Massive Data Storage and Retrieval

Final Report Submission

Submited by Nitin Reddy Karolla (nrk60)

Project Proposal

"Heading/Sub-Heading identification for provided paragraph"

The aim of this project is to develop a methodology that can be used to assign a title to a paragraph provided. Here we will try to build a corpus containing paragraphs of different sizes. Data will be picked up from various sources like Wikipedia, News Articles, Journals, and Forums etc. Following two sub-objectives will be kept in mind while trying to develop a methodology.

- Finding a single word from the paragraph that can be a title
- Finding the new word that has not occurred in the paragraph to be a title

Different approaches will be tried in identifying the title for two sub-objectives. We intend to read through the paragraph and identify a word or apply chunker that can best describe the whole paragraph. Also, we intend to give a title that could not be a word present in the paragraph.

Paragraphs are going to parsed independently and titles will be predicted without any knowledge of the pre or past occurring paragraphs. The research will involve identifying any relevant research and developing methodology using Natural Language Tools such as word vectors, tf-idf, word-net, pos taggers and but not just restricted to this. The scope of applicability of Machine learning will also be considered.

Data Collection

As there is no exactly same work conducted before as I am trying to achieve here, so the data is not readily available. I have put together some data collected form different sources. Close to 40 paragraphs are randomly collected from wikipedia articles each of close 5-10 sentences. Also sample of 250 news article is collected from the 'All the News' kaggle dataset.

In [5]:
```
import pandas as pd
```

In [6]:
```
#Reading the Paragraph Data
para_data1 = pd.read_csv('E:/NLP/Code/data_40.csv')

news = pd.read_csv("E:\\NLP\\Data\\articles1.csv")
sample = news['content'].sample(250, random_state = 22).reset_index(drop= True)

#Combine data
data = pd.DataFrame(para_data1["Paragraph"].append(sample).reset_index(drop= True
```

In [7]:
```
data.columns =['para']
```

In [8]:
```
#Let us look at one random paragraph from Wikipedia paragraphs
print(sample[125])
```

```
 (CNN) If you see a country music fan this week, you may want to give them a hu
g. If Twitter is any indication, folks are grieving over the announcement that
Blake Shelton and Miranda Lambert, the first couple of country music, are split
ting after four years of marriage. Some of it may be    or hyperbole, but news
of the breakup has been accompanied by the type of sadness worthy of a country
hit. Country stars Blake Shelton and Miranda Lambert split, One person even twe
eted, ”Now that Blake Shelton and Miranda Lambert are getting divorced I can sa
y with 100% confidence that love does not exist.” Of course, along with the gri
ef comes tons of speculation about what may have happened in the relationship b
etween ”The Voice” coach and his superstar wife.  Much is being made of Lamber
t’s tearful performance of ”The House That Built Me” at Cheyenne Frontier Days
in Wyoming on Saturday, days before the breakup was announced. Shelton told Peo
ple magazine that he played the song    which was originally written for him
for Lambert one night during a drive when the pair were engaged. She burst into
tears, he said, and he decided to give the song to her to record.  ”I took one
look at her and just said, ’If you have this strong of a reaction to a song, yo
u need to record it!’ ” Shelton told the publication in 2011.
```

In [9]:
```
#Let us look at one random paragraph from Wikipedia paragraphs
print(para_data1['Paragraph'][5])
```

```
Programmers typically implement plug-in functionality using shared libraries, w
hich get dynamically loaded at run time, installed in a place prescribed by the
host application. HyperCard supported a similar facility, but more commonly inc
luded the plug-in code in the HyperCard documents (called stacks) themselves. T
hus the HyperCard stack became a self-contained application in its own right, d
istributable as a single entity that end-users could run without the need for a
dditional installation-steps. Programs may also implement plugins by loading a
directory of simple script files written in a scripting language like Python or
Lua.
```

Approach

The main idea here is to identify title for the paragraphs, so we will try to fetch the central idea of the paragraph which could be suggested as the title for the paragraph. There can be lot of ways in which the central theme and idea of the paragraph can be obtained. Lets us look at one very naive approach using the words embeddings.

Word embedding are being used in the various forms in field of NLP. Considering word embeddings to calculate the paragraph vector can be a good start to the problem. Using the paragraph vector to check the closest word in the word embedding can be considered.

Idea 1 - Paragraph Vector

Post cleaning the paragraphs i.e. after removing the special characters, we tokenise the paragraph into words. Now, we remove the stop words as they generally dont really have strong meaning in the sentence or paragraph. Once we have the words, we can fetch the words vector for each of these words and calculate the average of the all word vector to obtain the paragraph vector. Post which we look for the top 10 word vectors that close to the paragraph vector and analyse it.

I have here considered the Glove Word Vector of 300d to calcuate the paragraph vector.

In [10]:
```
#Importing the required packages
import numpy as np

import re

import collections

from nltk.corpus import stopwords

from gensim.models import Word2Vec, KeyedVectors
from gensim.test.utils import datapath
from gensim.scripts.glove2word2vec import glove2word2vec
from gensim.test.utils import get_tmpfile
```

```
C:\Users\nitin\AppData\Local\Continuum\anaconda3\lib\site-packages\gensim\util
s.py:1212: UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [11]:
```
#Funcation to load the Word Embedding File
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in f:
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
```

In [12]:
```
#Reading the 300d Glove 6B word vector file
wordvector = loadGloveModel("glove.6B.300d.txt")
```

