
        self.n_heads = n_heads   #8
        self.single_head_dim = int(self.embed_dim / self.n_heads)   #512/8 = 64  . each key,query, value will be of 6

        #key,query and value matrixes    #64 x 64
        self.query_matrix = nn.Linear(self.single_head_dim , self.single_head_dim ,bias=False)  # single key matrix f
        self.key_matrix = nn.Linear(self.single_head_dim  , self.single_head_dim, bias=False)
        self.value_matrix = nn.Linear(self.single_head_dim ,self.single_head_dim , bias=False)
        self.out = nn.Linear(self.n_heads*self.single_head_dim ,self.embed_dim)

    def forward(self,key,query,value,mask=None):    #batch_size x sequence_length x embedding_dim    # 32 x 10 x 512

        """
        Args:
           key : key vector
           query : query vector
           value : value vector
           mask: mask for decoder

        Returns:
           output vector from multihead attention
        """
        batch_size = key.size(0)
        seq_length = key.size(1)

        # query dimension can change in decoder during inference.
        # so we cant take general seq_length
        seq_length_query = query.size(1)
```

```
        # 32x10x512
        key = key.view(batch_size, seq_length, self.n_heads, self.single_head_dim)  #batch_size x sequence_length x n
        query = query.view(batch_size, seq_length_query, self.n_heads, self.single_head_dim) #(32x10x8x64)
        value = value.view(batch_size, seq_length, self.n_heads, self.single_head_dim) #(32x10x8x64)

        k = self.key_matrix(key)       # (32x10x8x64)
        q = self.query_matrix(query)
        v = self.value_matrix(value)

        q = q.transpose(1,2)  # (batch_size, n_heads, seq_len, single_head_dim)    # (32 x 8 x 10 x 64)
        k = k.transpose(1,2)  # (batch_size, n_heads, seq_len, single_head_dim)
        v = v.transpose(1,2)  # (batch_size, n_heads, seq_len, single_head_dim)

        # computes attention
        # adjust key for matrix multiplication
        k_adjusted = k.transpose(-1,-2)  #(batch_size, n_heads, single_head_dim, seq_ken)  #(32 x 8 x 64 x 10)
        product = torch.matmul(q, k_adjusted)  #(32 x 8 x 10 x 64) x (32 x 8 x 64 x 10) = #(32x8x10x10)


        # fill those positions of product matrix as (-1e20) where mask positions are 0
        if mask is not None:
             product = product.masked_fill(mask == 0, float("-1e20"))

        #divising by square root of key dimension
        product = product / math.sqrt(self.single_head_dim) # / sqrt(64)

        #applying softmax
        scores = F.softmax(product, dim=-1)

        #mutiply with value matrix
        scores = torch.matmul(scores, v)  ##(32x8x 10x 10) x (32 x 8 x 10 x 64) = (32 x 8 x 10 x 64)

        #concatenated output
        concat = scores.transpose(1,2).contiguous().view(batch_size, seq_length_query, self.single_head_dim*self.n_he

        output = self.out(concat) #(32,10,512) -> (32,10,512)

        return output
```

4. Encoder



In the encoder section -

Step 1: First input(padded tokens corresponding to the sentence) get passes through embedding layer and positional encoding layer.

```
code hint
suppose we have input of 32x10 (batch size=32 and sequence length=10). Once it passes through embedding layer
it becomes 32x10x512. Then it gets added with correspondng positional encoding vector and produces output of
32x10x512. This gets passed to the multihead attention
```

Step 2: As discussed above it will passed through the multihead attention layer and creates useful representational matrix as output.

```
code hint
input to multihead attention will be a 32x10x512 from which key,query and value vectors are generated as abov
e and finally produces a 32x10x512 output.
```

Step 3: Next we have a normalization and residual connection. The output from multihead attention is added with its input and then normalized.