```
Loading Glove Model
Done. 400000  words loaded!
```

Data Clearning and formatting

In [13]:
```
data['cleaned'] = data['para'].apply(lambda x : re.sub('[^A-Za-z]+', ' ', x))
```

We here plan to keep only the strings i.e. remove all the characters that are not in alphabets including numbers.

In [14]:
```
data["para"][10]
```

Out[14]:
```
"The Bach family already counted several composers when Johann Sebastian was bo
rn as the last child of a city musician in Eisenach. After becoming an orphan a
t age 10, he lived for five years with his eldest brother, after which he conti
nued his musical development in Lüneburg. From 1703 he was back in Thuringia, w
orking as a musician for Protestant churches in Arnstadt and Mühlhausen and, fo
r longer stretches of time, at courts in Weimar—where he expanded his repertoir
e for the organ—and Köthen—where he was mostly engaged with chamber music. From
1723 he was employed as Thomaskantor (cantor at St. Thomas) in Leipzig. He comp
osed music for the principal Lutheran churches of the city, and for its univers
ity's student ensemble Collegium Musicum. From 1726 he published some of his ke
yboard and organ music. In Leipzig, as had happened in some of his earlier posi
tions, he had a difficult relation with his employer, a situation that was litt
le remedied when he was granted the title of court composer by the Elector of S
axony and King of Poland in 1736. In the last decades of his life he reworked a
nd extended many of his earlier compositions. He died of complications after ey
e surgery in 1750."
```

In [15]:
```
data["cleaned"][10]
```

Out[15]:
```
'The Bach family already counted several composers when Johann Sebastian was bo
rn as the last child of a city musician in Eisenach After becoming an orphan at
age he lived for five years with his eldest brother after which he continued hi
s musical development in L neburg From he was back in Thuringia working as a mu
sician for Protestant churches in Arnstadt and M hlhausen and for longer stretc
hes of time at courts in Weimar where he expanded his repertoire for the organ
and K then where he was mostly engaged with chamber music From he was employed
as Thomaskantor cantor at St Thomas in Leipzig He composed music for the princi
pal Lutheran churches of the city and for its university s student ensemble Col
legium Musicum From he published some of his keyboard and organ music In Leipzi
g as had happened in some of his earlier positions he had a difficult relation
with his employer a situation that was little remedied when he was granted the
title of court composer by the Elector of Saxony and King of Poland in In the l
ast decades of his life he reworked and extended many of his earlier compositio
ns He died of complications after eye surgery in '
```

We now create a paragraph vector function that would take in a paragraph p and return the average of embedding for the words present in the paragraph.

In [16]:
```
#Calculating Paragraph Vector
def para_vec(p):
    para_list = p.lower().split(" ")
    para_list = [word for word in para_list if word not in stopwords.words('engli
    count = 0
    feature_vec = np.zeros((300, ), dtype='float32')
    for i in para_list:
        if i in wordvector:
            count += 1
            feature_vec = np.add(feature_vec, wordvector[i])
    if (count > 0):
        feature_vec = np.divide(feature_vec, count)
    return feature_vec
```

In [98]:
```
#Applying the para_vec function on complete cleaned data
data['vector'] = data['cleaned'].apply(para_vec)
```

In [18]:
```
#Creating a word2vec model of the wordembedding so that methods of the word2vec c
glove_file = datapath('E:\\NLP\\Code\\glove.6B.300d.txt')
tmp_file = get_tmpfile("test_word2vec.txt")
glove2word2vec(glove_file, tmp_file)
```

Out[18]:
```
(400000, 300)
```

In [19]:
```
model = KeyedVectors.load_word2vec_format(tmp_file)
```

In [20]:
```
#Fetching the top ten word vectors that close to the paragraph vector
data['suggested_topics'] = data['vector'].apply(model.similar_by_vector)
```

```
C:\Users\nitin\AppData\Local\Continuum\anaconda3\lib\site-packages\gensim\matut
ils.py:737: FutureWarning: Conversion of the second argument of issubdtype from
`int` to `np.signedinteger` is deprecated. In future, it will be treated as `n
p.int32 == np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
```

In [21]:
```
data.to_csv("idea1_output.csv")
```

Observations

Some observations from Experiment 1 – Average Paragraph Vector Similar Words

• Paragraph 0 – Is a paragraph from article on Wikipedia and we can see that the Titles suggested are somewhat like – ‘Web’, ‘Online’, ‘Internet’. These topics are basically found to be the very general central theme of the paragraph. We can see from the paragraph, that most of the words occurring in the paragraph are related to the ‘Web’, ‘Online’ and ‘Internet’. I have referred such observation with Pattern column marked as Y. Around 16 of 40 wikipedia articles had tittle suggestions slightly close to the content of the paragraph.

In [22]:
```
print(data.loc[0,'para'])

print("Suggested topics for the paragraph are - ",data.loc[0,'suggested_topics'])
```

```
The online encyclopedia project Wikipedia is the most popular wiki-based websit
e, and is one of the most widely viewed sites in the world, having been ranked
in the top ten since 2007.[3] Wikipedia is not a single wiki but rather a colle
ction of hundreds of wikis, with each one pertaining to a specific language. In
addition to Wikipedia, there are tens of thousands of other wikis in use, both
public and private, including wikis functioning as knowledge management resourc
es, notetaking tools, community websites, and intranets. The English-language W
ikipedia has the largest collection of articles; as of September 2016, it had o
ver five million articles. Ward Cunningham, the developer of the first wiki sof
tware, WikiWikiWeb, originally described it as "the simplest online database th
at could possibly work".[4] "Wiki" (pronounced [?wiki][note 1]) is a Hawaiian w
ord meaning "quick".[5][6][7]
Suggested topics for the paragraph are -  [('web', 0.6910759210586548), ('onlin
e', 0.6712905764579773), ('example', 0.6659792065620422), ('only', 0.6428586840
629578), ('internet', 0.6358155012130737), ('addition', 0.6325753331184387),
('many', 0.6271064281463623), ('use', 0.6264867782592773), ('well', 0.625910222
530365), ('instance', 0.6258213520050049)]
```

• Paragraph 7 – central theme was captured well by the tittle suggested talking about ‘education’, ’graduate’.

In [23]:
```
print(data.loc[7,'para'])

print("Suggested topics for the paragraph are - ",data.loc[7,'suggested_topics'])
```

```
Cunningham was born in Michigan City, Indiana and grew up in Highland, Indiana,
staying there through high school.[4] He received his Bachelor's degree in inte
rdisciplinary engineering (electrical engineering and computer science) and his
master's degree in computer science from Purdue University, graduating in 1978.
[5] He is a founder of Cunningham & Cunningham, Inc. He has also served as Dire
ctor of R&D at Wyatt Software and as Principal Engineer in the Tektronix Comput
er Research Laboratory. He is founder of The Hillside Group and has served as p
rogram chair of the Pattern Languages of Programming conference which it sponso
rs. Cunningham was part of the Smalltalk community. From December 2003 until Oc
tober 2005, he worked for Microsoft Corporation in the "Patterns & Practices" g
roup. From October 2005 to May 2007, he held the position of Director of Commit
ter Community Development at the Eclipse Foundation.
Suggested topics for the paragraph are -  [('university', 0.6784282922744751),
('science', 0.6489746570587158), (',', 0.6472765207290649), ('research', 0.6392
382979393005), ('graduate', 0.6260223388671875), ('engineering', 0.622553229331
9702), ('technology', 0.6221566796302795), ('education', 0.6161079406738281),
('.', 0.6139761209487915), ('addition', 0.6116740703582764)]
```

• Paragraph 14 – ‘government’ and ‘states’ is a repeating word in the paragraph, so it seems obvious to be present as suggested tittle as the common repeating words skews the paragraph vector. This pattern was common across many paragraphs.

In [24]:
```
print(data.loc[14,'para'])

print("Suggested topics for the paragraph are - ",data.loc[14,'suggested_topics'])
```

```
The states of the Weimar Republic were effectively abolished after the establis
hment of Nazi Germany in 1933 by a series of Reichsstatthalter decrees between
1933 and 1935, and autonomy was replaced by direct rule of the National Sociali
st German Workers' Party in the Gleichschaltung process. The states continued t
o formally exist as de jure rudimentary bodies, but from 1934 were superseded b
y de facto Nazi provinces called Gaue. Many of the states were formally dissolv
ed at the end of World War II by the Allies, and ultimately re-organised into t
he modern states of Germany.
Suggested topics for the paragraph are -  [('which', 0.6483539342880249), ('sta
tes', 0.6448923349380493), ('government', 0.6426990628242493), ('however', 0.64
23750519752502), ('part', 0.6297308206558228), ('war', 0.6249045133590698), ('a
lthough', 0.621939480304718), ('.', 0.6192313432693481), ('rule', 0.61401474475
8606), ('governments', 0.6111968755722046)]
```

• Paragraph 28 – Is a short paragraph consisting about 3-4 lines. And titles suggested are interesting and made sense.

In [25]:
```
print(data.loc[28,'para'])

print("Suggested topics for the paragraph are - ",data.loc[28,'suggested_topics'])
```

```
When terrorism began to be treated as a more active threat after the September
11 attacks, BART increased its emphasis on infrastructure protection. The polic
e department hosts drills and participates in counter-terrorism working groups.
The agency has an officer assigned full-time to the FBI's Joint Terrorism Task
Force. Furthermore, a command officer is designated as a mutual-aid, counter-te
rrorism, and homeland-security liaison. BART's police dogs are certified in exp
losives detection.
Suggested topics for the paragraph are -  [('security', 0.7271169424057007),
('terrorism', 0.6751623153686523), ('enforcement', 0.6623741388320923), ('offic
ials', 0.6602051258087158), ('military', 0.6486713886260986), ('forces', 0.6454
729437828064), ('force', 0.6323009729385376), ('officers', 0.6316796541213989),
('terrorist', 0.6303951144218445), ('police', 0.6276746988296509)]
```

• Most frequent title suggested were ‘example’, ‘many’, ‘well’, ‘because’, ‘same’, ‘used’, 'but', 'that' for most of the paragraphs including the new articles (except very few news articles)

In [26]:
```
data['suggested_topics'][0][1][0]
```

Out[26]:
```
'online'
```

In [27]:
```
#checking the most frequent title suggested
wordcount={}
for i in data['suggested_topics']:
    for j in i:
        if j[0] not in wordcount:
            wordcount[j[0]] = 1
        else:
            wordcount[j[0]] += 1

topics = collections.Counter(wordcount)
topics.most_common(20)
```

Out[27]:
```
[('but', 248),
 ('that', 205),
 ('even', 197),
 ('because', 185),
 ('not', 152),
 ('what', 133),
 ('so', 122),
 ('.', 101),
 ('they', 98),
 ('would', 95),
 ('only', 64),
 ('saying', 52),
 ('should', 51),
 ('come', 43),
 ('well', 42),
 ("n't", 40),
 ('same', 37),
 ('one', 36),
 ('when', 35),
 ('last', 33)]
```

Idea 2 - Extracting subject matter (first step of chunker)

To identify title we need to understand what the subject is talking about. Extracting Subject out of a paragraph can be really helpful as this is the main subject of the paragraph. Currently we are expecting the subject to be the named entity among the nouns present in the paragraph.

We will extract the subject main word or phrase by first calculating the word frequency distribution. Then, the most frequent nouns are collected. We use NLTK's built-in methods to achieve that.

Extracting Subject

In [106]:
```
#Importing Libraries

from bs4 import BeautifulSoup
import requests
import re
import pickle
import nltk
from nltk.corpus import stopwords
from nltk import DefaultTagger, UnigramTagger, BigramTagger, TrigramTagger
stop = stopwords.words('english')