```
code hint
output of multihead attention which is 32x10x512 gets added with 32x10x512 input(which is output created by e
mbedding vector) and then the layer is normalized.
```

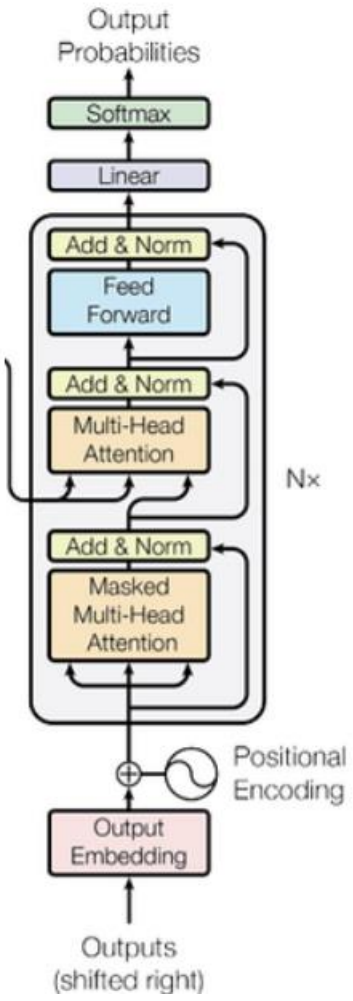
Step 4: Next we have a feed forward layer and a then normalization layer with residual connection from input(input of feed forward layer) where we passes the output after normalization though it and finally gets the output of encoder.

```
code hint
The normalized output will be of dimension 32x10x512. This gets passed through 2 linear layers: 32x10x512 ->
32x10x2048 -> 32x10x512. Finally we have a residual connection which gets added with the output and the layer
is normalized. Thus a 32x10x512 dimensional vector is created as output for the encoder.
```

In [7]:
```
class TransformerBlock(nn.Module):
    def __init__(self, embed_dim, expansion_factor=4, n_heads=8):
        super(TransformerBlock, self).__init__()

        """
        Args:
           embed_dim: dimension of the embedding
           expansion_factor: fator ehich determines output dimension of linear layer
           n_heads: number of attention heads

        """
        self.attention = MultiHeadAttention(embed_dim, n_heads)

        self.norm1 = nn.LayerNorm(embed_dim)
        self.norm2 = nn.LayerNorm(embed_dim)

        self.feed_forward = nn.Sequential(
                        nn.Linear(embed_dim, expansion_factor*embed_dim),
                        nn.ReLU(),
                        nn.Linear(expansion_factor*embed_dim, embed_dim)
        )

        self.dropout1 = nn.Dropout(0.2)
        self.dropout2 = nn.Dropout(0.2)

    def forward(self,key,query,value):

        """
        Args:
           key: key vector
           query: query vector
           value: value vector
           norm2_out: output of transformer block

        """

        attention_out = self.attention(key,query,value)  #32x10x512
        attention_residual_out = attention_out + value  #32x10x512
        norm1_out = self.dropout1(self.norm1(attention_residual_out)) #32x10x512
```

```
        feed_fwd_out = self.feed_forward(norm1_out) #32x10x512 -> #32x10x2048 -> 32x10x512
        feed_fwd_residual_out = feed_fwd_out + norm1_out #32x10x512
        norm2_out = self.dropout2(self.norm2(feed_fwd_residual_out)) #32x10x512

        return norm2_out


class TransformerEncoder(nn.Module):
    """
    Args:
        seq_len : length of input sequence
        embed_dim: dimension of embedding
        num_layers: number of encoder layers
        expansion_factor: factor which determines number of linear layers in feed forward layer
        n_heads: number of heads in multihead attention

    Returns:
        out: output of the encoder
    """
    def __init__(self, seq_len, vocab_size, embed_dim, num_layers=2, expansion_factor=4, n_heads=8):
        super(TransformerEncoder, self).__init__()

        self.embedding_layer = Embedding(vocab_size, embed_dim)
        self.positional_encoder = PositionalEmbedding(seq_len, embed_dim)

        self.layers = nn.ModuleList([TransformerBlock(embed_dim, expansion_factor, n_heads) for i in range(num_layers

    def forward(self, x):
        embed_out = self.embedding_layer(x)
        out = self.positional_encoder(embed_out)
        for layer in self.layers:
            out = layer(out,out,out)

        return out  #32x10x512
```

5. Decoder ¶



Now we have gone through most parts of the encoder.Let us get in to the components of the decoder. We will use the output of encoder to generate key and value vectors for the decoder.There are two kinds of multi head attention in the decoder.One is the decoder attention and other is the encoder decoder attention. Don't worry we will go step by step.

Let us explain with respect to the training phase. Firt

Step 1:

First the output gets passed through the embeddin and positional encoding to create a embedding vector of dimension 1x512 corresponding to each word in the target sequence.