# Noun Part of Speech Tags used by NLTK
NOUNS = ['NN', 'NNS', 'NNP', 'NNPS']
VERBS = ['VB', 'VBG', 'VBD', 'VBN', 'VBP', 'VBZ']
```

Lets us define some set of functions that can be used to extract the subject out of the paragraph provided. First of all, we need a function that will do the cleaning of the paragraph passed.

In [107]:
```
# Cleaning the paragraph
def clean_document(paragraph):
    paragraph = re.sub('[^A-Za-z .-]+', ' ', paragraph)
    paragraph = ' '.join(paragraph.split())
    paragraph = ' '.join([i for i in paragraph.split() if i not in stop])
    return paragraph
```

Next, let us define a function, which when called on paragraph will tokenize the sentences present in the paragraph.

In [108]:
```
#Tokenizing the sentences
def tokenize_sentences(paragraph):
    sentences = nltk.sent_tokenize(paragraph)
    sentences = [nltk.word_tokenize(sent) for sent in sentences]
    return sentences
```

Extracting entities would be a main step, if we want to identify the subject. We believe that the majority of the subject to be a named entity. To extract Named Entity we use the 'ne_chunk' along with 'pos_tag' from the package NLTK.

In [109]:
```
#Extract Named Entities using NLTK Chinking
def get_entities(paragraph):
    entities = []
    sentences = tokenize_sentences(paragraph)

    # Part of Speech Tagging
    sentences = [nltk.pos_tag(sent) for sent in sentences]
    for tagged_sentence in sentences:
        for chunk in nltk.ne_chunk(tagged_sentence):
            if type(chunk) == nltk.tree.Tree:
                entities.append(' '.join([c[0] for c in chunk]).lower())
    return entities
```

Also, we would like to know frequency of the word that are present in paragraph, the importance of this step will be eventually clear.

In [110]:
```
# Return the Word count frequency
def word_freq_dist(paragraph):
    words = nltk.tokenize.word_tokenize(paragraph)
    words = [word.lower() for word in words if word not in stop]
    fdist = nltk.FreqDist(words)
    return fdist
```

It is clear that the subject of the paragraph should be occuring majority of times. Any named entity present multiple times in paragraph and which is a noun is expected to be Subject of the paragraph than a named entity present one or two times. Based on this logic we try to find the subject which is most commonly repeating and return it.

In [115]:
```
#Extracting the main subject out of the paragraph
def extract_subject(document):
    # Get most frequent Nouns
    fdist = word_freq_dist(document)
    most_freq_nouns = [w for w, c in fdist.most_common(10)
                       if nltk.pos_tag([w])[0][1] in NOUNS]

    # Get Top 10 entities
    entities = get_entities(document)
    top_10_entities = [w for w, c in nltk.FreqDist(entities).most_common(10)]

    # Get the subject noun by looking at the intersection of top 10 entities
    # and most frequent nouns. It takes the first element in the list
    subject_nouns = [entity for entity in top_10_entities
                     if entity.split()[0] in most_freq_nouns]
    #print (subject_nouns)
    return subject_nouns[0]

def for_all_subjects(x):
    try:
        return extract_subject(x)
    except:
        return []
```

In [117]:
```
data['subject_nouns'] = data['para'].apply(for_all_subjects)
```

Let us look at the output and analyse it. As we can see below, - 'Gerald' is the main subject in the paragraph provided.

In [118]:
```
data['para'][23]
```

Out[118]:
```
'Upon the death of his uncle, the Bishop of St David\'s, in 1176, the chapter n
ominated Gerald as his successor. St David\'s had the long-term aim of becoming
independent of Canterbury, and the chapter may have thought that Gerald was the
man to take up its cause. Henry II of England, fresh from his struggle with Tho
mas Becket, promptly rejected Gerald, possibly because his Welsh blood and ties
to the ruling family of Deheubarth made him seem like a troublesome prospect, i
n favour of one of his Norman retainers Peter de Leia. According to Gerald, the
king said at the time: "It is neither necessary or expedient for king or archbi
shop that a man of great honesty or vigour should become Bishop of St. David
\'s, for fear that the Crown and Canterbury should suffer thereby. Such an appo
intment would only give strength to the Welsh and increase their pride".[3] The
chapter acquiesced in the decision; and Gerald, disappointed with the result, w
ithdrew to the University of Paris. From c.?1179-8, he studied and taught canon
law and theology. He returned to England and spent an additional five years stu
dying theology. In 1180, he received a minor appointment from the Bishop of St.
David\'s, which he soon resigned.[1]'
```

In [119]:
```
data['subject_nouns'][23]
```

Out[119]:
```
'gerald'
```

In [120]:
```
data['para'][56]
```

Out[120]:
```
' (CNN) Another ePrix, another victory for Sebastien Buemi.  The Renault eDams
driver made it three wins out of three for the   season at the Buenos Aires ePr
ix with another impressive drive at the Puerto Madero Street Circuit on Saturda
y.  The reigning world champion led for the majority of the   race after starti
ng third on the grid behind pole sitter Lucas di Grassi    a first for the ABT
Schaeffler Audi Sport driver    and Techeetah's   Vergne.  Vergne seized the le
ad from di Grassi on the third lap but by lap six it was Buemi who had hit the
front    and the Swiss driver never looked back.    The    took the checkered f
lag a comfortable three seconds clear of Vergne to seal a third consecutive win
    the first driver to achieve the feat in Formula E    and his ninth overall
in the   race series.  READ: How virtual racing breeds   success, READ: How 'hu
mble' star landed F1's hottest drive  Buemi won the   in Hong Kong last October
and then repeated the feat in Marrakech the following month.  This latest win g
ives the Swiss a maximum 75 points and a   cushion over nearest rival di Grass
i.   But with nine races to go, Buemi is determined not to take his foot off th
e gas.  "We need to build up some points . .. these things never last for eve
r," Buemi said.  "My team did a great job    let's hope it continues for a few
more races. At the end of the day I'm really happy."  Di Grassi, who pushed Bue
mi all the way in last year's championship, ended up claiming third to notch a
15th podium in 24 career starts in Formula E.   Vergne scored his first podium
of the season with Buemi's teammate Nico Prost coming home fourth followed by N
extEV's Nelson Piquet Jr. DS Virgin Racing's Jose Maria Lopez, the only Argenti
ne driver in the field, finished 10th to secure a point but there was disappoin
tment for teammate Sam Bird. The Briton was defending the title he won last yea
r but limped home last after suffering damage to the left rear of his car early
on in the race.  Buenos Aires ePrix top 10 finishers:  1. Sebastien Buemi  2.
Vergne  3. Lucas di Grassi  4. Nico Prost  5. Nelson Piquet Jr.  6. Loic Duval
7. Daniel Abt  8. Jerome D'Ambrosio  9. Oliver Turvey  10. Jose Maria Lopez  Yo
u can watch highlights of the Buenos Aires ePrix on Supercharged's February sho
w    click this link to go to our motorsport page where you will find show time
s.   Round four of the   Formula E World Championship takes place in Mexico on
April 1.  '
```

In [121]:
```
data['subject_nouns'][56]
```

Out[121]:
```
'buemi'
```

Idea 2 - Extracting subject matter (second step of chunker)

Clearly, we can see that Subjects have been correctly identified. Now, let us extract the verb and object, combining them with object and verb to obtain subject matter of the paragraph. Before we proceed any further, we would need to train trigram tagger, let us use the nltk corpus for this purpose.

In [67]:
```
train_sents = nltk.corpus.brown.tagged_sents()
train_sents += nltk.corpus.conll2000.tagged_sents()
train_sents += nltk.corpus.treebank.tagged_sents()
```

In [68]:
```
t0 = DefaultTagger('NN')
t1 = UnigramTagger(train_sents, backoff=t0)
t2 = BigramTagger(train_sents, backoff=t1)
tagger = TrigramTagger(train_sents, backoff=t2)
```

In [75]:
```
def merge_multi_word_subject(sentences, subject):

    if len(subject.split()) == 1:
        return sentences
    subject_lst = subject.split()
    sentences_lower = [[word.lower() for word in sentence]
                       for sentence in sentences]
    for i, sent in enumerate(sentences_lower):
        if subject_lst[0] in sent:
            for j, token in enumerate(sent):
                start = subject_lst[0] == token
                exists = subject_lst == sent[j:j+len(subject_lst)]
                if start and exists:
                    del sentences[i][j+1:j+len(subject_lst)]
                    sentences[i][j] = subject
    return sentences

def tag_sentences(subject, paragraph):
    """Returns tagged sentences using POS tagging"""
    trigram_tagger = tagger

    # Tokenize Sentences and words
    sentences = tokenize_sentences(paragraph)
    merge_multi_word_subject(sentences, subject)

    # Filter out sentences where subject is not present
    sentences = [sentence for sentence in sentences if subject in
                 [word.lower() for word in sentence]]

    # Tag each sentence
    tagged_sents = [trigram_tagger.tag(sent) for sent in sentences]
    return tagged_sents
```

The nouns that being identified are seperated for example, Donald is tagged as one Noun and Trump as another, but they occus together, so the above function will help us in mergeing multi noun words into single token.

Extract Subject Action Object (subject matter)

Based on the subject determined earlier, the action is identified with particular related to the subject using Verbs. The noun coming after verb is considered as object. Now, combining all the three gives us the subject matter we have been looking for.

In [82]:
```
def get_svo(sentence, subject):
    subject_idx = next((i for i, v in enumerate(sentence)
                        if v[0].lower() == subject), None)
    data = {'subject': subject}
    for i in range(subject_idx, len(sentence)):
        found_action = False
        for j, (token, tag) in enumerate(sentence[i+1:]):
            if tag in VERBS:
                data['action'] = token
                found_action = True
            if tag in NOUNS and found_action == True:
                data['object'] = token
                data['phrase'] = sentence[i: i+j+2]
                return data
    return {}
```

In [90]:
```
def final_out(document):
    document = clean_document(document)
    subject = extract_subject(document)
    tagged_sents = tag_sentences(subject, document)
    svos = [get_svo(sentence, subject) for sentence in tagged_sents]
    return svos
```

In [94]:
```
def for_all(x):
    try:
        return final_out(x)
    except:
        return 'NA'
```

In [95]:
```
data['Subject_Verb_Object']=data['para'].apply(for_all)
```

In [146]:
```
data.to_csv("Idea2.csv")
```

Results

Different phrases generated can be found below for a sample set of articles. The results for the overall have been saved in the output_sao csv file.

In [124]:
```
data['para'][56]
```

Out[124]:
```
' (CNN) Another ePrix, another victory for Sebastien Buemi.  The Renault eDams
driver made it three wins out of three for the   season at the Buenos Aires ePr
ix with another impressive drive at the Puerto Madero Street Circuit on Saturda
y.  The reigning world champion led for the majority of the   race after starti
ng third on the grid behind pole sitter Lucas di Grassi    a first for the ABT