```
code hint
Suppose we have a sequence length of 10. batch size of 32 and embedding vector dimension of 512. we have inpu
t of size 32x10 to the embedding matrix which produces and output of dimension 32x10x512 which gets added wit
h the positional encoding of same dimension and produces a 32x10x512 out
```

Step 2:

The embeddig output gets passed through a multihead attention layers as before(creating key,query and value matrixes from the target input) and produces an output vector. This time the major difference is that we uses a mask with multihead attention.

Why mask?

Mask is used because while creating attention of target words, we donot need a word to look in to the future words to check the dependency. ie, we already learned that why we create attention because we need to know contribution of each word with the other word. Since we are creating attention for words in target sequnce, we donot need a particular word to see the future words. For eg: in word "I am a strudent", we donot need the word "a" to look word "student".

```
code hint
For creating attention we created a triangular matrix with 1 and 0.eg:traingular matrix for seq length 5 look
s as below:

1 0 0 0 0
1 1 0 0 0
1 1 1 0 0
1 1 1 1 0
1 1 1 1 1

After the key gets multiplied with query, we fill all zero positions with negative inifinity, In code we will
fill it with a very small number to avoid division errors.
(with -1e 20)
```

Step 3:

As before we have a add and norm layer where we add with output of embedding with attention out and normalized it.

Step 4:

Next we have another multihead attention and then a add and norm layer. This multihead attention is called encoder-decorder multihead attention. For this multihead attention we create we create key and value vectors from the encoder output. Query is created from the output of previous decoder layer.

```
code hint:
Thus we have 32x10x512 out from encoder out. key and value for all words are generated from it. Similary quer
y matrix is generated from otput from previous layer of decoder(32x10x512).
```

Thus it is passed through a multihead atention (we used number of heads = 8) the through a Add and Norm layer. Here the output from previous encoder layer(ie previoud add and norm layer) gets added with encoder-decoder attention output and then normalized.

Step 5: Next we have a feed forward layer(linear layer) with add and nom which is similar to that of present in the encoder.

Step 6: Finally we create a linear layer with length equal to number of words in total target corpus and a softmax function with it to get probablity of each word.

In [8]:
```
class DecoderBlock(nn.Module):
    def __init__(self, embed_dim, expansion_factor=4, n_heads=8):
        super(DecoderBlock, self).__init__()

        """
        Args:
           embed_dim: dimension of the embedding
           expansion_factor: fator ehich determines output dimension of linear layer
           n_heads: number of attention heads

        """
        self.attention = MultiHeadAttention(embed_dim, n_heads=8)
        self.norm = nn.LayerNorm(embed_dim)
        self.dropout = nn.Dropout(0.2)
        self.transformer_block = TransformerBlock(embed_dim, expansion_factor, n_heads)


    def forward(self, key, query, x,mask):

        """
        Args:
           key: key vector
           query: query vector
           value: value vector
           mask: mask to be given for multi head attention
        Returns:
           out: output of transformer block

        """

        #we need to pass mask mask only to fst attention
        attention = self.attention(x,x,x,mask=mask) #32x10x512
        value = self.dropout(self.norm(attention + x))

        out = self.transformer_block(key, query, value)


        return out
```

```
class TransformerDecoder(nn.Module):
    def __init__(self, target_vocab_size, embed_dim, seq_len, num_layers=2, expansion_factor=4, n_heads=8):
        super(TransformerDecoder, self).__init__()
        """
        Args:
           target_vocab_size: vocabulary size of taget
           embed_dim: dimension of embedding
           seq_len : length of input sequence
           num_layers: number of encoder layers
           expansion_factor: factor which determines number of linear layers in feed forward layer
           n_heads: number of heads in multihead attention

        """
        self.word_embedding = nn.Embedding(target_vocab_size, embed_dim)
        self.position_embedding = PositionalEmbedding(seq_len, embed_dim)

        self.layers = nn.ModuleList(
            [
                DecoderBlock(embed_dim, expansion_factor=4, n_heads=8)
                for _ in range(num_layers)
            ]

        )
        self.fc_out = nn.Linear(embed_dim, target_vocab_size)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x, enc_out, mask):

        """
        Args:
            x: input vector from target
            enc_out : output from encoder layer
            trg_mask: mask for decoder self attention
        Returns:
            out: output vector
        """
```

```
    def forward(self, x, enc_out, mask):

        """
        Args:
            x: input vector from target
            enc_out : output from encoder layer
            trg_mask: mask for decoder self attention
        Returns:
            out: output vector
        """


        x = self.word_embedding(x)  #32x10x512
        x = self.position_embedding(x) #32x10x512
        x = self.dropout(x)

        for layer in self.layers:
            x = layer(enc_out, x, enc_out, mask)

        out = F.softmax(self.fc_out(x))

        return out
```

Finally we will arrange all submodules and creates the entire tranformer architecture.

In [9]:

```
class Transformer(nn.Module):
    def __init__(self, embed_dim, src_vocab_size, target_vocab_size, seq_length,num_layers=2, expansion_factor=4, n_h
        super(Transformer, self).__init__()

        """
        Args:
           embed_dim:  dimension of embedding
           src_vocab_size: vocabulary size of source
           target_vocab_size: vocabulary size of target
           seq_length : length of input sequence
           num_layers: number of encoder layers
           expansion_factor: factor which determines number of linear layers in feed forward layer
           n_heads: number of heads in multihead attention

        """

        self.target_vocab_size = target_vocab_size

        self.encoder = TransformerEncoder(seq_length, src_vocab_size, embed_dim, num_layers=num_layers, expansion_fac
        self.decoder = TransformerDecoder(target_vocab_size, embed_dim, seq_length, num_layers=num_layers, expansion_


    def make_trg_mask(self, trg):
        """
        Args:
            trg: target sequence
        Returns:
            trg_mask: target mask
        """
        batch_size, trg_len = trg.shape
        # returns the lower triangular part of matrix filled with ones
        trg_mask = torch.tril(torch.ones((trg_len, trg_len))).expand(
            batch_size, 1, trg_len, trg_len
        )
        return trg_mask
```

```
    def decode(self,src,trg):
        """
        for inference
        Args:
            src: input to encoder
            trg: input to decoder
        out:
            out_labels : returns final prediction of sequence
        """
        trg_mask = self.make_trg_mask(trg)
        enc_out = self.encoder(src)
        out_labels = []
        batch_size,seq_len = src.shape[0],src.shape[1]
        #outputs = torch.zeros(seq_len, batch_size, self.target_vocab_size)
        out = trg
        for i in range(seq_len): #10
            out = self.decoder(out,enc_out,trg_mask) #bs x seq_len x vocab_dim
            # taking the last token
            out = out[:,-1,:]

            out = out.argmax(-1)
            out_labels.append(out.item())
            out = torch.unsqueeze(out,axis=0)


        return out_labels

    def forward(self, src, trg):
        """
        Args:
            src: input to encoder
            trg: input to decoder
        out:
            out: final vector which returns probabilities of each target word
        """
        trg_mask = self.make_trg_mask(trg)
        enc_out = self.encoder(src)

        outputs = self.decoder(trg, enc_out, trg_mask)
        return outputs
```

6. Testing Code

Suppose we have input sequence oflength 10 and target sequence of length 10.

In [10]:
```
src_vocab_size = 11
target_vocab_size = 11
num_layers = 6
seq_length= 12


# let 0 be sos token and 1 be eos token
src = torch.tensor([[0, 2, 5, 6, 4, 3, 9, 5, 2, 9, 10, 1],
                    [0, 2, 8, 7, 3, 4, 5, 6, 7, 2, 10, 1]])
target = torch.tensor([[0, 1, 7, 4, 3, 5, 9, 2, 8, 10, 9, 1],
                       [0, 1, 5, 6, 2, 4, 7, 6, 2, 8, 10, 1]])

print(src.shape,target.shape)
model = Transformer(embed_dim=512, src_vocab_size=src_vocab_size,
                    target_vocab_size=target_vocab_size, seq_length=seq_length,
                    num_layers=num_layers, expansion_factor=4, n_heads=8)
model
```

```
torch.Size([2, 12]) torch.Size([2, 12])
```

Out[10]:
```
Transformer(
  (encoder): TransformerEncoder(
    (embedding_layer): Embedding(
      (embed): Embedding(11, 512)
    )
    (positional_encoder): PositionalEmbedding()
    (layers): ModuleList(
      (0): TransformerBlock(
        (attention): MultiHeadAttention(
          (query_matrix): Linear(in_features=64, out_features=64, bias=False)
          (key_matrix): Linear(in_features=64, out_features=64, bias=False)
          (value_matrix): Linear(in_features=64, out_features=64, bias=False)
          (out): Linear(in_features=512, out_features=512, bias=True)
        )
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (feed_forward): Sequential(
          (0): Linear(in_features=512, out_features=2048, bias=True)
```

In [11]:
```
out = model(src, target)
out.shape
```

Out[11]:
```
torch.Size([2, 12, 11])
```

In [12]:
```
# inference
model = Transformer(embed_dim=512, src_vocab_size=src_vocab_size,
                    target_vocab_size=target_vocab_size, seq_length=seq_length,
                    num_layers=num_layers, expansion_factor=4, n_heads=8)



src = torch.tensor([[0, 2, 5, 6, 4, 3, 9, 5, 2, 9, 10, 1]])
trg = torch.tensor([[0]])
print(src.shape,trg.shape)
out = model.decode(src, trg)
out
```