Schaeffler Audi Sport driver    and Techeetah's   Vergne.  Vergne seized the le
ad from di Grassi on the third lap but by lap six it was Buemi who had hit the
front    and the Swiss driver never looked back.    The    took the checkered f
lag a comfortable three seconds clear of Vergne to seal a third consecutive win
    the first driver to achieve the feat in Formula E    and his ninth overall
in the   race series.  READ: How virtual racing breeds   success, READ: How 'hu
mble' star landed F1's hottest drive  Buemi won the   in Hong Kong last October
and then repeated the feat in Marrakech the following month.  This latest win g
ives the Swiss a maximum 75 points and a   cushion over nearest rival di Grass
i.   But with nine races to go, Buemi is determined not to take his foot off th
e gas.  "We need to build up some points . .. these things never last for eve
r," Buemi said.  "My team did a great job    let's hope it continues for a few
more races. At the end of the day I'm really happy."  Di Grassi, who pushed Bue
mi all the way in last year's championship, ended up claiming third to notch a
15th podium in 24 career starts in Formula E.   Vergne scored his first podium
of the season with Buemi's teammate Nico Prost coming home fourth followed by N
extEV's Nelson Piquet Jr. DS Virgin Racing's Jose Maria Lopez, the only Argenti
ne driver in the field, finished 10th to secure a point but there was disappoin
tment for teammate Sam Bird. The Briton was defending the title he won last yea
r but limped home last after suffering damage to the left rear of his car early
on in the race.  Buenos Aires ePrix top 10 finishers:  1. Sebastien Buemi  2.
Vergne  3. Lucas di Grassi  4. Nico Prost  5. Nelson Piquet Jr.  6. Loic Duval
7. Daniel Abt  8. Jerome D'Ambrosio  9. Oliver Turvey  10. Jose Maria Lopez  Yo
u can watch highlights of the Buenos Aires ePrix on Supercharged's February sho
w    click this link to go to our motorsport page where you will find show time
s.   Round four of the   Formula E World Championship takes place in Mexico on
April 1.  '
```

In [126]:
```
data['Subject_Verb_Object'][56][1]['phrase']
```

Out[126]:
```
[('Buemi', 'NN'), ('hit', 'VBD'), ('front', 'NN')]
```

In [133]:
```
data['Subject_Verb_Object'][56][2]['phrase']
```

Out[133]:
```
[('Buemi', 'NN'),
 ('Hong', 'NNP'),
 ('Kong', 'NNP'),
 ('last', 'JJ'),
 ('October', 'NNP'),
 ('repeated', 'VBD'),
 ('feat', 'NN')]
```

In [134]:
```
data['Subject_Verb_Object'][56][3]['phrase']
```

Out[134]:
```
[('Buemi', 'NN'), ('determined', 'VBN'), ('take', 'VB'), ('foot', 'NN')]
```

'Buemi determined take foot' seems to be good title suggestion for the paragraph.

In [132]:
```
data['para'][27]
```

Out[132]:
```
"Litigation $50 million (originally $25 million) lawsuit by John Burris against
BART on behalf of Grant's mother and daughter was settled for $2.8 million; Gra
nt's father's lawsuit was denied\nOscar Grant III was a 22-year-old African-Ame
rican man who was fatally shot in the early morning hours of New Year's Day 200
9 by BART Police Officer Johannes Mehserle in Oakland, California. Responding t
o reports of a fight on a crowded Bay Area Rapid Transit train returning from S
an Francisco, BART Police officers detained Grant and several other passengers
on the platform at the Fruitvale BART Station. Two officers, including Mehserl
e, forced the unarmed Grant to lie face down on the platform. Mehserle drew his
pistol and shot Grant in the back. Grant was rushed to Highland Hospital in Oak
land and pronounced dead later that day. The events were captured on multiple o
fficial and private digital video and privately owned cell phone cameras. Owner
s disseminated their footage to media outlets and to various websites where it
became viral. Both peaceful and violent protests of police actions took place i
n the following days."
```

In [136]:
```
data['Subject_Verb_Object'][27][1]['phrase']
```

Out[136]:
```
[('Grant', 'NP'),
 ('father', 'NN'),
 ('lawsuit', 'NN'),
 ('denied', 'VBN'),
 ('Oscar', 'NP'),
 ('Grant', 'NP'),
 ('III', 'NNP')]
```

In [140]:
```
data['Subject_Verb_Object'][27][3]['phrase']
```

Out[140]:
```
[('Grant', 'NP'), ('lie', 'VB'), ('face', 'NN')]
```

In [141]:
```
data['Subject_Verb_Object'][27][5]['phrase']
```

Out[141]:
```
[('Grant', 'VB'), ('rushed', 'VBN'), ('Highland', 'NNP')]
```

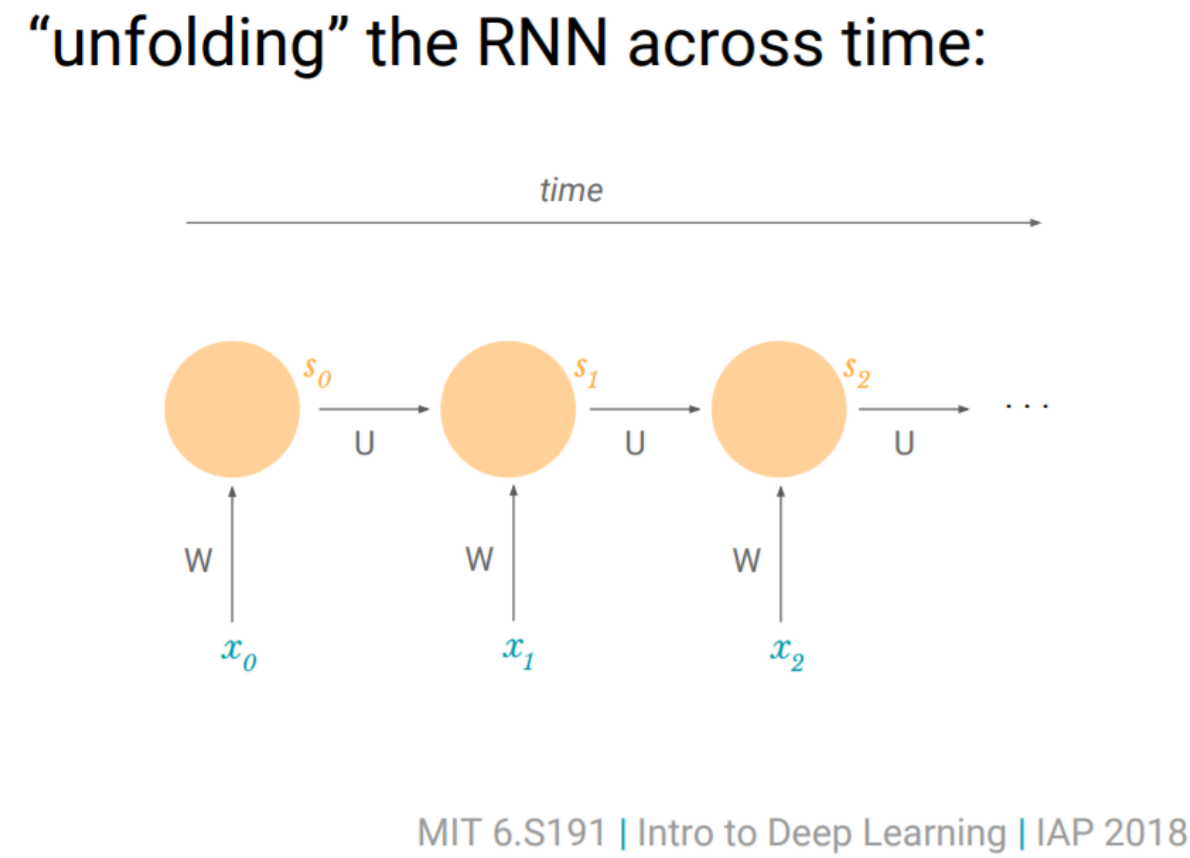
Both, 'Grant father lawsuit denied' and 'Grant rushed Highland' both looks like good titles for the article at hand. So far, we have tried to develop a methodology based on our understanding to determine the title. However, let us see if any supervised learning model such a deep neural networks can be applied here to predict titles for the paragraphs.

Application of Deep Learning

Deep Neural Networks are powerful models that have achieved excellent results recently in many different task. While Convoluted Neural Networks is good at solving most of the image related problems, Recurrent Neural Network could be a good choice for the problems involving sequence like current problem at hand.

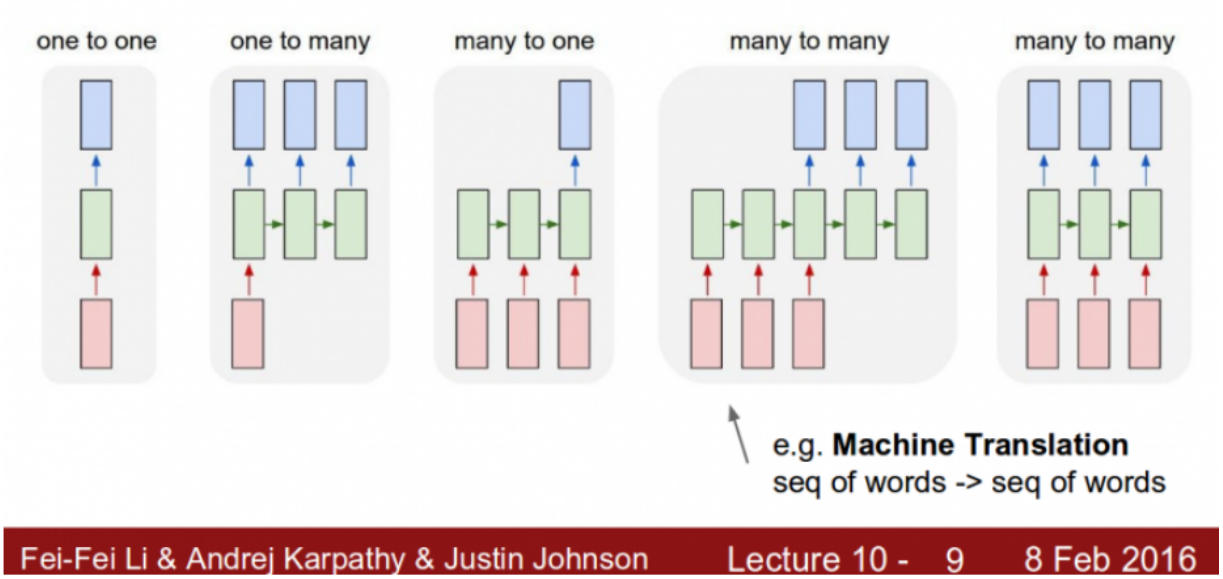
Recurrent Neural Networks

For instance, language as we saw earlier- the sequence of words define their meaning. We are trying to use such data for any reasonable output, we need a network which has access to some prior knowledge about the data to completely understand it. The architecture of Recurrent Neural Networks take into account the requirement of prior knowledge.



Sequence to Sequence

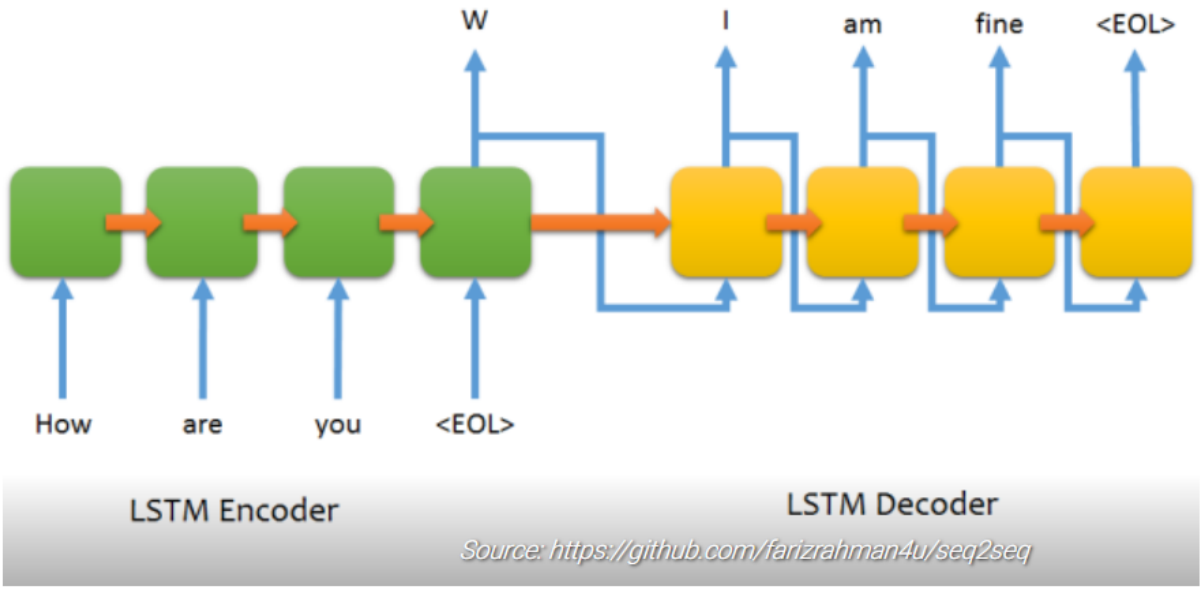
Recurrent Neural Networks is our go-to architecture, for any sequence modelling problems. But there are many types of sequence related problems possible.



Where the input as well as the output are a sequence, this kind of preblem is called sequence to sequence models. And, it is widely used in Machine Translation and Video Captioning.

Architecture and Technique

Sequence to sequence model has two parts – an encoder and a decoder. Both of them are basically two independant Neural Networks models combined. The encoder generates a one output for complete input and is passed into the decoder network to start predicting a sequence.



Let us consider training the Sequence to Sequence deep neural model to predict the title for the passage passed.

Data

Fake or real article dataset is considered to train the model, which is almost similar to the one dataset used earlier. To train a deep model with high accuracy we will need an substantial amount of data.

The model training and codes are present in the different notebook - 'Extension - Sequence to Sequence Model using Glove'.

Model Description

Details of the model are as described below : -