```
torch.Size([1, 12]) torch.Size([1, 1])
```

Out[12]:
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Conclusion:

Implement a transformer from scratch using the Pytorch library.

Viva Question:

- What are the types of embedding in transformer?
- What are the applications of Transformer
- What is the purpose of transformers in NLP?
- What is a Transformer?
- What is Self Attention?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator :	

Group 1

Assignment No 5

Title of the Assignment:
Morphology is the study of the way words are built up from smaller meaning bearing units. Study and understand the concepts of morphology by the use of add delete table.

Objective of the Assignment: To understand the fundamental concepts and techniques of natural language processing (NLP).

CO Relevance: CO1

Morphology:

Morphology is the study of the way words are built up from smaller meaning bearing units i.e., morphemes. A morpheme is the smallest meaningful linguistic unit. For eg:

- बच्चों(bachchoM) consists of two morphemes, बच्चा(bachchaa) has the information of the root word noun "बच्चा"(bachchaa) and ओं(oM) has the information of plural and oblique case.
- played has two morphemes play and -ed having information verb "play" and "past tense", so given word is past tense form of verb "play".

बच्चा	-ओं
लड़का	-ओं
play	-ed
want	-ed

Words can be analysed morphologically if we know all variants of a given root word. We can use an 'Add-Delete' table for this analysis.

Morph Analyser

Definition:

Morphemes are considered as smallest meaningful units of language. These morphemes can either be a root word(play) or affix(-ed). Combination of these morphemes is called morphological process. So, word "played" is made out of 2 morphemes "play" and "-ed". Thus finding all parts of a word(morphemes) and thus describing properties of a word is called "Morphological Analysis". For example, "played" has information verb "play" and "past tense", so given word is past tense form of verb "play".

Analysis of a word:

बच्चों (bachchoM) = बच्चा(bachchaa)(root) + ओं(oM)(suffix) (ओं=3 plural oblique) A linguistic paradigm is the complete set of variants of a given lexeme. These variants can be classified according to shared inflectional categories (eg: number, case etc) and arranged into tables.

Paradigm for बच्चा

Case/num	Singular	Plural
Direct	बच्चा(bachchaa)	बच्चे(bachche)
oblique	बच्चे(bachche)	बच्चों (bachchoM)

Algorithm to get बच्चों(bachchoM) from बच्चा(bachchaa)

1. Take Root बच्च(bachch)आ(aa)
2. Delete आ(aa)
3. output बच्च(bachch)
4. Add ओं(oM) to output
5. Return बच्चों (bachchoM)

Therefore आ is deleted and ओं is added to get बच्चों

Add-Delete table for बच्चा

Delete	Add	Number	Case	Variants
आ(aa)	आ(aa)	sing	dr	बच्चा(bachchaa)
आ(aa)	ए(e)	Plu	dr	बच्चे(bachche)
आ(aa)	ए(e)	Sing	ob	बच्चे(bachche)
आ(aa)	ओं(oM)	Plu	ob	बच्चों(bachchoM)

Paradigm Class

Words in the same paradigm class behave similarly, for Example लड़क is in the same paradigm class as बच्च, so लड़का would behave similarly as बच्चा as they share the same paradigm class.

Conclusion: Understanding the morphology of a word by the use of Add-Delete table successfully.

Viva Questions:

- What is a Morphology?
- What are types of Morphology?
- Why do we need to do Morphological Analysis?
- What is the application of morphology in linguistics?
- What is difference between inflectional and derivational morphology?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator :	

Group 2

Mini-Project

Title of the Assignment:

1. **Mini Project (Fine tune transformers on your preferred task)**

Finetune a pretrained transformer for any of the following tasks on any relevant dataset of your choice:

- **Neural Machine Translation**
- **Classification**
- **Summarization**

2. **Mini Project - POS Taggers For Indian Languages**

3. **Mini Project -Feature Extraction using seven moment variants**

4. **Mini Project -Feature Extraction using Zernike Moments**

Objective of the Assignment: To understand the concept of Mini-project.

Outcome: Students will be able to learn and understand concept project.

Theory:

Abstract:

Introduction:

Software Requirement Specification:

Graphical User Interface:

Source Code:

Testing document:

Conclusion:

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	