```
Maximum Input Sequence Length = 500
Maximum Target Sequence Length = 50
Maximum Input Vocabulary Size= 8000
Maximum Target Vocabulary Size = 2000

Dimension for the Glove Word Embeddings = 100

Training Details :
    Hidden layers for each Encode and Decoder Model = 100
    Batch Size = 16
    Epochs = 30 (as high as 150 were trained)
    Optimizer = rmsprop
    Activation = Softmax
    Loss = Categorical Cross Entropy
```

Summary of the model is below :-

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
encoder_inputs (InputLayer)     (None, None, 100)    0
__________________________________________________________________________________________________
decoder_inputs (InputLayer)     (None, None, 2001)   0
__________________________________________________________________________________________________
encoder_lstm (LSTM)             [(None, 100), (None, 80400       encoder_inputs[0][0]
__________________________________________________________________________________________________
decoder_lstm (LSTM)             [(None, None, 100),  840800      decoder_inputs[0][0]
                                                                 encoder_lstm[0][1]
                                                                 encoder_lstm[0][2]
__________________________________________________________________________________________________
decoder_dense (Dense)           (None, None, 2001)   202101      decoder_lstm[0][0]
==================================================================================================
Total params: 1,123,301
Trainable params: 1,123,301
Non-trainable params: 0
__________________________________________________________________________________________________
```

Results of the model

Let us look at some of the generated headlines :-

```
Generated Headline:  california today: today: the u.s. in the the world
Original Headline:  Bill Paxton, Star of ‘Big Love’ and Movie Blockbuster
s, Dies at 61

Generated Headline:  north and china is killed by the watch: china
Original Headline:  Oakland Fire Victims Included Performers, Educators a
nd Lawyers

Generated Headline:  china says u.s. american american have
Original Headline:  ‘Hamilton’ Inc.: The Path to a Billion-Dollar Broadwa
y Show

Generated Headline:  a new york times with a new york times
Original Headline:  Mute and Alone, He Was Never Short of Kind Words or F
riends
```

Most of these dont seem to make sense for the model predicted. However, the model has learnt to predict to sequence of words.

With the limited data and resources, the sequence to sequence model was not able to perform as expected, with accuracy of less than 10%. Adding of additional data led to memory error. Majority of the trainings led to prediction of single output for all the inputs due to not sufficient epochs or data. Irrespective of trying various combinations of the hyperparameters the model was not able to perform as expected.

Future Scope

The results looks promising in the paper - 'Generating News Headlines with Recurrent Neural Networks' by Konstantin Lopyrev by addition of attention layer in the encoder - decoder model. Training the model with attention layer and increasing data size can help us in achieving better results. This can be one further development to the project.

References and Acknowledgement

For Idea 1 and Idea 2 : -

https://streamhacker.com/2009/02/23/chunk-extraction-with-nltk/ (https://streamhacker.com/2009/02/23/chunk-extraction-with-nltk/)

https://www.kaggle.com/snapcrack/all-the-news (https://www.kaggle.com/snapcrack/all-the-news)

https://medium.com/@acrosson/extract-subject-matter-of-documents-using-nlp-e284c1c61824 (https://medium.com/@acrosson/extract-subject-matter-of-documents-using-nlp-e284c1c61824)

http://www.winwaed.com/blog/2011/11/08/part-of-speech-tags/ (http://www.winwaed.com/blog/2011/11/08/part-of-speech-tags/)

For Deep learning model :-

https://nlp.stanford.edu/courses/cs224n/2015/reports/1.pdf (https://nlp.stanford.edu/courses/cs224n/2015/reports/1.pdf)

https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html (https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html)

https://machinelearningmastery.com/define-encoder-decoder-sequence-sequence-model-neural-machine-translation-keras/ (https://machinelearningmastery.com/define-encoder-decoder-sequence-sequence-model-neural-machine-translation-keras/)

https://machinelearningmastery.com/develop-encoder-decoder-model-sequence-sequence-prediction-keras/ (https://machinelearningmastery.com/develop-encoder-decoder-model-sequence-sequence-prediction-keras/)

https://github.com/chen0040/keras-text-summarization (https://github.com/chen0040/keras-text-summarization)

https://nbviewer.jupyter.org/github/hamelsmu/Seq2Seq_Tutorial/blob/master/notebooks/Tutorial.ipynb (https://nbviewer.jupyter.org/github/hamelsmu/Seq2Seq_Tutorial/blob/master/notebooks/Tutorial.ipynb